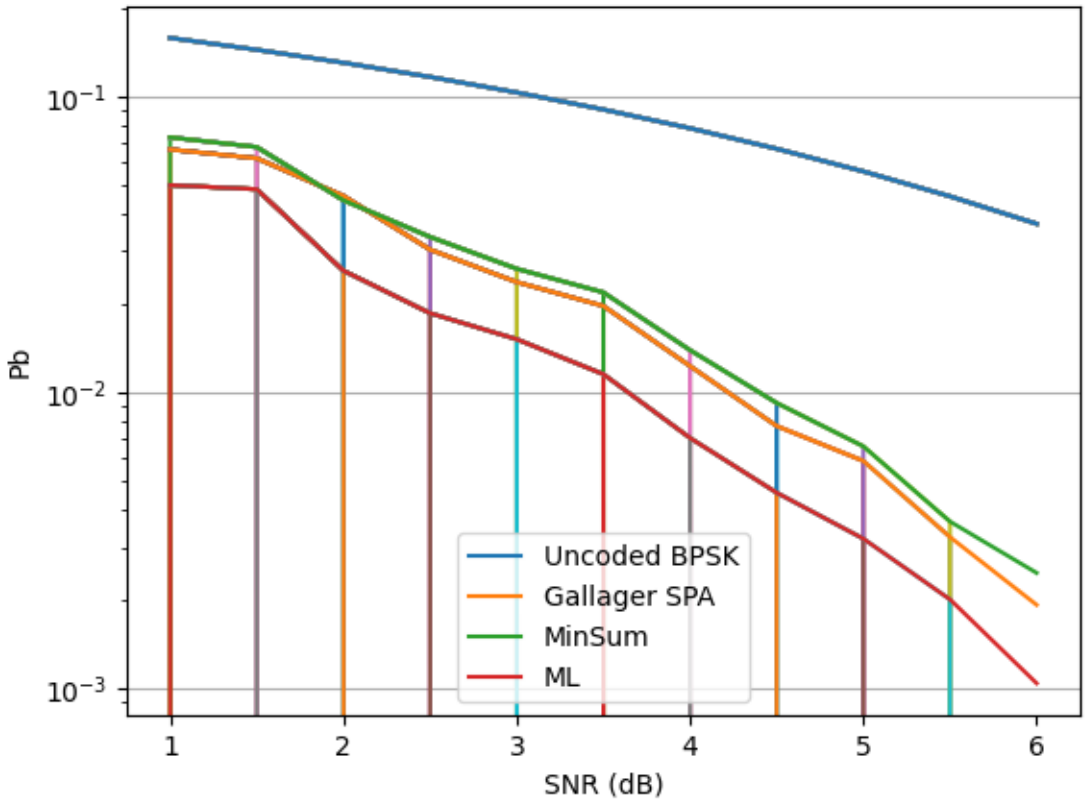
Mandatory3

Kristian Jensen

November 2021

Exercise 3



We can see that the ML-decoding always perform significantly better than MinSum & Gallager, however its decoding complexity increases by 2**k, where

k is the amount of rows in the generator matrix G. This is acceptable for smaller codes, which is why one want to consider Gallager or MinSum as decoding.
We can see that the Gallager-algorithm performs, overall, better than the Min-Sum algorithm. However it is more computational demanding. For application where one favors speed over precision the MinSum algorithm is preferable.

Exercise 4

$r = (-0.2, 0.3, -1.2, 0.5, -0.8, -0.6, 1.1)$
First we compute L_C, and find the two smallest values in each row.

$$L_C = \begin{pmatrix} -0.2 & 0.3 & -1.2 & 0 & -0.8 & 0 & 0 \\ 0 & 0.3 & -1.2 & 0.5 & 0 & -0.6 & 0 \\ -0.2 & 0.3 & 0 & 0.5 & 0 & 0 & 1.1 \\ -0.2 & 0 & -1.2 & 0 & -0.8 & -0.6 & 1.1 \end{pmatrix} |min\_vals| = \begin{pmatrix} 0.2 & 0.3 \\ 0.3 & 0.5 \\ 0.2 & 0.3 \\ 0.2 & 0.6b \end{pmatrix}$$

The computation of $L_{i \to y}$ from (5.7) in the lecture notes.
$L_{c1} = (-1)^4 : P = -1 : - + -- \to + - + +$
$L_{c2} = (-1)^4 = 1 : P = 1 : + - +- \to + - + -$
$L_{c3} = (-1)^4 = 1 : P = -1 : - + ++ \to + - - -$
$L_{c4} = (-1)^5 = -1 : P = 1 : - - - - + \to + + + + -$
Overall we get,

$$Lv = \begin{pmatrix} 0.3 & -0.2 & 0.2 & 0 & 0.2 & 0 & 0 \\ 0 & 0.5 & -0.3 & 0.3 & 0 & -0.3 & 0 \\ 0.3 & -0.2 & 0 & -0.2 & 0 & 0 & -0.2 \\ 0.6 & 0 & 0.2 & 0 & 0.2 & 0.2 & -0.2 \end{pmatrix}$$

$L^{tot} := (1.0, 0.4, -1.1, 0.6, -0.4, -0.7, 0.7)$
$\hat{v} := (1, 1, 0, 1, 0, 0, 1)$
$H * \hat{v} = \vec{0}$. meaning that $\hat{v}$ is a valid code-word, and the algorithm terminates.

Exercise 5

1 For a check node to not be able to uniquely determine the beliefs for its adjacent variable nodes needs to have more than one incoming '?'. i.e for a variable node to not be decoded uniquely it cannot appear in a check node function where it is the only variable node with belief equal ∞.
It is easy to see that that the code always can correct 1 error.
Then placing the '?' at the most destructive places are i) at variable nodes whose check nodes have few incoming edges, or ii) at variable nodes with many outgoing edges.
Placing 2 incoming erasures to any one check node, one of those erasures can be uniquely decoded in one iteration of belief. Meaning that in the second iteration there is only, at most, one error. Which, can be uniquely decoded.

2 r = (x, ?, x, ?, x, ?, x) $c1 = x + x + x + x = 0 c2 = ? + x + ? + x = ? + ? c3 = ? + x + ? + x = ? + ?$

r = (?, x, x, ?, ?, x, x) $c1 = ? + x ? + x = ? + ? c2 = x + x + x + ? = 0 c3 = ? + ? + x + x = ? + ?$

r = (?, ?, ?, x, x, x, x) $c1 = ? + ? + x + x = ? + ? c2 = ? + ? + x + x = 0 c3 = x + x + x + x = 0$

We see that there is no '?' appearing uniquely in any of the nodes sorted many to few edges = v7, v3, v5, v6, v1, v2, v4

3 (bonus) Characterize (find necessary and sufficient conditions) all the messages with exactly three erasures that cannot be decoded exactly.

For one check node not being able to uniquely decode one variable node, it needs to have more than one incoming belief = 0.
So placing the the ? in the most destructive places at the variable nodes whose check nodes have few incoming edges. We see that all check nodes have equal amount of incoming edges.
The next step is to target variable nodes with many edges.
setting ? at v6 and v7, we see that the only check node having one 0 incoming belief is check node 1. from this we get (?,x,x,x,x,?,?), (x,x,?,x,x,?,?), (x,x,x,x,?,?,?)
and then (keeping ?'s at v6 & v7) we target variable nodes w many outgoing edges.
(x,x,?,x,x,?,?), (x,x,x,x,?,?,?), (x,x,?,x,?,?,x)