Univerza v Ljubljani
Fakulteta za elektrotehniko

TURBO KODE

Seminarska naloga pri predmeta Digitalne komunikacije magistrskega študija

Patrik Ritoša

patrik.ritosa@fe.uni-lj.si

Ljubljana, 2005

Kazalo

Slovar kratic

	Angleški izrazi	*Slovenski izrazi*
APP	A Posterior Probability	Aposteriori verjetnost
BCH	Bose – Chaudhuri – Hocquenghem code	Bose – Chaudhuri – Hocquenghem kode
BER	Bit Error Rate	Pogostost bitnih napak
E_b/N_0	Bit Energy to Noise density ratio	Razmerje energije bita in gostote šuma
EXIT	Extrinsic Information Transfer diagram (method)	Pristranski informacijski doprinos
FEC	Forward Error Correction	Vnaprejšnji korekcija napak
LLR	Logarithm Likelihood Ratio	Logaritemsko podobnostno razmerje
MAP	Maximum A posteriori Probability	Maksimalna a posteriori verjetnost
RCS	Recursive Convolution Systematic code	Rekurzivna konvolucijska sistematična koda
SNR	Signal to Noise Ration	Razmerje signal šum

1. Uvod

Turbo kode so nedavno razvita družina zelo zmogljivih kod, za vnaprejšnjo detekcijo in popravljanje napak FEC (ang. Forward Error Correction). Prvoten njihov namen uporabe je bil v satelitskih in vesoljskih komunikacijah. Komunikacijah na velikih razdaljah, pri omejeni moči in pasovni širini ter ob prisotnosti motečega šuma. Od vseh do sedaj poznanih kodirnih postopkov, se turbo kode najbolj približajo Shannon-ovi meji. Teoretični meji za prenos prek šumnega kanala.

Metodo so prvič predstavili Berrou, Glavieux in Thitimjshima leta 1993 [1]. Od tedaj se razvoj in izboljšave izvaja na številnih univerzah po celem svetu.

Z uporabo turbo kod je mogoče izboljšati prenosne lastnosti, ne da bi bilo za to potrebno dodatno izrabiti resurse, ki jih imamo na razpolago (oddajno moč, večji poraba spektra). Pogosto so pogoji zveze, ki jih imamo na razpolago že maksimalno izkoriščeni in predstavlja dober kodirni postopek edino rešitev. Izboljšave oz. kodno ojačanje dobimo na račun kompleksnega dekodiranja. Pri tem se po sprejemu signala izvajajo potrebne obdelave in številne računske operacije.

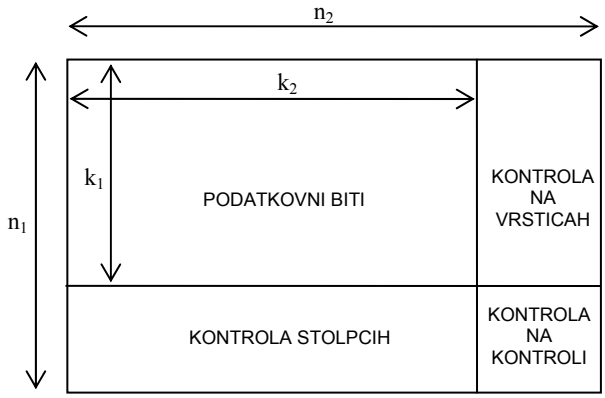
2. Povezovanje kod in osnove turbo kodiranja

Temeljni princip delovanja turbo kod je kombiniranje dveh ali več, med seboj neodvisnih, kodirnih postopkov. Če želimo doseči največji učinek je potrebno poskrbeti, naj bosta kodirna postopka čimbolj neodvisna med seboj. Med seboj je možno kombinirati tako blokovne kot konvolucijske kode.

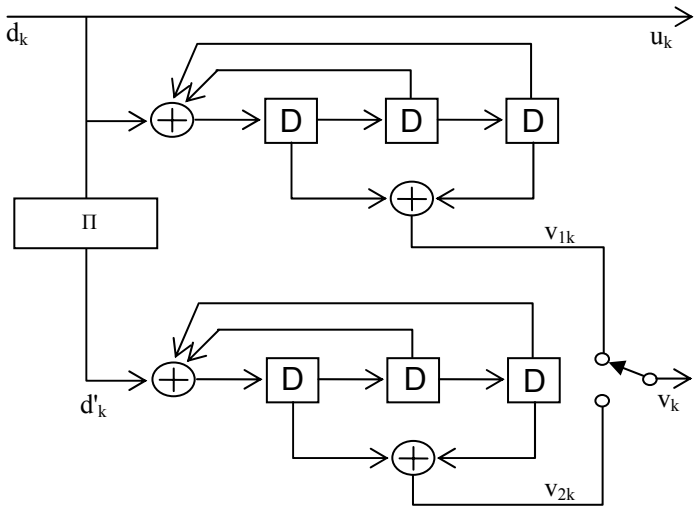
Blokovne kode je možno povezati med seboj, tako da naredimo produkt med dvema ali več kodami in dobimo 2-D ali več dimenzionalne kodirne sheme [2]. Primer 2-D produkta je prikazan na sliki 1, kjer so kontrolni biti vneseni z uporabo ene izmed znanih sistematičnih kodirnih postopkov, tako na vrsticah kot na stolpcih. S tem ko se na istih podatkovnih bitih istočasno izvaja kontrola napak po vrsticah in po stolpcih, je zagotovljeno delovanje dveh med seboj neodvisnih kodirnih postopkov. V obeh primerih se uporabi enak blokovni kodirni postopek.

Pri povezovanju konvolucijskih kod prav tako uporabimo dve ali več izmed nabora znanih konvolucijskih kod [3]. V prikazanem primeru na sliki 2 sta uporabljena dva enaka kodirnika in za doseganje neodvisnosti med kodama je uporabljen mešalnik bitov (ang. Interleaver) označen z Π. Ta element, na izbrani dolžini bitov (npr. 256), psevdo naključno premeče bite ter zagotovi, da vsak kodirnik dobi različno – premešano vhodno sekvenco. S tem je zagotovljena neodvisnost kodirnih postopkov. Poleg tega, se z uporabo mešalnika bitov učinkovito zaščiti proti napakam v izbruhih.

Povezovanje kodirnih postopkov je običajna kodirna praksa. Ponavadi se kode poveže tako, da se isti podatki najprej zakodirajo z eno kodo (notranja koda) in nato vse skupaj še z drugo kodo (zunanja koda). Tak način kodiranje se pogosto uporablja in s tujko se imenuje »concatenation«. V takem primeru delujeta kodi neodvisno ena od druge in lahko povezujemo tudi različne kode med samo.



Slika 1: Povezovanje bločnih kod



Slika 2: Povezava konvolucijskih kod

Pri turbo kodah želimo imeti čim večje sodelovanje med uporabljenimi kodami, zato povezujemo enake kode med sabo. Podatki se hkrati kodirajo z obema kodama. Glavni dobitek ali kodno ojačanje pridobimo prav na račun sodelovanja med kodami pri dekodiranju. To je izvedeno tako, da se informacijo o sprejetih bitih pridobljena s pomočjo ene kode, uporabi pri dekodiranju druge kode.

Na tak način sta na razpolago namesto enega, dva neodvisna podatka, kar omogoča pravilnejše dekodiranje rezultata. Postopek je še dalje izboljšan tako, da se opravi več dekodirnih iteracij. Dekodiran rezultat prve kode vpliva na drugo ter rezultat druge kode vpliva nazaj na prvo kodo. Tak cikel se lahko ponovi N krat.

Pri uporabi blokovnih kod se dekodiranje izvaja tako, da se na bitnem bloku (npr. 8x8) najprej dekodira podatkovne bite kot vrstice na podlagi vrstičnih kontrolnih bitov in nato kot stolpce s pomočjo stolpčnih kontrolnih bitov. Po končani prvi vrstični operaciji dekodiranja imamo na razpolago informacijo o sprejetih bitih, ki jo lahko nato uporabimo pri dekodiranju po stolpcih. Po prvem takem ciklu, lahko na podlagi pridobljenih informacij ponovno dekodiramo podatke kot vrstice in nato ponovno kot stolpce, itd. Po vsakem ciklu je verjetnost detekcije pravilnega rezultata večja.

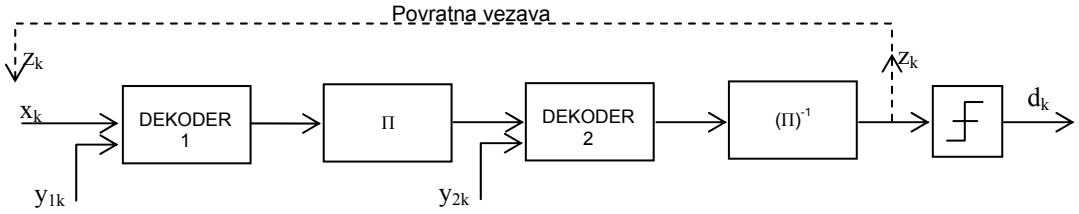
Dekodirnik na sliki 3 je primer turbo dekodiranja z uporabo konvolucijskih kod. V prvem koraku DEKODER 1 dekodira sprejete podatkovne bite x_k s pomočjo pripadajočih kontrolnih bitov y_{1k}. Dobljen rezultat se nato ponovno dekodira še z DEKODERJEM 2 in uporabo drugih kontrolnih bitov y_{2k}. Na izhodu prvega dekodirnika imamo bitno sekvenco, ki je bila popravljena s pomočjo prve vnesene redundance, zato so podatki na izhodu prvega dekoderja bolj verjetni (pravilnejši), kot sprejeti. Nato vhodna bitna sekvenca vstopi v drugi dekodirnik, ki dodatno popravi morebitne napake in na izhodu, po obeh iteracijah dobimo še pravilnejšo sekvenco.

Dobljen rezultat se nato ponovno vodi na vhod prvega dekodirnika, ki še dodatno izboljša rezultat, itn. lahko ponovimo več ciklov. Po vsakem ciklu dobimo bolj verjetno bitno sekvenco. Mešalnik Π ter obraten mešalnik $(\Pi)^{-1}$ sta v tem primeru potrebna, da zagotovita pravilno bitno sekvenco na vhodu posameznega dekodirnika.

$$x_k = u_k + n_k \quad (1)$$

$$y_{1k} = v_{1k} + n_k \quad (2)$$

$$y_{2k} = v_{2k} + n_k \quad (3)$$



Slika 3: Dekoder pri uporabi konvolucijskih kod

Oznake v sliki 3 so v povezavi z oznakami iz slike 2 dodatno utemeljene z izrazi 1,2 in 3 ter pomenijo: x_k predstavlja sprejete informacijske podatke z dodanim šumom n_k, y_{1k} in y_{2k} sta pripadajoči redunčni informaciji v_{1k} in v_{2k} s prav tako dodanim šumom n_k. V vseh treh primerih gre Gaussov šum z ničto srednjo vrednostjo ter varianco σ. Dodane šumne vrednosti n_k posamezni vrednosti so si med seboj povsem nekorelirane. z_k je informacija o izhodni bitni sekvenca, ki se po prvem ciklu prenese nazaj na vhod in pripomore pri ponovnem dekodiranju (dodatni obdelavi).

V osnovi je ideja dekodiranja za oba dekodirna postopka v ista. Za opisane iterativne postopke so nujno potrebni dekodirniki z mehkim odločanjem (ang. Soft decision). Dekodiranje zgleda tako, da se vhodnemu signalu, ki je na prvi pogled podoben šumu, postopoma iz iteracije v iteracijo izoblikujeta dva ločena nivoja.

3. Verjetnostni koncept

Pri turbo kodah se iz iteracije v iteracijo računajo verjetnosti pravilnega sprejema. Sprejete podatke se obdeluje tako, da je po vsaki iteraciji verjetnost pravilnega sprejema večja (pravilnejši rezultat). Za tako obdelavo je potreben primeren verjetnostni račun in logika [3, str. 447-480], ki to omogoči.

Celoten postopek temelji na računanju verjetnosti oddanega simbola na podlagi sprejetega signala. Matematični zapis problema se izraža z Bayes-ovim teoremom zapisanim v izrazu 4 in podaja aposteriori verjetnost dogodka APP (ang. A Posteriori Probability). Izraz govori o verjetnost, da je bil oddan določen simbol, če je bil sprejet določen signal. V izrazu prestavlja *d,* za binaren primer, dva možna oddana logična stanja ('0' in '1'), *p(x)* je porazdelitvena funkcija sprejetega signala. Ostali zapisi so verjetnosti ter pogojne verjetnosti posameznega dogodka.

$$P(d=i|x)=\frac{p(x|d=i)P(d=i)}{p(x)} \qquad i=0,1 \tag{4}$$

Ob sprejemu signala je za določitev logičnega stanja potrebno primerjati verjetnosti obeh možnih stanj (*P(d=0|x)* in *P(d=1|x)*) ter se odločiti za tisto stanje – hipoteho (H_1, H_2) z večjo verjetnostjo. Če povedano izrazimo z izrazom 4, dobimo izraz 5. *p(x)* se pokrajša, ker se nahaja na obeh straneh neenačbe. Če izraz 5 izrazimo še v obliki razmerja dobimo izraz 6. Pri pravilno načrtovanem sistemu je verjetnost nastopa obeh logičnih stanj ('0' in '1') enaka in z se lahko v izrazu 6 nadomesti kvocient verjetnosti kar z enoto.

$$p(x|d=1)P(d=1)\underset{H_2}{\overset{H_1}{>\!<}}p(x|d=0)P(d=0) \tag{5}$$

$$\frac{p(x|d=1)}{p(x|d=0)}\underset{H_2}{\overset{H_1}{>\!<}}\frac{P(d=0)}{P(d=1)}=1 \tag{6}$$

Zaradi lažjega računanja se uvede še logaritemsko vrednost razmerja izraženega z izrazom 6 in dobimo izraz 7. Imenuje se logaritemsko podobnostno razmerje LLR (ang. Log-Likehood Ratio) in se označuje z *L*.

$$L(d\mid x)=\log\left[\frac{p(x\mid d=1)P(d=1)}{p(x\mid d=0)P(d=0)}\right]=\log\left[\frac{p(x\mid d=1)}{p(x\mid d=0)}\right]+\log\left[\frac{P(d=1)}{P(d=0)}\right] \tag{7}$$

ali poenostavljeno zapisano, kjer $L_C(x)$ predstavlja prvi člen izraza 5 in *L(d)* drugi člen izraza 5.

$$L(\hat{d})=L_C(x)+L(d) \tag{8}$$

V izrazu 8 predstavlja *L(d)* vnaprejšnjo – apriori LLR vrednost podatkov, ki se prenašajo in $L_C(x)$ predstavlja izmerjeno LLR vrednost podatkov na podlagi sprejetega signala na koncu prenosnega kanala. Pri turbo kodah pride do izraza še ena LLR vrednost, ki se prišteje zraven $L_E(\hat{d})$. Ta predstavlja postransko vrednost (ang. Extrinsic), ki jo doprinese koda in igra

bistveno vlogo pri uspešnem delovanju kodirnega postopka. Pove koliko informacije o sprejetih bitih doda sama koda. Tako, z vsemi do sedaj omenjenimi LLR vrednosti, dobimo izraz 8, ki zajema vse omenjene parametre.

$$L(\hat{d}) = L_C(x) + L(d) + L_E(\hat{d}) \tag{9}$$

Logaritemsko podobnostno razmerje (LLR) ima tako lastnost, da se lahko npr. v izrazu 7 določi logične stanje sprejete vrednosti '0' in '1' iz predznaka izračunane LLR vrednosti. Pozitivna vrednost pomeni logično '1' in negativna logično '0'. Kar pa je najbolj pomembno, LLR vrednost je lahko zajema vrednosti od $-\infty$ do $+\infty$ in dobimo poleg podatka o logični vrednosti znaka tudi o verjetnost nastopa le tega. To pa nam omogoča mehko odločanje o sprejetih podatkov.

Primer:

$P_1(d=1\,|\,x) = 0.6$ $\qquad$ $P_2(d=1\,|\,x) = 0.99$

$P_1(d=0\,|\,x) = 0.4$ $\qquad$ $P_2(d=0\,|\,x) = 0.01$

$L_1(d\,|\,x) = \log\left[\frac{0.6}{0.4}\right] = 0.176$ $\qquad$ $L_2(d\,|\,x) = \log\left[\frac{0.99}{0.1}\right] = 1.996$

3.1. Uporaba verjetnostne logike pri turbo dekodiranju

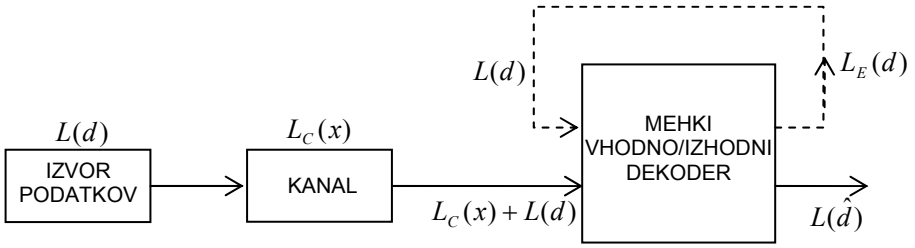
Pri turbo dekodiranju imamo opravka z računanjem LLR vrednosti vhodnih podatkov. Pri postopku izvedemo toliko iteracij, da zagotovimo tako LLR vrednost, ki nudi zadosti veliko verjetnost pravilnega rezultata. Za tak postopek je potreben dekoder z mehkim odločanjem, ki glede na sprejete LLR vrednosti oz. verjetnosti določenih vhodnih spremenljivk ustrezno uteženo vpliva na izhodne spremenljivke. Shematično je postopek prikazan na sliki 4.

Delovanje prikazanega dekoderja se začne z določanjem LLR vrednosti kanala $L_C(x)$. Apriori vrednost $L(d)$ v prvi iteraciji ne pripomore k rezultatu kajti je ne poznamo (obe logični stanji sta enako verjetni) in ima vrednost 0. Na podlagi dekodirnega postopka se izračuna trenutna izhodna vrednost $L(\hat{d})$ ter določi postransko LLR vrednost $L_E(d)$, ki izvira iz lastnosti kode in dekodirnika. Ob začetku drugega cikla, na podlagi rezultata iz prvega, imamo več podatkov o sprejeti sekvenci ter se lahko poleg LLR vrednosti kanala $L_C(x)$ tokrat doda še poznana apriori verjetnost, ki jo predstavlja postranska vrednost iz prvega cikla $L_{1E}=L_2(d)$.

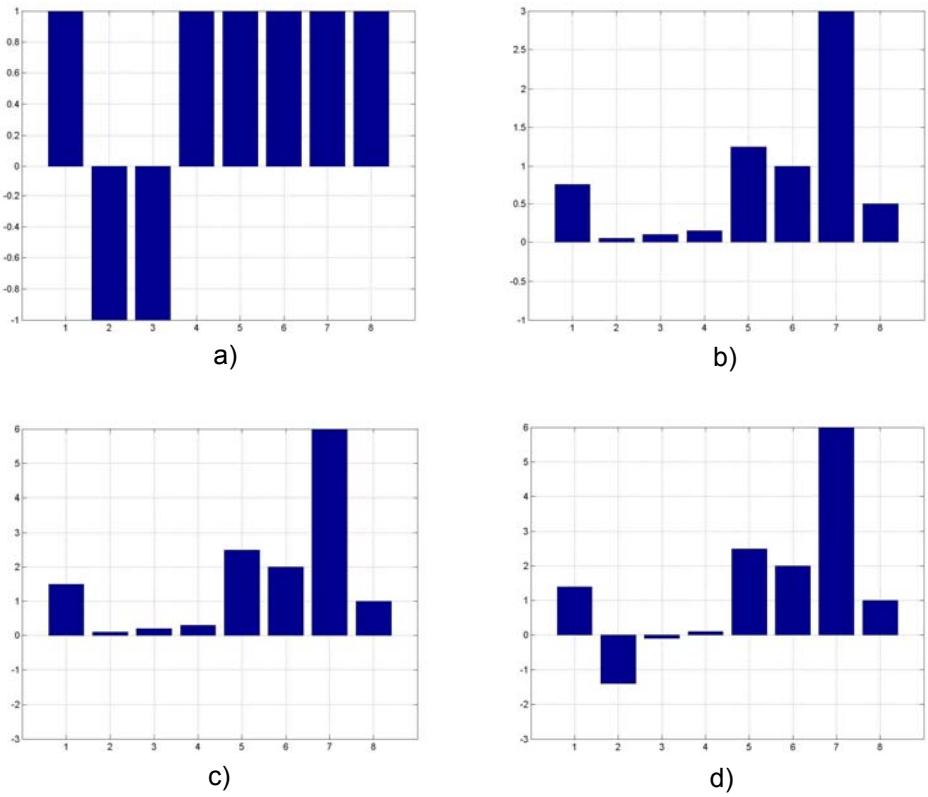
$$L(\hat{d}) = L_C(x) + L(d)$$

V vsakem nadaljnjem ciklu je na razpolago čedalje več informacij o vhodni sekvenci, kar pomeni, da se lahko dekodira rezultat z večjo verjetnostjo. Izraženo v LLR vrednostih pomeni, da se iz iteracije v iteracijo LLR vrednosti čedalje bolj povečujejo v pozitivno ali negativno smer.

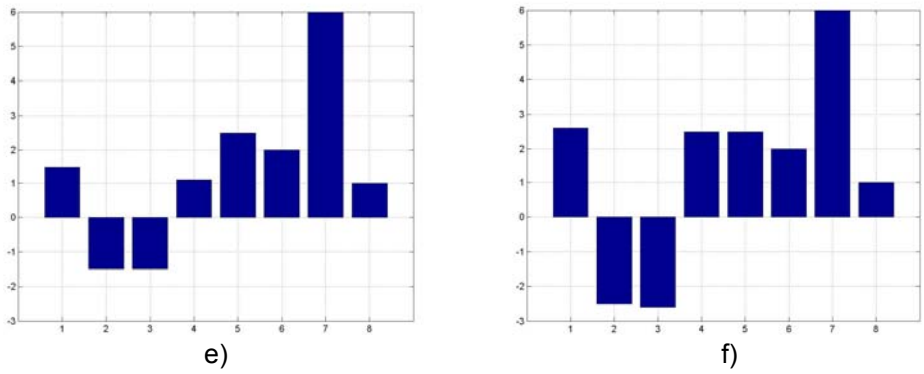
Na sliki 5 je prikazan učinek delovanja bločne turbo kode na osem bitni sekvenci 10011111, ki se prenaša z bipolarnimi vrednostmi '+1' in '-1'. Prvi štirje so informacijski biti, zadnji štirje pa kontrolni biti. Zaradi šuma (Gaussovega z srednjo vrednostjo 0 in varianco 1), se sekvenca pri prenosu pokvari in na sprejemu detektiramo sekvenco prikazano na sliki 2 b. Pri trdi detekciji take sekvence, bi dva bita napačno detektirali. Na slikah 2c, 2d, 2e, 2f, so prikazane postopoma popravljene sekvence. Že po prvi iteraciji bi detektirali pravilo bitno sekvenco, vendar z majhno verjetnostjo pravilnega rezultata. Po četrti iteraciji pa je rezultat zelo jasno določljiv (z veliko verjetnostjo).



Slika 4: Shematičen prikaz delovanja dekoderja z mehkim vhodno/izhodnim odločanjem



Slika 5: Primer delovanja dekodirnika bločne turbo kode prikazan z LLR vrednostmi: a) oddan bipolaren signal, b) sprejet signal, c) sprejet signal izražen z LLR vrednostmi, d) po prvem dekodirnem ciklu, e) po drugem dekodirnem ciklu, f) po četrtem dekodirnem ciklu



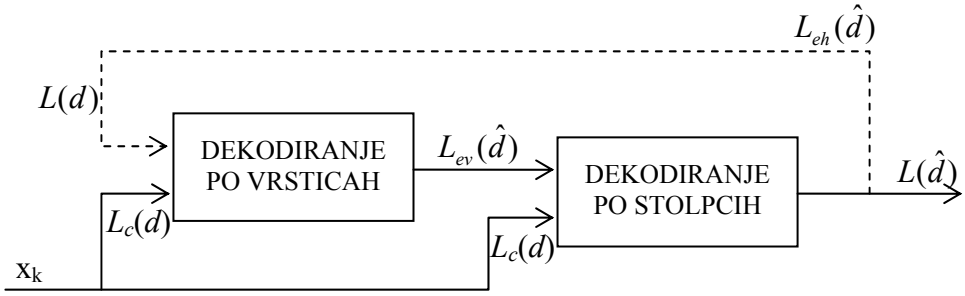
Slika 6: Primer delovanja dekodirnika bločne turbo kode prikazan z LLR vrednostmi: a) oddan bipolaren signal, b) sprejet signal, c) sprejet signal izražen z LLR vrednostmi, d) po prvem dekodirnem ciklu, e) po drugem dekodirnem ciklu, f) po četrtem dekodirnem ciklu

4. Delovanje bločnega turbo dekodirnika

Za primer delovanja bločnega turbo dekodirnika je uporabljena shema 2-D produktne kode prikazana na sliki 1. Zaradi preprostosti zgleda je na produktu 4x4, kot kontrola uporabljen en paritetni bit za vsako vrstico in en za vsak stolpec. Na sliki 7 je shematično prikazan postopek dekodiranja.

V prvem koraku, detektor z mehkim odločanjem, iz sprejetega signala detektira poslano sekvenco in jo posreduje do prvega dekoderja. Ko prvi dekoder sprejme podatke se začne dekodiranje. Vsakemu sprejetemu bitu se najprej določi ustrezna LLR vrednost kanala $L_C(d)$ ter izračuna pristranska vrednost kode $L_{ev}(\hat{d})$. V prvem dekodirnem ciklu, apriori vrednost $L(d)$ ni poznana zato se ji priredi vrednost 0. Po končanem prvem dekodiranju se isti podatki ponovno dekodirajo še v drugem dekoderju, tokrat kot stolpci. V tem primeru je apriori vrednost že znana in sicer se določi na podlagi postranske vrednosti $L_{ev}(\hat{d})$ izračunane v prvem dekodirniku ($L(d)= L_{ev}(\hat{d})$). V opisanem primeru se sprejeta sekvenca najprej dekodira kot vrstice in nato kot stolpci. Načeloma bi lahko bil vrstni red tudi obraten.

Po končanem drugem dekodiranju, se izračunane pristranske vrednosti $L_{eh}(\hat{d})$ iz drugega dekodirnika po povratni vezavi vodijo nazaj na vhod prvega dekodirnika, ki postanejo apriori vrednost za prvi dekoder. Tako se lahko prvo dekodiranje ponovi. Tak cikel se lahko poljubno krat ponovi, pri čemer se iz cikla v cikel rezultat izboljšuje.



Slika 7: Shematičen prikaz delovanja bločnega turbo dekodiranja

Za opisan postopek je potreben tak dekodirni proces, ki omogoča računanje LLR vrednosti iz iteracije v iteracijo. LLR vrednosti se računajo na podlagi: vrednosti sprejetih simbolov, predhodnega znanja o simbolih ter s pomočjo vnesenih kontrlonih bitov.

V tem primeru so pri kodirnem postopku kontrolni biti vneseni tako, da je zagotovljena pravilna pariteta bitne besede. Funkcija s katero je določena vrednost paritetnega bita zapisana z izrazom 10, kjer je z $\oplus$ označena XOR operacija.

$$d_p = d_1 \oplus d_2 \oplus d_3 \tag{10}$$

Pri dekodirnem postopku, z uporabo dekodirnika z mehkim odločanjem, logične operacije zapisane z izrazom 10 ni mogoče izvajati. Potrebna je taka funkcija, ki zna operirati z sprejetimi simboli (mehko odločanje). Korekcija napak, ki se pri dekodiranju izvrši, se izvaja na podlagi izračunane pristranske LLR vrednosti $L_e(\hat{d})$. Pri računanju pristranske LLR

vrednosti za posamezen bit, se upoštevajo do tedaj izračunane LLR vseh ostalih bitov v besedi (vrstici ali stolpcu).

Primer računanje za eno vrstico:

$$L_{ev}(\hat{d}_1) = f(L(d_2), L(d_3), L(d_4))$$
$$L_{ev}(\hat{d}_2) = f(L(d_1), L(d_3), L(d_4))$$
$$L_{ev}(\hat{d}_3) = f(L(d_1), L(d_2), L(d_4))$$
$$L_{ev}(\hat{d}_4) = f(L(d_1), L(d_2), L(d_3))$$

Funkcija, ki opravlja opisano nalogo je zapisana z izrazom 11 ali poenostavljeno z izrazom 12. Izračunane pristranske vrednosti $L_e(\hat{d})$, za določeno vhodno sekvenco imajo take vrednosti, da prispevajo k izboljšanju rezultata (glej sledeči primer).

$$L_{ev}(\hat{d}_4) = f(L(d_1), L(d_2), L(d_3)) = \log\left[\frac{e^{L(d_1)} + e^{L(d_2)} + e^{L(d_3)} + e^{L(d_1)}e^{L(d_2)}e^{L(d_3)}}{1 + e^{L(d_1)}e^{L(d_2)} + e^{L(d_1)}e^{L(d_3)} + e^{L(d_2)}e^{L(d_3)}}\right] \tag{11}$$

$$L_{ev}(\hat{d}_4) = f(L(d_1), L(d_2), L(d_3)) = \operatorname{sgn}[L(d_1)] \cdot \operatorname{sgn}[L(d_2)] \cdot \operatorname{sgn}[L(d_3)] \cdot \min[|L(d_1)|, |L(d_2)|, |L(d_3)|] \tag{12}$$

Primer:

Na oddajni strani je oddana bipolarna sekvenca štirih bitov d_{odd}. Pri sprejemu pride do popačitve oddanega signala zaradi dodanega šuma in detektor detektira vrednosti d_{spr}, ki so različne od oddanih. Pri trdi detekciji in upoštevanju paritetne kontrole bi sicer zaznali napako, vendar je ne bi mogli odpraviti.

Po opisanem dekodirnem postopku se dekodiranje začne z računanjem LLR vrednosti kanala $L_c(d)$ za vsak sprejeti simbol (postopek za Gaussov kanal [3, str. 483-484]).

$$d_{odd} = 1 \quad -1 \quad 1 \quad -1,$$
$$d_{spr} = 0.8 \quad 0.1 \quad 0.5 \quad -0.7$$
$$L_c(d) = 1.6 \quad 0.2 \quad 1.0 \quad -1.4$$

Ko so na razpolago vse LLR vrednosti sprejetih simbolov (v tem primeru so na razpolago samo LLR vrednosti kanala), se lahko z uporabo izraza 11 izračunajo pripadajoče pristranske LLR vrednosti in dobimo naslednje vrednosti.

$$L_e(d_1) = f(0.2, \quad 1.0, \quad -1.4) = -0.06$$
$$L_e(d_2) = f(1.6, \quad 1.0 \quad -1.4) = -0.38$$
$$L_e(d_3) = f(1.6, \quad 0.2, \quad -1.4) = -0.08$$
$$L_e(d_4) = f(1.6, \quad 0.2, \quad 1.0) = 0.06$$

Če upoštevamo izračunane pristranske vrednosti in z njimi korigiramo sprejeto sekvenco dobimo spodje vrednosti, ki tokrat ob trdem določanju dajo pravilen rezultat.

$$L(\hat{d}) = L_c(d) + L_e(d) = (1.6 \quad 0.2 \quad 1.0 \quad -1.4) + (-0.06 \quad -0.38 \quad -0.08 \quad 0.06) =$$
$$= (1.54 \quad -0.18 \quad 0.92 \quad -1.34)$$

Utemeljitev izraza 11 in 12, s pomočjo katerega dobimo pristranske LLR vrednosti je sledeč. Pri računanju prve vrednosti $L_e(d_1)$ je na podlagi ostalih treh vrednosti ugotovljeno, da je prišlo do napake (pariteta se ne ujem). Za zagotoviti pravilno pariteto bi bilo potrebno, prvi vrednosti, za katero računamo, zamenjati logično stanje. Vrednost $L_e(d_1)$=1.6 je potrebno spremeniti v negativno vrednost in zato funkcija določi negativno vrednost. Vendar, ker ima ena od LLR vrednosti s katerimi računamo majhno vrednost $L_e(d_2)$=0.2 (majhno verjetnost), se določi majhna korekcijska vrednost. Na podlagi manj verjetnih vrednosti ni mogoče korigirati bolj verjetno vrednost.

Pri računanju druge pristranske LLR vrednosti je scenarij podoben. Prav tako se ugotovi, da ni pravilna pariteta in prav tako jo je mogoče zagotoviti, le če vrednosti za katero računamo spremenimo logično stanje (spremenimo predznak). Vendar v tem primeru imajo ostali tri simboli velike LLR vrednosti, kar pomeni da je velika verjetnost, da so pravilni. Zato lahko simbol, za katerega računamo, popravimo z dosti večjo vrednostjo. Na podlagi zelo verjetnih vrednosti je mogoče korigirati manj verjetno vrednost. Z enakim postopkom (razmišljanjem) se pride še do ostalih dveh vrednoti.

4.1. Primer turbo dekodiranja 2D produktne kode

V primeru produktne kode se opisan postopek izvede najprej na vrsticah in nato na stolpcih Popravki narejeni pri dekodiranju vrstic pomagajo pri dekodiranju stolpcev. To je smiselno ker sta kodirna postopka po vrsticah in stolpcih nekorelirana in omogočata učinkovito odpravljanje napak. V nasprotnem primeru bi dosegli celo nasproten učinek.

Postopek na 2-D produktni kodi je prikazan na naslednjem primeru. Napačno sprejeti simboli so v razpredelnicah označeni z zasenčenimi polji.

Oddana sekvenca

-1	-1	1	1
-1	1	1	-1
1	1	-1	-1
1	-1	-1	1

a)

Sprejeta sekvenca

-0.5	0.4	1.6	1.2
-1.0	0.4	0.4	0.2
-0.2	1.1	-0.1	-1.2
2.3	-0.2	-0.7	0.4

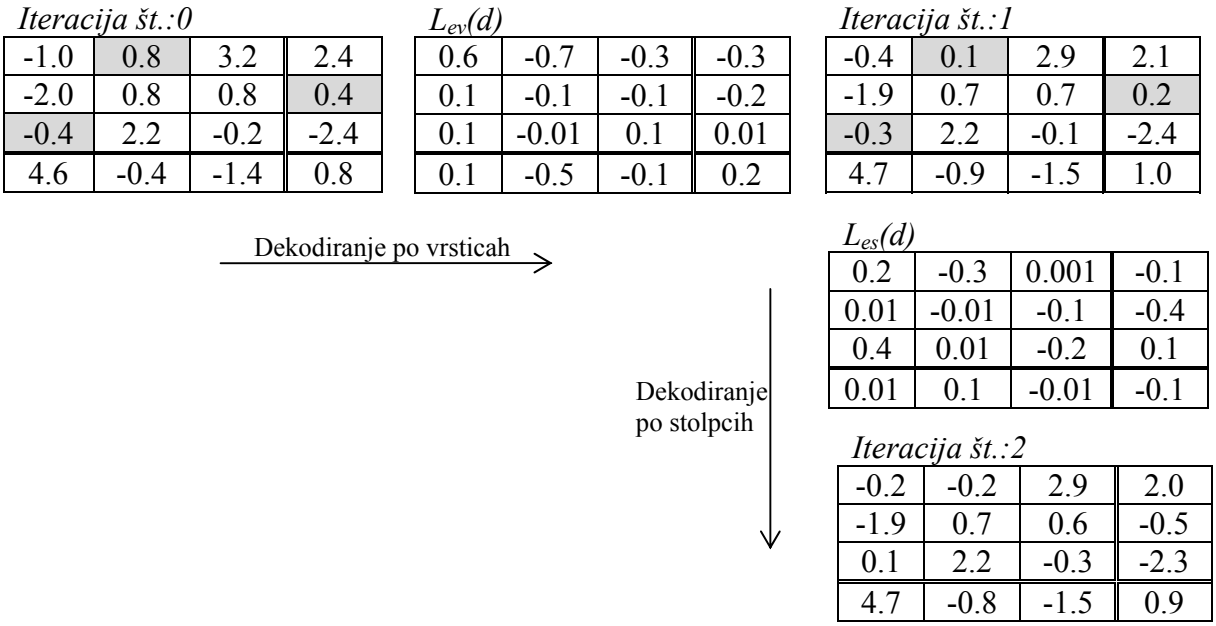
b)

Slika 8: Primer a) oddane bipolarne sekvence in b) sprejete sekvence, popačene zaradi dodanega šuma

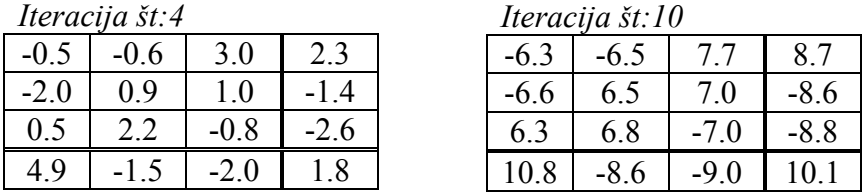
Tako kot v prejšnjem primeru se tudi v tem najprej izračunajo LLR vrednosti sprejetih simbolov. Prikazane so na sliki 9, kjer je označen tudi potek dekodiranja. V naslednjem koraku se za vse simbole izračunajo pristranske vrednosti $L_{ev}(\hat{d})$, z uporabo kontrole po

vrsticah. Izračunane pristranske vrednosti se prištejejo začetnim in dobimo korigirano sekvenco. S tem se konča prva iteracija.

Napake se zmanjšajo vendar so še vedno prisotne. V drugem koraku se s pomočjo popravljenih LLR vrednosti dekodira še po stolpcih (iteracija št.:2). Po dveh iteracija napake izginejo, vendar so rezultati podani z majhno verjetnostjo (majhne LLR vrednosti), zato je potrebno ponoviti več takih iteracij. Dobljene vrednosti se dekodirajo ponovno vrsticah in nato spet po stolpcih, itn. Na tak način dobimo boljši rezultat (iteracija št.:4, iteracija št.:10)



Slika 9: Primer dekodiranja po vrsticah in stolpcih ponazorjen z LLR vrednostmi



Slika 10: Dobljen rezultat po štirih in desetih iteracijah ponazorjen z LLR vrednostmi

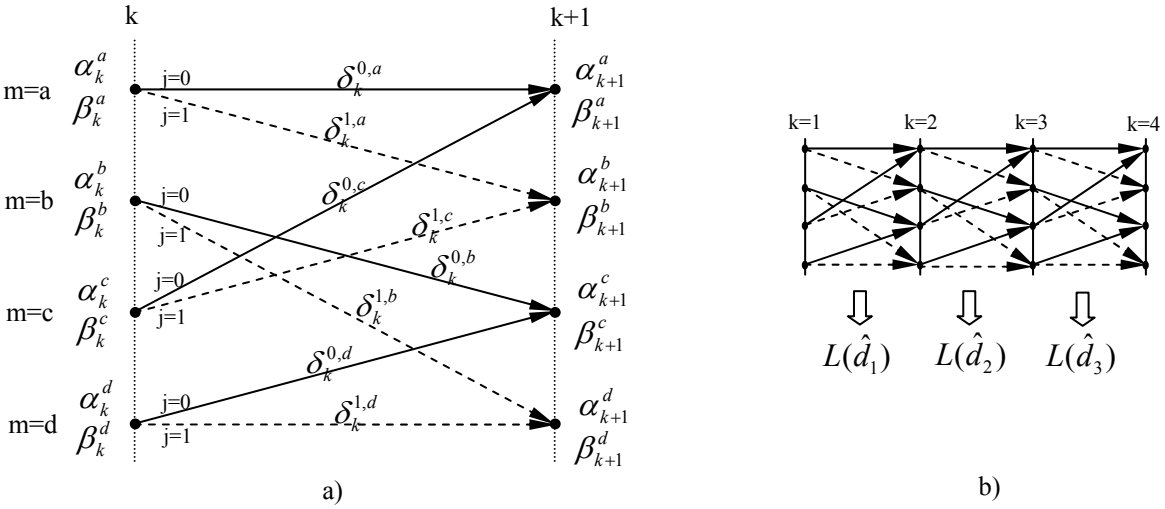
5. Delovanje kunvolucijskega turbo dekodirnika

Dekodirnik konvolucijske turbo kode je na prvi pogled zelo podoben Viterbi-jevemu mrežnemu dekodirnemu postopku, skozi katerega se proces prebija pri postopku dekodiranja. Razlog zakaj, kljub podobnosti ni mogoče uporabiti Viterbijevega dekodirnega postopek je v tem, da ne zna določiti aposteriori verjetnosti (APP) posameznega bita. Viterbijev dekodirni algoritem zna določiti le verjetnost pravilnega sprejema celotne bitne besede, kar pa ni uporabno pri iterativnem procesu določanja sprejetih simbolov. Za ta namen je bil razvit MAP algoritem (ang. Maximum A Posteriori), ki zahtevano aposteriori verjetnost zna izračunati in je temelj konvolucijskega turbo dekodiranja [3, str. 498-509].

5.1. MAP algoritem

Osnova MAP algoritma so vrednosti, tako imenovane metrike stanj prikazanih na sliki 11. Začnejo se računati, ko iz demodulatorja z mehkim odločanjem vstopi v dekoder sekvenca N simbolov. Za pripadajočo mrežno strukturo uporabljenih konvolucijskih kodirnikov-dekodirnikov se izračunajo tri vrste metrik:

1. Tako imenovana vnaprejšnja metrična stanja (ang. Forward metric) α_k^m, za vsako od m stanj.
2. Tako imenovana vzvratna metrična stanja (ang. Backward metric) β_k^m za vsako od m stanj.
3. Metrika veje (ang. Branch metric) $\delta_k^{i,m}$ za vsako vejo mrežnega diagrama.

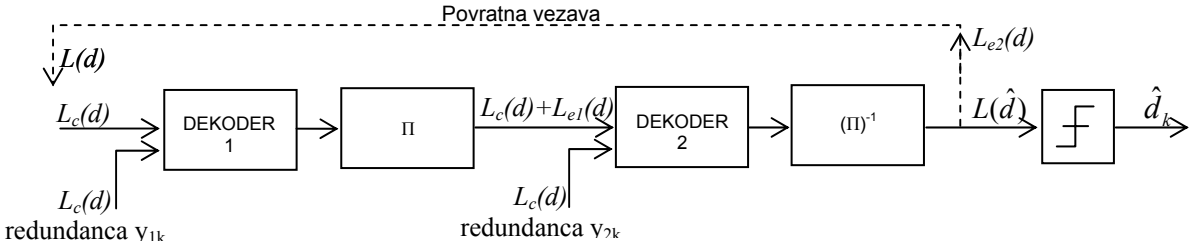


Slika 11: Metrike stanj in prehodov pri MAP algoritmu za primer koderja-dekoderja z 4 stanji (m=a,b,c,d)

Na podlagi izračunanih metrik mrežnega diagrama se nato po izrazu 13 določi še LLR vrednost za vsak posamezen bit. Vrednost posameznega dekodiranega bita je odvisna od pripadajočih vejnih metrik, ter vnaprejšnje in vzvratne metrike, iz katere te izhajajo in v katere vpadajo (slika 11b).

$$L(\hat{d}_k) = \log\left[\frac{\sum_m \alpha_k^m \delta_k^{1,m} \beta_{k+1}^{1,m}}{\sum_m \alpha_k^m \delta_k^{0,m} \beta_{k+1}^{0,m}}\right] + L_e(d_k) \tag{13}$$

V drugem koraku vstopijo enaki vhodni biti še v drugi dekoder. Zaradi mešalnika bitov (Π člena) vstopijo v različnem (premešanem) zaporedju, kot prej. Vstopijo v takem zaporedju, da ustreza drugemu dekoderju. V tem primeru, se zaradi drugačnega vpliva motnje na podatke, izračunajo drugačne metrike stanj (α_k^m, β_k^m in $\delta_k^{i,m}$). Pri ponovnem računanju LLR vrednosti bitov (izraz 13), se v tem primeru upoštevajo tudi apriori vrednosti, ki so kar izračunane vrednosti bitov iz prvega dekoderja $L_e(d_k) = L(\hat{d}_k)$. Po končanem drugem dekodiranju, se lahko isti podatki dekodirajo še enkrat v prvem dekoderju, kar da po več takih ciklih boljši rezultat.



Slika 12: Postopek dekodiranja konvolucijskih turbo kod

5.1.1. Računanje metrik stanj

Pri računanju metrik stanj izhaja postopek iz osnovnega že zapisanega izraza za aposteriori verjetnost ali z LLR razmerjem zapisano z izrazom 14. V izrazu dvakrat nastopata izraza za združeno verjetnost $P(d_k = i, S_k = m \mid R_1^N)$. V števcu je zapisana združena verjetnost za prehod ob logični '1' in v imenovalcu za prehod ob logični '0', za vsa stanja *m*. R_1^N predstavlja popačeno sekvenco bitov, ki vstopi v dekoder.

$$L(\hat{d}_k) = \log\left[\frac{\sum_m P(d_k = 1, S_k = m \mid R_1^N)}{\sum_m P(d_k = 0, S_k = m \mid R_1^N)}\right] \tag{14}$$

Z uporabo Bayes-ovega izreka in po krajšem računu [3, str. 498-502], lahko zgornji izraz za združeno verjetnost prevedemo v nekoliko drugačno, bolj razčlenjeno obliko. Zapisana je z izrazom 15 in zajema vse tri prej opisane metrike stanj.

$$P(d_k = i, S_k = m \mid R_1^N) =$$
$$= P(R_1^{k-1} \mid d_k = i, S_k = m, R_k^N) \cdot P(R_{k+1}^N \mid d_k = i, S_k = m \mid R_k) \cdot P(d_k = i, S_k = m, R_k) / P(R_1^N) =$$
$$= \alpha_k^m \cdot \beta_{k+1}^{i,m} \cdot \delta_k^{i,m} \tag{15}$$

Metrike stanja (α_k^m, β_k^m in $\delta_k^{i,m}$), lahko ob določenih dopustnih zanemaritvah in poenostavitvah zapišemo kot:

$$\alpha_k^m = P(R_1^{k-1} \mid S_k = m) \tag{16}$$

Metrika α_k^m je vnaprejšnja metrika in je odvisna samo od trenutnega stanja *m* in predhodnega elementa R_1^{k-1}.

$$\beta_{k+1}^{i,m} = P(R_{k+1}^N \mid d_k = i, S_k = m, R_k) \tag{17}$$

β_{k+1}^m je metrika povratnega stanja v trenutku (*k+1*), ki na podlagi prihodnjih vhodnih elementov nosi informacijo o verjetnosti trenutnega. Izražen kot pogojna verjetnost naslednjih stanj R_{k+1}^N v odvisnosti od trenutnega stanja S_k=*m* in prehoda d_k=*i*.

$$\delta_k^{i,m} = P(d_k = i, S_k = m, R_k) \tag{18}$$

$\delta_k^{i,m}$ je vejna metrika v času *k* in stanju *m*.

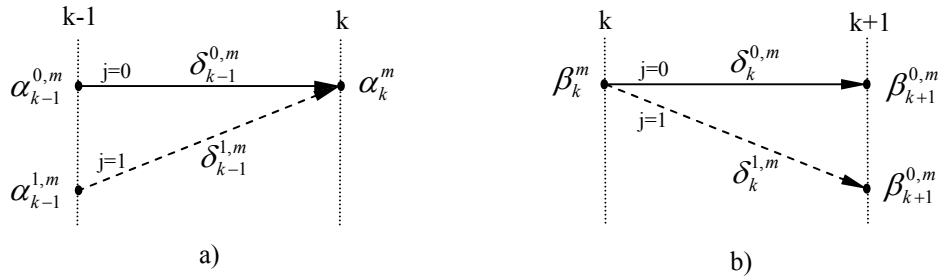
Enkrat, ko so določene metrike stanj že izračunane, lahko preprosto izračunamo še naslednje oz. predhodne po metodi prikazani na sliki 13. Vnaprejšnje metrike stanj α_k^m se zaradi njihove narave vedno začnejo računati od trenutku *k*=1 dalje. Za izračun naslednje uporabimo izraz 19, po sliki 13a. Za novo stanje upoštevamo metrike vseh poti, ki pritekajo novo stanje ter metrike stanj iz katerih izvirajo poti.

$$\alpha_k^m = \sum_{j=0}^{1} \alpha_{k-1}^{j,m} \cdot \delta_{k-1}^{j,m} \tag{19}$$

Povratno metriko stanj se začne računati po izrazu 20 in postopku prikazanem na sliki 13b, od trenutka *k*=*N* nazaj.

$$\beta_k^m = \sum_{j=0}^{1} \delta_k^{j,m} \cdot \beta_{k+1}^{j,m} \tag{20}$$

Na zadnje še način, kako preračunavati z metrikami prehodnih poti (vej). Izračunajo se s pomočjo izraza 21, katerega dobimo po krajši izpeljavi [3, str. 503-504] in velja za Gaussov prenosni kanal. *P(d=i)* je apriori verjetnost posameznega prehoda (sprejem logične '0' ali '1'), x_k in y_k so podatkovni in kontrolni biti popačeni zaradi šuma. u_k^i in $v_k^{i,m}$ so prav tako podatkovni in kontrolni biti, vendar zapisani v bipolarni obliki in ustrezajo posameznem vejnemu prehodu (glej naslednji primer!).

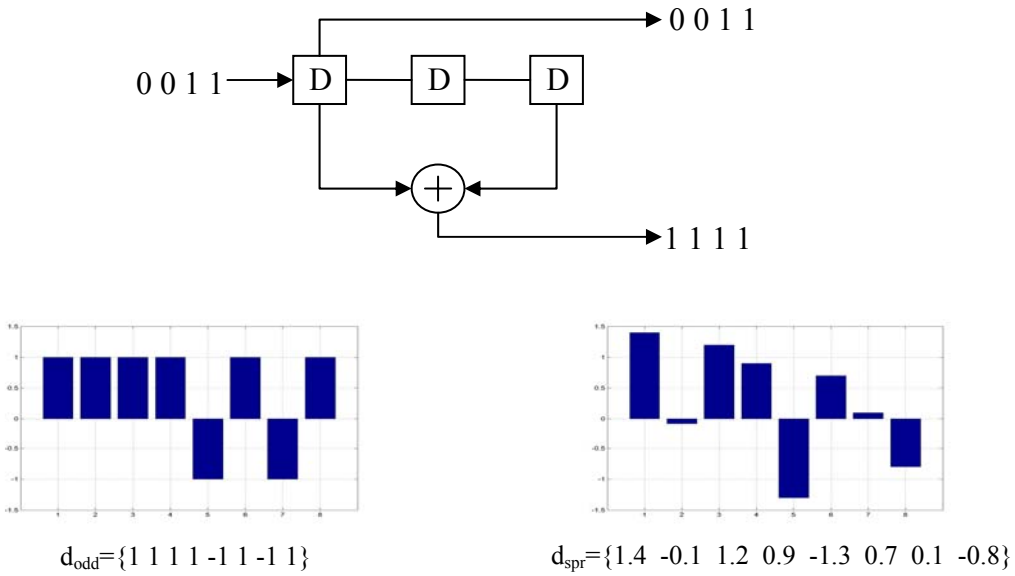


Slika 13: Računanje naslednje/predhodne metrike stanja

$$\delta_k^{i,m} = P(d=i)\exp(x_k u_k^i + y_k v_k^{i,m}) \tag{21}$$

5.2. Primer določanja metrik stanj pri MAP postopku

Primer delovanja MAP algoritma je predstavljen za konvolucijski kodirnik prikazan na sliki 14, s pripadajočim mrežnim diagramom na sliki 15. Na vhod kodirnika pride bitna sekvenca štirih bitov 1 1 0 0, ki se kodirajo v sekvenco 1 1 1 1 0 1 0 1. Skrajno levi bit je prvi bit, ki vstopi v kodirnik/dekodirnik. Taka sekvenca se nato pošlje v bipolarni obliki (1 1 1 1 -1 1 -1 1) preko šumnega kanala. Na sprejemu detektor detektira naslednje vrednosti: 1.38 -0.01 0.98 0.95 -1.00 0.68 0.10 -0.87, ki so močno popačena zaradi dodanega šuma.



Slika 14: Prikaz uporabljenega konvolucijskega kodirnika

V primeru, če bi tako bitno sekvenco preprosto detektirali z trdim odločanjem pri odločitvenim pragom pri 0, bi detektirali kar tri napačne bite od osmih. Pri prenosu štirih podatkovnih bitov, bi v tem primeru prenesli enega napačno.V nadaljevanju je prikazano

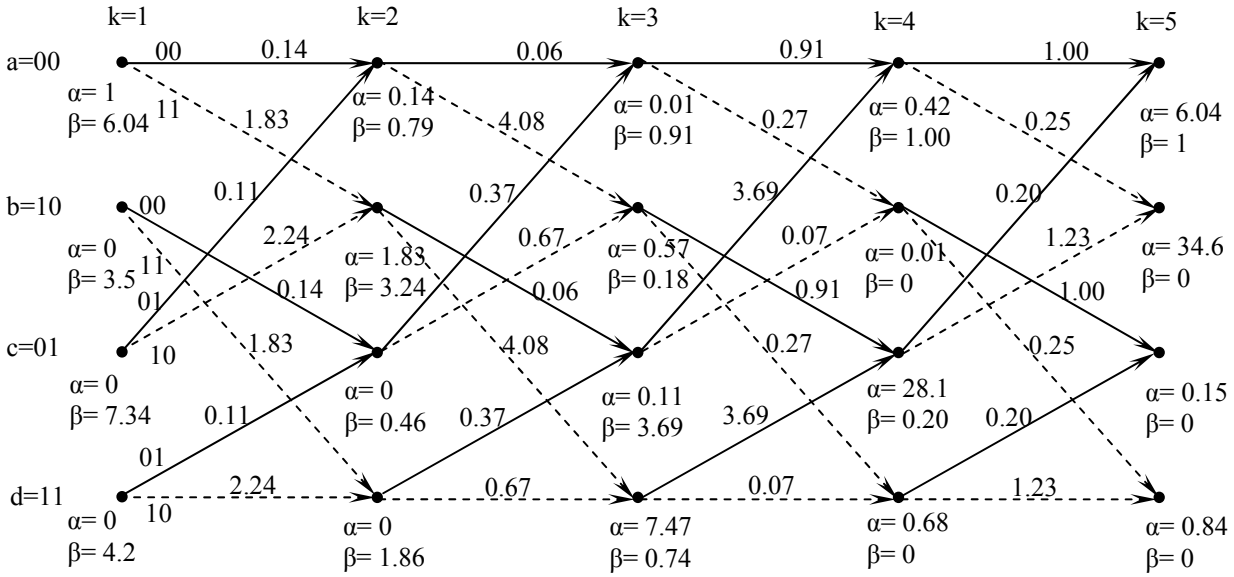
postopek kako se začne turbo dekoditanje z uporabo MAP algoritma in kakšne rezultate se da doseči.

Postopek se začne z mrežnim diagramom prikazanim na sliki 2. Z polno črto so označene veje, ki se zgodijo ob nastopu '0' na vhodu kodirnika/dekodirnika in črtkano črto veje ob nastopu '1'. Na vejah (prehodih) so za trenutek k=1 označena tudi pripadajoča izhodna stanja kodirnika (00, 11, 01 in 10), ki so potrebna za računanje metrik prehodov $\delta_k^{i,m}$. S temi se računaje tudi začne in sicer s pomočjo enačbe 21.

Vsako vejo odraža eno izhodno stanje, zato se pri računanju metrik prehodov upošteva pravilna bipolarna sekvenca za določeno vejo ($u_k^i, v_k^{i,m}$) ter dejanske vrednosti sprejetih bitov (x_k, y_k). Prvi člen vsakega para predstavlja podatkovne bite, medtem ko drugi kontrolne. Apriori verjetnost sprejetih bitov je $P(d=i)$=0.5, ker kot običajno v prvem koraku ne poznamo verjetnosti nastopa simbola in oba logična stanja sta enako verjetna. Izrazi, ki sledijo, so izračuni vejnih metrik za trenutek k=1.

$$\delta_k^{i,m} = P(d=i)\exp(x_k u_k^i + y_k v_k^{i,m})$$

$$\delta_1^{0,a} = 0.5\exp(1.4(-1) - 0.1(-1)) = 0.14 \qquad \delta_1^{1,a} = 0.5\exp(1.4\cdot(1) - 0.1\cdot(1)) = 1.83$$
$$\delta_1^{0,b} = 0.5\exp(1.4(-1) - 0.1(-1)) = 0.14 \qquad \delta_1^{1,b} = 0.5\exp(1.4\cdot(1) - 0.1\cdot(1)) = 1.83$$
$$\delta_1^{0,c} = 0.5\exp(1.4\cdot(-1) - 0.1\cdot(1)) = 0.11 \qquad \delta_1^{0,c} = 0.5\exp(1.4\cdot(1) - 0.1\cdot(-1)) = 2.24$$
$$\delta_1^{0,d} = 0.5\exp(1.4\cdot(-1) - 0.1\cdot(1)) = 0.11 \qquad \delta_1^{0,d} = 0.5\exp(1.4\cdot(1) - 0.1\cdot(-1)) = 2.24$$



Slika 15: Meržni diagram s pripadajočimi metrikami stanj

V trenutku k=1 se upoštevajo prve dve sprejete vrednosti (1.4 in -0.1), v drugem k=2 druge dve (1.2 in 0.9) ter tako do konca dobimo metrike za vse prehode.

V naslednjem koraku se izračunajo vnaprejšnja metrična stanja α_k^m in vzvratna metrična stanja β_k^m. Postopka sta praktično enaka, le da poteka računanje v obratne smeri. Začnejši z

α_k^m-ji, se najprej določijo metrike za trenutek *k*=1. Glede na to, da kodirnik na začetku vsebuje prazen pomikalni register, se nahaja v stanju a=00. Zato je to stanje v danem trenutku med vsemi najbolj verjetno in se mu priredi metriko $\alpha_1^a = 1$. Ostalim stanjem priredi vrednost 0 ($\alpha_1^b = 0, \alpha_1^c = 0, \alpha_1^d = 0$).

Vnaprejšnje metrike za nadaljnja prihodnja stanja se izračunajo s pomočjo enačbe 19. Postopek se nato izvaja do konca, tako da se izračunajo vsa vnaprejšnja stanja. Izračuni, ki sledijo so za trenutek *k*=2.

$$\alpha_k^m = \sum_{j=0}^{1} \alpha_{k-1}^{j,m} \cdot \delta_{k-1}^{j,m}$$

$$\alpha_2^a = \alpha_1^{0,a} \cdot \delta_1^{0,a} + \alpha_1^{0,c} \cdot \delta_1^{0,c} = 1 \cdot 0.14 + 0 \cdot 0.11 = 0.14$$
$$\alpha_2^b = \alpha_1^{1,a} \cdot \delta_1^{1,a} + \alpha_1^{1,c} \cdot \delta_1^{1,c} = 1 \cdot 1.83 + 0 \cdot 2.24 = 1.83$$
$$\alpha_2^c = \alpha_1^{0,b} \cdot \delta_1^{0,b} + \alpha_1^{0,d} \cdot \delta_1^{0,d} = 0 \cdot 0.14 + 0 \cdot 0.11 = 0$$
$$\alpha_2^d = \alpha_1^{1,b} \cdot \delta_1^{1,b} + \alpha_1^{1,d} \cdot \delta_1^{1,d} = 0 \cdot 1.83 + 0 \cdot 2.24 = 0$$

Pri računanju vzvratnih koeficientov gre na podoben način. V tem primeru je dodeljena metrika 1 vzvratnemu koeficientu v stanju *a*, v končnem trenutku *k=5* ($\beta_5^a = 1$). Ostalim pa 0 ($\beta_5^b = 0$, $\beta_5^c = 0$, $\beta_5^d = 0$). To lahko naredimo zaradi tega, ker običajno se ob koncu podatkovnih bitov doda še serijo ničel za ločitev bitnih besed. S tem se pomikalni register izprazni in se vrne v stanje a=00. To velja tudi za naš primer.

Postopek računanja ostalih vzvratnih metrik, se računa po izrazu 20 in je spodaj prikazan za trenutek *k*=4. Ponavlja se do trenutka *k*=1, tako da pridemo do vseh potrebnih vzvratnih metrik.

$$\beta_k^m = \sum_{j=0}^{1} \delta_k^{j,m} \cdot \beta_{k+1}^{j,m}$$

$$\beta_4^a = \delta_4^{0,a} \cdot \beta_5^{0,a} + \delta_4^{1,a} \cdot \beta_5^{1,b} = 1.00 \cdot 1 + 0.25 \cdot 0 = 1.00$$
$$\beta_4^b = \delta_4^{0,b} \cdot \beta_5^{0,c} + \delta_4^{1,b} \cdot \beta_5^{1,d} = 1.00 \cdot 0 + 0.25 \cdot 0 = 0$$
$$\beta_4^c = \delta_4^{0,c} \cdot \beta_5^{0,a} + \delta_4^{1,c} \cdot \beta_5^{1,b} = 0.20 \cdot 1 + 1.23 \cdot 0 = 0.20$$
$$\beta_4^d = \delta_4^{0,d} \cdot \beta_5^{0,c} + \delta_4^{1,d} \cdot \beta_5^{1,d} = 0.20 \cdot 0 + 1.23 \cdot 0 = 0$$

Ko imamo na razpolago vse metrike lahko po izrazu 14 določimo še LLR vrednosti za sprejete bite in dobimo sledeče rezultate.

$$L(\hat{d}_k) = \log\left[\frac{\sum_m \alpha_k^m \delta_k^{1,m} \beta_{k+1}^{1,m}}{\sum_m \alpha_k^m \delta_k^{0,m} \beta_{k+1}^{0,m}}\right] + L(d)$$

$$L(\hat{d}_1) = \log\left[\frac{\alpha_1^a \delta_1^{1,a} \beta_2^{1,b} + \alpha_1^b \delta_1^{1,b} \beta_2^{1,d} + \alpha_1^c \delta_1^{1,c} \beta_2^{1,b} + \alpha_1^d \delta_1^{1,d} \beta_2^{1,d}}{\alpha_1^a \delta_1^{0,a} \beta_2^{0,a} + \alpha_1^b \delta_1^{0,b} \beta_2^{1,c} + \alpha_1^c \delta_1^{0,c} \beta_2^{1,a} + \alpha_1^d \delta_1^{0,d} \beta_2^{1,c}}\right] + 0 =$$

$$= \log\left[\frac{1 \cdot 1.83 \cdot 3.24}{1 \cdot 0.14 \cdot 0.79}\right] = \log\left[\frac{5.93}{0.11}\right] = 1.73$$

$$L(\hat{d}_2) = \log\left[\frac{0.14 \cdot 4.08 \cdot 0.18 + 1.83 \cdot 4.08 \cdot 0.74}{0.14 \cdot 0.06 \cdot 0.91 + 1.83 \cdot 0.06 \cdot 3.69}\right] = \log\left[\frac{5.63}{0.41}\right] = 1.13$$

$$L(\hat{d}_3) = \log\left[\frac{0}{0.01 \cdot 0.91 \cdot 1 + 0.57 \cdot 0.91 \cdot 0.2 + 0.11 \cdot 3.69 \cdot 1 + 7.47 \cdot 3.69 \cdot 0.2}\right] = \log\left[\frac{0}{6.03}\right] = -\infty$$

$$L(\hat{d}_4) = \log\left[\frac{0}{0.42 \cdot 1.0 \cdot 1 + 28.08 \cdot 0.2 \cdot 1 + 7.47 \cdot 3.69 \cdot 0.2}\right] = \log\left[\frac{0}{11.54}\right] = -\infty$$

$L(\hat{d}_1) = 1.73 \Rightarrow$'1'
$L(\hat{d}_2) = 1.13 \Rightarrow$'1'
$L(\hat{d}_3) = -\infty \Rightarrow$'0'
$L(\hat{d}_4) = -\infty \Rightarrow$'0'

Iz rezultata se vidi, da smo z MAP algoritmom izračunali pravilno bitno sekvenco. Če bi uporabili še en neodvisen kodirnik/dekodirnik, bi lahko postopek dekodiranja ponovili. Pri drugem dekodiranju, bi v izrazu 14, rezultat prvega dekoderja upoštevali kot apriori znanje o iskani bitni sekvenci.

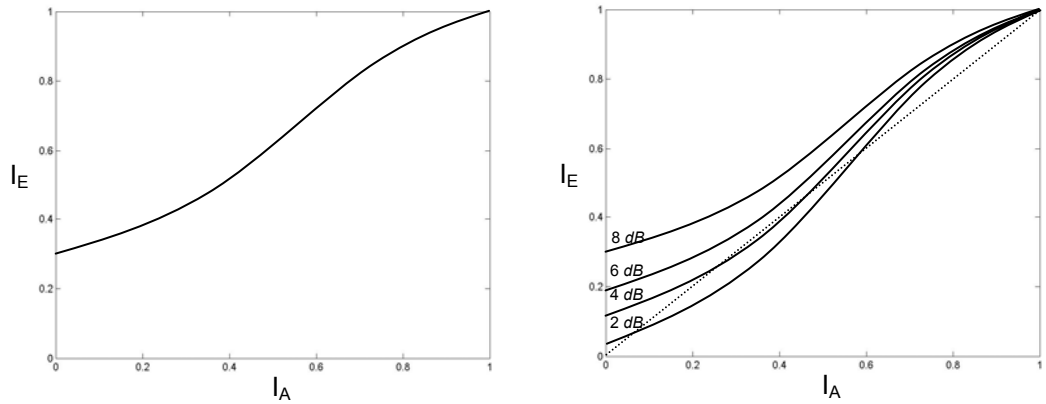
6. Konvergenca in število iteracij pri turbo kodah

Pri postopkih, ki smo jih do sedaj opisali je pomembno analizirati, koliko so učinkoviti [4]. Glede na to, da so turbo kode v osnovi sestavljene iz znanih bločnih ali konvolucjskih kod, se je pri učinkovitosti zlasti potrebno vprašati pri katerem razmerju signal-šum SNR (ang. Signal to Noise Ratio) ali razmerju E_b/N_0 (energija posameznega bita na gostoto prisotnega šuma) postopek še omogoča prenos pri izbrani pogostosti napak. Ostali značilni podatki kot so: kodno razmerje, evklidska distanca so v splošnem že znani iz same značilnosti uporabljene sestavne kode.

Cilj analize je določiti čim bolj preprost postopek, ki omogoča čim boljše delovanje, pri čim manjšem razmerju SNR (ali pripadajočem E_b/N_0). V primeru turbo kod, kjer se pri postopku dekodiranja opravlja veliko računskih operacij je preprostost ali učinkovitost določena z preprostostjo in številom operacij potrebnih, da postopek konvergira k rešitvi [5]. Pri analizi učinkovitosti je zlasti uporaben tako imenovan EXIT diagram (ang. Extrinsic Information Transfer) [6].

EXIT diagram je graf prikazan na sliki 16, ki podaja odvisnost med apriori informacijo (I_A) vhodnega znaka v dekodirnik in pristransko (Extrinsic) informacijo (I_E), ki jo dekodirnik doda. Govori o učinkovitosti dekoderja, da poveča informacijo vhodnega znaka. Na sliki 16b so prikazane omenjene odvisnosti (v obliki krivulje) za enak dekoder pri različnih vhodnih razmerjih SNR.

Pri večjih vhodnih SNR vrednosti dobi dekoder več vhodne informacije, zato lahko tudi doda več pristranske informacije in krivulja se premakne više. Osnovna oblika in višina krivulje je pri enaki vhodni SNR vrednosti odvisna od učinkovitosti ali uspešnosti uporabljenega dekoderja.



a) b)

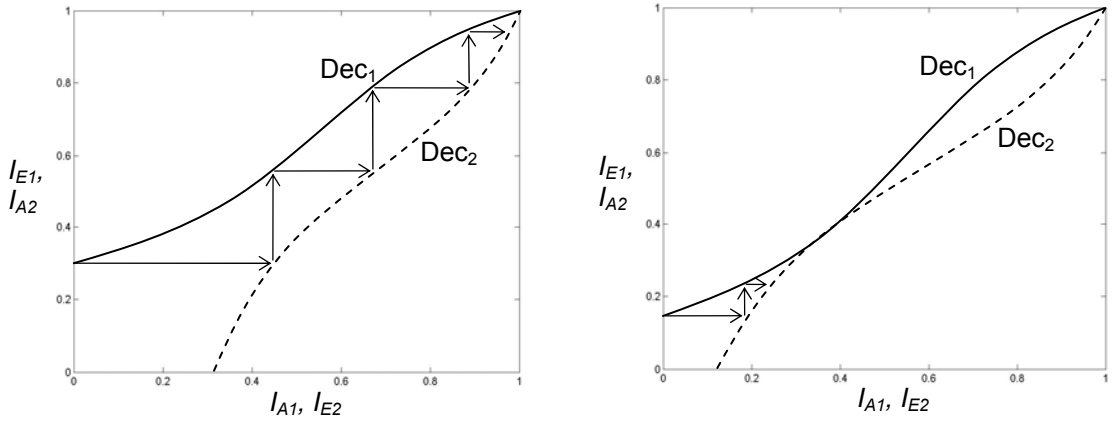
Slika 16: Primer EXIT diagrama

Pri turbo dekodirnih postopkih se uporabljata najmanj dva dekoderja in načeloma imata lahko oba lastni karakteristiki. Zato se običajno EXIT diagram riše kot dva grafa, enega čez drugega, ki istočasno prikazuje karakteristiki obeh dekoderjev. Primer je prikazan na sliki 17, kjer je prikazan tudi postopek konvergence.

Po prvi iteraciji pristranska vrednost prvega dekoderja I_{E1} postane apriori vrednost drugega I_{A2} in tako naprej, kot je ponazorjeno z puščicami. Če imata dekodirnika značilnosti, kot jih

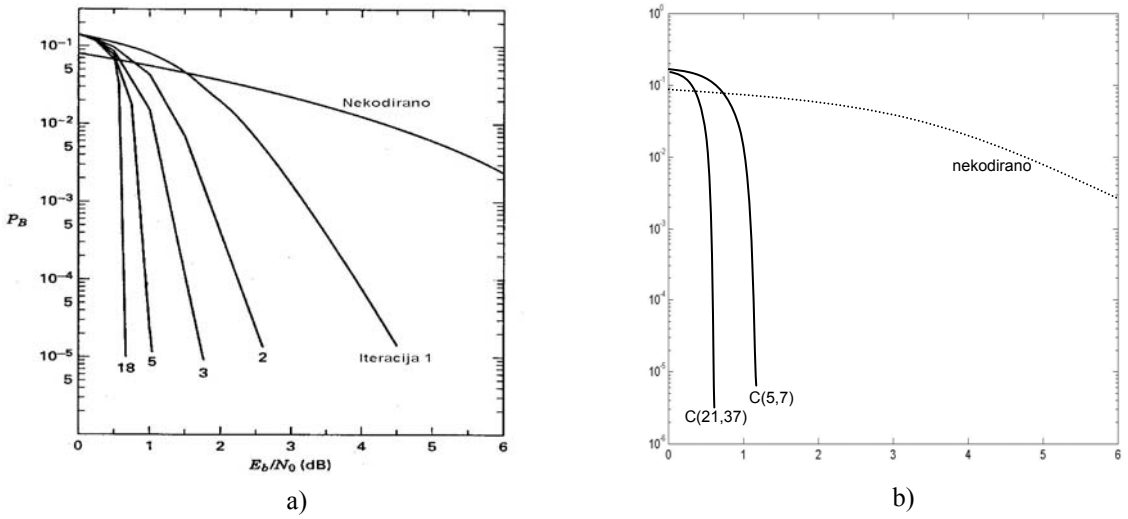
prikazuje EXIT diagram na slika 17a bo postopek hitro konvergiral. Že v nekaj iteracija bodo izhodni dekodirani simboli imeli veliko informacijo (1 bit bo nosil približno 1 bit informacije).

V primeru na sliki 17b, sta zaradi manjšega vhodnega SNR razmerja krivulji pomaknjeni bolj skupaj in postopek ne konvergira k končni vrednosti 1 ampak se ustavi na določeni točki prej. Zaradi tega, kljub povečevanju števila iteracij postopek ne bo dosegel boljšega rezultata. Če želimo izboljšati razmere je potrebno doseči večjo SNR razmerje ali izbrati bolšji kodirni/dekodirni postopek, ki ima pri isti SNR vrednosti bolj ugodne karakteristike (bolj razmaknjeni krivulji).



Slika 17: Prikaz konvergence turbo dekodiranja na EXIT diagramu a) SNR_1, b) $SNR_2 < SNR_1$

Če povežemo rezultate iz EXIT diagrama z grafoma na sliki 18, ki prikazujeta verjetnost napake P_B v odvisnosti od vhodnega SNR razmerja, jasno vidimo zakaj z povečevanjem števila iteracij ne dobimo sorazmerno boljše vrednosti. Ko se nahajamo blizu konvergenčne točke, bodisi točke iz slike 17a ali 17b, smo dosegli praktično največ kar se da z uporabljenim kodirnim/dekodirnim postopkom in z večanjem števila iteracij ni mogoče bistveno izboljšati rezultata.

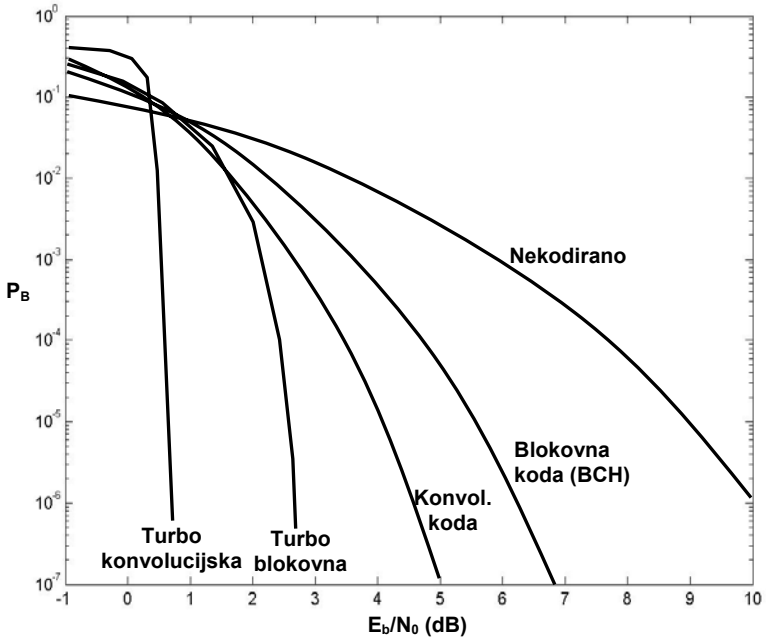


Slika 18: Primer učinkovitosti turbo kod a) konvolucijska turbo koda C(21,37) pri različnem številu iteracij in b) primerjava različnih konvolucijskih kod pri istem številu iteracij

Pri opazovanju delovanja različnih turbo kod pride do izraza oster prehod, pri katerem pri postopnem povečevanju vhodnega SNR razmerja preidemo iz stanja z velikim številom napak v stanje brez napak. Če ta pojav opazujemo iz grafu na sliki 18b, ki prikazuje verjetnost napake P_B v odvisnosti od vhodnega razmerja SNR, opazimo značilen oster prehod v obliki vodnega slapa (ang. waterfall). Opisan oster prehod v delovanju, se v EXIT diagramu opazi takrat, ko se krivulji razmakneta in se prvič pojavi neprekinjen tunel od začetne do končne vrednosti (I_{E2}=I_{E1}=1). Opisan prehod se zgodi zelo na hitro in pri zelo majhni spremembi vhodnega SNR razmerja (≈1 *dB* prirastka).

7. Praktične izvedbe turbo kod

Za konec so prikazane nekatere praktične izvedbe ter rezultati, ki jih je mogoče doseči z turbo kodami. Zlasti zanimiva je primerjava učinkovitosti turbo kod, s kodami iz katerih so sestavljene. Na sliki 19 so prikazani tipični rezultati, ki jih je mogoče doseči z uporabo različnih kodo. Iz slike se tudi lepo vidi na katerem področju je določena koda učinkovita in kje ima svoje meje.



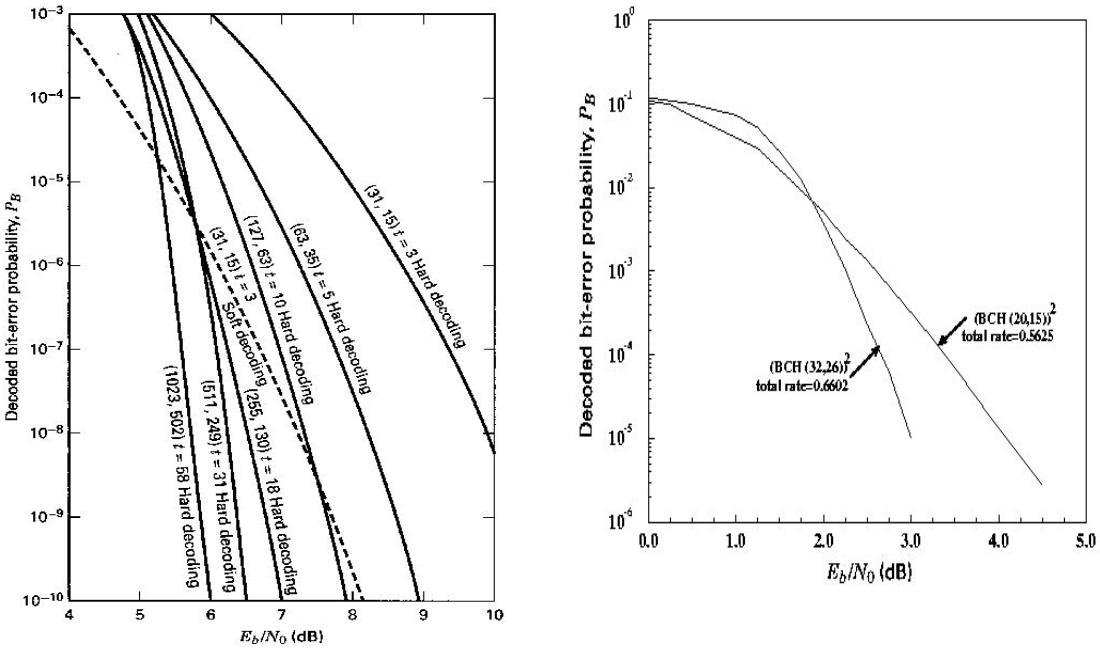
Slika 19: Primerjava učinkovitosti različnih kodirnih postopkov

Z istimi kodami in z dodatnimi računskimi operacijami pri dekodiranju, je mogoče doseči dosti boljše rezultate (turbo kode). Dodatno kodno ojačanje pride na račun kompleksnih računskih operacij, ki se opravljajo po sprejemu signala. To je bistvenega pomena, kajti s tem lahko izboljšamo kvaliteto prenosa ali pa povečamo prenos na kanalu, ki deluje na meji zmogljivosti.

Na sliki 20a so prikazani rezultati za posamezne BCH kode. Na posamezni krivulji so označeni pripadajoči parametri za uporabljene kode, kjer je z *t* označeno največje število napak, ki jih koda lahko popravi. Lahko opazimo kako se z dodajanjem redundance izboljšuje učinkovitost kode in do katere meje jo ima smisel dodajati.

Na sliki 20b so prikazani rezultati za bločne turbo kode, ki uporablja blokovne BCH kode (prikazan rezultat je po štirih opravljenih iteracijah). Iz primerjave rezultatov je mogoče oceniti dodatno kodno ojačanje, ki ga je mogoče doseči pri uporabi ekvivalentnih kot povezanih v turbo kodo.

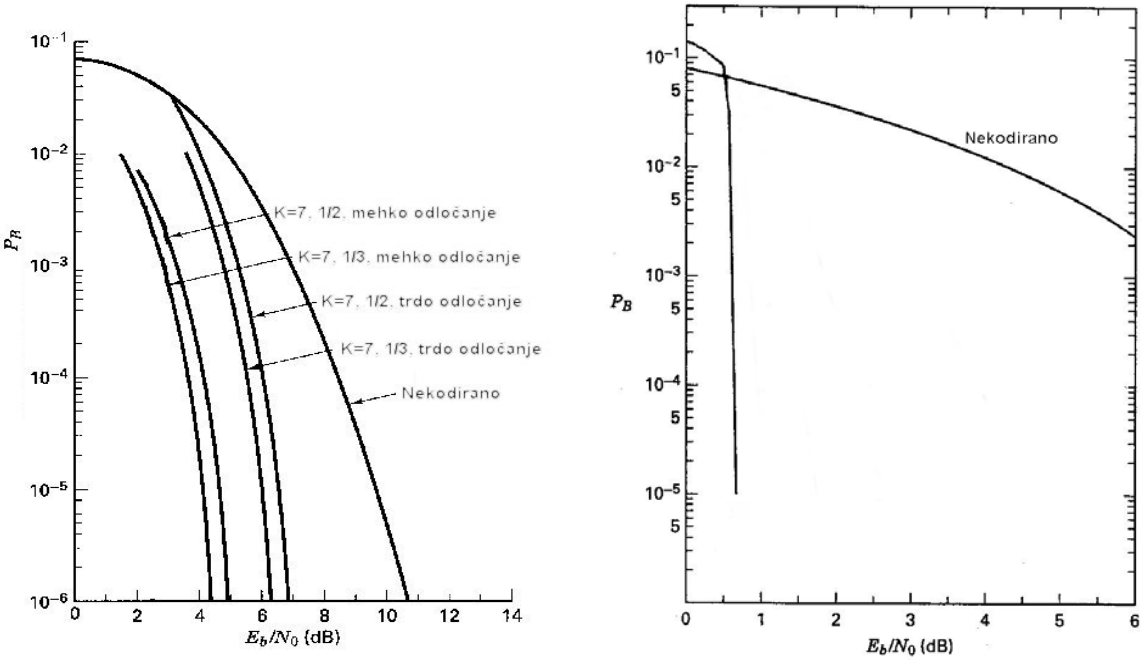
Tako kot za bločne kode, je narejena primerjava tudi za konvolucijske kode. Na sliki 21a so prikazani rezultati, katere je mogoče doseči z različnimi konvolucijskimi kodami. V tem primeru so rezultati nekoliko boljši kot pri uporabi BCH kod, vendar tudi v tem primeru se jasno vidi pri kateri meji se učinkovitost kode ustavi.



a) b)

Slika 20: Učinkovitost a) različnih BCH kod, b) bločnih turbo kod z uporabo BCH kod

Na sliki 21b so prikazani rezultati za konvolucijsko turbo kodo. Uporabljeni sta dve enaki konvolucijski kodi (K=5, 1/2, C(37,21)) med katerima je vstavljen mešalnik bitov na bloku 65536 bitov. Prikazan rezultat je dobljen po 18 iteracijah. Tudi v tem primeru dobimo relativno veliko kodno ojačanje (približno 4dB) in se že močno približamo Shannon-ovi teoretični meji prenosa prek šumnega kanala.



a) b)

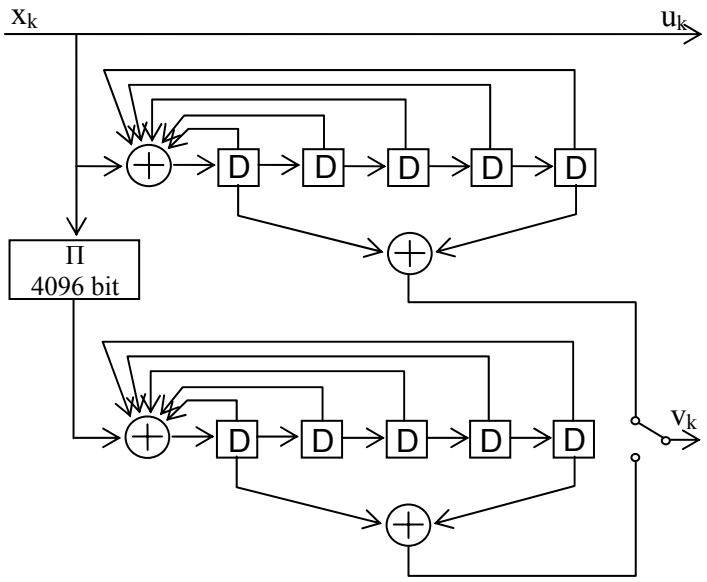
Slika 21: Učinkovitost a) različnih konvolucijskih kod, b) konvolucijske turbo kode

7.2. Prikaz delovanja turbo kod

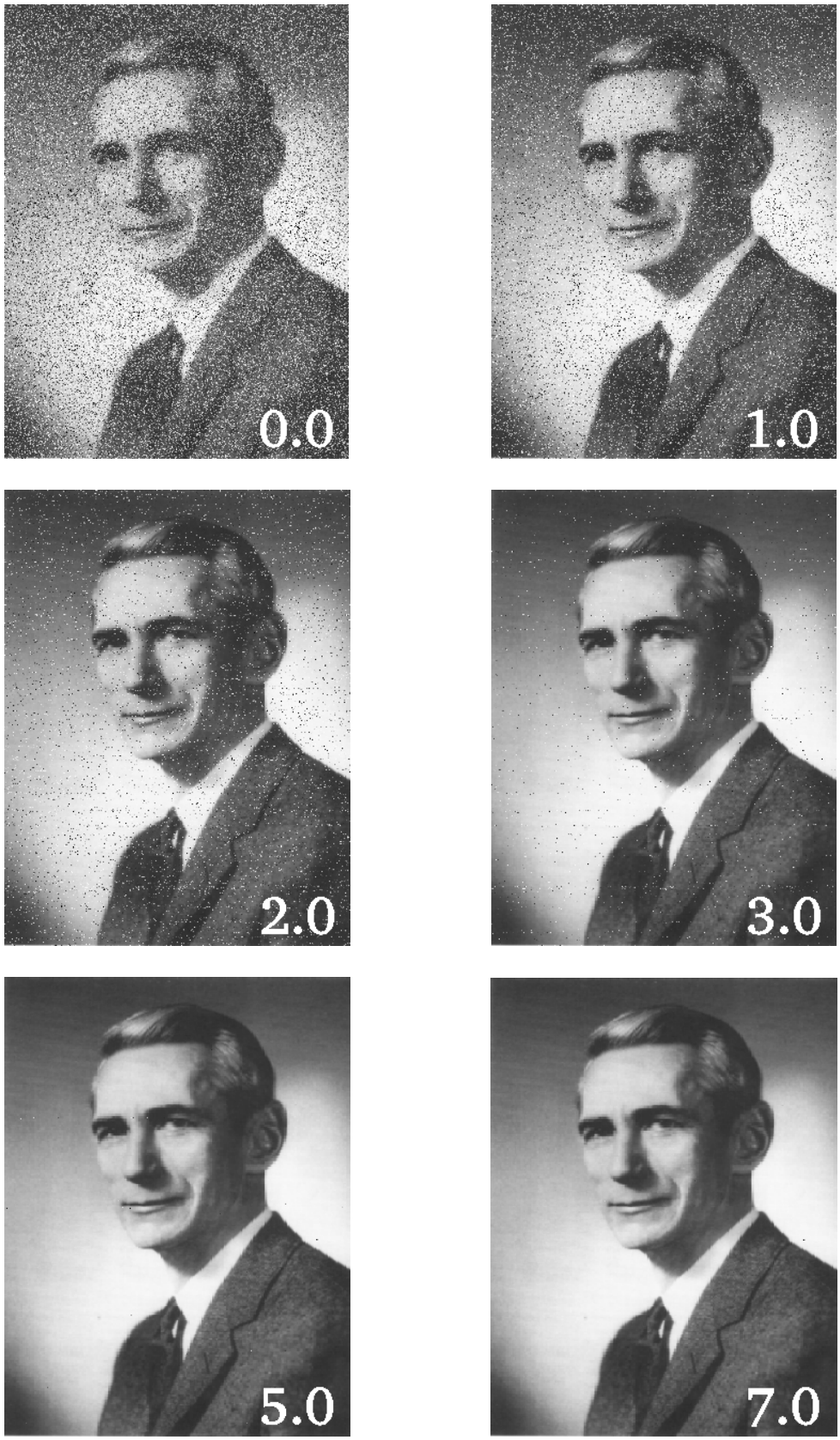
Na spodnjem primeru je prikazano delovanje turbo kode pri prenosu digitalizirane slike. Za poizkus je uporabljena slika formata 426x323 slikovnih elementov in 8 bitov za sivino. Prenos je izveden prek Gaussovega šumnega kanala pri razmerju E_b/N_0= 1.2 *dB*. Za kodiranje podatkov sta uporabljeni dve rekurzivni sistematični konvolucijski kodi RSC(37,21), med katere je vstavljen mešalnih bitov dolžine 4096 bitov. Kodno razmerje uporabljenega turbo kodirnika je *1/2*. Kodirnik je prikazan na sliki 21.

Iz vzorcev sprejetih digitalnih sliki, prikazanih na sliki 22, je mogoče opaziti rezultat po različnem številu iteracij turbo dekodiranja. Že s subjektivno oceno je mogoče določiti po kolkih iteracija, je dosežen dober rezultat. Število iteracije je na posamezni sliki označeno v spodnjem desnem kotu.

Uporabljena koda ima prag delovanja (razmerje E_b/N_0 pri katerem preide iz stanja z veliko napakami v stanje z zelo majhnim številom napak) pod izbranim razmerjem E_b/N_0=1.2 *dB*. Zaradi tega, kot je razvidno na slikovnih vzorcih, postopek dekodiranja hitro konvergira k rešitvi in možno je doseči rezultat praktično brez napak.



Slika 22: Konvolucijski turbo kodirnik uporabljen za prikaz primer



Slika 23: Rezultati dobljeni po: 0, 1, 2, 3, 5, 7 opravljenih dekodirnih iteracijah

8. Zaključek

V nalogi so na kratko predstavljene bločne in konvolucijske turbo kode, ter temeljni principi in ideje delovanja. Trenutno so to najbolj učinkovite kode, ki se med vsemi znanimi najbolj približajo Shannon-ovi teoretični meji prenosa prek šumnega kanala. Znanstveniki na področju kodiranja in digitalnega prenosa se strinjajo, da turbo kode delujejo na sami praktični meji (pri 0.5 *dB* E_b/N_0).

Na kratko so opisani postopki povezovanja bločnih in konvolucijskih kod (turbo kodiranje). Gre za postopke odvisnega povezovanja, kar pomeni da uporabljene kodirne postopka čim bolj sodelujejo med seboj, z namenom doseganja čim boljših rezultatov.

Opisana je logika, ki se uporablja bi turbo dekodiranju in povezava iz katere je taka logika izpeljana. Govor je o aposteriori verjetnosti – verjetnosti oddanega simbola na podlagi sprejetega. S pomočjo omenjene logike, so izpeljani in prikazani postopki za bločno in konvolucijsko dekodiranje.

Na preprostih, poenostavljenih primerih je prikazano delovanje bločnega in kovolucijskega dekodiranja. Nakazane so zmogljivosti vnaprejšnje detekcije in korekcije napak.

Sledi še kratka obravnava konvergence turbo dekodirnih postopkov. Na vse zadnje opisani postopki zahtevajo veliko število zahtevnih računskih operacij in se je smiselno vprašati ali postopek sploh konvergira h končni rešitvi. Za obravnavo problema je uporabljen EXIT diagram. To je metoda, ki nudi pregleden grafičen vpogled v dogajanje.

Na koncu je narejena še kratka primerjava in zmogljivost različnih praktičnih izvedb turbo kod ter kod iz katerih so le te sestavljene (odvisnost verjetnosti napak od razmerja E_b/N_0). Na praktičnem primeru prenosa slike je ilustrirano delovanje.

9. Literatura

[1] C. Berrou, A. Glavieux, P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes”, Proceedings of IEEE International Communications Conference, 1993.

[2] B. Sounto, J. Carlos, F. Ceresa, “Block turbo shemes using THC codes”.

[3] B. Sklar, “Digital communications: Fundamentals and Applications”, Prentice Hall PTR, 2000 New Jersey.

[4] X. Jaspar, L. Vandendorpe, “Performance and convergence analysis of joint source-channel turbo schemes with variable length codes”.

[5] F. Brannstrom, T. Aulin, L. Rasmussen, A. Grant, “ Convergence analysis of interative detectors for narrow-band multiple access”.

[6] I. Land, P. A. Hoeher, J. Huber, “Analytical derivation of EXIT charts for simple block codes and for LDPC codes using information combining”.