Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Πολυτεχνική Σχολή

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Μελέτη αλγορίθμων για message passing decoding και σχεδίαση LDPC κωδίκων

Εργασία στο μάθημα "Κώδικες Διόρθωσης Σφαλμάτων"

Ιανουάριος 2021

Χαραλαμπίδης Βασίλειος

Παπαδόπουλος Ιωάννης

Παχατουρίδου Μαρία

Περίληψη

Η παρούσα εργασία ξεκινάει με την υλοποίηση ενός αλγορίθμου για βελτιστοποίηση υπολογισμών περιθωρίων όταν το factor graph που αναφέρεται στην παραγοντοποίηση της συνάρτησης υπολογισμού είναι δένδρο. Δίνεται ένα μικρό παράδειγμα όπου φαίνεται η διαφορά σε σχέση με ένα brute force αλγόριθμο. Το δεύτερο κομμάτι ασχολείται με την πραγματοποίηση ενός belief propagation decoder που κάνει τους υπολογισμούς βασισμένος στις αρχές των factor graphs. Προσομοιώνεται η λειτουργία του αποκωδικοποιητή για ένα συγκεκριμένο κώδικα και παρατηρούνται τα αποτελέσματα που μπορεί να πετύχει σε κανάλια BSC με διαφορετική πιθανότητα σφάλματος. Τέλος ασχολούμαστε με την σχεδίαση κωδίκων LDPC και υλοποιούμε ένα συγκεκριμένο παράδειγμα. Γίνεται συζήτηση επί των ερωτημάτων που γεννιούνται από τη θεωρητική επίδοση του υπό εξέταση κώδικα σε σχέση με αυτή που παρατηρούμε στις προσομοιώσεις. Προσπαθούν να αποδοθούν εξηγήσεις σε όσα παρατηρήθηκαν και μας τράβηξαν την προσοχή.

Χαραλαμπίδης Βασίλειος (8681), *xaralampidisbasilis@gmail.com*

Παπαδόπουλος Ιωάννης (8586), *ioapapchr@ece.auth.gr*

Παχατουρίδου Μαρία (8418), *mpachato@ece.auth.gr*

Επιβλέποντες

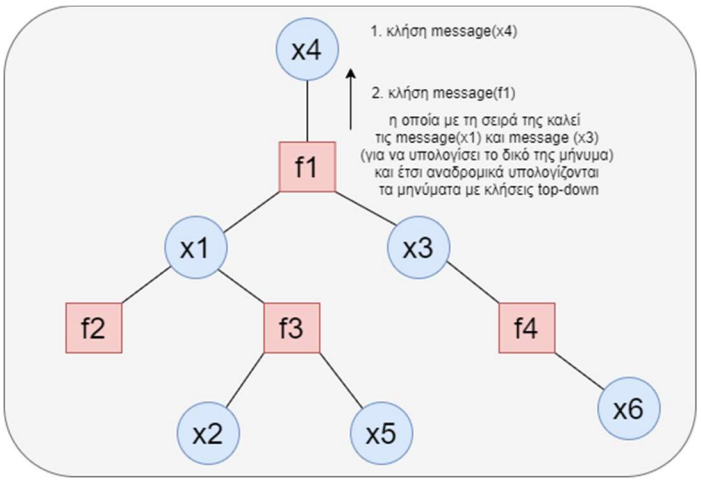
Γεωργιάδης Λεωνίδας, Καθηγητής

Καραγιαννίδης Γεώργιος, Καθηγητής

1. Υλοποίηση message passing αλγορίθμου υπολογισμού περιθωρίων (marginals)

Στο κομμάτι αυτό δημιουργήθηκε ένας αλγόριθμος για τον υπολογισμό της περιθώριας κατανομής για μια μεταβλητή σύμφωνα με τις σημειώσεις της θεωρίας. Ο βασικός αλγόριθμος βρίσκεται στο αρχείο *marginal.m* και περισσότερες πληροφορίες για το πώς λειτουργεί μπορείτε να βρείτε στο ξεχωριστό αρχείο αλλά και στα σχόλια εντός του αρχείου του κώδικα.

Όσον αφορά την υλοποίηση υπήρχαν αρκετές επιλογές. Κάποιες από αυτές ήταν η χρήση κάποιας βιβλιοθήκης δομής δένδρου ή γράφου ή και χωρίς τη χρήση καμίας βιβλιοθήκης παραμόνο με αναδρομικές κλήσεις αναλόγως με την μορφή του πίνακα που θα περιείχε την πληροφορία σύνδεσης των κόμβων(bipartite matrix). Τελικά χρησιμοποιήθηκε η δομή του γράφου με αναδρομικές κλήσεις μιας συνάρτησης message. Δηλαδή αρχικά καλείται η συνάρτηση υπολογισμού του μηνύματος στην μεταβλητή που θέλουμε, η οποία όμως με τη σειρά της καλεί όλους τους γείτονες της (εκτός από τον κόμβο πατέρα) να της στείλουν τα δικά τους μηνύματα, για να βγάλει το τελικό αποτέλεσμα. Αυτό γίνεται αναδρομικά μέχρι να φτάσουμε σε κάποιο φύλλο όπου ο υπολογισμός του μηνύματος είναι τετριμμένος.



[εικ. 1] Δομή γράφου όπου φαίνεται η σειρά με την οποία γίνεται ο τελικός υπολογισμός της περιθώριας.

Από τα κύρια προβλήματα που αντιμετωπίσαμε στην ανάπτυξη του αλγορίθμου ήταν η υλοποίηση με όσο πιο γενικό τρόπο γίνεται της διαδικασίας: δηλαδή για οποιαδήποτε συνάρτηση με οποιοδήποτε πλήθος μεταβλητών και οποιεσδήποτε πράξεις πολλαπλασιασμού και πρόσθεσης. Τέλος προσπαθήσαμε να βελτιστοποιήσουμε όσο περισσότερο γινόταν τον αλγόριθμο και να μειώσουμε την πολυπλοκότητα του.

Παράδειγμα :

Δημιουργήσαμε ένα σύντομο παράδειγμα για να φανεί καθαρά πόσο σημαντική είναι η επιτάχυνση των υπολογισμών που μπορεί να επιτευχθεί με τη χρήση ενός τέτοιου αλγορίθμου. Έστω λοιπόν ότι θέλουμε να υπολογίσουμε το άθροισμα για όλες τις πιθανές τιμές του x4 :

$$g(x_4) = \sum_{x_1,x_2,x_3,x_5,x_6} f(x_1,x_2,x_3,x_4,x_5,x_6)$$

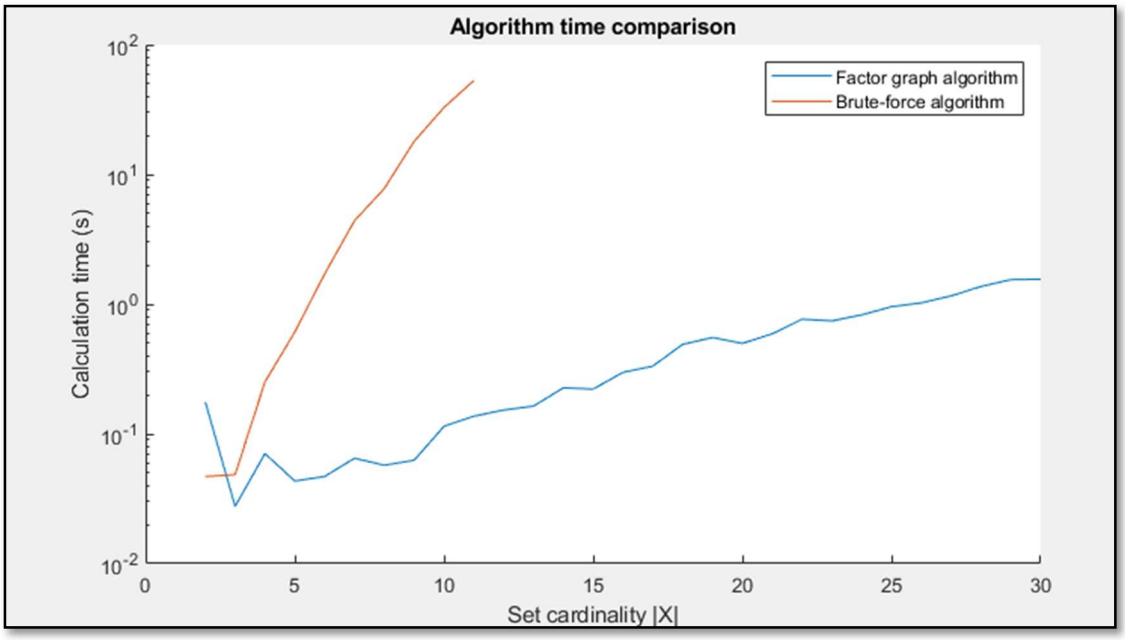
$$= \sum_{x_1,x_2,x_3,x_5,x_6} f_1(x_1,x_3,x_4) \cdot f_2(x_1) \cdot f_3(x_1,x_2,x_5) \cdot f_4(x_3,x_6)$$

όπου τα x παίρνουν τιμές σε ένα σύνολο της μορφής {1,2,….N} το οποίο μεταβάλουμε για να δείξουμε την αύξηση της πολυπλοκότητας. Χάριν πληρότητας, οι συναρτήσεις παραγοντοποίησης είναι:

$$f_1(x_1,x_3,x_4) = \frac{x_1 + x_3}{(x_4 - x_1)^2 + 1} \qquad f_2(x_1) = {x_1}^2$$

$$f_3(x_1,x_2,x_5) = {x_5}^{x_2} + x_1 \qquad f_4(x_3,x_6) = x_3 + {x_6}^2 + 1$$

Το factor graph που αντιστοιχεί σε αυτές τις συναρτήσεις είναι αυτό που φαίνεται στην εικόνα 1. Οι πράξεις έγιναν με δύο τρόπους: μια με τον αλγόριθμο που χρησιμοποιεί το γεγονός ότι η αρχική συνάρτηση f παραγοντοποιείται (οπότε εκτελεί το εν λόγω message passing) και μια με έναν brute-force αλγόριθμο. Αυξάνοντας τον πληθάριθμο του συνόλου όπου μπορούν να πάρουν τιμές τα x, μπορούμε να δούμε γιατί ο υπολογισμός μέσω factor graph είναι πολύ πιο αποδοτικός.



[εικ. 2] Σύγκριση υπολογιστικού κόστους σε υπολογισμό marginal από αλγόριθμο που εκμεταλλεύεται το factor graph σε σχέση με την brute-force μέθοδο

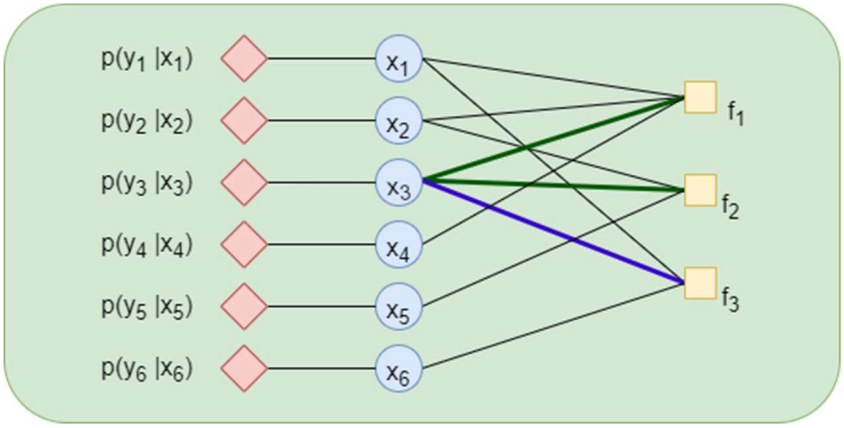
Σημείωση: Οι χρόνοι για μικρά προβλήματα ενδέχεται να μην είναι απόλυτα ακριβείς καθώς έχουν μεγάλες αποκλίσεις.

2. Υλοποίηση αλγορίθμων αποκωδικοποίησης γραμμικών κωδίκων για bit-decoding

Στο δεύτερο κομμάτι της εργασίας ασχοληθήκαμε με την υλοποίηση ενός belief propagation αποκωδικοποιητή για εφαρμογή σε ένα δυαδικό συμμετρικό κανάλι (BSC). Έχουμε δυο υλοποιήσεις: τις *bitDecode.m* και *decode.m* που διαφέρουν λιγάκι ως προς κάποιες βελτιστοποιήσεις, αν και το παράδειγμα(βλέπε παρακάτω) έδειξε πως έχουν τελικά πολύ παρόμοια αποτελέσματα.

Η *decode.m* είναι και αυτή που χρησιμοποιήθηκε και στο 3ο κομμάτι για το δυαδικό κανάλι με διαγραφές (BEC). Μπορεί να λάβει σαν όρισμα οποιεσδήποτε ακολουθίες καθώς δέχεται σαν είσοδο χαρακτήρες και είναι πολύ πιο ευέλικτη στη χρήση της.

Ένα βασικό στοιχείο της *bitDecode.m* είναι πως κατασκευάζει έναν «χάρτη» (var_msg_needed, check_msg_needed) που περιέχει τις θέσεις των μηνυμάτων τα οποία πρέπει να υπολογιστούν για να σταλεί το νέο μήνυμα κατά μήκος μιας ακμής. Αυτό βοηθάει πολύ στο να βελτιστοποιηθεί η διαδικασία: σε κάθε επανάληψη ξέρουμε αμέσως ποια μηνύματα πρέπει να συνυπολογίσουμε για κάθε ακμή, επομένως δεν εκτελούμε περιττές αναζητήσεις.



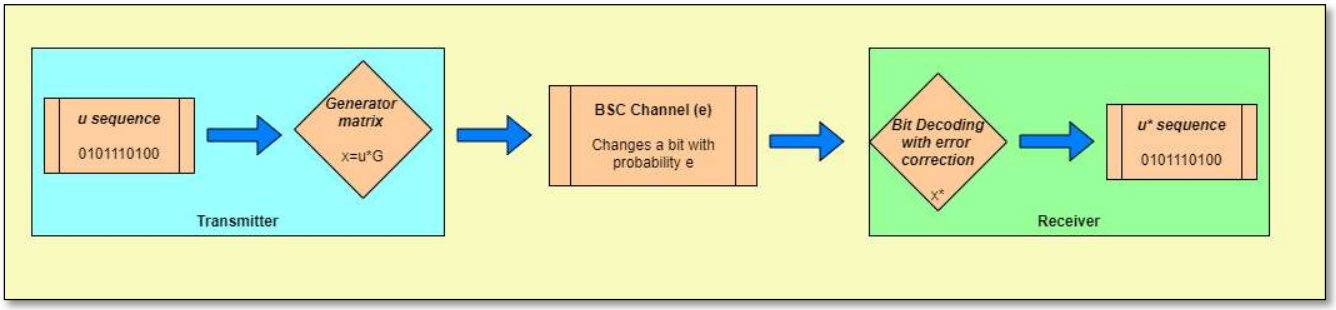
[εικ. 3] *Διάγραμμα όπου φαίνονται με πράσινο οι ακμές που θα χρησιμοποιούνταν για τον υπολογισμό της μπλε ακμής(με φορά x3 ->f3), χωρίς φυσικά τη συμμετοχή της πληροφορίας του καναλιού-που θα πρεπει να συμπεριληφθεί. Οι πράσινες είναι οι ακμές που θα αποθηκεύονταν στην θέση του "χάρτη" για την μπλε ακμή.*

Μπορούμε να φανταστούμε έναν τέτοιο αποκωδικοποιητή σαν ένα σύστημα ψηφοφορίας. Κάθε variable node-ψηφοφόρος έχει το δικό του συντελεστή βαρύτητας στην απόφαση για την κωδικολέξη η οποία στάλθηκε και επηρεάζει κάποιους από τους άλλους ψηφοφόρους. Μετά από έναν ορισμένο αριθμό συνομιλιών μεταξύ των ψηφοφόρων, εξάγεται -ή όχι- η απόφαση για την πιο πιθανή ακολουθία που μπορεί να στάλθηκε.

Η πολυπλοκότητα των αλγορίθμων αυτών θεωρητικά είναι ανάλογη του μήκους κώδικα n σε έναν LDPC μιας και ο αριθμός των ακμών είναι ανάλογος του n. Οι μετρήσεις που κάναμε έδειξαν μια γραμμική συμπεριφορά για το χρόνο εκτέλεσης, όμως υπήρχαν κάποιες ανώμαλες συμπεριφορές που πιθανόν οφείλονταν είτε στο γεγονός ότι δεν χρησιμοποιήσαμε επαληθευμένους κώδικες είτε στην κατανάλωση υπολογιστικών πόρων εκείνη τη στιγμή από τον υπολογιστή

Παραδείγματα:

Δημιουργήσαμε ένα βασικό μοντέλο διάδοσης για να ελέγξουμε τις επιδόσεις των αλγορίθμων που κατασκευάσαμε.



[εικ. 4] *Βασικό μοντέλο διάδοσης. Μελετήθηκαν τα σφάλματα έως την κωδικοποιημένη ακολουθία που φτάνει στον πομπό πριν την αποκωδικοποίηση(μέχρι x* , χωρίς u*)*

Αρχικά δημιουργούμε τυχαίες ακολουθίες u (μπορεί να αντιστοιχούν στα bits που θέλει να στείλει ο πομπός). Έπειτα χρησιμοποιήσαμε γραμμική κωδικοποίηση με έναν Generator Matrix G για να δημιουργηθεί η κωδικοποιημένη ακολουθία που θα σταλεί.

Αξίζει να σημειωθεί ότι υποθέτουμε γνωστό τον parity check matrix και από εκεί βρίσκουμε έναν generator matrix. Η διαδικασία αυτή έγινε με τη βοήθεια κάποιων σημειώσεων του διδάσκοντα και του βιβλίου *A first course in coding theory, Raymond Hill*[2].

Σημείωση: Ο generator matrix που βρίσκεται εδώ δεν είναι βέλτιστος καθώς αν εντοπιστεί λάθος κωδικολέξη που απέχει από την πραγματική, δεν είναι σίγουρο ότι αυτή θα αντιστοιχεί σε πηγαία ακολουθία με όσο το δυνατό μικρότερη απόσταση από τη σωστή(όπως θα γινόταν σε έναν κώδικα Gray όπου κοντινές κωδικολέξεις θα αντιστοιχούσαν σε λάθος 1 bit). Ακόμα, όπως διαπιστώσαμε κάπως αργά, η εν λόγω διαδικασία (gm_from_pcm) πολλές φορές οδηγεί σε generator matrix οι οποίοι έχουν μηδενικές στήλες, και παρόλο που είναι σωστοί (παράγουν κωδικολέξεις οι οποίες είναι αποδεκτές από τον αντίστοιχο parity check matrix) δημιουργούν προφανώς περιττά bits τα οποία είναι μονίμως 0. Στο παράδειγμα χρησιμοποιήσαμε έναν generator matrix χωρίς μηδενικές στήλες.

Από τον πομπό οι κωδικολέξεις περνούν, μέσω του καναλιού που αλλάζει τα bit με πιθανότητα e, στον δέκτη που υλοποιείται με έναν belief propagation decoder, ο οποίος διορθώνει κάποια από τα λάθη που γίνονται στο κανάλι. Μετά ακολουθεί η διαδικασία της αποκωδικοποίησης μέσω ενός lookup table, όμως δεν ασχοληθήκαμε με αυτό το κομμάτι. Η προσοχή μας λοιπόν αποδόθηκε στην απόδοση του αλγορίθμου: πόσα από τα λάθη που έγιναν μπορεί να διορθώσει;

***~SINGLE INSTANCE*:**

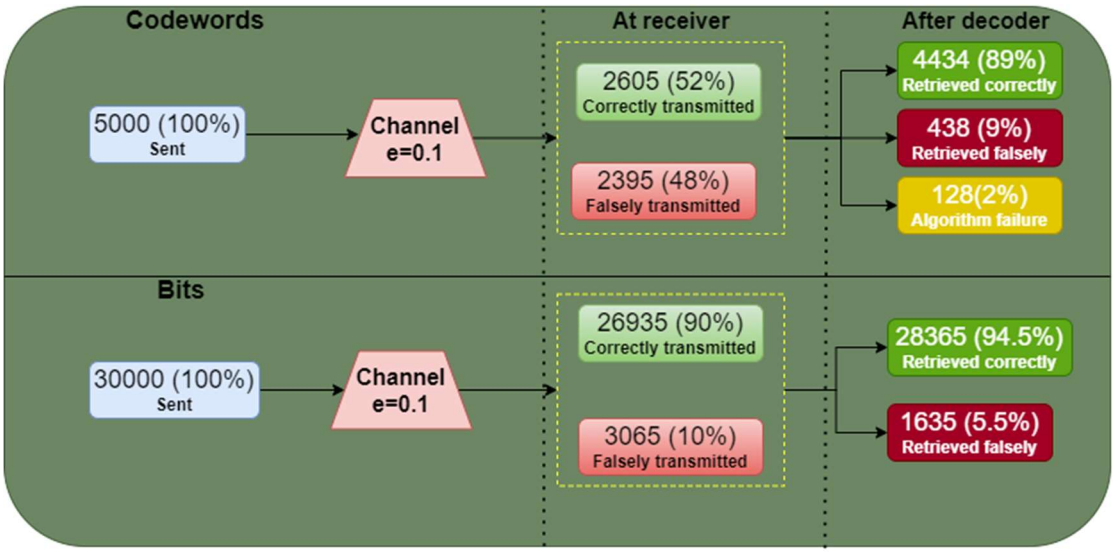
Σε μία μόνο μετάδοση με 5000 κωδικολέξεις, πιθανότητα σφάλματος του καναλιού e=0.1 (BSC κανάλι) και ζεύγος generator matrix - parity check matrix που φαίνονται παρακάτω, τα αποτελέσματα έδειξαν τα εξής (propagation_test_BSC.m):

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Generator Matrix

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Parity Check Matrix



[εικ. 5] Αποτελέσματα της προσομοίωσης για τον belief propagation decoder για το κανάλι BSC και για πιθανότητα σφάλματος καναλιού e=0.1

Υπενθύμιση: Τα δεδομένα αυτά αφορούν τις κωδικολέξεις που στάλθηκαν και λήφθηκαν, προτού κοιτάξουμε τον lookup table και αντιστοιχίσουμε την κωδικολέξη(6 bits) που λήφθηκε με την πιθανή πηγαία ακολουθία (3 bits) που πιστεύουμε ότι στάλθηκε.

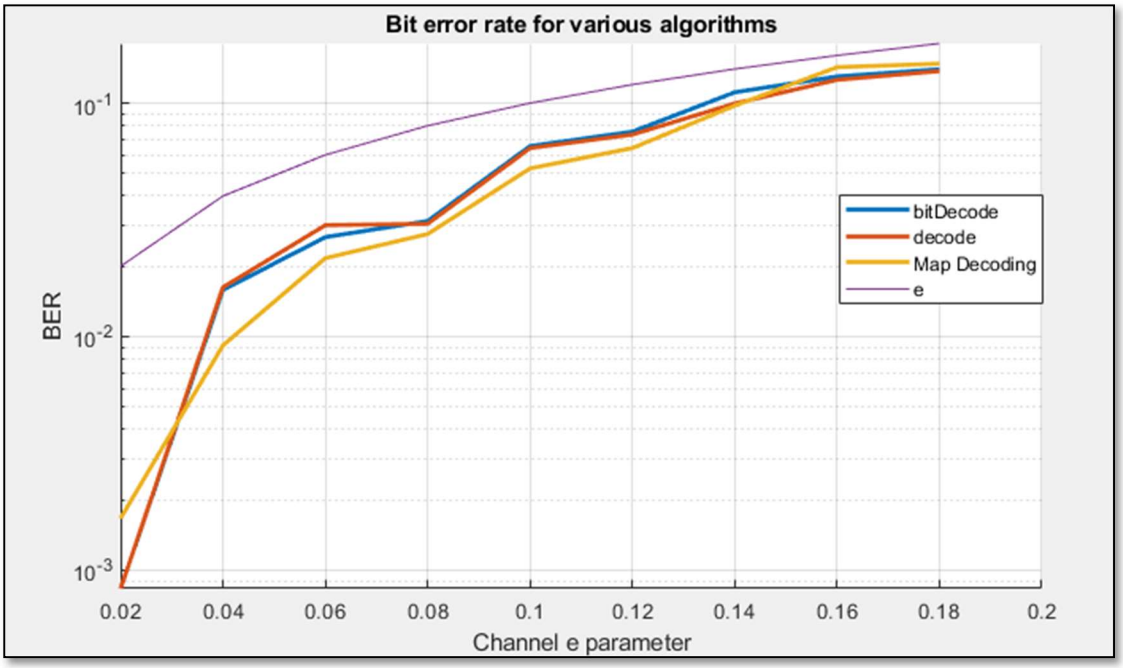
Βλέπουμε λοιπόν πως σε ένα κανάλι με πιθανότητα σφάλματος 10%, ο belief propagation bit decoder πετυχαίνει τελικό BER της τάξης του 5.5%, διορθώνει δηλαδή περίπου τα μισά σφάλματα. Βέβαια σε κάποιες περιπτώσεις ο αλγόριθμος αποτυγχάνει και θα μπορούσε τότε να ξαναζητήσει από τον πομπό να του στείλει το μήνυμα (Algorithm failure) και να πετύχει καλύτερα αποτελέσματα. **Σε αυτές τις περιπτώσεις ορίσαμε η ακολουθία που αποφασίζει ο δέκτης ότι στάλθηκε, να είναι αυτή που έλαβε, δηλαδή με όσα λάθη έγιναν (**που όμως δεν θα μπορούσε να αντιστοιχηθεί μέσω ενός lookup table). Τις φορές που ο δέκτης κάνει λάθος εκτίμηση (Retrieved falsely) έχει συγκλίνει σε κωδικολέξη η οποία όμως δεν είναι αυτή που στάλθηκε. Αν χρησιμοποιούσαμε κώδικα Gray τότε η «κοντινή» κωδικολέξη (6 bit) που θα έβρισκε ο decoder θα αντιστοιχούσε και σε «κοντινή» αποκωδικοποιημένη ακολουθία (3 bit) , που θα σήμαινε και μικρό τελικό (μετά την αποκωδικοποίηση) σφάλμα.

***~MULTIPLE INSTANCES*:**

- *<u>Παραλλαγή1:</u>*
  Οι BP decoders (belief propagation decoders: bitDecode, decode) ***όταν αναγνωρίζουν ότι κάνουν fail, επιστρέφουν όποια πιστεύουν ότι είναι εκείνη τη στιγμή η ακολουθία****(που συνήθως καταλήγει να έχει ΠΟΛΛΑ λάθη).*

Κάναμε την παραπάνω διαδικασία για διαφορετικές τιμές της παραμέτρου e του καναλιού(διαφορετική δηλαδή πιθανότητα σφάλματος) και για έναν parity check matrix 6x12 και έχουμε την απόδοση των αλγορίθμων παρακάτω. Οι **bitDecode** και **decode** είναι οι αλγόριθμοι που υλοποιήθηκαν από τους συγγραφείς ενώ η **Map Decoding** είναι μια πρόχειρη υλοποίηση ενός Block Map Decoder που βρίσκει δηλαδή την πιο πιθανή κωδικολέξη που μπορεί να έχει μεταδοθεί δεδομένου της λαμβανόμενης ακολουθίας (αν δυο κωδικολέξεις ισαπέχουν από τη λαμβανόμενη, επιλέγει μια -αλλά όχι στην τύχη, επιλέγει την τελευταία με τη σειρά ανίχνευσης).

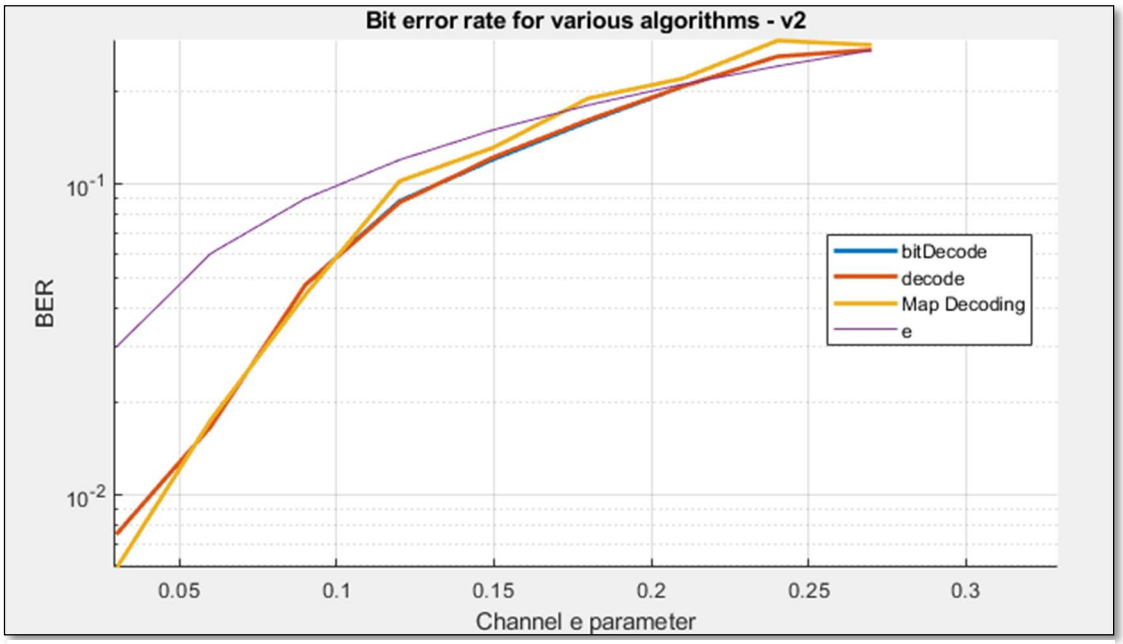
Σημείωση: Προτείνουμε να μη χρησιμοποιήσετε τη Map Decoding μαζικά: είναι πολύ αργή και καθόλου βέλτιστα φτιαγμένη.



[εικ. 6] *Σύγκριση των επιδόσεων των αλγορίθμων belief propagation decoding και του block map decoding*

Βλέπουμε στο παραπάνω διάγραμμα πως οι belief propagation decoders (BP decoders) καταφέρνουν πολύ καλά αποτελέσματα, συγκρίσιμα μάλιστα με τα αποτελέσματα ενός block map decoder, που στην πράξη δεν θα μπορούσε εύκολα να υλοποιηθεί λόγω πολυπλοκότητας. Επίσης , όπως περιμέναμε οι δυο υλοποιήσεις (bitDecode, decode) έχουν πολύ παρόμοιες αποδόσεις.

- *<u>Παραλλαγή 2:</u>*
  Οι BP decoders ***όταν αναγνωρίζουν ότι κάνουν fail, επιστρέφουν την ακολουθία που πήραν εξ αρχής, δηλαδή την λαμβανόμενη ακολουθία****, με ότι λάθη αυτή περιέχει και χωρίς αυτή να είναι κωδικολέξη. Προφανώς αυτό δεν θα είχε κανένα νόημα στην πραγματικότητα αλλά είναι ένας τρόπος να αναλογιστούμε την απόδοση του αλγορίθμου αν γινόταν εκ νέου αποστολή των λέξεων αυτών από τον πομπό.*

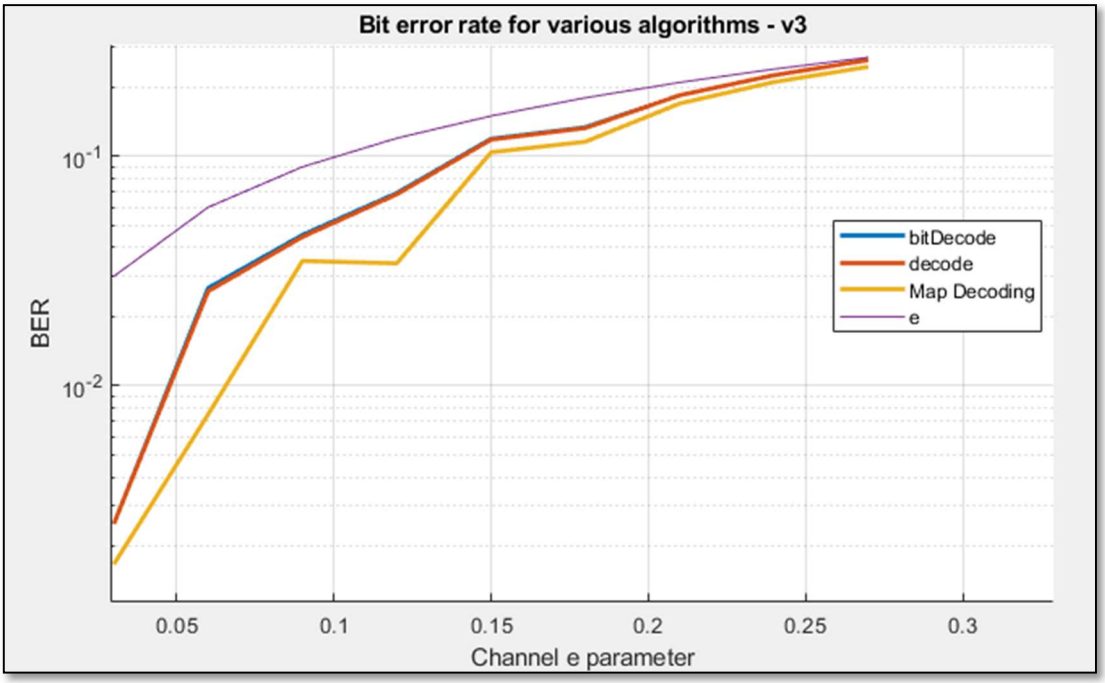


[εικ. 7] *Σύγκριση των επιδόσεων των αλγορίθμων belief propagation decoding και του block map decoding - v2*

Εδώ οι BP decoders βλέπουμε πως έχουν συγκρίσιμες και μάλιστα πολλές φορές καλύτερες επιδόσεις από τον block map decoder. Αυτό βέβαια γίνεται γιατί στην παραλλαγή αυτή οι BP decoders έχουν ένα σοβαρό πλεονέκτημα: δεν κάνουν πολλά λάθη στις δύσκολες περιπτώσεις, ενώ ο block map decoder πρέπει να αποφασίσει μια κωδικολέξη και έτσι καταλήγει συχνά να αλλάζει πολλά bits. Για να συγκρίναμε αυτούς τους αλγορίθμους «επί ίσους όρους» θα έπρεπε να «βοηθήσουμε» και τον map decoder όταν δυσκολεύεται , δηλαδή όταν δύο ή περισσότερες κωδικές λέξεις ισαπέχουν από τη λαμβανόμενη. Το γεγονός ότι όταν το e ανεβαίνει πολλές φορές οι αλγόριθμοι έχουν χειρότερο BER απ' ότι αυτό που θα είχαμε χωρίς την διόρθωση λαθών οφείλεται στο γεγονός ότι σε κάποιες κωδικολέξεις τα λάθη θα είναι τόσο πολλά που θα αρχίζουν να μοιάζουν περισσότερο σε άλλες κωδικολέξεις!

- *<u>Παραλλαγή 3 (BONUS):</u>*
  Στέλνοντας μόνο μηδενικά *------- και οι BP decoders* ***όταν αναγνωρίζουν ότι κάνουν fail, επιστρέφουν την ακολουθία που πήραν εξ αρχής, δηλαδή την λαμβανόμενη ακολουθία****, με ότι λάθη αυτή περιέχει και χωρίς αυτή να είναι κωδικολέξη*

Στο επόμενο γράφημα φαίνονται τα αποτελέσματα της προσομοίωσης όταν δεν στέλνουμε οποιεσδήποτε κωδικολέξεις, αλλά στέλνοντας πάντα τη μηδενική. Οι αλγόριθμοι φαίνεται να έχουν πολύ καλύτερες μέσες αποδόσεις και κυρίως ο map decoder φαίνεται να είναι καλύτερος από τους άλλους και αυτό πιθανολογούμε ότι συμβαίνει επειδή η μηδενική κωδικολέξη με τη συγκεκριμένη κωδικοποίηση είναι μια «εύκολη» κωδικολέξη, απέχει δηλαδή πολύ (σε σύγκριση με τις υπόλοιπες) από τους γείτονες της.



[εικ. 8] *Σύγκριση των επιδόσεων των αλγορίθμων belief propagation decoding και του block map decoding – v3*

Σχόλιο: Ο λόγος που τα BERS των αλγορίθμων, ακόμα και του Block Map decoder, είναι τόσο υψηλός είναι γιατί τα bits που λαμβάνονται είναι ***ισοδύναμα*** *και επομένως κανένας αποκωδικοποιητής δεν μπορεί να ξεχωρίσει τα σωστά από τα λάθος, με αποτέλεσμα να αποφασίζει πολλές φορές τελικά λάθος κωδικολέξεις. Για παράδειγμα είδαμε τον Map decoder να κάνει πολλά λάθη όταν υπήρχαν κωδικολέξεις που «ισαπέχουν» από τη λαμβανόμενη. Στην επόμενη ενότητα φαίνεται καθαρά πόσο καθοριστικά αλλάζουν αυτά τα αποτελέσματα όταν ο δέκτης είναι σίγουρος για το ποια bits είναι σωστά. Αυτό θα μπορούσε να γίνει σαν προσομοίωση και εδώ, δίνοντας συντελεστές βαρύτητας στις πιθανότητες p(y|x) αναλόγως με το πόσο κοντά είναι το λαμβανόμενο σύμβολο στα αναμενόμενα πχ με μια BPAM διαμόρφωση. Μια τέτοια ανάλυση όμως θα απαιτούσε να χτίσουμε την διαμόρφωση και δεν «χώρεσε» στην παρούσα εργασία.*

3. Μελέτη και σχεδίαση LDPC κωδίκων, υλοποίηση αποκωδικοποιητή και προσομοίωση

Σε αυτό το κομμάτι έγινε η προσπάθεια σχεδίασης από την αρχή και η μελέτη της επίδοσης ενός LDPC κώδικα. Έστω ότι θέλουμε να πετύχουμε μετάδοση σε **κανάλι BEC** με **e=0.4** και πως ο μέγιστος βαθμός κόμβων και για τους variable nodes και για τους check nodes είναι 6. Οι κατανομές (λ, ρ) που προέκυψαν από την εκτέλεση του αλγορίθμου distributionPairLDPC.m φαίνονται παρακάτω:

$$\lambda_i(x) = 0.5822 \cdot x + 0.4178 \cdot x^2 \qquad \rho_i(x) = x^4 \qquad (r_i = 0.53)$$

Σημείωση: οι δείκτες i, f αναφέρονται στην initial(αρχική, σχεδιαστική) και final(τελική πραγματική)

Για να υλοποιήσουμε όμως έναν πραγματικό κώδικα πρέπει να βρούμε τις κατανομές (Λ,P) και γι' αυτό χρειάζεται να επιλεγεί μια ακόμη παράμετρος: το μήκος της κωδικολέξης n και κατόπιν να γίνει μια στρογγυλοποίηση. Θα μπορούσε αυτοματοποιημένα να γίνεται η εύρεση του ελάχιστου n για το οποίο η νέα κατανομή (Λ,P) -που θα προσεγγίζει όσο το δυνατόν περισσότερο αυτή που υπολογίσαμε- έχει ένα multiplicative gap (ποσοστιαία απόσταση από τον επιθυμητό ρυθμό *r*) μικρότερο από ένα ορισμένο όριο. Επιλέξαμε ένα σχετικά μικρό n=12 για να μπορούμε να δείξουμε τα αποτελέσματα στο χαρτί. Οι κατανομές που προέκυψαν έχουν ως εξής:

$$\Lambda_f(x) = 8 \cdot x^2 + 4 \cdot x^3 \qquad P_f(x) = x^3 + 5 \cdot x^5$$
$$\lambda_f(x) = 0.5714 \cdot x + 0.4286 \cdot x^2 \qquad \rho_f(x) = 0.107 \cdot x^2 + 0.893 \cdot x^4 \qquad (r_f = 0.5)$$

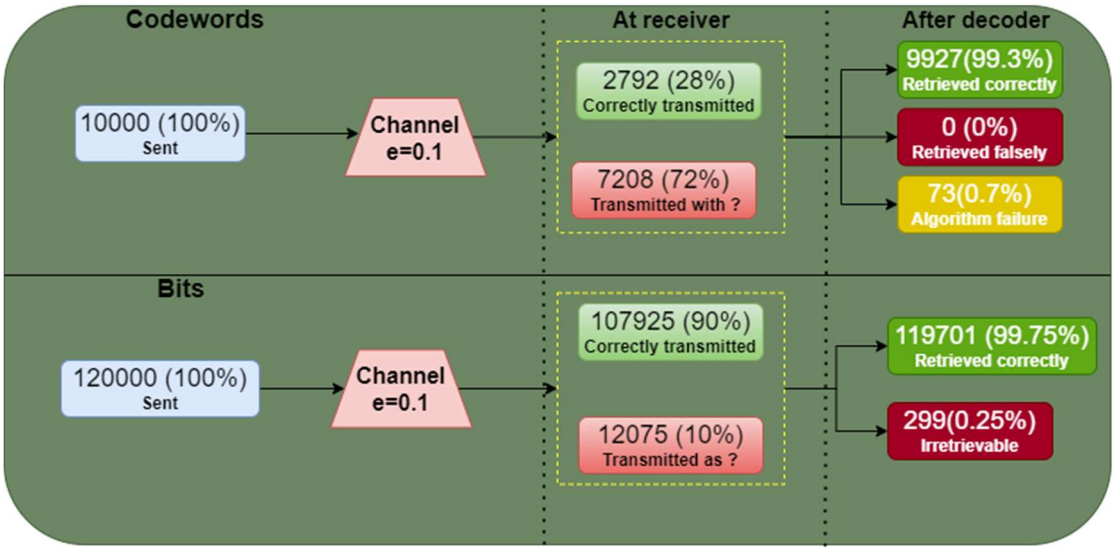
Με τις κατανομές αυτές φτιάχνουμε έναν LDPC πίνακα, εδώ η διαδικασία έγινε με το χέρι, υπάρχει όμως τρόπος να γίνει αυτόματα (αλγόριθμος **Standard Ensemble LDPC(Λ,P)**). Επίσης κατασκευάστηκε και ένας generator matrix για τον parity check που προέκυψε.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Parity Check Matrix

*Σημείωση προς τους επιβλέποντες: Η κατασκευή των πινάκων αυτών με το χέρι βοήθησε στο να έχουμε μια επαφή με την απόδοση τους. Ακόμη, ο αλγόριθμος που δεδομένης μια κατανομής (Λ, P) κατασκευάζει έναν πίνακα **Standard Ensemble LDPC(Λ,P)** και ο αλγόριθμος που κατασκευάσαμε για την παραγωγή ενός Generator Matrix δεδομένου ενός Parity Check Matrix **gm_from_pcm** είχαν το πρόβλημα ότι ορισμένες στήλες τους είναι μηδενικές. Έτσι δεν θα είχαν την απόδοση που ίσως περιμέναμε και σχεδιάσαμε, γι' αυτό και χρησιμοποιήσαμε πίνακες που επαληθεύσαμε συγκεκριμένα.*

Κάναμε εκ νέου μια μετάδοση σε ένα κανάλι BEC με e=0.1 , για να δούμε αρχικά αν ο κώδικας που δημιουργήσαμε , μαζί με τον belief propagation αλγόριθμο αποκωδικοποίησης μπορεί να ανταποκριθεί σε ένα κανάλι αρκετά πιο «εύκολο» από αυτό για το οποίο τον σχεδιάσαμε.

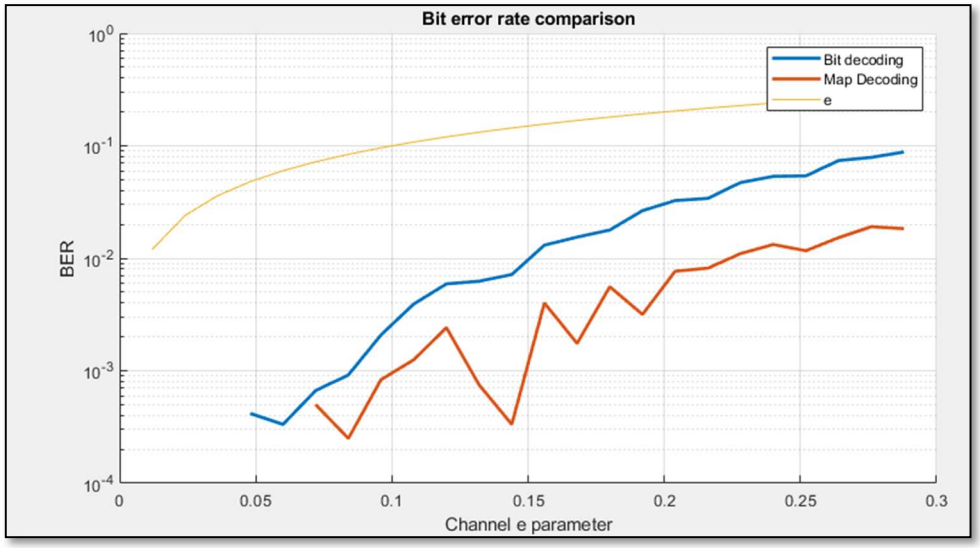


[εικ. 9] *Αποτελέσματα της προσομοίωσης για τον belief propagation decoder για το κανάλι BEC και για πιθανότητα σφάλματος καναλιού e=0.1*

Όπως βλέπουμε, ο αποκωδικοποιητής μπορεί να διορθώσει πάρα πολλά λάθη! Να σχολιάσουμε εδώ ότι όσες κωδικολέξεις δεν μπορεί να αποκωδικοποιήσει τις αφήνει ως έχουν, με ερωτηματικά, οπότε αυτές είναι που τελικά οδηγούν στα “Irretrievable” bits.

Φαίνεται να έχουμε σχετικά καλά αποτελέσματα, όμως γιατί ενώ σχεδιάσαμε έναν κώδικα ο οποίος είναι για πιθανότητα e=0.4 δεν μπορεί να διορθώσει όλα τα λάθη που συμβαίνουν σε μια μετάδοση με e=0.1; Πώς άραγε θα συμπεριφερόταν αν τον χρησιμοποιούσαμε σε κανάλι με e=0.4; Τι θα έπρεπε να κάνουμε για να πετύχουμε τελικά πολύ μικρότερη πιθανότητα να μην μπορέσει να ανιχνευθεί ένα bit;

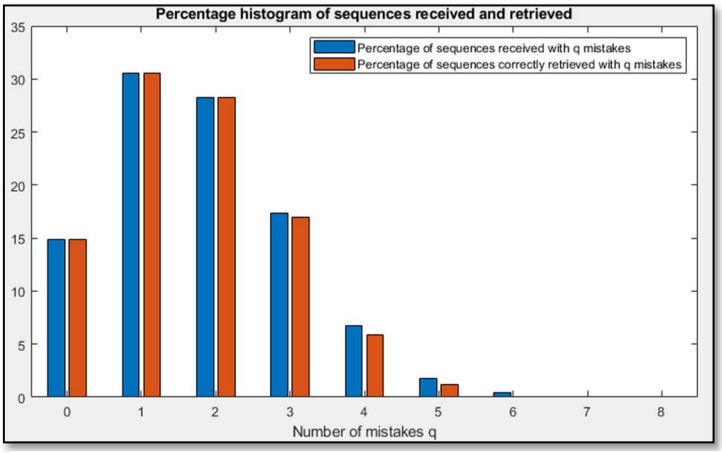
Παρακάτω φαίνεται το BER του BP αποκωδικοποιητή μας για διάφορες τιμές της παραμέτρου καναλιού e σε σύγκριση με τις αποδόσεις τον MAP decoder. Σημειώστε ότι το BER του BP decoder - που καταχρηστικά χρησιμοποιείται σαν όρος- αναφέρεται ουσιαστικά στα ερωτηματικά που δεν μπόρεσε να διορθώσει - αφού σε αυτά διαφέρει η απεσταλμένη ακολουθία από την αποκωδικοποιημένη - καθώς σε καμία από τις προσομοιώσεις δεν υπήρχαν bits που ήταν ερωτηματικά αλλά ανίχνευσε λανθασμένα. Το BER του MAP decoder αναφέρεται προφανώς στα bits που ανίχνευσε λανθασμένα καθώς πάντα αποφασίζει για κάποια κωδικολέξη.



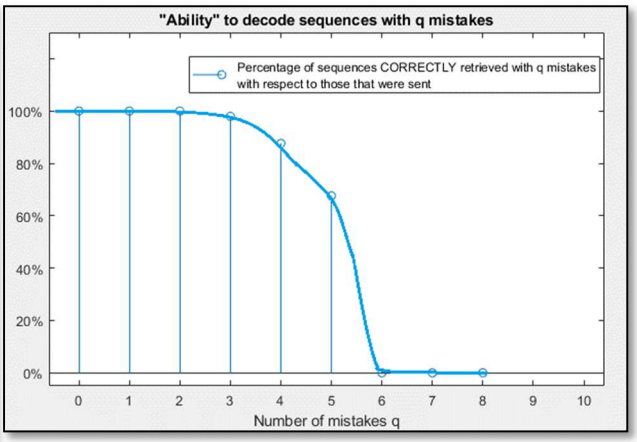
[εικ. 10] *Το BER που πετυχαίνουμε για τον επί εξέταση κώδικα για διάφορες τιμές της παραμέτρου e του καναλιού*

Τι ακριβώς εξασφαλίσαμε λοιπόν κάνοντας την σχεδίαση για παράμετρο καναλιού e=0.4; Σύμφωνα με τους *Richardson & Urbanke*[1] θα έπρεπε ο κώδικας που σχεδιάσαμε να μπορεί να διορθώνει το μεγαλύτερο ποσοστό των λαθών που γίνονται, αν όχι όλα καθώς το μετρούμενο e^{BP} του είναι 0.4. Σ' αυτό το σημείο θέλουμε να προτείνουμε μια απάντηση σ' αυτή την ερώτηση χωρίς να είμαστε σίγουροι ότι είναι σωστή, αν και βγάζει πολύ καλό νόημα.

Η -μάλλον απλή- απάντηση είναι ότι όταν γίνει μετάδοση μέσω BEC με e=0.4 υπάρχουν πολλές κωδικολέξεις στις οποίες θα έχουν αλλοιωθεί περισσότερα από το 40% (στην περίπτωση μας 4.8-) των bits και αυτές είναι οι οποίες δεν μπορούν να ανιχνευθούν! Οι ακολουθίες λοιπόν στις οποίες 5 ή περισσότερα από τα 12 bits έχουν αλλοιωθεί είναι μάλλον αυτές οι οποίες δεν μπορούν να αποκωδικοποιηθούν από τον BP decoder. Το παρακάτω ιστόγραμμα δείχνει τις σωστές εκτιμήσεις κωδικολέξεων, σε σχέση με το πλήθος άγνωστων bit που είχαν όταν λήφθηκαν σε μια μετάδοση για το αρχικό μας παράδειγμα με e=0.1.



[εικ. 11] *Κωδικολέξεις που έχουν q λάθη{ΜΠΛΕ} και που αποκωδικοποιήθηκαν επιτυχώς {ΠΟΡΤΟΚΑΛΙ}*



[εικ. 12] Ποσοστό λέξεων που αποκωδικοποιούνται σωστά επί των λέξεων που στάλθηκαν με σφάλματα q

Παρατηρούμε λοιπόν πως ο belief propagation decoder μπορεί και διορθώνει σχεδόν όλες τις λέξεις που έχουν έως 3 λάθη όταν το μήκος του κώδικα είναι 12. Φαίνεται λοιπόν ότι για να έχουμε καλά αποτελέσματα στο κανάλι με e=0.4 θα έπρεπε να σχεδιάσουμε έναν κώδικα με μεγαλύτερο μήκος, ώστε να είναι πιθανότερο η τυχούσα κωδικολέξη να έχει λάθη περίπου 40%. Αυτό συμβαδίζει με κάτι που μπορεί να καταλάβει κανείς πως είναι απαραίτητο «διαισθητικά» για να μπορεί ένας κώδικας να διορθώσει πολλά σφάλματα : κάθε κωδικολέξη να ταυτοποιείται επαρκώς από ένα ποσοστό των bit της. Όταν έχει χαθεί ποσοστό bit μεγαλύτερο από αυτό που ορίζει η κωδικοποίηση ότι επιτρέπεται για να μπορέσουμε να διακρίνουμε μια κωδικολέξη, η αναγνώριση της από ένα σημείο και μετά γίνεται αδύνατη.

Feedback

Η ενασχόληση με την εργασία ήταν πολύ εποικοδομητική. Είδαμε πολλά θεωρητικά και πρακτικά προβλήματα και βρήκαμε προσεγγιστικές λύσεις. Η εργασία ήταν πολύ χρήσιμη προγραμματιστικά γιατί μάθαμε τεχνικές που θα μπορέσουμε να αξιοποιήσουμε στο μέλλον. Ήταν λίγο πιο απαιτητική απ' όσο θα περιμέναμε αλλά άξιζε τον κόπο.

Στο πρώτο ζητούμενο, η γενική μορφή που έπρεπε να έχει ο αλγόριθμος ήταν αρκετά δύσκολη στην υλοποίηση και ίσως δεν προσέφερε τόσο πολύ τελικά στα μετέπειτα κομμάτια της εργασίας. Θα ήταν πιο εύκολο να γίνει υλοποίηση για πιο συγκεκριμένα παραδείγματα, ή τουλάχιστον μόνο για τις πράξεις της συνηθισμένης πρόσθεσης και πολλαπλασιασμού.

Το δεύτερο ζητούμενο είχε κατάλληλο βαθμό δυσκολίας και ταυτόχρονης κατανόησης της θεωρίας, και ήταν ξεκάθαρος ο στόχος του ερωτήματος και το τι προσφέρει. Είχαμε το κίνητρο να σχεδιάσουμε τον κώδικα στο matlab για να τον δούμε πραγματικά να διορθώνει σφάλματα.

Το τρίτο ζητούμενο είχε τον μεγαλύτερο βαθμό θεωρητικής και πρακτικής δυσκολίας. Δεν ήταν εκ πρώτης όψεως εμφανείς οι λεπτομέρειες (βλ (λ,ρ)-> (Λ,Ρ)) που χρειάστηκε να αντιμετωπίσουμε για να σχεδιάσουμε από την αρχή έως το τέλος έναν LDPC κώδικα και υπήρχαν πολλές κρυμμένες επομένως δυσκολίες. Άξιζε όμως να δούμε την συμπεριφορά ενός κώδικα στο τέλος σε σχέση με αυτά που περιμέναμε.

Για ότι χρειαστείτε να συζητήσουμε / να εξηγήσουμε επικοινωνήστε μαζί μας!

Αναφορές

[1] Richardson T. & Urbanke R. (2008). Modern Coding Theory

[2] Raymond Hill (1986). A first course in coding theory

[3] Tavakoli, H., Attari, M. A., & Peyghami, M. R. (2011, October). Optimal rate for irregular LDPC codes in binary erasure channel. In 2011 IEEE Information Theory Workshop (pp. 125-129). IEEE

[4] Shokrollahi, A. (2003). LDPC codes: An introduction. Digital Fountain, Inc., Tech. Rep, 2, 17.

[5] Guruswami, V. (2006). Iterative decoding of low-density parity check codes (A Survey)

[6] The Cutting-Plane Method for Solving Convex Programs
https://www.mathworks.com/help/optim/ug/miqp-portfolio-problem-based.html

[7] Linear Programming and Mixed-Integer Linear Programming
https://www.mathworks.com/help/optim/linear-programming-and-mixed-integer-linear-programming.html?s_tid=CRUX_lftnav