UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gregor Kralj

PARITETNE KODE Z NIZKO GOSTOTO

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01641/2010
Datum: 15.03.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GREGOR KRALJ**

Naslov: **PARITETNE KODE Z NIZKO GOSTOTO**
LOW-DENSITY PARITY-CHECK CODES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Paritetne kode z nizko gostoto (angl. low-density parity-check codes) so razred linearnih bločnih kod za odpravljanje napak, ki se lahko pojavijo pri prenosu podatkov po komunikacijskem kanalu. V zadnjem času se uporabljajo v številnih aplikacijah, saj omogočajo delovanje zelo blizu kapacitete kanala ob sprejemljivi zahtevnosti kodirnega in dekodirnega postopka.

V diplomskem delu obravnavajte paritetne kode z nizko gostoto. Predstavite enega od postopkov za konstrukcijo kod, kodiranje sporočil in dekodiranje prejetega besedila z iterativnim postopkom na osnovi izročanja sporočil. Predstavljene algoritme tudi preizkusite na paritetnih kodah z različnimi dolžinami bloka: preverite moč odpravljanja napak in časovno zahtevnost dekodiranja za različne vrednosti šuma.

Mentor:

doc. dr. Arjana Žitnik

Žitnik

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Franc Solina

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gregor Kralj

PARITETNE KODE Z NIZKO GOSTOTO

Mentorica: doc. dr. Arjana Žitnik

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.

Št. naloge: 01641/2010
Datum: 15.03.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GREGOR KRALJ**

Naslov: **PARITETNE KODE Z NIZKO GOSTOTO**
LOW-DENSITY PARITY-CHECK CODES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Paritetne kode z nizko gostoto (angl. low-density parity-check codes) so razred linearnih bločnih kod za odpravljanje napak, ki se lahko pojavijo pri prenosu podatkov po komunikacijskem kanalu. V zadnjem času se uporabljajo v številnih aplikacijah, saj omogočajo delovanje zelo blizu kapacitete kanala ob sprejemljivi zahtevnosti kodirnega in dekodirnega postopka.

V diplomskem delu obravnavajte paritetne kode z nizko gostoto. Predstavite enega od postopkov za konstrukcijo kod, kodiranje sporočil in dekodiranje prejetega besedila z iterativnim postopkom na osnovi izročanja sporočil. Predstavljene algoritme tudi preizkusite na paritetnih kodah z različnimi dolžinami bloka: preverite moč odpravljanja napak in časovno zahtevnost dekodiranja za različne vrednosti šuma.

Mentor:

doc. dr. Arjana Žitnik

Žitnik

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Franc Solina

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Gregor Kralj,

z vpisno številko 63020087,

sem avtor diplomskega dela z naslovom:

Paritetne kode z nizko gostoto.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Arjane Žitnik,

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela Fri«.

V Ljubljani, dne Podpis:

Zahvala

Zahvalil bi se vsem, ki so mi kakorkoli pripomogli pri izdelavi diplomskega dela, posebej pa staršem, ki so me vseskozi spodbujali in materialno podpirali.

Največja zahvala pa gre mentorici doc. dr. Arjani Žitnik za neizmerno požrtvovalnost in strokovno vodenje. Brez nje diplomsko delo ne bi nastalo v taki obliki.

Za vse nas, ki skupaj oblikujemo ta svet

»Če je vaš mehanizem narejen iz plastike, potem bo to tako. In znanstveniki dejansko razmišljajo o tej smeri, da bi celoten mehanizem naredili iz plastike. Potem ne bi potreboval hrane, ne bi potreboval ljubezni, ne bi potreboval ničesar. Včasih, če bi se kaj pokvarilo,bi te poslali v delavnico. Vsak dan pa bi hodil na črpalko, da bi natočili gorivo vate in spet bi bil dober. Tedaj bi bil perfekcionist, tedaj bi bil popoln.

A življenje, kakršno je, je občutljivo; ni iz plastike, je zelo občutljivo. Nimaš žic, imaš živce. In ravnovesje se ves čas giblje sem in tja. Nič ni gotovega in vse se preliva eno v drugo. Zato si živ.

Človek življenja se ne vznemirja, nepopolnost ga ne skrbi. Sploh ne razmišlja o pojmih popolnosti; samo živi v trenutku, kolikor se da celovito, kolikor se le da polno. In čim bolj polno živi, tem bolj postaja sposoben živeti.

Napoči nov dan…živi preprosto, ne da bi vsiljeval kakršnekoli ideale, ne da bi razmišljal o pojmih, ne da bi se delal pravila in predpise o življenju. Preprosto živi in se radosti.«

Sosan, *Hsin Hsin Ming*

Kazalo

Seznam simbolov

Poglavje 1

$\oplus$	operacija seštevanja po modulu 2
$\rightarrow$	iz tega sledi
$\sim$	ekvivalentnost kodov
φ	preslikava, ki sporočilu priredi kodno besedo, $\varphi : s \rightarrow x$
Ψ	preslikava, ki prejeti besedi priredi sporočilo, $\Psi : y \rightarrow \hat{s}$
a	amplituda BPSK modulacije
$\mathcal{C}$	kod, njegove elemente imenujemo kodne besede
d_H	Hammingova razdalja
d_{min}	minimalna Hammingova razdalja;
e	napaka, ki nastane zaradi delovanja šuma
$erfc$	komplementarna funkcija napake
$N(\mu, \sigma)$	Gaussova porazdelitev gostote verjetnosti, kjer je μ matematično upanje in σ standardna deviacija
M	število različnih kodnih besed v kodu
p	verjetnost napačnega prejetega znaka (bita)
Q	verjetnost napačno prenesenega bita je enaka Gaussovi Q funkciji
s sistema	sporočilo, ki ga oddajnik pošlje preko komunikacijskega $s = (s_1, s_2, \dots, s_k)$
$\hat{s}$	prejeto sporočilo $\hat{s}$; ni nujno, da je le-to enako poslanemu sporočilu s
t	teža kodne besede
V_2	kodna abeceda, množica {0,1}
V_2^*	$V_2^* = \bigcup_{i=1}^{\infty} V_2^i$

$x = (x_1, \ldots, x_n)$	kodne besede
$y = (y_1, \ldots, y_n)$	prejete besede

Poglavje 2

G	generatorska matrika
G_{sys}	sistematična generatorska matrika
H	paritetna matrika
H_{sys}	sistematična paritetna matrika
I	identična matrika; matrika ničel, z enicami na diagonali
k	dolžina sporočila, dimenzija koda
n	dolžina kode
R	razmerje koda; $R = \frac{k}{n}$
T	transponiranje matrike; zamenjava vrstic in stolpcev matrike

Poglavje 3

c	testno vozlišče
$colOnes$	matrika skrajšanega zapisa mest enic v posameznem stolpcu paritetne matrike H
d_o	minimalna razdalja PEG koda
g_o	dolžina najkrajšega cikla Tannerjevega grafa
p_j^x	ocena verjetnosti, da je posamezen bit prejete besede y enak 0 ali 1
Q_{ij}^x	simbol v_j pošlje vsem svojim otrokom, testnim vozliščem
c_i,	ocenjeno vrednost, da je testno vozlišče v stanju x
R_{ij}^x	testno vozlišče c_i pošlje vsem nadrejenim vozliščem v_j, ocenjeno vrednost, da je nadrejeno vozlišče v stanju x
$rowsOnes$	matrika skrajšanega zapisa mest enic v posamezni vrstici paritetne matrike H
v	simbol, nastopa v Tannerjevem grafu

w_c	stopnja testnega vozlišča
w_r	stopnja simbola

Poglavje 4

$AvgBer$	napaka na dekodirani besedi po 20-ih dekodirnih iteracijah
$AvgCherr$	napaka pri prenosu skozi kanal pri danem SNR
$AvgIter$	povprečno število iteracij za ugotovitev poslane kodne besede
SNR	razmerje moči signala proti moči šuma
E	energija signala
$\frac{N_0}{2}$ močnostni	motnja belega Gaussovega šuma je enaka dvostranski spektralni gostoti $\frac{N_0}{2}$

Seznam kratic

APP	angl., A posteriori probability
BPSK	angl., Binary Phase-Shift Keying
BSC	angl., Binary Symmetric Channel
LDPC	angl., Low-Density Parity-Check Codes
PEG	angl., Progressive Edge Growth
SNR	angl., Signal-to-Noise Ratio
SPA	angl., Sum Product Algorithm

Povzetek

Pri prenosu podatkov preko raznih komunikacijskih sistemov prihaja zaradi nezaželenih vplivov do pojava napak. Eden od najučinkovitejših načinov za povečanje zanesljivosti prenosa je uporaba kod za popravljanje napak. V zadnjem času je pogosta uporaba paritetnih kod z nizko gostoto, saj ob napredku tehnologije omogočajo delovanje blizu kapacitete komunikacijskega kanala ob sprejemljivi zahtevnosti kodirnega ter dekodirnega postopka.

V diplomskem delu obravnavamo paritetne kode z nizko gostoto. Opišemo postopek PEG (angl. Progressive Edge Growth) za generiranje paritetnih matrik teh vrst kod ter predstavimo kodirni postopek in iterativni dekodirni postopek na osnovi izročanja sporočil. Za model komunikacijskega kanala vzamemo Gaussov kanal z belim šumom. Nazadnje naredimo še primerjavo učinkovitosti opisanega dekodirnega postopka na matrikah različnih dimenzij, ki smo jih generirali s postopkom PEG. Pri tem spreminjamo moč šuma in preverjamo pojav napak pri različnem številu iteracij. Izkaže se, da je pri izbranih matrikah boljše uporabiti paritetne matrike večjih dimenzij kot paritetne matrike manjših dimenzij.

Ključne besede:

- kodi za popravljanje napak,
- paritetne kode z nizko gostoto,
- iterativni dekodirni algoritem z izročanjem sporočil,
- Gaussov kanal z belim šumom.

Abstract

Transmition of data over various communication systems is lossy, because of all the noise affecting the communication channel. One of the most useful ways to achieve reliable transmision are error correcting codes. Recently, low-density parity-check codes became very popular. These codes can come arbitrarily close to the Shannon limit and because of the technology development their coding and decoding algorithms are reasonably fast.

In this graduation thesis, low density parity check (LDPC) codes are studied. The progressive edge growth (PEG) algorithm for generating such codes is presented and iterative decoding algorithm (SPA) is described. For the communication channel model, additive white Gaussian noise channel is used. In the last section of the thesis, efficiency of the presented decoding algorithm is examined for matrices of various dimensions, that were generated with algorithm PEG. The rate of bit errors or the number of iterations for correct decoding was counted for different levels of noise. Results show that in the tested cases, parity check codes of larger dimensions perform better than the parity check codes of lower dimensions.

Keywords:

- Error Correcting Codes,
- Low-Density Parity-Check Codes,
- Sum-Product Algorithm,
- Additive white Gaussian noise.

1. Uvod

Že dolgo vemo, da prihaja pri prenosu podatkov preko komunikacijskih sistemov do napak. Naj gre za enostaven prenos govora preko telekomunikacijskega sistema ali pa za digitalen prenos podatkov s sond na oddaljenih planetih, izkaže se, da podatki na strani oddajanja niso nujno enaki podatkom na strani sprejemanja.

V preteklosti se je izkazalo, da je možno z boljšimi tehnološkimi rešitvami in različnimi zapisi podatkov bistveno izboljšati prenos podatkov. Še vedno pa ne moremo vplivati na razne zunanje vplive, kot so elektromagnetni vplivi, medsebojno motenje frekvenc, razna sevanja pa tudi pokvarljivost naprav zaradi raznih vzrokov. Skratka, zaenkrat še ni videti, da bi bil kdaj prenos podatkov po raznih sistemih popolnoma zanesljiv.

Kakorkoli že, četudi bi morda dobili že enkrat popolnoma dovršeno tehnologijo, nikoli ne bomo uspeli nadzirati vplive narave na tehnologijo. Največja ironija pa je v tem, da je še največja težava v nadzoru samih akterjev, ki sodelujejo v procesu. V veliko primerih se je namreč pokazalo, da je ob pojavu raznih težav in katastrof kriv ravno človek sam.

Eden od načinov, kako povečati zanesljivost prenosa, je uporaba kod za popravljanje napak. Sporočilo pred pošiljanjem najprej kodiramo tako, da mu dodamo nekaj kontrolnih bitov, s pomočjo katerih lahko ugotovimo, ali je pri prenosu prišlo do napak in kje so se napake pojavile.

V zadnjem času se v ta namen vse bolj uporablja poseben razred kod, imenovanih paritetne kode z nizko gostoto (angl., Low-Density Parity-Check Codes). Le-te so podane s paritetno matriko, ki omogoča preverjanje, ali je pri prenosu prišlo do napake. Njihovo ime izhaja iz lastnosti paritetne matrike, da ima majhno število enic v primerjavi s številom ničel. Kode so bile sicer odkrite že v šestdesetih letih, vendar so bile zaradi omejenih zmogljivosti tehnologije do sedaj pozabljene. Sedaj, ko teh omejitev ni več, pa so se spet začele pogosteje uporabljati. Ob dobro generiranih paritetnih matrikah in uporabi iterativnih dekodirnih postopkov so LDPC kode namreč zelo učinkovite pri popravljanju napak pri prenosu.

S pomočjo predstavitve LDPC kod s Tannerjevim grafom lahko zelo enostavno opišemo dekodirni algoritem z izmenjevanjem sporočil (angl. Sum-Product Algorithm). S posebnim iterativnim postopkom namreč dobimo z izmenjevanjem sporočil med točkami tega grafa oceno poslane besede, za katero nato preverimo, ali je enaka poslani besedi.

Znanih je več postopkov za generiranje paritetnih matrik LDPC kod, eden najuspešnejših je algoritem PEG (angl., Progressive Edge Growth), čigar namen je kar najbolj povečati najkrajši cikel v Tannerjevemu grafu. Z daljšimi cikli v Tannerjevem grafu se namreč poveča minimalna razdalja koda in s tem je večja zanesljivost prenosa.

V diplomskem delu obravnavamo paritetne kode z nizko gostoto ter opišemo postopek PEG za generiranje teh vrst kod. Omenjeni dekodirni postopek z izmenjevanjem sporočil je na konkretnem primeru paritetne matrike podrobno pojasnjen. Sam komunikacijski sistem in vpliv šuma v komunikacijskem kanalu bomo simulirali z enim danes najbolj uporabnih modelov, to je Gaussov kanal z belim šumom (angl. Additive white Gaussian noise).

V zadnjem delu preverjamo, kako se spreminja učinkovitost dekodirnega postopka na matrikah različnih dimenzij s postopnim spreminjanjem šuma prisotnega v komunikacijskem kanalu. Matrike, ki jih preizkušamo, smo pridobili s postopkom PEG. Izmed večjega števila generiranih matrik smo izbrali tiste z največjo ocenjeno minimalno razdaljo. Na izvedenem številu iteracij smo potem analizirali, v koliko primerih uspe prejemnik ugotoviti poslano kodno besedo.

V dodatku je vključena vsa izvorna koda, potrebna za simuliranje opisanega komunikacijskega kanala in dekodirnega postopka, napisana v programu MATLAB 7.7.0 R2008b, tako da si bo lahko bralec natančno ogledal celoten proces delovanja komunikacijskega sistema.

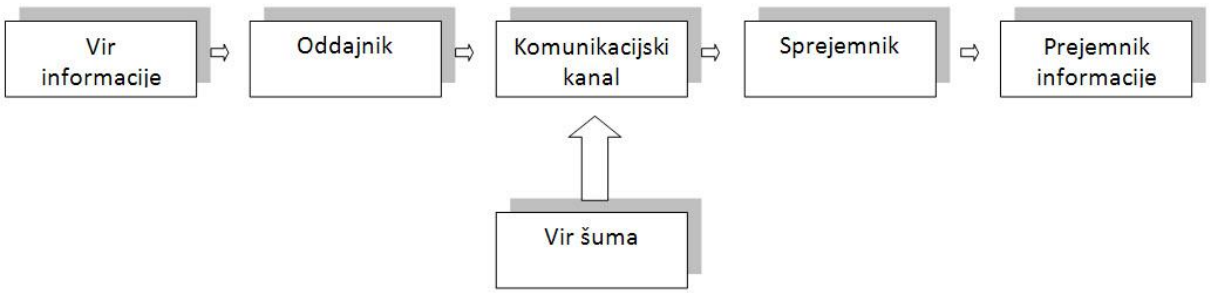
2. Komunikacijski sistem

V tem poglavju si bomo najprej pogledali namen posameznih komponent komunikacijskega sistema ter njihovo osnovno delovanje. Zaradi raznih nezaželenih vplivov, ki jih je težko izničiti ter kontrolirati, poimenovali jih bomo *šum*, med prenosom podatkov prihaja do pojava napak. Šum, ki deluje na komunikacijski kanal, je možno obvladovati na različne načine. V praktični uporabi se je kot model za komunikacijski kanal zadnje čase zelo izkazal kanal z uporabljenim belim Gaussovim šumom, ki ga bomo v nadaljevanju poglavja natančneje analizirali. V zaključku tega poglavja pa bomo preučili, kako s pomočjo kodiranja podatkov doseči, da bo nastalo kar najmanj napak. Najprej pa si poglejmo delovanje splošnega komunikacijskega sistema.

Leta 1948 je Claude Shannon v dokumentu *A Mathematical Theory of Communication [8]* predstavil koncepte in teoreme, sedaj priznane kot osnova teorije informacij. Predstavil je komunikacijski sistem kot množico povezanih enot, namenjen za prenašanje informacij. Sestavljajo ga:

- *vir informacije*, to je začetna točka, kjer so sporočila poslana,
- *oddajnik*, to je kodirna naprava, ki pretvori sporočilo v tako obliko signala, da ga je možno pošiljati po komunikacijskem kanalu,
- sam *komunikacijski kanal*, ki je medij, po katerem se signal pošilja,
- *sprejemnik*, to je dekodirna naprava, ki pretvarja prejeti signal v izvorno sporočilo, in je obratna funkciji oddajnika,
- *prejemnik informacije*, to je končna točka, kateremu je sporočilo namenjeno in
- *vir šuma*, ki vpliva na komunikacijski kanal in na popačenje poslanega signala.

Privzeto je, da so viri šuma neodvisni od vira pošiljanja informacij.



Slika 1.1: Shema komunikacijskega sistema

Množico $V_2 = \{0,1\}$ bomo imenovali kodna abeceda. Opazimo, da je V_2 končni obseg za operaciji seštevanja in množenja po modulu 2. Nizu znakov iz množice V_2 pravimo *beseda*.

a	b	$a + b$	$a \cdot b$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

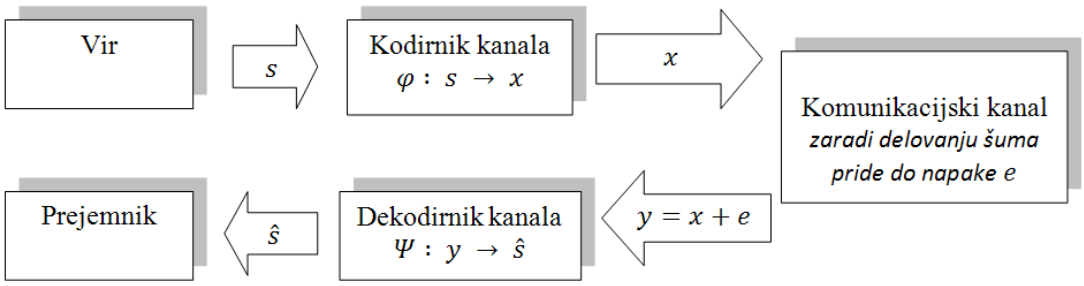
Slika 1.2: Operacije nad V_2

Naj bo $\mathcal{C}$ podmnožica ${V_2}^*$; ${V_2}^* = \bigcup_{i=1}^{\infty} {V_2}^i$. Potem $\mathcal{C}$ imenujemo *kod*, njegove elemente pa *kodne besede*. Kodne besede so lahko različnih dolžin. V naši obravnavi se bomo omejili na primer, ko imajo vse kodne besede enako dolžino. Če so vse kodne besede sestavljene iz enakega števila simbolov n, potem kod imenujemo *bločni kod* dolžine n. Število različnih kodnih besed v kodu označimo z M in velja $M \leq 2^n$.

Da bi lahko zagotovili zanesljiv prenos, morajo biti sporočila kodirana; *kodirnik* kanala preslika vsako besedo dolžine k v njegovo kodno zamenjavo dolžine n, torej vsakemu sporočilu s mora ustrezati končna sekvenca $x = (x_1, \ldots, x_n)$; $x_i \in V_2$. Preslikavo $\varphi : s \rightarrow \mathcal{C}$ imenujemo *kodirni postopek*. Eden od pomembnejših parametrov koda je razmerje med dolžino sporočila dolžine k in dolžino kodne besede n in se imenuje *razmerje koda*; $R = \frac{k}{n}$ *[6]*.

Prejemnik poskuša obnoviti izvorno sporočilo s tako, da obdeluje prejeto sekvenco $y = (y_1, \ldots, y_n)$; $y_i \in V_2$. Preslikavo, kjer prejemnik poskuša iz te sekvence obnoviti poslano kodno besedo, $\Psi : y \rightarrow \hat{s}$, imenujemo *dekodirni postopek*. Ta postopek je v osnovi obraten kodirnemu postopku, napravo, ki opravlja to delo, pa imenujemo pripadajoče *dekodirnik*. Naloga kodirnika in dekodirnika kanala je, da naredita prevajanja informacije v kanalu odporno na šum.

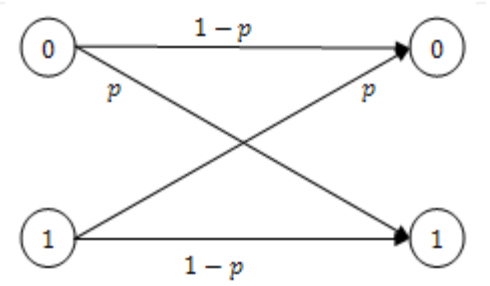
Zaradi vpliva šuma prisotnega v kanalu je lahko prejeto sporočilo $\hat{s}$ drugačno od poslanega s, kar poimenujemo *dekodirna napaka*.



Slika 1.3: Shema delovanja komunikacijskega sistema

2.1 Komunikacijski kanal

Za osnovni model prenosa podatkov preko komunikacijskega kanala bomo definirali *dvojiški simetrični kanal* (angl. binary symmetric channel; BSC*)*. Podatki, ki jih pošljemo preko tega kanala so elementi množice V_2, splošno poznani kot *biti*. Verjetnost napake pri prenosu p; $0 \leq p < \frac{1}{2}$ je simetrično enaka za oba bita, in iz teh dveh lastnosti izhaja tudi njegovo ime.



Slika 1.4: Verjetnost napake p za kanal BSC je simetrično enaka za elemente V_2

V nadaljevanju si bomo ogledali kanal z dodanim belim Gaussovim šumom (angl. Additive white Gaussian noise; AWGN) *[2]*. Motnja v kanalu AWGN je normalno porazdeljena z povprečno vrednostjo μ in standardno deviacijo σ; gostota verjetnosti je podana z izrazom $p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.

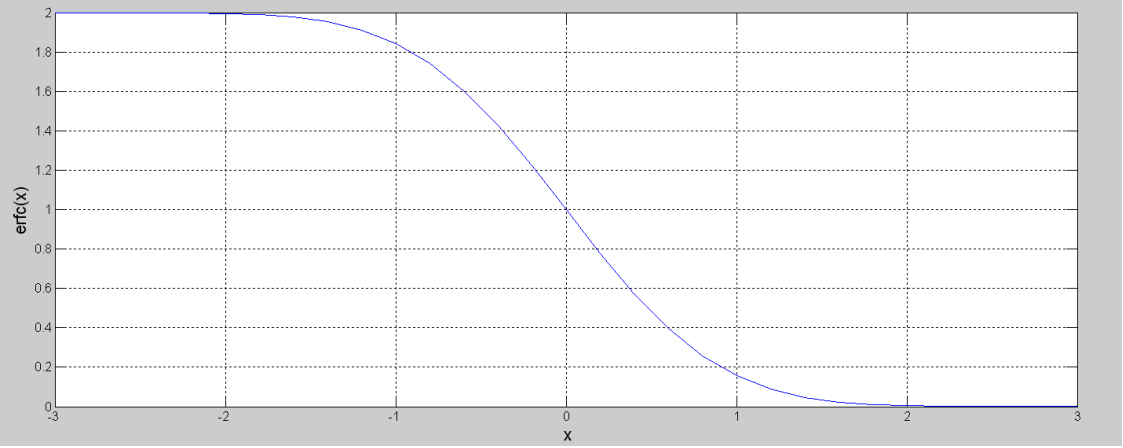
BPSK modulacija

Naj predstavlja $x = (x_1, \ldots, x_n)$ kodno besedo iz V_2^n. Vhodne podatke (bite) najprej pretvorimo v format za prenos po kanalu. Prenosni sistem je definiran tako, da z uporabo BPSK (angl. *Binary Phase-Shift Keying*) *modulacije* in *amplitude* a preslikuje elemente $x \in V_2$ v prenosni format: $(2 \cdot x_i - 1) \cdot a$, pri tem bo v našem primeru amplituda $a = 1$. Tako se bo na primer beseda (1,0,0,1) preslikala v besedo $(1, -1, -1, 1)$. Tako moduliran prenosni format kodirnik nato pošlje kot vhod v kanal AWGN.

Verjetnost napačno prenesenega bita

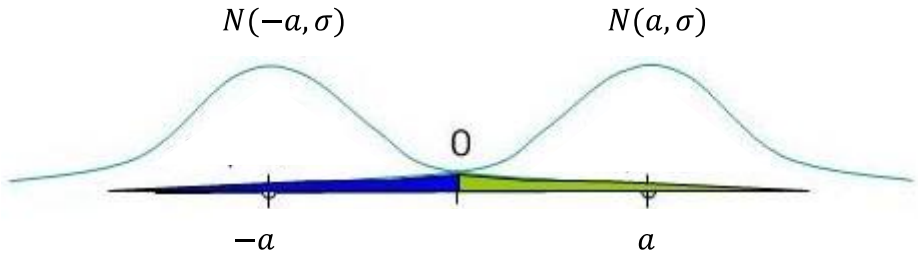
Napaka pri prenosu se bo pojavila, če je motnja nasprotno predznačena od podatka in po absolutni vrednosti večja od a (glej sliko 1.6). Verjetnost napake pri prenosu je enaka $P_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{t-u}{\sigma}}\, dt = 1 - Q(\frac{x-\mu}{\sigma})$. Funkcijo Q lahko izračunamo s

pomočjo *komplementarne funkcije napake* (angl. complementary error function; *erfc*): $Q(x) = \frac{1}{2} \cdot erfc(\frac{x}{\sqrt{2}})$, pri tem velja: $erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2}\, dt$.



Slika 1.5: Komplementarna funkcija napake

Na spodnji sliki je razvidno, da pride do dekodirne napake v bližini povprečne vrednosti μ. Omejili se bomo na primer, ko je $\mu = 0$. Verjetnost napačnega dekodiranja (označena dela na sliki 1.6) je tako odvisna od standardne deviacije šuma.



Slika 1.6: Verjetnosti napačnega dekodiranja

Po postopku, podanem v literaturi *[4]*, velja, da je: $P(x_i|y_i) = \frac{1}{1+e^{-\frac{2ay_i}{\sigma^2}}}$. In predvsem ta podatek, je glavna prednost modela komunikacijskega kanala AWGN v primerjavi z kanalom BSC, saj pove, kako blizu je sprejeta vrednost glede na oddano. Bralec pa si lahko v razdelku 4.4 ter v razdelku 5.2 ogleda, kako je v našem primeru kanal AWGN realiziran.

2.2 Kodi za popravljanje napak

Kodiranje lahko izkoristimo za popravljanje napak, ki so nastale med prenosom, kar bomo opisali v tem razdelku. Naj bosta x in y besedi iz ${V_2}^n$; $x = (x_1, x_2,..., x_n)$ in $y = (y_1, y_2, \ldots., y_n)$. *Hammingova razdalja* med x in y je definirana kot število mest, na katerih se ti dve besedi razlikujeta:

$$d_H(x,y) = |\{i: x_i \neq y_i, 0 \leq i \leq n\}| = \sum_{i=1}^{n} x_i \oplus y_i .$$

Hammingova razdalja ima naslednje lastnosti:

- $d(a,b) \geq 0,\ a \neq b,$
- $d(a,a) = 0,$
- $d(a,b) = d(b,a),$
- $d(a,c) \leq d(a,b) + d(b,c).$

Definirali bomo tudi *težo besede* t kot število neničelnih elementov besede:

$t(x) = |\{i: x_i \neq 0, 0 \leq i \leq n\}|$. Velja torej $d_H(x,y) = t(x + y)$.

Naj bo $x \in {V_2}^n$ poslana kodna beseda in $y \in {V_2}^n$ sprejeta beseda, popačana zaradi vpliva šuma e: $y = x + e$. Pri dekodiranju želimo poiskati tisto besedo, ki se od prejete besede čim manj razlikuje. Pravilu, da prejeto besedo y dekodiramo v tisto kodno besedo x, za katero je $d_H(x,y)$ najmanjša, pravimo *pravilo najmanjše razdalje*. V primeru več enakih izbir se odločimo za naključno, oziroma zahtevamo ponovno pošiljanje besede.

Iz tega sklepa sledi, da je potrebno za varen prenos izbrati $M \leq 2^n$ kodnih besed tako, da so Hammingove razdalje med njimi v prostoru ${V_2}^n$ čim večje. Čemu je tako? Namreč, četudi pride do napak pri prenosu, se prejeta kodna beseda ne bo preveč oddaljila od poslane besede in se ne bo zgodilo, da bi se bolj približala kaki drugi kodni besedi.

Glede na podan kod $\mathcal{C}$ lahko definiramo *minimalno Hammingovo razdaljo*, poznano tudi kot *najmanjša razdalja koda*, $d_{min} = d_H(\mathcal{C})$, ki nam pove minimalno razdaljo med vsemi pari kodnih besed v $\mathcal{C}$ in s tem zmožnost uspešnega dekodiranja:

$$d_{min} = \min_{x,y \in \mathcal{C}} \{d_H(x,y);\ x \neq y\}.$$

Kod $\mathcal{C}$ *je* (n, M, d_{min}) *–kod*, če velja:

- n je bločna dolžina,
- M nam daje število kodnih besed,
- d_{min} pa je minimalna Hammingova razdalja.

Primer:

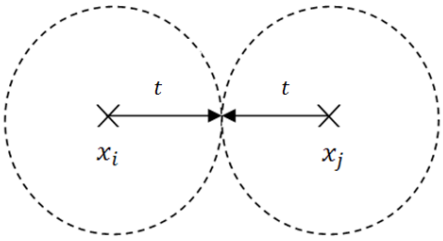
Kod $\mathcal{C} = \{000,011,110,101)$ je $(3,4,2)$ −kod, saj ima $M = 4$ kodne besede, pri tem ima vsaka dolžino $n = 3$, minimalno razdaljo pa ugotovimo s primerjanjem vsake kodne besede z vsako drugo in tako ugotovimo:

$d(0\mathbf{00}, 0\mathbf{11}) = d(\mathbf{00}0, \mathbf{11}0) = d(\mathbf{0}0\mathbf{0}, 1\mathbf{0}1) = d(\mathbf{0}1\mathbf{1}, 1\mathbf{1}0) = d(\mathbf{01}1, \mathbf{10}1) = d(1\mathbf{10}, 1\mathbf{01}) = 2,$

torej je $d_{min} = 2$.

Hammingova krogla

Hammingova krogla s središčem x in polmerom t je množica besed iz ${V_2}^n$, ki so od x oddaljene za največ t. Če je razdalja koda vsaj $2t + 1$, se Hammingove krogle s središči v kodnih besedah x in polmerom t med seboj ne sekajo. Iz tega sledi, da če je na sprejeti besedi manj kot t napak, tedaj ne bomo pobegnili izven krogle s središčem v izvorni kodni besedi in posledično ne bo prišlo do napak pri dekodirnem postopku. V primeru, če je razdalja večja, pa imamo težave pripisati kateri vhodni besedi pripada prispela beseda.



Slika 1.6: Hammingovi krogli s polmerom t

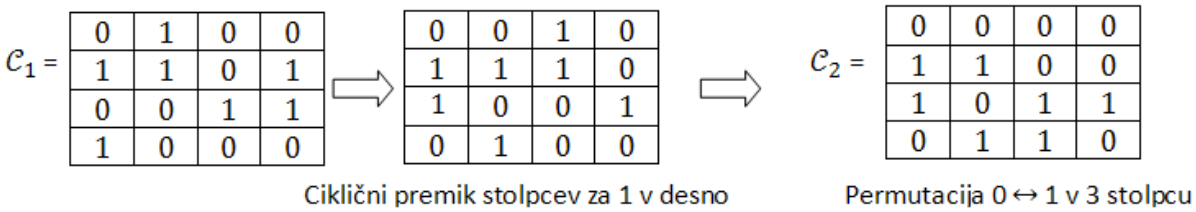
Če poznamo minimalno razdaljo koda, lahko izračunamo polmer t okrog kodnih besed, da bodo krogle disjunktne takole:

$$d_{min} = 2t + 1 \rightarrow t = \left\lfloor \frac{(d_{min}-1)}{2} \right\rfloor.$$

Kod z minimalno razdaljo d_{min} lahko torej uspešno popravi vse napake na največ $\left\lfloor \frac{(d_{min}-1)}{2} \right\rfloor$ mestih. Lahko pa se zgodi, da je pri dani vhodni kodni besedi x vpliv šuma premočan in je napaka prevelika in takrat se lahko zgodi, da je prejeta beseda bolj podobna kateri drugi kodni besedi.

Ekvivalentnost kodov

Koda $\mathcal{C}_1$ in $\mathcal{C}_2$ sta *ekvivalentna*, če lahko $\mathcal{C}_2$ dobimo iz $\mathcal{C}_1$, z zaporedjem transformacij in sicer s permutacijo stolpcev ali permutacijo simbolov v izbranem stolpcu kodne matrike. V tem primeru zapišemo $\mathcal{C}_1 \sim \mathcal{C}_2$. Ekvivalentna koda sta enakovredna glede popravljanja napak, ker je moč popravljanja napak odvisna samo od razdalje med posameznimi kodnimi besedami. Na sliki 1.7 je prikazano, kako iz koda $\mathcal{C}_1$ dobimo ekvivalenten kod $\mathcal{C}_2$.

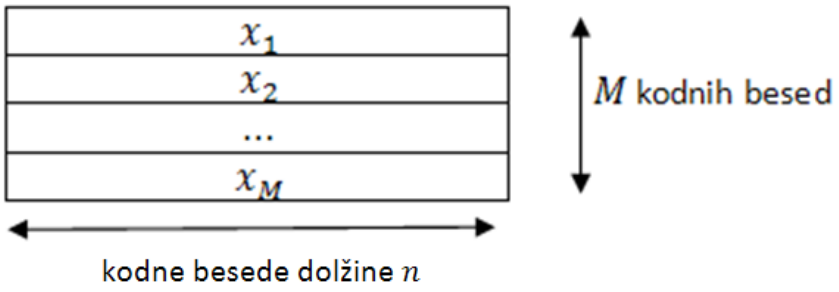


Slika 1.7: Ekvivalentna koda nam dajeta enako Hammingovo razdaljo

3. Linearni bločni kodi

Glede na to, kako iz sporočila sestavimo kodno besedo, lahko kode za odkrivanje napak razdelimo na dva velika razreda, namreč na konvolucijske kode in bločne kode. Oba načina sta podprta v praktični uporabi, v naši obravnavi pa se bomo osredotočili na linearne bločne kode, predvsem na tip paritetnih kod z nizko gostoto, ki jih bomo podrobneje predstavili v 4. poglavju.

Eden od glavnih problemov pri teoriji kodiranja je za dana n in M poiskati (n, M, d_{min}) −kod z največjo možno razdaljo, saj je od razdalje neposredno odvisna moč popravljanja napak. To je v splošnem težak problem. Izkaže se, da je problem lažje obvladljiv, če ima kod neko matematično strukturo. V primeri linearnih kod to pomeni, da je kodiranje možno izvajati preko manipulacij z matrikami.



Slika 2.1: Kodu $\mathcal{C} = \{x_1, x_2, \dots, x_M\}$ priredimo kodno matriko velikosti $M \times n$

Vemo, da je V_2 končen obseg, saj ima namreč operaciji seštevanja in množenja z vsemi potrebnimi lastnostmi. Kod $\mathcal{C}$ je *linearen*, če je vektorski podprostor prostora V_2^n. Potem ima *bazo*, ki jo lahko zapišemo kot množico linearno neodvisnih kodnih besed $\{x_1, x_2, ..., x_k\}$. To pa nato pomeni, da lahko katerokoli kodno besedo $x \in \mathcal{C}$, zapišemo kot linearno kombinacijo elementov baze:

$$x = u_1 x_1 + u_2 x_2 + \dots + u_k x_k; \quad u_i \in V_2, \quad 1 \leq i \leq k.$$

Elemente baze nato zložimo v matriko G velikosti $k \times n$, ki jo imenujemo *generatorska matrika G koda $\mathcal{C}$*. Vpeljimo še naslednjo oznako; $[n, k, d_{min}]$ −*kod* je *linearen* kod z bločno dolžino n dimenzije k in minimalno razdaljo d_{min} .

Zaradi dejstva, da je matrika G matrika dimenzije $k \times n$, obstaja tudi $m \times n$ dimenzionalna matrika H, za katero velja $G \cdot H^T = 0$, pri čemer velja $m = n - k$. Ali drugače povedano, za vsako kodno besedo $x \in \mathcal{C}$ velja $x \cdot H^T = 0$. Matriko H imenujemo *paritetna matrika koda $\mathcal{C}$*, Drugo ime zanjo je *nadzorna matrika*.

Primer:

Naj bo $\mathcal{C} = \{0000,1110,0011,1101\}$. Ker je poljubna linearna kombinacija kodnih besed spet kodna beseda, je kod linearen. Ena od njegovih baz je tako $\{1110,0110\}$.

$\mathcal{C} =$

~~0~~	~~0~~	~~0~~	~~0~~
1	1	1	0
0	0	1	1
~~1~~	~~1~~	~~0~~	~~1~~

$G =$

1	1	1	0
0	0	1	1

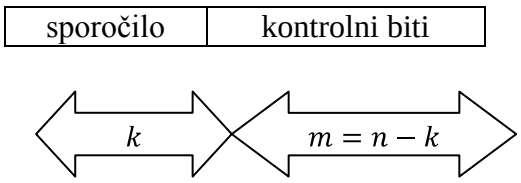
Slika 2.2: S iskanjem linearno neodvisnih besed koda $\mathcal{C}$ dobimo generatorsko matriko G

3.1 Kodirni postopek z generatorsko matriko *G* in paritetno matriko *H*

Podano imamo sporočilo $s = (s_1, s_2, \dots, s_k)$; $s_i \in V_2$, ki ga želimo prenesti preko komunikacijskega kanala, ter generatorsko matriko G koda $\mathcal{C}$. Kodirni postopek se prične z množenjem sporočila in generatorske matrike; $x = s \cdot G = s_1x_1 + s_2x_2 + \dots + s_kx_k$, pridobljen x pa je *kodna beseda*, saj je linearna kombinacija besed iz G.

Če ima matrika G posebno obliko $G = (I_{k \times k} | P_{k \times m})$, kjer je I identična matrika dimenzije $k \times k$, potem pravimo da je *sistematična*. Pravimo tudi, da je G v *standardni obliki*. Generatorsko matriko v taki obliki bomo označevali z G_{sys}. Za vsak kod obstaja ekvivalenten kod, ki ima generatorsko matriko v sistematični obliki. Dano matriko lahko v sistematično obliko pretvorimo z Gaussovo eliminacijo in po potrebi z zamenjavo stolpcev.

Prednost uporabe kodirnega postopka s sistematično obliko, je v tem, da sedaj prejemnik ob pravilnem sprejemu besede enostavno prebere poslano sporočilo s, saj ta predstavlja prvi del kodne besede preostanek pa so tako imenovani *kontrolni biti* (slika 2.3).



Slika 2.3: Kodna beseda, iz katere enostavno preberemo sporočilo

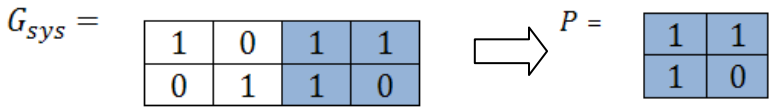
Dodatna prednost je tudi v tem, da v primeru sistematične oblike generatorske matrike zelo enostavno najdemo njej ustrezno paritetno matriko H: $H_{sys} = (P_{m \times k}{}^{T} | I_{m \times m})$, kjer je I identična matrika dimenzije $m \times m$, in zadošča enakosti $G \cdot H_{sys}{}^{T} = 0$

Primer :

Generatorska matrika G iz slike 2.2 ni v sistematični obliki. Z zamenjavo drugega in četrtega stolpca dobimo generatorsko matriko v sistematični obliki za ekvivalenten kod.

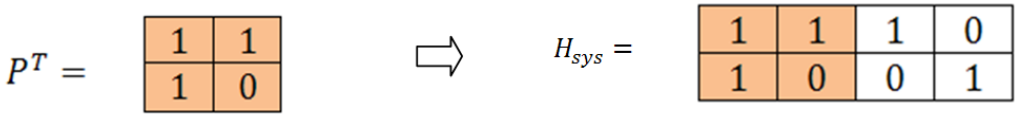
Slika 2.4: Generatorsko matriko pretvorimo v sistematično obliko z zamenjavo stolpcev

Po formuli $G = (I_{k \times k} | P_{k \times m})$, je označeni del na spodnji sliki matrika P, na levi strani je identiteta I.



Slika 2.5: Branje desnega dela matrike G_{sys}

Sestavimo sedaj matriko $H_{sys} = (P_{m \times k}{}^{T} | I_{m \times m})$. V tem primeru je $P^T = P$:



Slika 2.6: Končno sestavljanje sistematične paritetne matrike H_{sys}

3.2 Dekodirni postopek

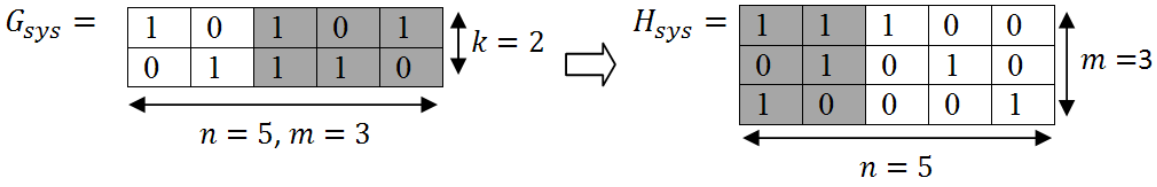
Dekodiranje je postopek, ki poišče kodno besedo x, ki je v nekem smislu najbližja sprejeti besedi y. Iskanja se lahko lotimo izčrpno, kjer primerjamo prejeto besedo z vsemi vhodnimi kodnimi besedami in izberemo tisto, ki ji je najbližja. Drug način je vpeljava koncepta *sindroma* d, definiranega kot $d = y \cdot H^T$, kjer je H paritetna matrika danega koda $\mathcal{C}$. Če prenašamo kodno besedo x, in je prejeta beseda enaka $y = x + e$, potem je sindrom prejete besede enak:

$$d = y \cdot H^T = (x + e) \cdot H^T = e \cdot H^T = (d_1,\ldots, d_m),$$

kar pomeni, da je sindrom odvisen samo od vektorja napake. Če je vektor napake ničelni vektor, potem bo sindrom sestavljen iz samih ničel in prejeta beseda bo veljavna kodna beseda. V primeru pojava enice v sindromu bo zaznana napaka v prejetem vektorju. Možno je seveda, da bo sindrom ničelni vektor tudi v primeru napake in sicer v primeru, če je vektor napake enak prejeti kodni besedi; število in mesta pozicij napak so take, da je prejeta beseda pretvorjena v drugo kodno besedo. Tega primera sindrom ne more zaznati. Posledica tega je, da obstaja 2^m vektorjev napak, ki generira enake sindrome.

Primer:

Naj bo $\mathcal{C}$ $[5,2,3]$ – kod, podan z generatorsko matriko G_{sys} in paritetno matriko H_{sys}:



Slika 2.7: Dekodirni postopek – podani sta sistematična generatorska matrika G_{sys} in pripadajoča paritetna matrika H_{sys}

Sindrom prejetega vektorja je enak sindromu vektorja napake. Reševanje naslednjega sistema enačb nam bo podala oceno veljavne kodne besede. Denimo, da sprejmemo kodno besedo $y = (11110)$.

$d = y \cdot H^T =$ | **1** | **1** | **1** | **1** | **0** | $\cdot$ | 1 | 0 | 1 |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

$=$ | 1 | 0 | 1 |
|---|---|---|

Slika 2.8: Dekodirni postopek: Izračun sindroma d

Opazimo pa lahko, da obstaja za ta primer sindroma (101) 2^k vzorcev , torej $2^2 = 4$ različni vzorci napak, ki generirajo enak sindrom.

e	$s = e \cdot {H_{sys}}^T$
10000	101
11110	101
00101	101
01011	101

Slika 2.9: Dekodirni postopek: Branje napake pripadajoča sindromu d

Seveda, se zdaj pojavi vprašanje, katera izmed 4 besed je bila poslana, oziroma katera je kar najbolj podobna besedi, ki smo jo prejeli. Smiselno je privzeti, da je napaka tista, ki ima najmanjšo težo (poglavje 2.2). V našem primeru je napaka $e = (10000)$. Poslana kodna beseda pa je torej:

$$y = x + e \rightarrow x = y - e = (11110) - (10000) = (\mathbf{01110}) .$$

4. Paritetne kode z nizko gostoto

Paritetne kode z nizko gostoto (angl. Low-Density Parity-Check ali s kratico LDPC) so razred linearnih bločnih kod. Njihovo ime izhaja iz lastnosti paritetne matrike, da ima zelo malo število enic v primerjavi s številom ničel. Prvi jih je predstavil Gallagher v šestdesetih letih, vendar so se začele bolj uporabljati šele v zadnjih desetih letih. Razlog je bil v tehničnih zmogljivostih, predvsem v omejenih kapacitetah pri iterativnem dekodiranju, saj njihova uporaba sloni na uporabi velik matrik.

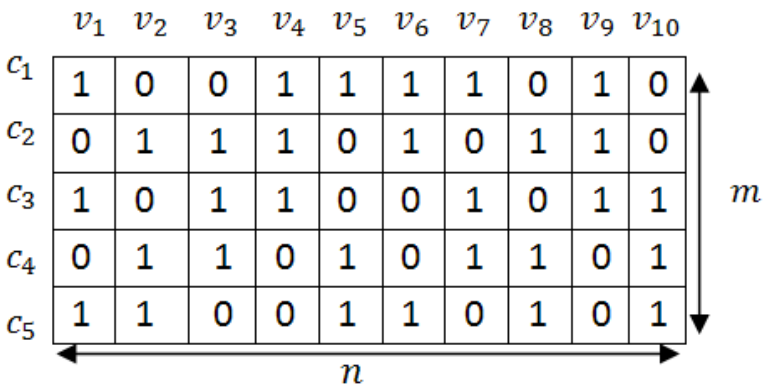
Prednosti LDPC kod so, da je implementacija tudi zaradi načina paralelizacije manj kompleksna in zelo enostavna v primerjavi z drugih dekodirnimi algoritmi, zelo preprosta je tudi njihova konstrukcija. Njihova najmočnejša stran pa je, da omogočajo ob dobrih postavitvah precej veliko medsebojno Hammingovo razdaljo in s tem ob uporabi velikih matrik odlične rezultate pri odkrivanju možnih napak pri prenosu. Če pa si pogledamo njene slabše strani, lahko omenimo da imajo precej višjo kompleksnost kodiranja, poleg tega je velikokrat potrebno več iteracij dekodirnega postopka v primerjavi z drugimi algoritmi, ki to zmorejo bolj učinkovito *[1,2,3,4]*.

V naslednjih razdelkih bomo predstavili LDPC kode na dva načina in sicer z matrično ter z grafično predstavitvijo s Tannerjevim grafom. Nadaljevali bomo s postopkom PEG (angl. Progressive Edge Growth), ki služi za generiranje paritetnih matrik LDPC kod. Podrobno si bomo pogledali dekodirni postopek SPA (angl. Sum-Product Algorithm), ki temelji na izročanju sporočil med točkami Tannerjevega grafa. Postopek iterativno spreminja prejeto besedo, dokler prejemnik ne ugotovi, da je popravljena beseda enaka neki kodni besedi oziroma ni preseženo predpisano število iteracij.

4.1 Matrična predstavitev

LDPC kod je linearen bločni kod, ki ga podamo s pomočjo paritetne matrike. V celotnem poglavju bomo število stolpcev paritetne matrike označevali kot n, število vrstic pa z m. LDPC kod je *regularen*, če ima njegova paritetna matrika isto število enic v vsakem stolpcu in isto število enic v vsaki vrstici, ter je *neregularen*, če število enic v vrsticah ali stolpcih ni konstantno. Označimo z w_r povprečno število enic v vsaki vrstici matrike in z w_c število enic v vsakem stolpcu. Za paritetne matrike LDPC kodov velja, da sta števili w_c in w_r znatno manjši od števila vrstic oziroma stolpcev. Velja $w_r = w_c \cdot \frac{n}{m}$, razmerje koda R pa dobimo nato kot $R = \frac{w_c - w_r}{w_c}$.

Na sliki 3.1 je primer regularne paritetne matrike z $w_c = 3$ in $w_r = 6$. Razmerje koda R je zato enako 0.5.



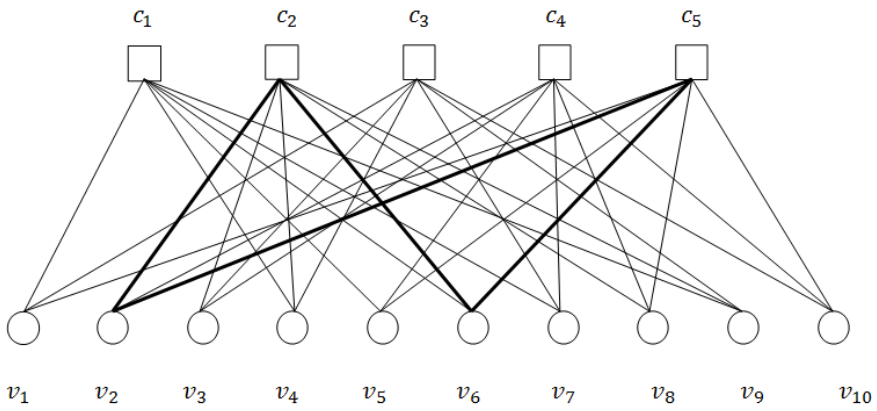
Slika 3.1: Regularna paritetna matrika

4.2 Grafična predstavitev

Tanner je predvsem zaradi boljšega pregleda nad dekodirnim algoritmom uvedel predstavitev LDPC kod z dvodelnim grafom, ki ga sedaj imenujemo *Tannerjev graf*. Točke grafa je ločil na dve množice; na *simbole* v in *testna vozlišča* c. Simbol v_j je povezan z testnim vozliščem c_i natanko tedaj, ko je v paritetni matriki H na presečišču vrstice i in stolpca j enica. Če so testna vozlišča povezana s simbolom, pravimo, da so *otroci* tega simbola. Če je nek simbol povezan s testnimi vozlišči, pravimo da je *starš* teh testnih vozlišč. Če je kod regularen, je testno vozlišče povezano z w_c simboli in vsak simbol je povezan z w_r testnimi vozlišči.

Na sliki 3.2 je prikazan Tannerjev graf, ki pripada paritetni matriki s slike 3.1. Ima $n = 10$ simbolov in $m = 5$ testnih vozlišč. Vsako testno vozlišče se povezuje z šestimi simboli, $w_c = 6$, vsak simbol se povezuje z tremi testnimi vozlišči, $w_r = 3$.

Dolžino najkrajšega cikla v Tannerjevem grafu bomo označevali z g_0. Na sliki 3.2 je s poudarjenimi povezavami prikazano, da je v tem primeru dolžina najkrajšega cikla enaka dolžini sploh najkrajšega možnega cikla v dvodelnem grafu: $g_0 = 4$. Dobro je, če dolžina zadostuje pogoju $g_0 > 4$ zaradi učinkovitejšega dekodiranja. V nadaljevanju bomo obravnavali postopek PEG, katerega cilj je kar največja dolžina najkrajšega cikla v Tannerjevem grafu.



Slika 3.2: Tannerjev graf, ki pripada paritetni matriki s slike 3.1

4.3 Postopek PEG za konstrukcijo LDPC kod

Paritetno matriko za LDPC kod je možno generirati povsem naključno. Pri tem dobimo dobre rezultate dekodiranja, pomembno je le, da je matrika redko posejana z enicami in da se upošteva pravilo razporeditve; v našem primeru so to enake stopnje simbolov in enake stopnje testnih vozlišč. Sicer pa obstajajo različne tehnike generiranja matrik, ki omogočajo enostavnejše kodiranje in dekodiranje v primerjavi s povsem naključnim generiranjem. To so: kode osnovane na končnih geometrijah, Reed-Solomonovih kodah, kodah Gilberta in druge *[1]*.

V zadnjem času se je še posebej izkazal postopek PEG, s katerimi so bile skonstruirane nekatere do sedaj najboljše paritetne matrike *[15]*. Osnovna ideja postopka PEG je konstrukcija Tannerjevega grafa, pri katerem poskušamo dobiti kar največjo dolžino najkrajšega cikla ob upoštevanju stopenj simbolov in stopenj testnih vozlišč.

Opišimo sedaj algoritem PEG:

VHOD algoritma :

m - število testnih vozlišč,
n - število simbolov,
w_r – stopnje simbolov,
w_c - stopnje testnih vozlišč.

metajezik ALGORITMA:

Naj bodo $c_i,\ 1 \leq i \leq m$, testna vozlišča in $v_j,\ 1 \leq j \leq n$, simboli. Na začetku ni povezav med testnimi vozlišči in simboli.

od $j = 1$ do n *ponavljaj:*

v prvem koraku *naredi*:
izberemo tisto testno vozlišče c_i , ki ima najmanjšo stopnjo
poveži simbol v_j in testno vozlišče c_i

od drugega koraka naprej do w_r *ponavljaj*:
Če lahko, poveži v_j z nekim prostim, še nepovezanim tesnim vozliščem c_i, sicer pa preglej poti, ki vodijo iz trenutno aktualnega simbola v_j.

pri tem za vsako pot *naredi*:
Na tej poti nadaljuj do največje možne globine in se tam poveži testnim vozliščem c_i, ki ima najmanjšo stopnjo. Pri tem upoštevaj, da je v našem obravnavanem primeru največja stopnja testnega vozlišča w_c .

IZHOD algoritma:

Paritetna matrika H z m vrsticami in n stolpci.

Postopek PEG je že implementiran in dostopen na spletu *[13]*. S pomočjo tega algoritma kot zgled sestavimo paritetno matriko z $m = 5$ testnimi vozlišči in $n = 10$ simboli. Zaradi velikih dimenzij se pojavljajo težave posebej pri večjih paritetnih matrikah. Zato bomo matrike podajali v skrajšani obliki. Zapis pojava posameznih enic v posamezni vrstici paritetne matrike bomo pisali v pripadajočo zaporedno vrstico matrike, ki jo bomo imenovali $rowsOnes$. Podobno bomo pojav posameznih enic v posameznem stolpcu paritetne matrike zapisali v pripadajočo zaporedno vrstico matrike, ki jo bomo imenovali $colOnes$.

$rowsOnes$	p1	p2	p3	p4	p5	p6
1	1	4	5	6	7	9
2	2	3	4	6	8	9
3	1	3	4	7	9	10
4	2	3	5	7	8	10
5	1	2	5	6	8	10

Slika 3.3: Skrajšan zapis enic po vrsticah matrike H s slike 3.1

$colOnes$	p1	p2	p3
1	1	3	5
2	2	4	5
3	2	3	4
4	1	2	3
5	1	4	5
6	1	2	5
7	1	3	4
8	2	4	5
9	1	2	3
10	3	4	5

Slika 3.4: Skrajšan zapis enic po stolpcih matrike H s slike 3.1

Konstrukcija generatorske matrike

Kako pridobiti generatorsko matriko iz podane generirane paritetne matrike H? Kot smo že omenili, nam postopek PEG vrne paritetno matriko, ki jo lahko zapišemo v obliki: $H = (A_{m \times k} | B_{m \times m})$. Matriko H pretvorimo z Gaussovo eliminacijo in po potrebi z zamenjavami stolpcev v sistematično obliko $H_{sys} = (P_{k \times m} | I_{k \times k})$. Generatorsko matriko v sistematični obliki dobimo nato enostavno tako: $G_{sys} = (I_{m \times m} | P_{m \times k}{}^{T})$. V primeru, da je bila potrebna zamenjava stolpcev, ustrezne stolpce zamenjamo tudi v prvotni matriki H. Podobno kot paritetno matriko H, zapišemo tudi pripadajočo generatorsko matriko G v skrajšani obliki; naštejemo le mesta pojavitev enic. Na sliki 3.5 je generatorska matrika, ki ustreza paritetni matriki s slike 3.1.

Generatorska matrika G	1	2	3	4	5	6	7	8	9	10
	5	1	1	1	2	1	2	3	4	5
	0	2	2	4	0	0	0	0	0	0
	0	3	5	5	0	0	0	0	0	0

Slika 3.5: Skrajšan zapis enic transponirane generatorske matrike G

4.4 Dekodiranje z izročanjem sporočil

Spoznali smo že, da na poslano kodno besedo x preko komunikacijskega kanala vpliva šum in povzroči, da prejeta beseda y ni nujno enaka poslani kodni besedi; to zapišemo kot $y = x + e$. V našem obravnavanem dekodirnem postopku bomo privzeli, da kodno besedo pošiljamo preko kanala AWGN (glej poglavje 2.1). To pomeni, da vsakemu bitu poslane kodne besede ustreza neko realno število pri prejeti besedi. Prejeto besedo pretvorimo v dvojiški vektor tako, da negativno število spremenimo v 0, pozitivno pa v 1.

Kako za dani kod, podan s paritetno matriko H preverimo, da je prejeta beseda y kodna beseda? Spomnimo se še enkrat poglavja 3.2, ko smo vpeljali koncept sindroma, definiranega s formulo: $s = y \cdot H^T = (x + e) \cdot H^T$. Ker je $x \cdot H^T = 0$ natanko tedaj, ko je *x* kodna beseda, lahko za preverjanje kodnih besed izračunamo sindrom.

Opisali bomo dekodirni *algoritem z izročanjem sporočil* (angl., Sum-Product algorithm ali SPA). Le-tega je najbolj priročno opisati na Tannerjevem grafu, kjer imamo povezave med dvema vrstama vozlišč; simboli v_j, ki predstavljajo prenesene bite in testnimi vozlišči c_i, ki predstavljajo *paritetne enačbe*, s katerimi so simboli povezani. Vrstice paritetne matrike podajajo simbole, ki sodelujejo v paritetnih enačbah, in sicer tako, da posamezna vrstica opisuje pripadajočo paritetno enačbo, pozicije enic (stolpcev paritetne matrike namreč) pa določajo indekse simbolov, sodelujočih v paritetni enačbi. Ko preverjamo parnost pravzaprav računamo sindrom.

Naj bo d sindrom: $s = y \cdot H^{T=}(x + e) \cdot H^T$, kjer je y prejeta beseda. Ker vemo, da je v primeru $s = 0$ vektor napake ničelni vektor in je prejeta beseda veljavna kodna beseda, lahko sedaj za v neki dekodirni iteraciji preračunani d naposled s paritetnimi enačbami zapišemo, kdaj pravzaprav vemo, da je napaka odpravljena in dekodiranje uspešno.

Primer:

Zapišimo paritetne enačbe za paritetno matriko s slike 3.3. Z d označimo besedo, za katero preverjamo parnost.

$$d_1 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_9 = 0,$$
$$d_2 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_8 \oplus d_9 = 0,$$
$$d_1 \oplus d_3 \oplus d_4 \oplus d_7 \oplus d_9 \oplus d_{10} = 0,$$
$$d_2 \oplus d_3 \oplus d_5 \oplus d_7 \oplus d_8 \oplus d_{10} = 0,$$
$$d_1 \oplus d_2 \oplus d_4 \oplus d_6 \oplus d_8 \oplus d_{10} = 0.$$

Algoritem z izročanjem sporočil deluje iterativno. Po vsaki iteraciji z računanjem sindroma preverimo, ali smo že dobili kodno besedo. V primeru uspeha bo prejemnik enostavno prebral sporočilo iz kodne besede. Iterativni postopek uspešno vrne poslano besedo, če ima Tannerjev graf obliko drevesne strukture, kar pomeni, da ne vsebuje ciklov. Izkazalo se je, ker je popolno odsotnost ciklov težko zagotoviti, da je možno z velikimi matrikami, ta pojav zelo omiliti.

Iterativni postopek se začne s tem, da vsak simbol v_j pošlje vsem svojim otrokom, testnim vozliščem c_i, verjetnost, ki jo bomo označili s Q_{ij}^x, da je testno vozlišče c_i v stanju $\{0,1\}$. To oceno pa simbol pridobi od vseh ostalih sosedov otroka, ki mu to vrednost trenutno pošilja. Vsako testno vozlišče c_i nato pošlje staršu v_j, verjetnost, ki jo bomo označili z R_{ij}^x, da je nadrejeno vozlišče v stanju x. To ocena pa testno vozlišče pridobi od vseh ostalih sosedov starša, ki mu to vrednost trenutno pošilja. Z pregledom vrednosti nad vsemi otroci posameznih simbolov v_j nato izračunamo oceno posameznega bita j dekodirane besede d. Izmenjava sporočil se ustavi, če v neki iteraciji izračun sindroma nad ocenjeno dekodirano besedo d izpolni pogoj $d \cdot H^T = 0$. Tedaj prejemnik ve, da ima pravo kodno besedo in lahko prebere sporočilo oddajnika *[3]* .

Otroke nekega simbola dobimo tako, da pregledamo vse njegove povezave v Tannerjevim grafu ali pa opazujemo pojavitev opazovanega simbola v paritetnih enačbah. Te pa lahko bolj enostavno dobimo ravno v pripadajoči vrstici matrike $colOnes$ (slika 3.4), saj nam posamezna vrstica $colOnes$ pove, s katerimi otroci je povezan. Otrokom se pripiše indeks pojavitve v opazovani vrstici. Starše nekega testnega vozlišča pa vidimo, da pregledamo vse njegove povezave v Tannerjevim grafu ali pa preberemo nastopajoče indekse simbolov in sicer pod tisto zaporedno številko paritetne enačbe, ki nam jo da indeks opazovanega testnega vozlišča. Z drugimi besedami, bolj enostavno jih lahko dobimo v pripadajoči vrstici matrike $rowsOnes$ (slika 3.2), saj nam posamezna vrstica $rowsOnes$ pove, s katerimi starši je testno vozlišče povezano.

Algoritem SPA lahko z metajezikom opišemo takole:

m - število testnih vozlišč,
n - število simbolov,
w_r – stopnje simbolov,
w_c - stopnje testnih vozlišč.

VHOD algoritma :

Redko posejana paritetna matrika H zapisana v skrajšani obliki z dimenzijami $m \times w_c$, pri tem je:
m - število testnih vozlišč,
w_c - stopnja testnih vozlišč.

Podani so tudi:

$n -$ število simbolov,
w_r $-$ stopnja simbolov,
$y -$ prejeti vektor ,
$\sigma -$ standardna deviacija šuma,
$L -$največje število iteracij.

metajezik ALGORITMA:

1. korak

Izračunaj verjetnosti p_j^0 in p_j^1 posameznih bitov prejetega $\boldsymbol{y_j}$, pri čemer $1 \leq j \leq n$:

$p_j^1 = P(x_j = 1|y_j) = 1/(1 + e^{\frac{-2ay_j}{\sigma^2}}),\ p_j^0 = 1 - p_j^1.$

Nato kreiraj tabeli Q_{ij}^0 in Q_{ij}^1 enakih dimenzij kot paritetna matrika H, ter jim dodeli vrednosti:

$Q_{ij}^x\ =\ p_j^x.$

2. korak

Kreiraj tabeli R_{ij}^0 in R_{ij}^1 enakih dimenzij kot paritetna matrika H, ter jim dodeli vrednosti:

$\delta R_{ij}\ =\ \prod_{t \neq j} Q_{i,t}^0\ -\ Q_{i,t}^1\ ,$

$R_{ij}^0 = \frac{1}{2} \cdot (1 + \delta R_{ij}),\ R_{ij}^1\ =\ 1 - R_{ij}^0.$

3. korak

Ocena posameznih bitov $\widehat{d_j}^{\,0}$ ter $\widehat{d_j}^{\,1}$ dekodirane besede $\widehat{d_j}$, pri čemer $1 \leq j \leq n$:

$\alpha_j\ = \frac{1}{p_j^0 \prod_t R_{t,j} + p_j^1 \prod_t R_{t,j}},$

$\widehat{d_j}^{\,0} = \alpha_j\ p_j^x \prod_t R_{t,j}\ ,\ \widehat{d_j}^{\,1} = 1 - \widehat{d_j}^{\,0}.$

Če dekodirana beseda $\widehat{d_j}$ zadostuje pogoju $\widehat{d_j}\ \cdot\ H^T = 0$, tedaj končaj.
Če je števec števila iteracij L enak 0, tedaj končaj.
Sicer nadaljuj, ter zmanjšaj števec števila iteracij L za 1.

4. korak

Posodobi verjetnosti Q_{ij}^0 in Q_{ij}^1, ki služijo kot vhod v naslednjo iteracijo:

$$\alpha_{ij} = \frac{1}{p_j^0 \prod_{t \neq i} R_{t,j} + p_j^1 \prod_{t \neq i} R_{t,j}}.$$

Osvežene vrednosti Q_{ij}^x so enake:

$$Q_{ij}^0 = \alpha_{ij}\, p_j^0 \prod_{t \neq i} R_{t,j}\ ,$$
$$Q_{ij}^1 = 1 - Q_{ij}^0\,.$$

Nadaljuj z 2. korakom.

IZHOD algoritma:

Veljavna kodna beseda $\hat{d}$ ali sporočilo o neuspehu.

Ocena časovne zahtevnosti

1. korak: računanje p_j^x: $2 \cdot n$, določitev vrednosti $Q_{ij}^x : m \cdot w_c$,
2. korak: določitev vrednosti $R_{ij}^x : m \cdot w_c \cdot (\frac{1}{2} \cdot (w_c - 1))$,
3. korak: izračun besede $\hat{d_j}$: $n \cdot w_r$,
4. korak: izračun vrednosti Q_{ij}^x: $m \cdot w_c \cdot (w_r - 1)$.

Skupaj: $\mathcal{O}(\, m \cdot {w_c}^2 + n \cdot w_r + m \cdot w_c \cdot w_r) \approx \mathcal{O}(n)$, če predpostavimo, da sta w_r in w_c majhni konstanti. Kar potem pomeni, da se z večanjem matrik čas dekodiranja povečuje linearno. To dejstvo bomo kasneje na testiranju različnih dimenzij matrik s pridom izkoriščevali.

4.4.1 Začetek algoritma

V nadaljevanju bomo na konkretnem primeru predstavili delovanje dekodirnega algoritma z izmenjevanjem sporočil.

Poslana kodna beseda x

Sporočilo, ki ga oddajnik želi poslati, bomo generirali naključno. Izvorno kodno besedo x nato dobimo preko formule $x = sporočilo\ s \cdot G_{sys}$ (poglavje 2.1). S poudarjenim delom je prikazan del sporočila v kodni besedi.

Kodna beseda x	1	2	3	4	5	6	7	8	9	10
	0	0	1	1	1	0	1	1	1	0

Slika 3.6 Oddajnik dobi z množenjem sporočila in generatorske matrike kodno besedo

Prejemnik prejme vrednost y

Oddana kodna beseda pri potovanju skozi specifičen komunikacijski kanal in delovanju (različnih nezaželenih) vplivov spremeni svojo prvotno (poslano) obliko. Oddajnik iz take prejete besede poskuša z nekim algoritmom pridobiti originalno kodno besedo.

Prejeta vrednost y	1	2	3	4	5	6	7	8	9	10
	-0.036	-1.713	0.294	0.490	0.931	-1.717	0.962	-0.098	0.707	-0.255
Trdo dekodiranje y>0	0	0	1	1	1	0	1	**0**	1	0

Slika 3.7: Poslana kodna beseda se preko potovanja skozi specifičen kanal deformira

Če bi prejemnik uporabil tehniko *trdega dekodiranja*, ki za posamezen element prejetega $y > 0$ predpostavi, da je bil izvorni poslani bit na tem mestu enak 1, sicer pa 0, bi bila napaka v primerjavi z poslano kodno besedi na 8 bitu.

	1	2	3	4	5	6	7	8	9	10
p_j^0	0.5478	0.9999	0.1721	0.0678	0.0068	0.9999	0.0058	0.6281	0.0224	0.7960
p_j^1	0.4522	0.0001	0.8279	0.9322	0.9932	0.0001	0.9942	0.3719	0.9776	0.2040

Slika 3.8: Verjetnost posameznega bita za vsak element prejetega $\boldsymbol{y}$

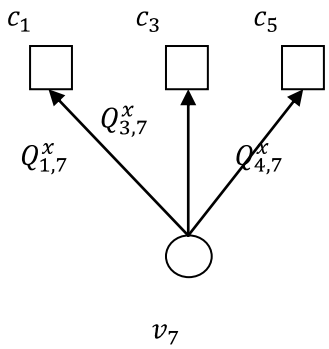
Izračun vrednosti Q_{ij}^x v prvi iteraciji

V postopku SPA ocenjujemo *pogojno verjetnost* (angl. A Aposteriori Probability ali APP), da je bil pri dani prejeti besedi y poslani bit besede x enak 0 ali 1. Predpostavimo, da je bila kodna beseda poslana po kanalu AWGN z amplitudo signala a in standardno deviacijo σ, formule smo izpeljali v poglavju 2.1:

V našem primeru je amplituda $a = 1$ standardna deviacija pa $\sigma = 0.6118$.

Primer:

Na sliki 3.9 je podan odseka Tannerjevega grafa iz katerega so jasno razvidni vsi otroci simbola $v_7, (j = 7)$, to so testna vozlišča $\{c_1, c_3, c_4\}$, $i \in \{1, 3, 4\}$, ter kako simbol v_7 pošlje vsem tem navedenim otrokom vrednost Q_{ij}^x (ki je v prvi iteraciji enaka informaciji p_j^x).



Slika 3.9: Grafični prikaz potovanja, kako simbol v_j pošlje svojim otrokom vrednost Q_{ij}^x

Q_{ij}^0	p1	p2	p3	p4	p5	p6
1	0.5478	0.0678	0.0068	0.9999	0.0058	0.0224
2	0.9999	0.1721	0.0678	0.9999	0.6281	0.0224
3	0.5478	0.1721	0.0678	0.0058	0.0224	0.7960
4	0.9999	0.1721	0.0068	0.0058	0.6281	0.7960
5	0.5478	0.9999	0.0068	0.9999	0.6281	0.7960

Slika 3.10: V implementaciji algoritma pripišemo glede na sliko 3.7 vrednosti $Q_{ij}^0 = {p_j}^0$

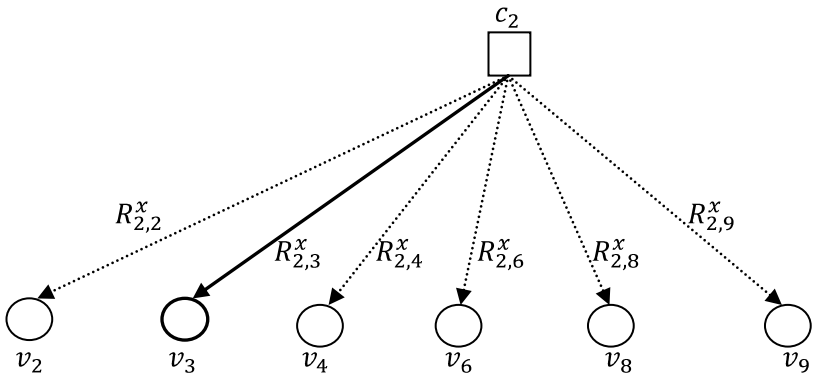
Q_{ij}^1	p1	p2	p3	p4	p5	p6
1	0.4522	0.9322	0.9932	0.0001	0.9942	0.9776
2	0.0001	0.8279	0.9322	0.0001	0.3719	0.9776
3	0.4522	0.8279	0.9322	0.9942	0.9776	0.2040
4	0.0001	0.8279	0.9932	0.9942	0.3719	0.2040
5	0.4522	0.0001	0.9932	0.0001	0.3719	0.2040

Slika 3.11: Podobno pripišemo vrednosti $Q_{ij}^1 = {p_j}^1$

4.4.2 Naslednje iteracije

Primer izračuna vrednosti R_{ij}^x

Na sliki 3.12 je odsek Tannerjevega grafa iz katerega so jasno razvidni vsi starši testnega vozlišča c_2 $(i = 2)$, to so simboli $\{v_2, v_3, v_4, v_6, v_8, v_9\}$, $j \in \{2,3,4,6,8,9\}$, ter kako testno vozlišče c_2 pošlje vsem tem navedenim staršem vrednost R_{ij}^x.



Slika 3.12: Grafični prikaz potovanja sporočil, kako testno vozlišče c_i pošlje svojim staršem vrednost R^x_{ij}

Poglejmo, kako poteka izračun vrednosti $R^x{}_{2,3}$. Najprej iz druge zaporedne vrstice $rowsOnes$, ki je prikazana na sliki 3.3, sestavimo pripadajočo paritetno enačbo: $d_1 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 = 0$. Testno vozlišče c_2 bo torej poslal staršu v_3 informacijo, da je bit v_3 enak: $v_3 = 0$ (pri tem seveda, da je pogoj paritetne enačbe izpolnjen).

Izračunu $R^0{}_{2,3}$ pa poteka takole:

$$\delta R_{2,3} = (Q^0_{2,2} - Q^1_{2,2}) + (Q^0_{2,4} - Q^1_{2,4}) + (Q^0_{2,6} - Q^1_{2,6}) + (Q^0_{2,8} - Q^1_{2,8}) + (Q^0_{2,9} - Q^1_{2,9}),$$

$$\rightarrow R^0{}_{2,3} = \frac{1}{2} \cdot (1 + \delta R_{2,3}) \ , \ R^1{}_{2,3} = 1 - R^0{}_{2,3}.$$

R^0_{ij}	p1	p2	p3	p4	p5	p6
1	0.9024	0.4555	0.4610	0.5385	0.4611	0.4597
2	0.4307	0.6057	0.5802	0.4307	0.2294	0.5726
3	0.6584	0.4769	0.4825	0.4847	0.4841	0.5256
4	0.4515	0.5739	0.5491	0.5490	0.3108	0.4181
5	0.4253	0.4929	0.5072	0.4929	0.4721	0.4879

Slika 3.13: Prikaz izračunov $R^0{}_{i\ j}$

R_{ij}^1	p1	p2	p3	p4	p5	p6
1	0.0976	0.5445	0.5390	0.4615	0.5389	0.5403
2	0.5693	0.3943	0.4198	0.5693	0.7706	0.4274
3	0.3416	0.5231	0.5175	0.5153	0.5159	0.4744
4	0.5485	0.4261	0.4509	0.4510	0.6892	0.5819
5	0.5747	0.5071	0.4928	0.5071	0.5279	0.5121

Slika 3.14: Prikaz izračunov $R^1{}_{i\ j}$

Ocena dekodirane besede $\widehat{d}$

Iz izračuna koeficientov Q_{ij}^x, ki so osnova vrednostim R_{ij}^x, nato sledi ocena verjetnosti bitov dekodirane besede $\hat{d}$. Če nam ocenjeni $\hat{d}$ izpolni enačbo $\hat{d} \cdot H^T = 0$, to pomeni, da je $\hat{d}$ veljavna kodna beseda, ali z drugimi besedami, uspešno smo ugotovili kakšna je izvorna poslana kodna beseda preko AWGN kanala. Posledično pa s tem tudi sporočilo, ki je bilo poslano preko komunikacijskega sistema.

Primer:

Poglejmo si sedaj, kako poteka izračun vrednosti bita d_7. Izpišemo si vse otroke simbola v_7, ki jih lahko enostavno izpišemo iz sedme vrstice tabele $colOnes$, ti pa so: $\{c_7, c_{11}, c_{17}\}$. Formula za izračun je tako:

$$\alpha_7 = \frac{1}{p_7^0 R_{1,7}^0 R_{3,7}^0 R_{4,7}^0 + p_7^1 R_{1,7}^1 R_{3,7}^1 R_{4,7}^1},$$

$$\widehat{d_j}^{\,0} = \alpha_7 \, R_{1,7}^0 \, R_{3,7}^0 \, R_{4,7}^0, \qquad \widehat{d_j}^{\,1} = 1 - \widehat{d_j}^{\,0}.$$

$$\widehat{d_7} = \begin{cases} 1; & \widehat{d_1}^{\,1} \geq \widehat{d_0}^{\,0} \\ 0; & \widehat{d_0}^{\,0} < \widehat{d_1}^{\,0} \end{cases}$$

j	$d0$	$d1$	$\widehat{d_j}$	x
1	0.9411	0.0589	0	0
2	0.9998	0.0002	0	0
3	0.2816	0.7184	1	1
4	0.0727	0.9273	1	1
5	0.0073	0.9927	1	1
6	0.9999	0.0001	0	0
7	0.0057	0.9943	1	1
8	0.1685	0.8315	1	1
9	0.0239	0.9761	1	1
10	0.7474	0.2526	0	0

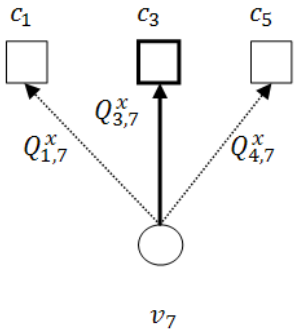
Slika 3.15: Produkt ocenjenega d z paritetno matriko H na sliki 3.3 nam da ničelni vektor, zato je d veljavna kodna beseda

Posodabljanje vrednosti Q_{ij}^x

Če enačba na ocenjeni dekodirni besedi $\hat{d} \cdot H^T = 0$ še ni izpolnjena, nadaljujemo z izračunom posodobljenih vrednosti Q_{ij}^x na spodaj opisan način. Postopek se nadaljuje spet s delom iteracije, ki je opisan v poglavju 4.4.2. Možno je tudi, da po nekem vnaprej določenem številu možnih iteracij še vedno ne dobimo veljavne kodne besede. V tem primeru postopek dekodiranja ustavimo. Beseda bo dekodirana napačno, vendar je napaka opažena.

Primer:

Na sliki 3.16 je odsek Tannerjevega grafa iz katerega so jasno razvidni vsi otroci simbola $v_7 \rightarrow j = 7$, to so testna vozlišča $\{c_1, c_3, c_4\}$, $i = \{1,3,5\}$, ter kako simbol v_7 pošlje vsem tem navedenim otrokom posodobljeno vrednost Q_{ij}^x.



Slika 3.16: Grafični prikaz potovanja sporočil, kako simbol v_7 pošlje svojim otrokom vrednost Q_{ij}^x

$$\alpha_{3,7} = \frac{1}{p_7^0 R_{1,7}^0 R_{4,7}^0 + p_7^1 R_{1,7}^1 R_{4,7}^1},$$

$$Q_{3,7}^0 = \alpha_{3,7}\, p_7^0 R_{1,7}^0 R_{4,7}^0 \quad ,$$

$$Q_{3,7}^1 = \alpha_{3,7}\, p_7^1 R_{1,7}^1 R_{4,7}^1 \quad .$$

4.4.3 Zaključek algoritma

Dekodirni postopek z izmenjevanjem sporočil se zaključi s pošiljanjem pripadajočega dela ocenjene kodne besede prejemniku.

$\widehat{d_J}^T$	1	2	3	4	5	6	7	8	9	10
	0	0	1	1	1	0	1	1	1	0

Slika 3.17: Prejemnik prebere sporočilo iz pripadajočega dela kodne besede $\hat{d}$

5. Simulacija in rezultati testiranja

V tem poglavju si bomo najprej pogledali, kako pridobimo paritetno matriko in generatorsko matriko po postopku PEG, opisanem v razdelku 4.3, pri tem pa bomo uporabljali program dostopen na spletu *[13]*. Nadaljevali bomo z opisom še enega programa, ki je prav tako dostopen na spletu *[14]*, s katerim ocenimo minimalno razdaljo generirane paritetne matrike. Za oba omenjena programa so podani parametri za zagon in njihov pomen.

V nadaljevanju tega poglavja so opisane podrobnosti našega komunikacijskega kanala AWGN, pri katerem moteči šum opišemo kot beli Gaussov šum. Ko prejemnik preko komunikacijskega kanala dobi vektor y, sledi dekodirni iterativni postopek z izmenjevanjem sporočil, opisan v razdelku 4.4. Po končanem procesu (glede na število iteracij), bomo na grafu prikazali relativno število napak, ki ga označimo z BER (angl. Bit Error Rate).

V zadnjem delu pa bomo primerjali odvisnost med dimenzijami generiranih LDPC matrik s postopkom PEG in uspešnostjo dekodiranja pri različnih stopnjah šuma. Ob dobro generiranih LDPC matrikah različnih velikostih bi moral dekodirni postopek pri matrikah večjih dimenzij delovati učinkoviteje, kot pri tistih z manjšimi dimenzijami (pri istem šumu).

5.1 Generiranje paritetne in generatorske matrike s postopkom PEG

Za generiranje paritetne in generatorske matrike za LDPC kodo uporabimo program PEG, opisan v poglavju 4.3. Algoritem je implementiran in prosto dostopen na spletu *[12]* pod imenom PEG. Program poženemo z ukazom:

PEG –numN 10 –numM 5 –codeName LDPC105.txt –degFileName Reg_3.txt –outputMode 2

–numN : število stolpcev N paritetne matrike H

–numM : število vrstic M paritetne matrike H

–codeName : oblika skrajšanega zapisa matrik H in G se shrani v datoteko z imenom codeName,

–degFileName : podamo stopnjo simbolov in s tem razvrstitev enic po posameznih stolpcih paritetne matrike H v datoteki degFileName,

–*outputMode* : opcija 2 pomeni, da vsebuje izpis tako paritetno kot pripadajočo generatorsko matriko .

Stopnje simbolov podamo v datoteki, kot je prikazano na spodnjem primeru:

Reg_3.txt

1 *[število različnih stopenj]* 3 *[w_c - vektor stopenj v naraščujočem vrstnem redu, v tem primeru imamo semo en element]* 1.0 *[vektor razporeditve uteži, v našem primeru imamo regularno, to je vseskozi enako]*

Primer izhodne datoteke za stopnje, podane v datoteki Reg_3.txt je prikazan spodaj:

10	*[število stolpcev n]*
5	*[dolžina sporočila k]*
5	*[število vrstic m]*
3	*[število vrstic skrajšanega zapisa generatorske matrike G]*
6	*[stopnja testnih vozlišč w_r. oz. širina skrajšane paritetne matrike H]*
[skrajšan transponiran zapis enic v posamezni vrstici G]	
5 1 1 1 2 1 2 3 4 5	
0 2 2 4 0 0 0 0 0 0	
0 3 5 5 0 0 0 0 0 0	
[skrajšan zapis pozicij enic v posamezni vrstici paritetne matrike H]	
1 4 5 6 7 9	
2 3 4 6 8 9	
1 3 4 7 9 10	
2 3 5 7 8 10	
1 2 5 6 8 10	

5.1 Preverjanje minimalne razdalje

Za preverjanje minimalne razdalje uporabimo program dostopen na spletu *[14]*. Uporabljeni algoritem je hevrističen in poskuša poiskati kodne besede z majhno težo, s tem da dekodira besede, ki so bile dobljene iz ničelne kodne besede, ki ji je bil prištet majhen šum. Beseda z najmanjšo težo je zgornja meja za minimalno razdaljo koda. Dlje ko algoritem teče, boljše bo ocena. Zaženemo ga z naslednjim ukazom:

minDist –code LDPC256128.txt –outFileCW res256128.txt

–*code:* datoteka, kjer je shranjen zapis paritetne matrike H v skrajšani obliki,

–*outFileCW:* izhodna datoteka, kjer se shranjuje izpis besed z majhnimi težami.

Primer izhodne datoteke po izčrpnem iskanju:

```
[minimalna |  [število besed na
   razdalja] |            tej razdalji  ]
Weight=1    num of codewords=0
Weight=2    num of codewords=0
Weight=3    num of codewords=0
Weight=4    num of codewords=0
Weight=5    num of codewords=0
Weight=6    num of codewords=0
Weight=7    num of codewords=0
Weight=8    num of codewords=0
Weight=9    num of codewords=0
Weight=10    num of codewords=0
Weight=11    num of codewords=0
Weight=12    num of codewords=1
         [najdena beseda]
13 44 59 88 122 142 157 183 200 202 227 237
Weight=13    num of codewords=0
Weight=14    num of codewords=1
1 26 74 79 105 108 124 130 143 164 182 220 224 240
Weight=15    num of codewords=0
Weight=16    num of codewords=15
Weight=17    num of codewords=0
Weight=18    num of codewords=28
    [izpis 28 besed na razdalji 18]
….
Weight=19    num of codewords=0
Weight=20    num of codewords=66
…
Weight=21    num of codewords=0
Weight=22    num of codewords=162
…
Weight=23    num of codewords=0
Weight=24    num of codewords=398
…
Weight=25    num of codewords=0
Weight=26    num of codewords=902
….
Weight=27    num of codewords=0
Weight=28    num of codewords=2082
…
Weight=29    num of codewords=0
```

5.2 Simuliranje AWGN kanala

Ko iz sporočila pridobimo kodno besedo, jo najprej pretvorimo v format za prenos po kanalu z BPSK modulacijo. Še enkrat se spomnimo, da se pri tem 1 preslika v 1, 0 pa v -1. Tako modulirano kodno besedo bomo imenovali *izvoren signal*.

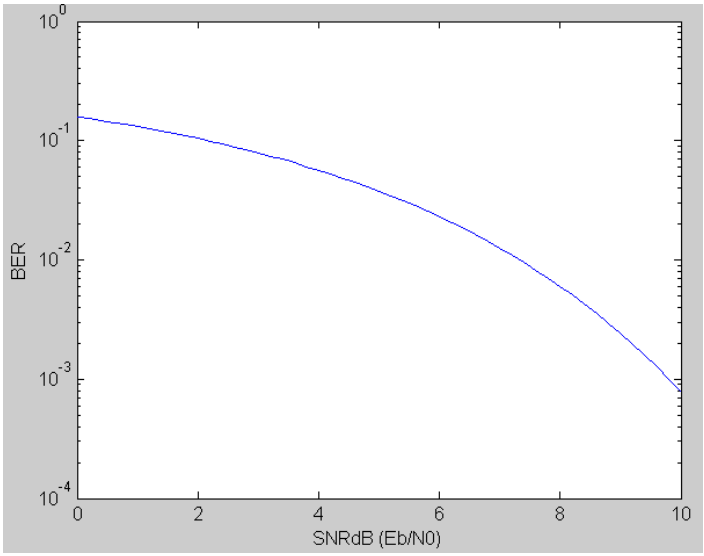
Vemo, da je verjetnost napačnega dekodiranja odvisna od standardne deviacije šuma. Le-tega bomo podajali preko običajnega označevanja razmerja moči signala proti moči šuma, kar bomo označevali kot SNR (angl., Signal-to-Noise ratio). Formula za SNR pravi: $SNR = \frac{E_b}{N_0}$, pri tem je E_b *energija signala* ter N_0 enostranska *močnostna gostota*. To je pomembno zato, ker je značilni beli Gaussov šum za kanal AWGN določen z Gaussovo porazdelitvijo $N(0, \sigma^2)$, pri čemer je standardna deviacija σ določena kot: $\sigma^2 = \frac{N_0}{2}$. Običajno se podaja mero SNR v decibelih, zato za nadaljnji postopek *normalizirani SNR* pridobimo kot: $SNR = 10^{SNRdB/10}$.

S preurejanjem formule za $SNRdb = \frac{\sigma^2{}_{Signal}}{\sigma^2{}_{\check{S}um}}$ dobimo popravljeni *umerjen signal* na tak način: $umerjen\ signal = \frac{\sigma_{\check{S}um}}{\sigma_{Signal}} \cdot \sqrt{SNR} \cdot izvorni\ signal$. Postopek je odlično opisan na spletu in sicer na tem naslovu *[9,10,11]*.

Vrednost besede y, ki jo dobi prejemnik pri upoštevanju teh posebnih lastnosti, ki veljajo za kanal AWGN je torej:

$$y = \ umerjen\ \ signal + \sqrt{\frac{N_0}{2}} \cdot \ naključno\ porazdeljen\ normalni\ šum\ .$$

Prejemnik bo nato na prejeti besedi y, s pomočjo dekodirnega algoritma z izmenjavanjem sporočil opisanega v poglavju 4.4, poskušal ugotoviti poslano kodno besedo. Definirati moramo še BER (angl. Bit Error Rate), kot razmerje števila mest v katerem se prejeta beseda razlikuje od poslane v primerjavi z njegovo dolžino. Učinkovitost dekodirnega postopka bomo namreč prikazovali na dvodimenzionalnemu grafu, kjer bo na horizontalni osi mera $SNRdb$ ter na vertikalni osi mera BER. Teoretično mejo BER pri danem šumu brez kodiranja nam podaja že omenjena $Q-$funkcija (poglavje 2.1). Funkcijo Q pa nato lahko izrazimo s komplementarno funkcijo napake: $Q(SNRdB) = \frac{1}{2} \cdot erfc(\frac{\sqrt{SNR}}{\sqrt{2}})$.



Slika 4.1: Funkcija Q nam pove teoretično mejo BER pri prenosu brez kodiranja

5.3 Učinkovitost dekodirnega postopka

Zanimala nas bo učinkovitost dekodirnega postopka SPA glede na različne stopnje šuma. Z algoritmom PEG, ki je dostopen na spletu *[13]*, sem generiral nekaj matrik različnih dimenzij. Kratki cikli v Tannerjevem grafu niso zaželeni, saj zmanjšujejo učinkovitost dekodirnega postopka. Na sliki 4.2 so podani rezultati iskanja najkrajšega cikla Tannerjevega grafa algoritma PEG za testirane matrike različnih dimenzij.

Matrika dimenzije	Najkrajši cikel
256×128	vsi simboli pripadajo ciklom dolžine 8
512×256	506 simbolov ima cikel 8, 6 simbolov cikel 10
1024×512	vsi simboli pripadajo ciklom dolžine 10

Slika 4.2: Rezultati iskanja najkrajših ciklov posameznih LDPC matrik

Ker je moč koda glede popravljanja napak odvisna predvsem od razdalj med kodnimi besedami, smo izbrali kode s čim večjimi minimalnimi razdaljami. Minimalno razdaljo smo preverjali z algoritmom, dostopnim na spletu *[14]*. Sicer je problem iskanja minimalne razdalje zelo težaven, med tem, ko se da najkrajši cikel poiskati hitro. Kot pa smo že spoznali, velja, da dajejo kodi z večjo minimalno razdaljo boljše rezultate dekodiranja. V nadaljevanju so podani rezultati merjenj minimalnih razdalj generiranih LDPC kod s slike 4.2. V prvih vrsticah so teže, v drugi pa število najdenih kodnih besed z dano težo. Za podrobnosti delovanja algoritma pa naj si bralec pogleda na *[14]*. Omeniti je še potrebno, da algoritem deluje zelo počasi, zato smo pri manjših dimenzijah lahko preizkusili več kodnih besed kot pri matrikah večjih dimenzij.

d0	14	16	18	20	22	24	26	28
n	3	21	109	349	1270	3432	9216	21658

Slika 4.3: Rezultati merjenj minimalnih razdalj za LDPC matriko velikosti 256 x 128

d0	22	26	28	30	32	34	36	38	40	42	44	46	48
n	1	1	2	6	10	25	39	73	71	289	587	1114	1976

Slika 4.4: Rezultati merjenj minimalnih razdalj za LDPC matriko velikosti 512 x 256

d0	62	66	70	72	74	76	78	80	82	84	86	88	90	92	94	96	98
n	1	2	2	5	2	7	14	25	42	59	96	154	223	386	565	832	1307

Slika 4.5: Rezultati merjenj minimalnih razdalj za LDPC matriko velikosti 1024 x 512

Učinkovitost dekodiranja

Za posamezno matriko smo testirali dekodirni postopek na 20 sporočilih doline 512 bitov (vsakemu sporočilu smo priredili od 1 do 4 kodne besede glede na dimenzijo koda). Za vsak kodno besedo smo izvedli do 20 iteracij dekodirnega algoritma. Rezultati teh merjenj so zbrani v pripadajočih tabelah, pri čemer $AvgCherr$ pomeni, kakšna je napaka pri prenosu skozi kanal pri danem SNR (v decibelih), $AvgBer$ pomeni, kakšna je napaka na besedi po 20 iteracijah, $AvgIter$, pa nam daje število iteracij potrebnih za uspešno dekodiranje. V nadaljevanju so podani rezultati teh testiranj, pri tem so jasno označena mesta, kjer pride do izboljšav pri posameznih velikostih matrik:

SNRdb 256x128	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
Avg CHerr	40.850	37.358	33.550	30.300	27.192	23.942	20.383	16.442	14.608	12.025	9.275
Avg BER	0.134	0.102	0.072	0.035	0.013	0.0028	0.0009	0	0	0	0
Avg iter	19.858	19.008	17.125	13.567	9.7583	6.825	4.633	2.942	2.533	2.083	1.783

Slika 4.6: Rezultati dekodirnega postopka SPA za LDPC matriko velikosti 256 x 128

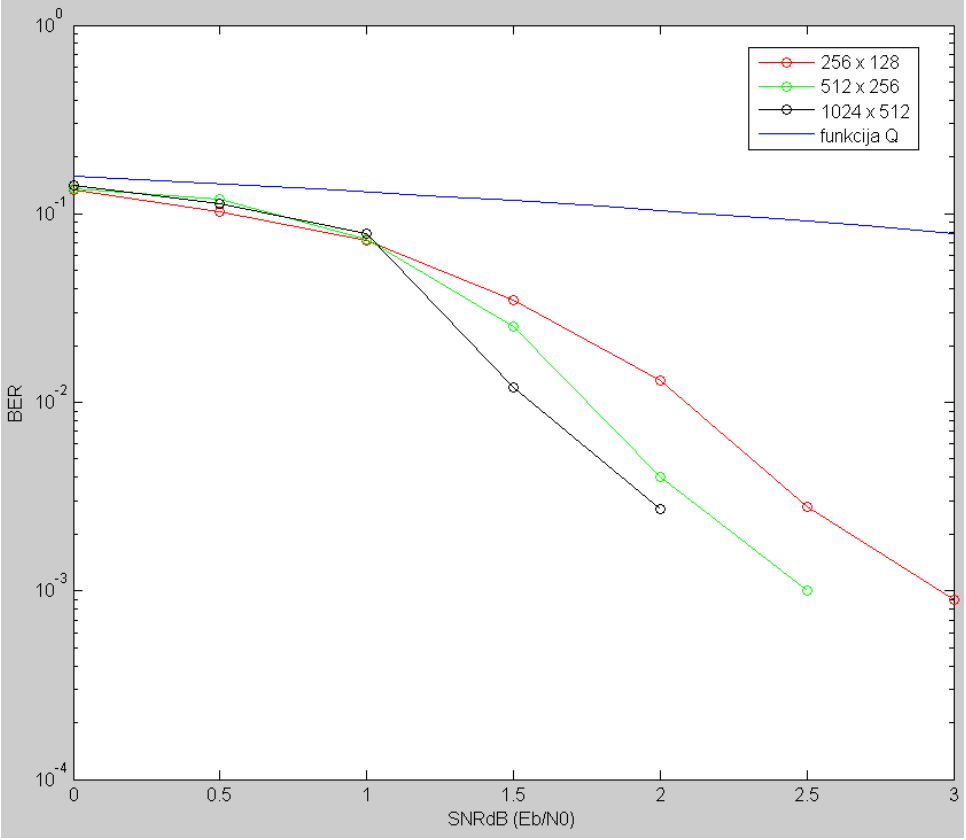
SNRdb 512x256	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
Avg CHerr	83.40	75.317	67.02	58.217	52.933	46.550	39.550	35.033	29.883	23.933	19.483
Avg BER	0.136	0.12	0.073	**0.0254**	**0.004**	**0.001**	**0**	0	0	0	0
Avg iter	20	20	19.167	14.167	9.433	6.417	4.117	3.683	3.033	2.600	2.083

Slika 4.7: Rezultati dekodirnega postopka SPA za LDPC matriko velikosti 512x256

SNRdb 1024x512	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
Avg CHerr	165.900	148.033	132.833	117.100	107.500	95.300	79.467	66.400	56.900	45.933	9.033
Avg BER	0.141	0.113	0.079	**0.012**	**0.0027**	**0**	0	0	0	0	0
Avg iter	20	20	19.467	14.367	10.733	6.833	5.033	3.733	3.133	2.733	2.300

Slika 4.8: Rezultati dekodirnega postopka SPA za LDPC matriko velikosti 1024 x 512

Na sliki 4.9 je prikazana napaka BER za posamezne velikosti matrik. Povedati moramo, da pri matriki večjih velikosti pri zmernem šumu hitreje ugotovi kodno besedo kot pri matrikah manjše velikosti, zato je pač tam BER enak 0. Na grafu je tudi lepo vidna primerjava med posameznimi matrikami in funkcijo Q, ki podaja mejo, če podatki niso kodirani.



Slika 4.9: Primerjava merjenja BER med različnimi velikostmi LDPC matrik

Kot smo že omenil, je testiranje minimalne razdalje zelo zamudno, sicer bi morda lahko našli boljše matrike pri enakih dimenzijah oziroma bi lahko preizkusili učinkovitost dekodiranja na več raznovrstnih matrikah. Bralec si lahko v ta namen pogleda lepo število generiranih paritetnih matrik na spletu *[15]*.

Opazili pa smo, da se morda ob kljub ne najboljšim generiranim paritetnim matrikam, učinkovitost dekodiranja izboljšuje. V razdelku 4.4 pa smo spoznali, da se časovna zahtevnost glede na dimenzije matrik povečuje linearno, tako da se zahtevnost ob uporabi večjih matrik ne povečuje tako drastično, da bi postal dekodirni proces prezahteven oziroma neuporaben.

6. Sklep

Torej, po vsem tem, smo spoznali komponente komunikacijskega sistema in kateri so njegovi kritični elementi. Spoznali smo, kako s pomočjo kod za popravljanje napak kodiramo podatke in kako z izračunom sindroma preverimo, ali je prejeta beseda kodna beseda oziroma ali je prišlo do napake. Z vpeljavo koncepta Hammingove krogle smo lahko tudi jasno določili, kakšna je lahko še največja napaka, ki se zgodi prenosom zaradi nezaželenih vplivov, da lahko prejemnik pravilno določi poslano sporočilo.

Nato smo spoznali eno od realizacij tega sistema, pri čemer smo nezaželene vplive med komuniciranjem simulirali z belim Gaussovim šumom. Glavna prednost tega načina v primerjavi z osnovnim dvojiškim simetričnim je, da nam lahko pove, kako blizu je sprejeta vrednost glede na oddano. To smo potem v dekodirnem postopku z izmenjavanjem sporočil s pridom izkoriščali.

Večji del te naloge pa smo se ukvarjali z redko posejanimi LDPC kodami, ki smo jih potem, predvsem z namenom pri uporabi dekodirnega postopka, predstavili na Tannerjevem grafom. S prosto dostopnim programom, ki uporablja algoritem PEG, smo generirali nekaj matrik s kar največjim najkrajšim ciklom Tannerjevega grafa. Nato pa smo z drugim prosto dostopnim programom preverjali njihove minimalne razdalje, saj smo želeli pridobiti kode s čim večjo minimalno razdaljo.

Na primeru manjše matrike smo s pomočjo Tannerjevega grafa, na katerem smo lahko spremljali potovanje sporočil, demonstrirali delovanje dekodirnega algoritma z izmenjevanjem sporočil. Z izračunom sindroma na paritetnih enačbah in vrednostjo, ki pove kako blizu je sprejeta vrednost glede na oddano, smo nazorno prikazali, kako dekodirnik ugotovi poslano kodno besedo in jo nato pošlje prejemniku.

Nato smo se ukvarjali s testiranjem, kjer smo s pomočjo algoritma PEG generirali nekaj matrik in preračunali njihove minimalne razdalje. Zanimalo nas je, kakšna je pri različnih stopnjah šuma resnična učinkovitost dekodirnega postopka. Merjenje napak s pomočjo dekodirnega algoritma je dejansko pokazal, da se kode z večanjem minimalne razdalje izboljšujejo. Moramo pa omeniti, da rezultati ne pridejo do izraza ob ekstremno velikem šumu ali pa v primeru praktično neizgubnega kanala. Potrebno je upoštevati, da je v praksi daleč najbolj pogosta srednja vrednost šuma in tam so se nam naše predpostavke zelo dobro potrdile.

Verjetno bi se dalo še precej razpravljati, kje so tiste vrzeli, ki bi jih lahko še bolje realizirali. Rezultati so pokazali, da dajejo trenutno LDPC kode ob sedanji tehnologiji resnično dobre rezultate. Tukaj bi lahko omenil, da bi lahko preizkusili še veliko več razmerij kodov in razporeditvi tež. Bralec si naj v ta namen pogleda enciklopedijo LDPC kod na *[15]*. O realizaciji šuma z belim Gaussovim šumom, katero smo v tej nalogi uporabili, bi sicer tudi lahko diskutirali. Ta kanal se izkaže kot dober približek in je

dovolj enostaven za analizo. Poznani pa so še drugi modeli komunikacijskega kanala, ki se lahko izkažejo za boljše v določenih situacijah.

Kanal, ki smo ga opisali v tej nalogi je danes eden najbolj realiziranih, LDPC kode pa zelo pogosto uporabljene. Nazorno pa smo predstavili enega od način, kako čim bolj zanesljivo prenesti podatke prejemnika k oddajniku. In hitrost ter zanesljivost informacij je tisto, k čemur danes vsi težimo.

DODATEK A : Izvorna koda napisana v programu Matlab

Branje paritetne ter pripadajoče generatorske matrike iz datoteke, ustvarjene z algoritmom PEG

Vhod:

- `fileNM` ; datoteka, generirana z algoritmom PEG,
- `RegCWD` ; datoteka razporeditev uteži.

Izhod:

- `rowsOnes` , `colOnes`; skrajšan zapis paritetne matrike H,
- `matrixG` ; skrajšan zapis njene pripadajoče generatorske matrike G.

Izvorna koda:

```
function [rowsOnes,colOnes,matrixG] = readFile(fileNM,RegCWD)
%branje podatkov datoteke kreirana z  PEG algoritmom

fidNM=fopen(fileNM,'r');
tableOfElements=fscanf(fidNM,'%d');
%dolžina Y skrajšane paritetne matrike H
rowsH=tableOfElements(3);
%širina X paritetne matrike H
colsH=tableOfElements(1);
%dolžina sporočila
varK=tableOfElements(2);
%dolžina Y skrajšane matrike G
rowsG=tableOfElements(4);
%stopnja vsakega testnega vozlišča oz. širina skrajšane paritetne
matrike H
rowWeightH=tableOfElements(5);

fidWD=fopen(RegCWD,'r');
tableWD=fscanf(fidWD,'%d');
%iz zunanje datoteke, stopnja vsakega simbola paritetne matrike H
colWeight=tableWD(2);

%iz datoteke preberemo skrajšano matriko G
for i=1:rowsG
       for j=1:colsH
       tableOfG(i,j)=tableOfElements(6+colsH*(i-1)+(j-1));
       end
end

%potrebujemo samo prvi (levi) del tabele G, na desni strani je
identiteta
 matrixG=tableOfG(1:rowsG,1:varK);


%branje pozicij enic po posameznih vrsticah paritetne matrike H
for i=1:rowsH
```

```
        for j=1:rowWeightH
        rowsOnes(i,j)=tableOfElements(6+colsH*rowsG+(i-1)*rowWeightH+(j-
1));
        end
end

%branje pozicij enic po posameznih stolpcih
%ker ne vem koliko je x, najprej vpišem j
%na prvo prosto mesto vrstice colOnes(temp,x)
%če pa je na prvem mestu vrstice že vpisan element različen od 0
%celo vrstico rotiramo v desno in zapišemo element na prvo mesto

colOnes=zeros(colsH,colWeight);

for i=1:rowsH
    for j=1:rowWeightH
        if rowsOnes(i,j)~=0
            colOnes(rowsOnes(i,j),1)=i;

colOnes(rowsOnes(i,j),:)=circshift(colOnes(rowsOnes(i,j),:),[1 1]);
        end
    end
end
```

Po kodirnem postopku dobimo kodno besedo

Vhod:

- `message`; sporočilo,
- `matrixG`; skrajšan zapis generatorske matrike G.

Izhod:

- `codeword`; kodna beseda, katero dobimo po kodirnem postopku.

Izvorna koda:

```
function codeword = returnCodeword(matrixG,message)

%sestavljanje kodne besede
%messageOnes nam pove mesta enic v sporočilu message
messageOnes=find(message);

bit=zeros(1,2*length(message));

%križanja mesta enic v sporočilu in generatorski matriki
for i=1:length(matrixG(1,:))
    for j=1:length(matrixG(:,1))
        if matrixG(j,i)~=0 && indexOfEl(messageOnes,1,matrixG(j,i))~=0
            bit(i)=bit(i)+1;
        end
    end
end
bit = mod(bit,2);
```

```
for i=1:length(messageOnes)
    bit(length(message)+messageOnes(i))=1;
end

codeword=bit;
```

Modulacija signala, BPSK

Vhod:

- `bitseq`; kodna beseda,
- `amplitude`; amplituda BPSK modulacije .

Izhod:

- `signal`; modulirana kodna beseda pri neki amplitudi.

Izvorna koda:

```
function [signal]=bpsk(bitseq,amplitude)
for i=1:length(bitseq)
   if bitseq(i)==1
      signal (i)=amplitude;
   else
      signal (i)=-amplitude;
   end
end
```

AWGN kanal z 'belim Gaussovim šumom'

Vhod:

- `signal`; modulirana kodna beseda ,
- `SNRdb`; razmerje moči signali proti moči šuma.

Izhod:

- `signalWithNoise`; prenesena beseda preko komunikacijskega kanala,
- `deviation`; deviacija te prenesene besede.

Izvorna koda:

```
function [signalWithNoise,deviation]= genSignalForSNR(signal,SNRdb)

%deviation=sqrt(1/N * sum(Xi-mean(X))^2)= 'ro'
%variance = deviation^2
%=>SNR=10^(SNRdb/10)  <=SNRdB=10log10(SNR);
%SNR = var(signal)/var(noise)
%ro(signal)^2 = ro(noise)^2 * SNR
% 'scaledSignal'(t) = ro(noise)*sqrt(SNR)*s(t) / ro(signal)
```

```
% Generate Additive White Gaussian Noise with zero mean and unit
variance
% SNRdb=Eb/N0 =>SNR=10^(SNRdb/10)
% EB is signal power
% signal_energy=(signal*signal')/length(signal);
% N0=signal_energy/(10^(SNRdb/10));
% awgnNoise = sqrt(N0/2)*randn(1,length(signal));
% ==> By definition SNR is the ratio of signal power to noise power
% signalPower = (norm(scaledSignal)^2)/length(scaledSignal);
% noisePower = (norm(noise)^2)/length(noise);

signal_energy=(signal*signal')/length(signal);
N0=signal_energy/(10^(SNRdb/10));
awgnNoise = sqrt(N0/2)*randn(1,length(signal));

%deviation=std, variance=var
scaledSignal = std(awgnNoise)*(sqrt(10^(SNRdb/10)))*signal/std(signal);

%==> Alternative way of calculating Signal and noise power from their
variance
signalPower = var(scaledSignal);
noisePower = var(awgnNoise);
deviation = std(awgnNoise);

%Calculate Signal to noise ratio for the scaledSignal and generated
Noise
SNRratio = signalPower/noisePower;
%measuredSNR is in dB
measuredSNR=10*log10(SNRratio);

%Add the scaled signal with the generated noise
signalWithNoise = scaledSignal + awgnNoise;

%plotting commands
subplot(3,1,1);
plot(scaledSignal);
title('Input Signal');

subplot(3,1,2);
plot(awgnNoise);
title('Generated Noise');

% subplot(3,1,3);
% plot(signalWithNoise);
% title(['Signal + Noise for SNR= ',num2str(measuredSNR),' dB']);
```

Izračun verjetnosti posameznih bitov poslane besede preko kanala

Vhod:

- `signalWithNoise`; prenesena beseda preko komunikacijskega kanala,
- `deviation`; deviacija prenesene besede.

Izhod:

- `p0,p1;` ocena verjetnosti, da ima posamezen bit prenesene besede vrednost 0 ali 1.

Izvorna koda:

```
function [p0,p1] = ChannelInf(signalWithNoise,deviation)

%pretvorba bitov kodne besed v format polarnih koordinat
%bit 1 se pretvori v bit +1
%bit 0 se pretvori v bit -1

%poglejmo si kakšen je izvorni preneseni vektor, brez dekodiranja
transferedWord=zeros(1,length(rx_waveform));

%implementacija SPA, začetne vrednosti kanala
%Qzero(i,j)=p0(j); Qone,j)=p1(j);

for i=1:length(rx_ signalWithNoise)
p1(i) = 1 /(1 + exp(-2* signalWithNoise(i)/(deviation^2)));
p0(i) = 1 - p1(i);

end
```

Dekodirni algoritem SPA

Vhod:

- `codeword_X`; kodna beseda za namene testiranja,
- `deviation`; deviacija prenesene besede,
- `rowsOnes` , `colOnes`; skrajšan zapis paritetne matrike *H*,
- `signalWithNoise`; prenesena beseda preko komunikacijskega kanala,
- `loops`; število iteracij dekodirnega postopka SPA.

Izhod:

- `BER;` število napak ocenjene dekodirane besed po končanem številu iteracij v primerjavi z kodno besedo.

Izvorna koda:

```
function [BER] =
decodeWithSPA(codeword_X,rowsOnes,colOnes,signalWithNoise,deviation,loop
s)

%v implementaciji dekodirnega algoritma izračunamo začetne verjetnosti
%posameznih bitov glede na prejeti pokvarjeni y
[p0,p1] = ChannelInf(signalWithNoise,deviation);

%p0 in p1 pripišemo posamezni Qij0 in Qij1, glede na pravilo
%Qij0=p(j)0 in Qij1=p(j)1

    for i=1:length(rowsOnes(:,1))
        for j=1:length(rowsOnes(1,:))
            QijZero(i,j)=p0(rowsOnes(i,j));
```

```
        end
    end
    QijOne=1-QijZero;

BER=zeros(1,loops);

%začetek iterativnega algoritma
for iter=1:loops

    %preračun Rij
    RijZero = tableRij(QijZero,QijOne,rowsOnes);
    RijOne = 1-RijZero;

    %ocena  posameznih bitov dekodirane besede
    [indexD,valueD0,valueD1] =
calculateBitDiff(p0,p1,rowsOnes,colOnes,RijZero,RijOne);
    %dejanska napaka (testiranje)
    BER(iter)=sum(xor(codeword_X,indexD))/length(codeword_X);
    %če dekodirana beseda D zadostuje pogoju d x transp(H) = 0
    %potem je d veljavna kodna beseda

    if testingCodeWord(rowsOnes,indexD)==0
%         numOfIteration=iter;
        break;   %BER(iter)=0;
    end

    %preračun osveženih Qij
    for row=1:length(rowsOnes(:,1))
        for col=1:length(rowsOnes(1,:))
            [QijZero(row,col),QijOne(row,col)]=
QijIter(p0,p1,rowsOnes,colOnes,RijZero,RijOne,row,rowsOnes(row,col));
        end
    end

end
```

Postopek preverjanja, ali je ocenjena dekodirna beseda veljavna kodna beseda

Vhod:

- `codeword_X`; kodna beseda za namene testiranja,
- `rowsOnes`; skrajšan zapis paritetne matrike H.

Izhod:

- `ifZeroThenCW;` če je rezultat preverjanja ničelni vektor, tedaj smo našli veljavno kodno besedo.

Izvorna koda:

```
%preverjanje izpolnjenosti paritetnih enačb oz,
%H * preračune vrednosti bitov vektorja d = 0
function ifZeroThenCW = testingCodeWord(rowsOnes,codeWord)

%messageOnes nam pove pozicije enic v sporočilu
```

```
% testingCW=zeros(1,length(rowsOnes));
testingCW=0; ifZeroThenCW=0;
messageOnes=find(codeWord);

for i = 1:length(rowsOnes(:,1))
    for j=1:length(messageOnes)
         if(indexOfEl(rowsOnes,i,messageOnes(j)) ~= 0)
         %testingCW(i)=testingCW(i)+1;
           testingCW=testingCW+1;
         end
    end

    if mod(testingCW,2)~=0
        ifZeroThenCW=1;
        break;
    end
    testingCW=0;
end
```

Postopek izračuna vrednosti $R^0{}_{ij}$ ter $R^1{}_{ij}$

Vhod:

- `QijZero,QijOne`; začetne vrednosti v prvi iteraciji, v naslednjih iteracijah pa osvežene preračunane vrednosti ,
- `rowsOnes`; skrajšan zapis paritetne matrike H.

Izhod:

- `RijZero`; posodobljenje vrednosti $R^0{}_{ij}$, pri tem velja $R^1{}_{ij} = 1 - R^0{}_{ij}$.

Izvorna koda:

```
% function [RijZero,RijOne] = tableOfRij(f0,f1,rowsOnes)
%preračunavanje vseh Rij-ev za naslednjo iteracijo
%RijZero=1/2 * (1 + productRijPerCombIter(i,j))
%j ne sme biti enak i-ju

function RijZero = tableRij(QijZero,QijOne,rowsOnes)

for i=1:length(rowsOnes(:,1))
       for j=1:length(rowsOnes(1,:))
       RijZero(i,j) =
0.5*(1+productRijPerCombIter(QijZero,QijOne,rowsOnes,i,indexR(rowsOnes,i
,indexOfEl(rowsOnes,i,rowsOnes(i,j)))));
       end
end
```

Pomožni postopek za izračun produktov vrednosti $R^0{}_{ij}$ ter $R^1{}_{ij}$

Vhod:

- `QijZero,QijOne`; začetne vrednosti v prvi iteraciji, v naslednjih iteracijah pa osvežene preračunane vrednosti ,
- `rowsOnes`; skrajšan zapis paritetne matrike H,
- `row`; indeks (zaporedne vrstice) paritetne enačbe
- `indexList`; vsi potrebni indeksi tabele Q^0_{ij} in Q^1_{ij}

Izhod:

- `sum`; zmnožek po indeksi tabele Q^0_{ij} in Q^1_{ij}

Izvorna koda:

```
%productRijPerCombIter je enak produktu QijZero - QijOne

function sum =
productRijPerCombIter(QijZero,QijOne,rowsOnes,row,indexList)
res=1;

for i=1:length(indexList)
    iy=indexOfEl(rowsOnes,row,indexList(i));

    res = res * (QijZero(row,iy)-QijOne(row,iy));

end
sum=res;
```

Postopek izračuna vrednosti $Q^0{}_{ij}$ ter $Q^1{}_{ij}$

Vhod:

- `codeword_X`; kodna beseda za namene testiranja,
- `rowsOnes,colOnes`; skrajšan zapis paritetne matrike H,
- `p0,p1`; ocena verjetnosti, da ima posamezen bit prenesene besede vrednost 0 ali 1,
- `RijZero,RijOne`; tabele $R^0{}_{ij}$ ter $R^1{}_{ij}$,
- `ri,rj`; indeksi (`ri,rj`) za branje elementov tabel $R^0{}_{ij}$ ter $R^1{}_{ij}$.

Izhod:

- `QijZero,QijOne`; posodobljenje vrednosti $Q^0{}_{ij}$ ter $Q^1{}_{ij}$, pri tem velja $Q^1{}_{ij} = 1 - Q^0{}_{ij}$.

Izvorna koda:

```
%QijOne=normalizacijska konstanta alfa * pj(1) * produkt  RmnOne ;
%m ne sme bit enak i-ju

%npr da so v 8. vrsti rowsOnes elementi:   6     8    15    22    26
33
%za Q0(8,26) pridobimo [indexTestR(colOnes,26,8)], torej vse elemente
%vrstice 26 colOnes razen elementa 8. To so npr (3,14).
%mulzero=alfa*f0(26)*R0(3,[pozicija v 3 vrstici rowsOnes kjer se nahaja
26])
%mulone=alfa*f1(26)*R1(3,[pozicija v 3 vrstici rowsOnes kjer se nahaja
26])
%Qzero(8,26)=mulzero/(mulzero+mulone)
Qone(8,26)=mulone/(mulzero+mulone);
%poudariti je potrebno, da so vrednosti RijZero,RijOne iz prejšnje
iteracije

function [QijZero,QijOne] =
QijIter(p0,p1,rowsOnes,colOnes,RijZero,RijOne,ri,rj)
```

```
mulQzero=1; mulQone=1;
temp=indexTestR(colOnes,rj,ri);

for i=1:length(temp)


mulQzero=mulQzero*RijZero(temp(i),indexOfEl(rowsOnes,temp(i),rj));
       mulQone=mulQone*RijOne(temp(i),indexOfEl(rowsOnes,temp(i),rj));
end

Qzero=p0(rj)*mulQzero; Qone=p1(rj)*mulQone;


QijZero=Qzero/(Qzero + Qone);
QijOne=Qone/(Qzero + Qone);
```

Postopek za prikaz vseh sosednih simbolov nekega elementa v paritetni enačbi

Vhod:

- `rowsOnes,colOnes`; skrajšan zapis paritetne matrike H,
- `rowIndex;` index vrstice paritetne matrike H,
- `withoutThisEl;` prikaže vse sosede razen tega elementa.

Izhod:

- `tableOfIndex`; vsi indeksi sosedov v neki paritetni enačbi razen elementa `withoutThisEl`.

Izvorna koda:

```
%če imamo v vrstici 2 matrike rowsOnes elemente
%2     3    14    18    27    31
%vrne indexR(rowsOnes,2,indexOfEl(rowsOnes,2,18))
%indexOfEl nam pove zaporedno pozicijo elementa 18 v vrstici 2
%elemente 2     3    14    27    31

function tableOfIndex = indexR(rowsOnes,rowIndex,withoutThisEl)

for i=1:length(rowsOnes(1,:))
       if(i~=withoutThisEl)
       tableOfIndex(i)=rowsOnes(rowIndex,i);
       end
end

tableOfIndex(tableOfIndex==0)=[];
```

Postopek za prikaz vseh sosednih testnih vozlišč nekega elementa v paritetni enačbi

Vhod:

- `colOnes`; skrajšan zapis paritetne matrike H,
- `rowIndex;` index vrstice paritetne matrike H,
- `notThisEl;` prikaže vse sosede razen tega elementa.

Izhod:

- `tableOfIndex`; vsi indeksi sosedov v neki paritetni enačbi razen elementa `withoutThisEl`.

Izvorna koda:

```
function tableOfIndex = indexTestR(colOnes,rowindex,notThisEl)
%potrebno za izračun novih Q(i,j)
%za npr Q(17,2) velja da je v 17 vrstici , stolpec 2 (enka)
%pogledamo v ColOnes vrstica 2 in izpisemo vse elemente razen 17
for i=1:length(colOnes(1,:))
       if(colOnes(rowindex,i)~=notThisEl)
       tableOfIndex(i)=colOnes(rowindex,i);
       end
end

tableOfIndex(tableOfIndex==0)=[];
```

Postopek za prikaz vseh sosednih testnih vozlišč nekega elementa v paritetni enačbi

Vhod:

- `rowsOnes`; skrajšan zapis paritetne matrike H,
- `ri,rj;` indeksi (`ri,rj`) za branje elementov tabele `rowsOnes`.

Izhod:

- `id`; zaporedni indeks elementa (`ri,rj`) v tabeli `rowsOnes`.

Izvorna koda:

```
%vrne zaporedno stevilko elementa rj v vrstici ri

function id = indexOfEl(rowsOnes,ri,rj)
% za vrstico 5 matrike rowsOnes=[ 5     9    12    21    29    34 ]
%vrne indexOfEl(rowsOnes,5,21), da je id=4

idy=0;
for i=1:length(rowsOnes)
      if(i==ri)
         for(j=1:length(rowsOnes(1,:)))
         if(rowsOnes(i,j)==rj)
         idy=j;
         end
         end
      end
end
id=idy;
```

Postopek za ocenjevanje novih vrednosti bitov dekodirane besede

Vhod:

- `rowsOnes,colOnes`; skrajšan zapis paritetne matrike H,
- `p0,p1;` ocena verjetnosti, da ima posamezen bit prenesene besede vrednost 0 ali 1,
- `RijZero,RijOne;` tabele $R^0{}_{ij}$ ter $R^1{}_{ij}$.

Izhod:

- `indexD`; ocenjena dekodirana beseda $\hat{d}$,
- `valueD0,valueD1;` osvežene vrednosti, da ima posamezen bit prenesene besede vrednost 0 ali 1.

Izvorna koda:

```
%oceno bita dj dobim kot:
% dj1=normalizacijska konstanta alfa*pj(1)*produkt(RijOne)

function [indexD,valueD0,valueD1] =
calculateBitDiff(p0,p1,rowsOnes,colOnes,RijZero,RijOne)

indexD=zeros(1,length(colOnes));
% valueD0=zeros(1,length(colOnes));
% valueD1=zeros(1,length(colOnes));

for i=1:length(colOnes)
        d0temp=1; d1temp=1;

        for j=1: length(colOnes(1,:))
d0temp=d0temp*RijZero(colOnes(i,j),indexOfEl(rowsOnes,colOnes(i,j),i));
d1temp=d1temp*RijOne(colOnes(i,j),indexOfEl(rowsOnes,colOnes(i,j),i));
        end

        prodD0=d0temp*p0(i); prodD1=d1temp*p1(i);

        D0=prodD0/(prodD0+prodD1);
        D1=prodD1/(prodD0+prodD1);

        %primerjamo oceni d1 proti d0
        if(D1>0.5)
            indexD(i)=1;
        else
            indexD(i)=0;
        end



valueD0(i)=D0;
valueD1(i)=D1;
end
```

Testiranje učinkovitosti dekodirnega postopka

Izhod:

- primer ocenjevanje učinkovitosti dekodirnega postopka, pri čemer merimo število napak dekodirane besede v primerjavi z kodno besedo.

Izvorna koda:

```
function testingLDPCcodes()

%merjenje napake BER za SNR v decibelih
SNRdb = [0:0.5:5];
fclose('all');
```

```
loops_SPA=20;
BER=zeros(120,loops_SPA);
Iterations=zeros(120,1);
napakaKanala=zeros(120,1);
%branje matrik iz datotek
[rowsOnes,colOnes,matrixG] = readFile('LDPC256128.txt','Reg_3.txt');

%rezultati BER
for snrdb=0:0.5:5
    '************************************************************'
    snrdb
    fclose('all');
    for i=1:120
        %pripadajoci i-ti del sporocila
        sendMessage=sporocilo(128*(i-1)+1:128*i);
        %kodna beseda
        codeword_X = returnCodeword(matrixG,sendMessage);
        tx_waveform=bpsk(codeword_X,1);
        %prejeta beseda y
        [signalWithNoise,deviation] =
genSignalForSNR(tx_waveform,snrdb);
        %SPA postopek na prejeti besed y
        [BER(i,:),Iterations(i)] =
decodeWithSPA(codeword_X,rowsOnes,colOnes,signalWithNoise,deviation,loop
s_SPA);
        %za testiranjem podatek, kolikšna je napaka kanala
        napakaKanala(i)=sum(xor(codeword_X,signalWithNoise>0));
    end
    %izpis
    napakaKanala
    povprecnaNapakaKanala=sum(napakaKanala)/length(napakaKanala)
    BER(:,20)
    povprecniBER=sum(BER(:,20))/120
    Iterations
    povprecnoIteracij=sum(Iterations)/length(Iterations)
end

%rezultati in prikaz
SNRdb = [0:0.5:3];
BER256128=[0.134 0.102 0.072 0.035  0.013 0.0028 0.0009];
BER512256=[0.136 0.12  0.073 0.0254 0.004 0.001  0];
BER1024512=[0.141 0.113 0.079 0.012  0.0027 0     0];
semilogy(SNRdb,BER256128,'o-r');
hold
semilogy(SNRdb,BER512256,'o-g');
semilogy(SNRdb,BER1024512,'o-k');
SNRdb2 = [0:0.3:3];
%funkcija Q
[theoretical_error_p]=Qfunct(SNRdb2);
semilogy(SNRdb2,theoretical_error_p,'-');
xlabel('SNRdB (Eb/N0)');
ylabel('BER');
```

DODATEK B : Kazalo slik in preglednic

VIRI

[1] G. Kabatiansky, E. Krouk, S. Semenov, *Error Correcting Coding and Security for Data Networks - Analysis of the Superchannel Concept*, John Willey & Sons, 2005, pogl. 2 , 5

[2] Todd K. Moon, *Error Correction Coding - Mathematical Methods and Algorithms*, Utah State University: John Willey & Sons, 2005, pogl. 1, 15

[3] Jorge Castiñeira Moreira, Patrick Guy Farrell, *Essentials of Error-control coding*, John Willey & Sons, 2006, pogl. 1, 2, 8

[4] Robert H. Morelos-Zaragoza, *The Art of Error Correcting Coding*, San Jose State University: John Willey & Sons, 2006, pogl. 1, 8

[5] John G. Proakis, *Digital communications*, Department of Electrical and Computer Engineering, Northeast University: McGraw-Hill, 1995, pogl. 5

[6] Pavešić Nikola, *Informacija in kodi*, Fakulteta za elektrotehniko, 1997, pogl. 8, 9

[7] William E. Ryan: *A introduction to LDPC codes*
Dostopno na:
http://www.ece.arizona.edu/~ryan/New%20Folder/ryan-crc-ldpc-chap.pdf

[8] C.E. Shanon : *A mathematical theory of communication*
Dostopno na:
http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

[9] *Generating a signal waveform with required SNR in Matlab*
Dostopno na:
http://gaussianwaves.blogspot.com/2010/02/generating-signal-waveform-with.html

[10] James E. Gilley: *Bit-Error-Rate Simulation UsingMatlab*
Dostopno na:
http://www.efjohnsontechnologies.com/resources/dyn/files/75831/_fn/bit-error-rate simulation_using_matlab.pdf

[11] Jian Sun: *A introduction to Low Density Parity (LDPC) Codes*
Dostopno na:
http://www.csee.wvu.edu/wcrl/public/slideldpc.pdf

[12] *Signal Energy*
Dostopno na:
http://www.mathworks.de/matlabcentral/newsreader/view_thread/138172

[13] *Progressive edge growth parity check matrix construction*
Dostopno na:
http://www.inference.phy.cam.ac.uk/mackay/PEG_ECC.html

[14] *On the Computation of the Minimum Distance of Low-Density Parity-Check Codes*
Dostopno na:
http://www.inference.phy.cam.ac.ukmackayMINDIST_ECC.html

[15] Encyclopedia of Sparse Graph Codes
Dostopno na:
http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

Drevo. Veliko drevo z miljoni vej. Veje so ločene. Če se oklepaš vej, kako boš prišel do korenin? Čim niže greš, toliko manj je vej; ko se spuščaš, mnogoterost izginja in prideš do enega samega debla, nerazdeljenega - v njem so vse veje, toda samo ni razdeljeno. Iz njega prihaja vse, iz njega prihaja mnogo, toda eno ostaja eno. To je koren. Vir.

Razločevanje – to je dobro, tisto je slabo, to mi je všeč, tistega ne maram – to razločevanje je sam temelj tvojega uma. Ko razločevanje izgine, um zgrmi v prepad. Dosegel boš njegov vir. In v tistem viru je ves smisel, so vse ekstaze in vsi blagoslovi.

Sosan, *Hsin Hsin Ming*