UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Turbo kode

SEMINARSKA NALOGA

Avtor: **Mitja PUGELJ**

Mentor: **prof. dr. Aleksandar JURIŠIĆ**

26. marec 2010

Kazalo

1 Uvod

Turbo kode so ene izmed kod za odkrivanje in odpravljanje napak. Z njimi želimo zagotoviti zanesljiv prenos sporočila po nezanesljivem mediju. Nezanesljiv medij je takšen medij, ki je pod vplivom šuma iz okolja in je zato sporočilo, ki ga nosi, izpostavljeno spreminjanju.

Pred pojavom turbo kod so se za prenos sporočil večinoma uporabljale konvolucijske kode ali serijsko povezane konvolucijske in bločne kode [15]. Leta 1993 pa je ekipa francoskih raziskovalcev [2] predstavila nov razred kod, sestavljenih iz vzporedno povezanih konvolucijskih kod. Turbo kode so naredile velik korak naprej v učinkovitosti, saj omogočajo zanesljiv prenos sporočila z oddajno močjo blizu Shannonove teoretične meje.

Cilj seminarske naloge je predstaviti turbo kode in njihovo uporabo. Začnemo s predstavitvijo ozadja kodiranja za namen popravljanja napak pri prenosu in konvolucijskih kod, ki so osnovni gradnik turbo kod. Podrobno si ogledamo pristop k kodiranju in odkodiranju. Zaključimo s pregledom vseh ključnih faktorjev, ki vplivajo na uspešnost turbo kod in s pregledom uporabe turbo kod v praksi.

2 Osnove

V tem poglavju predstavimo ozadje, ki ga potrebujemo za boljše razumevanje turbo kod.

2.1 Informacijska stopnja r

Zmožnost odkrivanja in odpravljanja napak naravno prihaja iz dodatnih informacij. Če označimo z m število bitov, ki jih želimo prenesti in z n število bitov, ki jih dejansko prenesemo po kanalu ($m - n$ je število redundantnih bitov), potem kvocient

$$r = \frac{m}{n}$$

označuje *informacijsko stopnjo*. Manjša kot je informacijska stopnja, več redundantnih bitov dodajamo sporočilo. Jasno je, da večje število dodatnih bitov pomeni zanesljivejše sporočanje. Navadno želimo prenašati karseda malo podatkov,

zato želimo pri ciljnih pričakovani napaki čim višjo informacijsko stopnjo kode.

BER BER (bit error ratio) je kvocient med številom napačno sprejetih bitov in številom vseh sprejetih bitov. Označuje velikost napake pri prenosu sporočila. Tukaj s sprejetimi biti mislimo na bite po končanem odkodiranju.

Shannononva meja Shannon je leta 1948 v članku *Mathematical Theory of Communication* [11] pokazal, da je mogoče prenesti sporočilo po vsakem kanalu s poljubno natančnostjo. Članek, ki je postal osnova teorije informacij, definira pojem kapacitete kanala C kot parametra, ki zajame vse lasnosti kanala. Kapaciteto definira kot:

$$C = \frac{1}{2}\log_2\left(1 + 2\frac{E_s}{N_o}\right).$$

V kolikor želimo uspešno prenesti sporočilo po kanalu, velja omejitev

$$r < C.$$

Informacijska stopnja je torej navzgor omejena s kapaciteto kanala, ki jo po Shannonu izračunamo iz količine E_b/N_o. E_b/N_o je v grobem mera za moč signala glede na moč šuma.

Preurejena enačba

$$\frac{E_b}{N_o} \geq \frac{1}{2r}(2^{2r} - 1)$$

nam ponuja tudi spodnjo mejo za moč signala glede na moč šuma (E_b/N_o) pri izbrani informacijski stopnji r. Učinkovitost turbo kod se kaže v tem, da je potrebna oddajna moč blizu (okoli 0.7dB) teoretični meji za izbran r.

2.2 Razmaknjenost kode

Hammingova razdalja med dvema besedama (nizoma bitov) je število mest na katerih se znaki (biti) razlikujejo. Razmaknjenost kode je najmanjša Hammingova razdalja med dvema kodnima beseda.

Razmaknjenost kode je tesno povezana s sposobnostjo kode za detekcijo in

popravljanj napak. Če sta dve zakodirani besedi blizu skupaj je ob napaki težje oceniti katera beseda je bila poslana. Želimo torej, da ima koda karseda veliko razmaknjenost.

2.3 Konvolucijske kode

Zgradba samih turbo kod ni preveč kompleksna, a da bi dobro razumeli njihovo delovanje, potrebujemo razumevanje njenih gradnikov. Zato bomo uvodoma najprej predstavili delovanje konvolucijskih kod, ki so pogosto osnovni gradnik turbo kod. Več o konvolucijskih kodah in pa tudi o kodiranju z namenom detekcije in odpravljanja napak v splošnem si lahko bralec prebere v [9].

2.3.1 Nekaj oznak

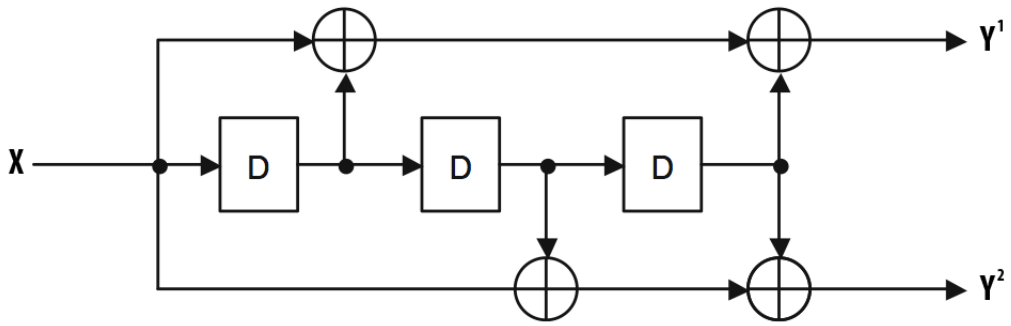
Z $X = X_1, X_2, X_3, ...$ označimo vhodno zaporedje bitov v kodirno metodo. Vhodno zaporedje je sestavljeno iz bitov, ki jih želimo prenesti po kanalu. Z Y^i označimo i-ti to zaporedje izhodnih bitov $Y_1^i, Y_2^i, Y_3^i, ...$ Vsa Y^i zaporedja so nato prenešena po kanalu in sprejeta na prejemnikovi strani. Zaradi prisotnosti šuma so prejeta zaporedja Y^i spremenjena glede na naravo šuma. V primeru dodatnega belega Gaussovega šuma sprejemnik sprejme vrednosti Z^i:

$$Z_k^i = (2 * Y_k^i - 1) + R_k^i,$$

kjer je R_k^i od Y_k^i neodvisna normalno porazdeljena slučajna spremenljivka s srednjo vrednostjo $\mu = 0$ in varianco σ^2. Z_k^i je torej zvezna in ne več diskretna 0-1 spremenljivka. V primeru ničnega šuma, je vrednost $Z_k^i = 1$, ko je $d_k = 1$, in $Z_k^i = -1$, ko je $d_k = 0$.

2.3.2 Konvolucijska koda

Poglejmo si primer preproste konvolucijske kode na sliki 1. Z X_k je označen k-ti vhodni bit, ki ga želimo prenesti, Y_k^1 in Y_k^2 sta k-ta izhodna bita, imamo torej dva izhodna bita za vsak vhodni bit. Z `D` so označeni registri, ki s prihodom novega



Slika 1: Zgradba konvolucijske kode.

bita, privzamejo vrednost predhodnega registra. Izhoda kode Y_k^1 in Y_k^2 sta binarni vsoti bitov izbranih registrov.

Označimo število registrov z v. Stanje kode sestavjajo vrednosti v v registrih in je torej odvisno od trenutnega vhodnega bita in $v - 1$ predhodnih bitov.

Za kodo na sliki imamo za vsak vhodni bit, dva izhodna. Informacijska stopnja podane kode je torej 1/2. Seveda lahko z dodatnimi izhodi dosežemo tudi nižje informacijske stopnje: 1/3, 1/4,..

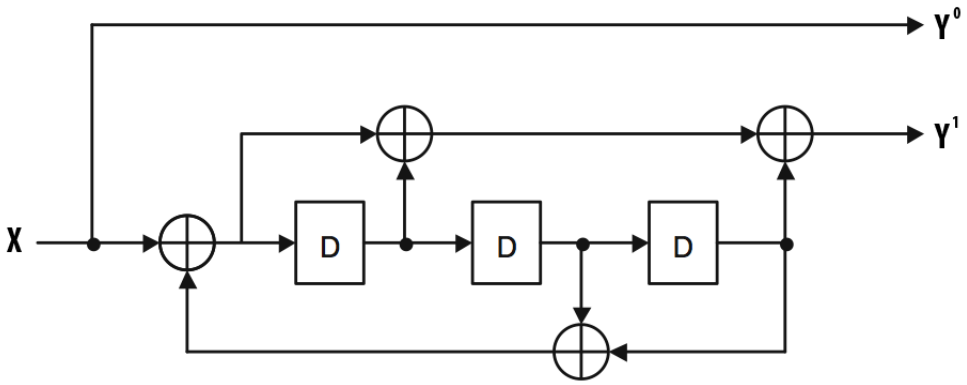
2.3.3 Rekurzivna sistematična konvolucijska koda

Slabost konvolucijske kode na sliki 2 je, da v izhodnem zaporedju bitov ni bitov, ki bi neposredno odražali vhodno informacijo. To lahko popravimo tako, da enega od izhodov predstavlja kar nespremenjeni vhod,

$$Y^1 = X,$$

drugi izhodi pa ostanejo, kot v prejšnjem primeru, definirani z vsoto izbranih registrov po modulu 2. Takšnim kodam pravimo sistematične, ker ločijo podatke, ki jih želimo prenesti, od redundantnih podatkov za namen detekcije in odprave napak.

Konvolucijske kode lahko modificiramo tudi tako, da enega od paritetnih izhodov preusmerimo nazaj na vhod. Izhodni (paritetni) biti so tako odvisni od vseh



Slika 2: Zgradba sistematične rekurzivne konvolucijske kode.

predhodnih bitov in ne samo od zadnjih $v - 1$ bitov.

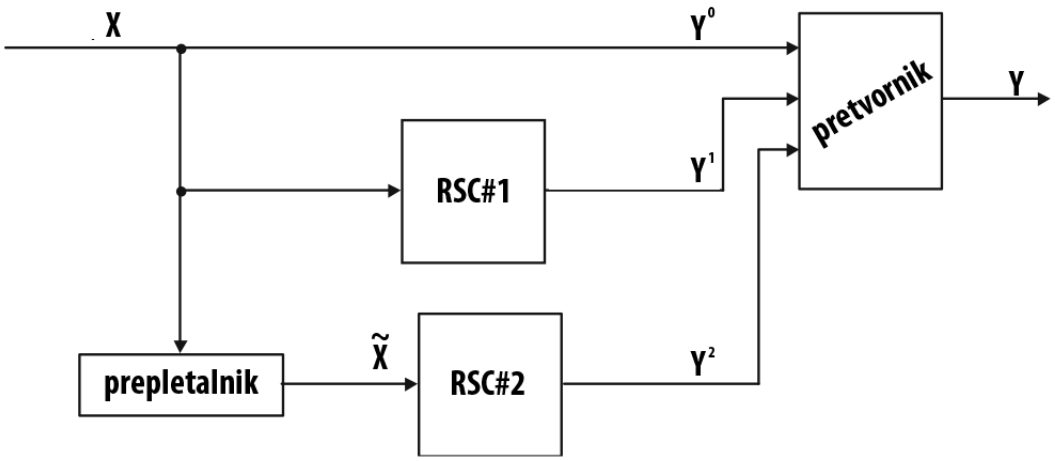
Če združimo obe spremembi, dobimo rekurzivno sistematično konvolucijsko (RSC, recursive systematic convolutional) kodo. Primer kode je na sliki 2

3 Zgradba turbo kode

3.1 Kodiranje

Turbo koda je sestavljena iz dveh vzporedni RSC kod. Čeprav so možne tudi izvedbe z več kodami in celo poljubnim tipom kod za odpravljanje napak, se bomo osredotočili na uporabo dveh RSC kod, kot je bilo to prvotno predstavljeno v [2] in kot se najpogosteje tudi uporablja. Poglejmo si primer turbo kode na sliki 3.

- Vhodni bit X_k je takoj preslikan v sistematični izhod Y^0.
- Zgornja RSC prejme nespremenjeno zaporedje bitov in ga zakodira. Sistematični izhod zavrže, paritetni izhod pa predstavlja Y^1 izhod turbo kode.
- Spodnja RSC prejme iste bite kot zgornja a v drugačnem vrstnem redu - psevdo permutirane v t.i. prepletalniku (interleaverju). Več o prepletalniku



Slika 3: Zgradba turbo kode.

v 3.2. Tako kot zgornja koda, tudi spodnja zavrže sistematični izhod, pariteni pa tu predstavlja Y^2 izhod turbo kode.

Vsi trije izhodi Y^0, Y^1 in Y^2 so v pretvorniku združeni v serijsko zaporedje, ki ga nato posredujemo po kanalu. Več o pretvorniku v razdelku 3.5.

3.2 Prepletalnik

Naloga prepletalnika je permutacija zaporedja bitov. Čeprav gre za navidez preprosto operacijo, je ravno prepletalnik ključen razlog za uspeh turbo kod. Permutacija bitov se zagotovi različen vhod v zgornjo in spodnjo kodo, čeprav gre v osnovi za iste podatke. Prav prepletalnik je tisti, ki v kodo vnese nekakšno naključnost, ki je zaželena pri kodiranju in hkrati ohranja strukturiranost.

Uspešnost kode namreč ni odvisna zgolj od razmaknjenosti, temveč tudi od števila besed, ki imajo minimalno hammingtonovo razdaljo do drugih besed blizu razmaknjenosti kode. Želimo, da je takšnih besed čim manj.

Različno vhodno zaporedje, kljub istim podatkom zagotavlja majhno verjetnost, da bi za neko vhodno zaporedje obe kodi zakodirali v kodno besedo z majhno razdaljo do drugih besed. Tako se poveča razmaknjenost kode in drastično zmanjša množica besed, ki so si blizu.

Prepletalnik seveda ne permutira celotnega (neskončnega) zaporedja bitov temveč samo določen del sporočila - blok. Turbo koda zato postane **bločna koda**. Kot bomo videli je prenos sporočila v blokih nujen tudi zaradi pristopa k odkodiranju.

Način permutiranja mora biti znan tako kodirniku kot odkodirniku in je zato vnaprej določen. Za manjše velikosti blokov lahko vnaprej izračunamo naključno mapiranje bitov in shranimo pare (star položaj, nov položaj) v tabelo. To tabelo nato uporabljamo pri vseh prenosih. Pri večjih blokih postane tabela velika, pa tudi izračun dobre naključne permutacije postane zamuden.

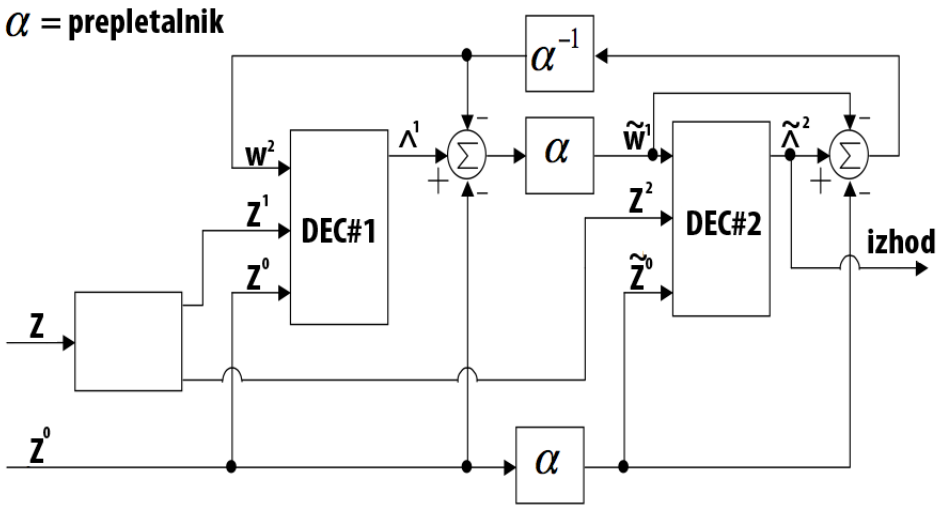
V tem primeru lahko lahko uporabimo turbo-prepletalnik (turbo-interleaver), ki deluje na velikih blokih (velikost bloka N, $N = 2^n \times 2^n$, $n \geq 3$), vrača dobre rezultate in ne shranjuje mapiranja v tabelo. Uporablja enostaven algoritem, ki določa preslikovo med starim in novim položajem bita v bloku [8]. Za vsak bit, ki se nahaja znotraj bloka v vrstici i in stolpcu j, sta novi koordinati i' in j' določeni z naslednjimi pravili:

- $i' = (2^{n-1} + 1)(i + j) \pmod{2^n}$
- $\zeta = i + j \pmod 8$
- $j' = (j + 1)(\zeta) - 1 \pmod{2^n}$
- $P(\zeta) = \{17, 37, 19, 29, 41, 23, 13, 7\}; \quad 0 \leq \zeta \leq 7$

3.3 Odkodiranje

Odkodirnik turbo kod je sestavljen iz dveh odkodirnikov DEC_1 in DEC_2, ki skrbita za odkodiranje RSC kod v kodirniku turbo kode. V primeru večih RSC kod na strani pošiljatelja, bi imel odkodirnik turbo kode več primernih odkodirnikov. Delovanje DEC_1 in DEC_2 bomo podrobneje opisali v 3.3.1, najprej pa si poglejmo kako poteka globalni proces odkodiranja.

Ena od novosti, ki so jo turbo kode prinesle v svet kodiranja, je iterativno odkodiranje. Ker se odkodirnika ukvarjata z istimi podatki (le drugače zakodiranimi), si lahko pomagata pri tvorjenju skupne ocene poslanega sporočila. DEC_1 tako pomaga DEC_2, nato DEC_2 posreduje dodatno informacijo DEC_1, ki ponovno posreduje dodatno informacijo DEC_2, itd. Zaradi narave procesa, dobimo na vsakem koraku boljšo oceno za poslano sporočilo. Odkodirnik je prikazan na sliki 4.



Slika 4: Zgradba odkodirnika turbo kode.

Najprej opišemo odkodirni proces v splošnem, pozneje pa predstavimo prilagojen odkodirnik konvolucijskih kod.

Odkodirnik DEC_1 ima 3 vhode:

- Z^0 niz sistematičnih bitov
- Z^1 niz paritetnih bitov
- w^2 zunanja dodatna informacija

in podobno DEC_2:

- Z^0 niz sistematičnih bitov
- Z^2 niz paritetnih bitov
- w^1 zunanja dodatna informacija

Definirajmo LLR (log likehood ratio), logaritemsko razmerje podobnosti Λ_k^1, Λ_k^2:

$$\Lambda_k^1 = log\frac{P[d_k = 1|Z^0, Z^1, w^2]}{P[d_k = 0|Z^0, Z^1, w^2]} \tag{1}$$

$$\Lambda_k^2 = log\frac{P[d_k = 1|Z^0, Z^2, w^1]}{P[d_k = 0|Z^0, Z^2, w^1]} \tag{2}$$

DEC_i vrne Λ_k^i, razmerje med verjetnostjo, da je zakodiran bit d_k enak 1 in verjetnostjo, da je enak 0, pri znanem sistematičnem in paritetnem vhodu ter zunanji informaciji w^i.

Zunanji informaciji w^1, w^2 sta zaporedoma lastni odkodirnikoma DEC_1 in DEC_2 ter ju DEC_2 in DEC_1 zaporedoma ne moreta dobiti iz sistematičnega vhoda in lastnega paritetnega vhoda. w torej služi izmenjavi informacij med odkodirnikoma. Pomembno je, da odkodirnik posreduje drugemu odkodirniku le sebi lastno, unikatno informacijo, sicer lahko postane odkodirni proces nestabilen. Zato w ni preprosto enak LLR oceni, temveč od nje odštejemo del vhoda v odkodirnik:

$$w^1 = \Lambda^1 - Z^0 - w^2 \tag{3}$$

$$w^2 = \Lambda^2 - Z^0 - w^1 \tag{4}$$

Enačbi veljata za celotno zaporedje bitov v bloku, ki ga odkodiramo, zato opuščamo $_k$ notacijo. Odkodiranje se začne s sprejemom bloka sistematičnih Z^0 in paritetnih bitov. Paritetni biti se razdelijo v dva niza Z^1, Z^2 v skladu z združevanjem v pretvorniku na strani pošiljatelja. Z^0 in Z^1 sta vhod v DEC_1 (w^2 ne nosi še nobene informacije), ki tvori oceno Λ_k^1 vrednosti bita d_k za vsak k. Iz enačbe (3) dobimo w^1, ki ga nato posredujemo v DEC_2 kot dodatno informacijo o prenešenem sporočilu. Zaradi uporabe prepletalnika v kodiranju, ga moramo ustrezno uporabiti tudi v odkodiranju. Bite preslikane s prepletalnikom označimo z $\tilde{}$.

Sedaj imamo pripravljen vhod v DEC_2: $\tilde{Z^0}$, Z^2 in $\tilde{w_1}$. DEC_2 vrne LLR oceno $\tilde{\Lambda^2}$ na podlagi katere iz enačbe (4) dobimo w^2. Tu se odločimo, če smo z oceno zadovoljni ali pa w^2 posredujemo DEC_1 in opravimo še eno iteracijo odkodiranja. Ko smo z oceno zadovoljni, $\tilde{\Lambda^2}$ usmerimo na izhod odkodirnika.

Kot smo omenili v 2.3.1, dobi sprejemnik na vhod zvezno količino za vsak bit v zaporedju. Ta zvezna vrednost predstavlja verjetnost, da je vrednost dejanskega poslanega bita 1 ali 0. Pravimo, da dobi odkodirnik na vhod mehke vrednosti, kot nasprotje trdih, diskretnih vrednosti.

Tudi tekom odkodirnega postopka celoten čas operiramo na mehkih vrednostih. Vsi vhodi v DEC^1 in DEC^2 in prav tako izhodna LLR ocena so mehke ocene za vrednost bita. Sprejemnik na koncu odkodirnega postopka seveda pričakuje odločitev o podatkih, zato izhod odkodirnega postopka filtriramo z enostavnim

filtrom:

$$d_k = \begin{cases} 1, & \text{če} \quad \Lambda^2 > 0 \\ 0 & \text{sicer.} \end{cases} \tag{5}$$

V odkodirnem postopku imamo še vedno eno neznanko in sicer zgradbo odkodirnikov DEC^1 in DEC^2. Kot smo že omenili, se morata odkodirnika prilegati uporabljenima kodirnima funkcijama na strani pošiljatelja. Če uporabljamo RSC kodo za kodiranje, potem moramo v DEC^1 oz DEC^2 uporabiti enega od algoritmov za odkodiranje konvolucijskih kod, npr. Viterbijev algoritem ali modificiran BAHLov algoritem. Bahlov algoritem je optimalna [13] metoda za kode, ki minimizirajo napako na bitu, torej tudi za turbo kode. Po drugi strani Viterbijev algoritem [2] minimizira napako na celotnem poslanem nizu. Omenili smo tudi, da mora algoritem imeti sposobnost mehkega odločaja - kot vhod lahko torej sprejme in na izhod vrne mehke vrednosti. Zato Viterbijev algoritem ni primeren in ga je potrebno ustrezno spremeniti.

Njegova spremenjena različica SOVA (soft output Viterbi algorithm) ima [7] precej manjšo zahtevnost od omenjenega BAHLovega algoritma, a vrne nekoliko slabše rezultate. Pri izbiri odkodirnega postopka torej spet nihamo med časovno in prostorsko zahtevnostjo in želeno kvaliteto odkodiranja.

Bralec si lahko več o samem odkodirnem postopku prebere v [2], [15], [13], [7], [1] in drugih virih, v naslednjem poglavju pa, povzeto po [12], predstavimo izpeljavo poenostavljenega Bahlovega algoritma.

3.3.1 Odkodirni algoritem

Za kodirnik z v spominskimi celicami označimo stanje kodirnika ob koraku k z S_k. S_k je v-terica odvisna samo od predhodnih izhodnih bitov. Izhodni bit $d_k = Y_k^0$ je povezan s prehodom iz stanja S_k v stanje S_{k+1}. Definirajmo še množico vseh 2^v možnih stanj m in jo imenujmo $\mathbf{B}$. Vhod v sprejemnik je zaporedje

$$K_1^N = (K_1, K_2, \ldots, K_k, \ldots, K_N),$$

označimo še podzaporeje

$$K_i^j = (K_i, K_{i+1}, \ldots, K_j),$$

$$\text{kjer je } \; K_k^k = K_k = \{Z_k^0, Z_k^1\}.$$

Definirajmo **razmerje podobnosti**, kot kvocient dveh pogojnih verjetnosti

$$\lambda_k = \frac{P(d_k = 0|K_1^n)}{P(d_k = 1|K_1^n)}. \tag{6}$$

$P(d_k = i|K_1^N), i = 0, 1$ je aposteriorna verjetnost za bit d_k. Odkodirnik lahko na podlagi λ_k določi najverjetnejši poslani bit $\hat{d}_k$ z enostavnim filtrom

$$\hat{d}_k = \begin{cases} 1 & ; \lambda_k \geq 1 \\ 0 & ; \lambda_k < 0 \end{cases}.$$

V nadaljevanju razvijemo enačbo (6) v obliko iz katere bo razviden način izračuna λ_k. Definirajmo

$$\lambda_k^{i,m} = P(d_k = i, S_k = m|K_1^N), \tag{7}$$

kjer je $m \in \mathbf{B}$. Če izraz seštejemo po vseh stanjih m dobimo ravno aposteriorno verjetnost za vrednost d_k.

$$P(d_k = i|K_1^N) = \sum_m \lambda_k^{i,m}, \quad i \in 0, 1.$$

Enačbo (6) lahko sedaj zapišemo z uporabo pravkar izpeljane vsote

$$\lambda_k = \frac{\sum_m \lambda_k^{0,m}}{\sum_m \lambda_k^{1,m}}.$$

Razširimo (7) z uporabo Bayesovega pravila

$$\begin{aligned}\lambda_k^{i,m} &= P(d_k = i, S_k = m|K_1^N) \\ &= \frac{P(d_k = i, S_k = m, K_1^N)}{P(K_1^N)} \\ &= \frac{P(K_1^{k-1}|d_k = i, S_k = m, K_k^N)P(K_{k+1}^N|d_k = i, S_k = m, K_k)P(d_k = i, S_k = m, K_k)}{P(K_1^N)}\end{aligned}$$

Pri prehodu iz druge na tretjo vrstico smo dogodek K_1^N razumeli kot presek dogodkov K_1^{k-1}, K_{k+1}^N in K_k in uporabili enačbo:

$$P(A \cap B \cap C) = P(B \cap C|A)P(A) \tag{8}$$
$$= P(C|B \cap A)P(B|A)P(A). \tag{9}$$

Definirajmo tri komponente $\alpha_k^m, \beta_{k+1}^{f(i,m)}, \delta_k^{i,m}$ s katerimi izrazimo $\lambda_k^{i,m}$. Pozneje bomo komponente preoblikovali na način, ki bo omogočal lažji izračun.

$$\alpha_k^m = P(K_1^{k-1}|d_k = i, S_k = m, K_k^N) \tag{10}$$

Pogoj $S_k = m$ fiksira stanje v katerem je bil kodirnik, ko je kodiral bit d_k. Zato lahko pogoj $d_k = i$ odstranimo iz enačbe. Če vzamemo v obzir fiksirano stanje S_k, potem nam zakodirani biti po stanju S_k, to so K_k^N, ne dajejo nobene dodatne informacije o bitih K_1^{k-1}. Tako se enačba poenostavi v:

$$\alpha_k^m = P(K_1^{k-1}|S_k = m). \tag{11}$$

Definirajmo še $\beta_{k+1}^{f(i,m)}$. Funkcija $f(i, m)$ je funkcija, ki vrne naslednje stanje glede na vhod i in dano stanje m.

$$\beta_{k+1}^{f(i,m)} = P(K_{k+1}^N|d_k = i, S_k = m, K_k) \tag{12}$$

S podobnim razmislekom kot pri α_k^m lahko tudi tu enačbo precej poenostavimo.

$$\beta_{k+1}^{f(i,m)} = P(K_{k+1}^N|S_{k+1} = f(i, m)) \tag{13}$$

Definirajmo še $\delta_k^{i,m}$:

$$\delta_k^{i,m} = P(d_k = i, S_k = m, K_k). \tag{14}$$

Tako lahko enačbo za $\lambda_k^{i,m}$ zapišemo kot:

$$\lambda_k^{i,m} = \frac{\alpha_k^m \delta_k^{i,m} \beta_{k+1}^{f(i,m)}}{P(K_1^N)}. \tag{15}$$

Želimo najti enostaven način za izračun verjetnosti $\alpha_k^m, \beta_{k+1}^{f(i,m)}$ in $\delta_k^{i,m}$. Pokažemo, da je moč α_k izračunati rekurzivno:

$$\begin{aligned}
\alpha_k^m &= P(K_1^{k-1}|S_k = m) \\
&= \sum_{\bar{m}\in\mathbf{B}} \sum_{j=0}^{1} P(d_{k-1} = j, S_{k-1} = \bar{m}, K_1^{k-1}|S_k = m) \\
&= \sum_{\bar{m}\in\mathbf{B}} \sum_{j=0}^{1} P(K_1^{k-2}|S_k = m, d_{k-1} = j, S_{k-1} = \bar{m}, K_{k-1}) P(d_{k-1} = j, S_{k-1} = \bar{m}, K_{k-1}|S_k = m) \\
&= \sum_{\bar{m}\in\mathbf{B}}^{1} P(K_1^{k-2}|S_{k-1} = h(j,m)) P(d_{k-1} = j, S_{k-1} = h(j,m), K_{k-1}) \\
&= \sum_{j=0}^{1} \alpha_{k-1}^{h(j,m)} \delta_{k-1}^{j,h(j,m)},
\end{aligned}$$

kjer je $h(j,m)$ predhodno stanje glede na trenutno stanje m in vhod j. Pri prehodu iz tretje v četrto vrstico upoštevamo, da stanje fiksirano stanje S_{k-1} določa bit $d_{k-1} = j$, naslednje stanje in pa naredi informacijo o K_{k-1} nepotrebno.

S podobnimi razmislek lahko tudi β_k^m preoblikujemo v rekurzivno enačbo

$$\begin{aligned}
\beta_k^m &= P(K_k^N | S_k = m) \\
&= \sum_{\bar{m} \in \mathbf{B}} \sum_{j=0}^{1} P(d_k = j, S_{k+1} = \bar{m}, K_k^N) | S_k = m) \\
&= \sum_{\bar{m} \in \mathbf{B}} \sum_{j=0}^{1} P(K_{k+1}^N | S_k = m, d_k = j, S_{k+1} = \bar{m}, K_k) P(d_k = j, S_{k+1} = \bar{m}, K_k | S_k = m) \\
&= \sum_{j=0}^{1} P(K_{k+1}^N | S_{k+1} = f(j,m)) P(d_k = j, S_k = m, K_k) \\
&= \sum_{j=0}^{1} \delta_k^{j,m} \beta_{k+1}^{f(j,m)}.
\end{aligned}$$

Algoritem najprej izračuna α_k^m po naraščajočih k, nato še β_k^m po padajočih k. Ob predpostavki, da blok začne in konča v stanju 0^v določimo, začetne vrednosti rekurzije:

$$\begin{aligned}
\alpha_1^0 &= 1 \\
\alpha_1^m &= 0 \quad \forall m \neq 0^v \\
\beta_{N+1}^0 &= 1 \\
\beta_{N+1}^m &= 0 \quad \forall m \neq 0^v.
\end{aligned}$$

V primeru, da blok konča v neničelnem stanju (neuspešen prenos celotnega bloka) pa je $\beta_{N+1}^m = 1$ za vsa možna stanja m. Verjetnost $\delta_k^{j,m}$ lahko izračunamo iz prehodne verjetnosti diskretnega kanala brez spomina in apriorne verjetnosti.

$$\delta_k^{i,m} = P(d_k = i, S_k = km, K_k) \quad \text{po def.} \tag{16}$$

Ponovno uporabimo enačbo (9)

$$= P(K_k | d_k = i, S_k = m) P(S_k = m | d_k = i) P(d_k = i). \tag{17}$$

Ker je stanje S_k neodvisno od nezakodiranega izhodnega bita v istem koraku d_k in ob predpostavki, da so vsa stanja enako verjetna, je verjetnost $P(S_k = m | d_k = i) =$

$\frac{1}{2^v}$. Definirajmo še $\zeta_k^i = P(d_k = i)$ in upoštevajmo $K_k = Z_k^0, Z_k^1$.

$$= \frac{P(Z_k^0|d_k = i, S_k = m)P(Z_k^1|d_k = i, S_k = m)\zeta_k^i}{2^v} \tag{18}$$

Za Gaussov aditivni beli kanal z $\mu = 0$ in varianco σ^2, dobimo

$$\delta_k^{i,m} = \frac{\zeta_k^i}{2^v\sqrt{2\pi}\sigma}\int e^{-\frac{\left(Z_k^0-(2i-1)\right)^2}{2\sigma^2}}dZ_k^0\ \frac{1}{\sqrt{2\pi}\sigma}\int e^{-\frac{\left(Z_k^1-(2c^{i,m}-1)\right)^2}{2\sigma^2}}dZ_k^1$$

kjer je χ_k konstanta, dZ_k^0 in dZ_k^1 zaporedoma differenciala Z_k^0 in Z_k^1, ter $c^{i,m}$ zakodiran bit glede na $d_k = i$ in $S_k = m$.

$$\delta_k^{i,m} = \frac{2\zeta_k^i}{2^v\sqrt{2\pi}\sigma}e^{-\frac{\left(Z_k^0-(2i-1)\right)^2}{2\sigma^2}}e^{-\frac{\left(Z_k^1-(2c^{i,m}-1)\right)^2}{2\sigma^2}} \tag{19}$$

$$= \frac{2\zeta_k^i}{2^v\sqrt{2\pi}\sigma}e^{\frac{-1}{2\sigma^2}((Z_k^0-(2i-1))^2+(Z_k^1-(2c^{i,m}-1))^2)} \tag{20}$$

$$= \frac{2\zeta_k^i}{2^v\sqrt{2\pi}\sigma}e^{\frac{-1}{2\sigma^2}((Z_k^0)^2-4Z_k^0i+2Z_k^0+\cancel{4i^2}-\cancel{4i}+(Z_k^1)^2-4Z_k^0c^{i,m}+2Z_k^1+\cancel{4(c^{i,m})^2}-\cancel{4c^{i,m}})} \tag{21}$$

$$= \frac{2\zeta_k^i}{2^v\sqrt{2\pi}\sigma}e^{\frac{-1}{2\sigma^2}((Z_k^0)^2-4Z_k^0i+2Z_k^0+(Z_k^1)^2-4Z_k^0c^{i,m}+2Z_k^1)} \tag{22}$$

$$= \frac{2\zeta_k^i}{2^v\sqrt{2\pi}\sigma}e^{\frac{-1}{2\sigma^2}((Z_k^0)^2+2Z_k^0+(Z_k^1)^2+2Z_k^1)}e^{\frac{2}{\sigma^2}(Z_k^0i+Z_k^0c^{i,m})} \tag{23}$$

$$= \chi_k\zeta_k^i\exp\left(L_c\left(Z_k^0i + Z_k^1c^{i,m}\right)\right), \tag{24}$$

kjer je $L_c = \frac{2}{\sigma^2}$, χ pa zajema vse ostale konstantne vrednosti.

Sedaj lahko napišemo λ_k, mehko oceno bita d_k kot

$$\lambda_k = \frac{\zeta_k^0}{\zeta_k^0}\exp\left(-L_cZ_k^0\right)\frac{\sum_m\alpha_k^m\exp\left(L_cZ_k^1c^{0,m}\right)\beta_{k+1}^{f(0,m)}}{\sum_m\alpha_k^m\exp\left(L_cZ_k^1c^{1,m}\right)\beta_{k+1}^{f(1,m)}}.$$

$$\lambda_k = \zeta_k\exp\left(-L_cZ_k^0\right)\zeta_k' \tag{25}$$

$\zeta_k = \frac{\zeta_k^0}{\zeta_k^1}$ je razmerje vhodnih apriornih verjetnosti, medtem, ko je ζ_k' izhodna zunanja informacija odkodirnika, ki smo jo v začetku razdelka označili z w^i za odko-

dirnik i.

3.3.2 Algoritem

Z definirami vrednostmi α, β in δ, lahko podamo oris algoritma, ki izračuna λ.

1. Začni pri $k = 1$ in izračunaj $\delta_i(K_k, m)$ za vse simbole - v našem primeru $i \in 0, 1$.

2. Nastavi $\beta_N^i(S_b^i(0)) = 1$ in $\beta_N^i(m) = 0$ za $m \neq 0$. Začni pri $k = N - 1$ in izračunaj $\beta_k^i(m)$.

3. Nastavi $\alpha_1^i(0) = \delta_i(K_1, 0)$ in $\alpha_1^i(m) = 0$ za $m \neq 0$. Začni pri $k = 2$ in izračunaj preostale vrednosti $\alpha_k^i(m)$.

4. Izračunaj λ_k in nato $\lambda = \sum_k \lambda_k$.

3.3.3 log-MAP

V prešnjem razdelku podani algoritem je znan tudi pod imenom MAP (maximum a aposteriori). Da bi zmanjšali kompleksnost strojne implementacije algoritma, ga logaritmiramo (logaritmiramo dobljene enačbe). S tem dobimo enačbi (1) in (2). Podrobnosti logaritemske transformacije si lahko bralec ogleda v [12].

3.4 Začetno in končno stanje

Kodiranje bloka vedno pričnemo z nekim fiksnim stanjem turbo kode - najbolj enostavno je, da vrednosti v vseh v registrih nastavimo na 0. Da bi bilo odkodiranje lažje, je potrebno, da sprejemniku ni znano samo začetno, temveč tudi končno stanje turbo kode [6].

Če uporabljamo nerekurzivno konvolucijsko kodo znotraj turbo kode, lahko stanje kode ponastavimo z enostavnim vnosom v-ih bitov vrednosti 0 na vhod. Postopek je pri RSC kodah zaradi povratne informacije kompleksnejši. Opazimo pa, da lahko zaradi binarnega seštevanja (oz. XOR operacije) med povratnim signalom in vhodom na vhod pošljemo enake bite, kot prihajajo iz povratnega

signala. S tem postavimo RSC kodo v začetno stanje v v korakih. Zaradi prepletalnika moramo postopek ponoviti za vsak kodo. Tako potrebujemo $2v$ bitov, če upoštevamo še prenešene paritetne bite, je vseh dodatnih bitov $4v$. Posledično se informacijska stopnja v primeru dveh RSC kodirnokov zmanjša z $1/3$ na

$$r = \frac{m}{3m + 4v}.$$

Pri večjih dolžinah bloka (m) postane zmanjšanje zanemarljivo.

3.5 Pretvornik

Osnovna naloga pretvornika je združiti več vzporednih signalov v enega zaporednega, ki ga nato prenesemo po kanalu. Vendar lahko v pretvorniku realiziramo še eno nalogo.

Turbo koda, ki smo jo opisovali tekom razdelka, je imela informacijsko stopnjo $1/3$. Omenili smo, da lahko z dodajanjem večih kodirnikov/odkodirnikov, dosežemo tudi nižje stopnje $1/4, 1/5, \ldots$. Kako pa doseči višje informacijske stopnje, npr. $2/3$?

Jasno je, da moramo za višjo informacijsko stopnjo prenesti manj bitov. To nalogo zaupamo pretvorniku. Pretvornik opušča nekatere paritetne bite na način, da imamo še vedno dovolj podatkov za odkodiranje sporočila in da hkati število prenešenih bitov ustreza ciljni informacijski stopnji.

Na strani prejemenika, moramo mesta, kjer smo bite opustili, nastaviti na 0, zato mora biti postopek opuščanja bitov znan tako pošiljatelju kot prejemniku in je kot večina parametrov turbo kod del protokola, ki ga uporabljamo.

4 Dejavniki za učinkovitost turbo kod

4.1 Prepletalnik

Velik donos k uspešnosti turbo kod prinese spretna uporaba prepletalnika, ki drastično poveča razmaknjenost turbo kode in omogoči iterativno odkodiranje.

Velikost bloka Zaradi uporabe prepletalnika, se koda spremeni v bločno, saj permutacijo opravljamo nad celotnim blokom bitov. Kvaliteta pseudo permutacije v prepletalniku je seveda odvisna tudi od velikosti bloka - večji bloki omogočajo boljše[1] permutacije. Boljše permutacije pa pomenijo večjo razmaknjenost kode, torej boljšo učinkovitost kode. Seveda bloka ne moremo poljubno povečati, saj ravno zaradi bločnega procesiranja večji bloki pomenijo večjo latenco pri kodiranju in odkodiranju, kar za navadno ni zaželeno. Velikost bloka je torej kompromis med učinkovitostjo v smislu odpravljanja napak in hitrostjo kode.

4.2 Informacijska stopnja kode

Kot pri ostalih kodah, na samo uspešno vpliva tudi informacijska stopnja kode. Razlogi za to so bili že omenjeni v 2.1.

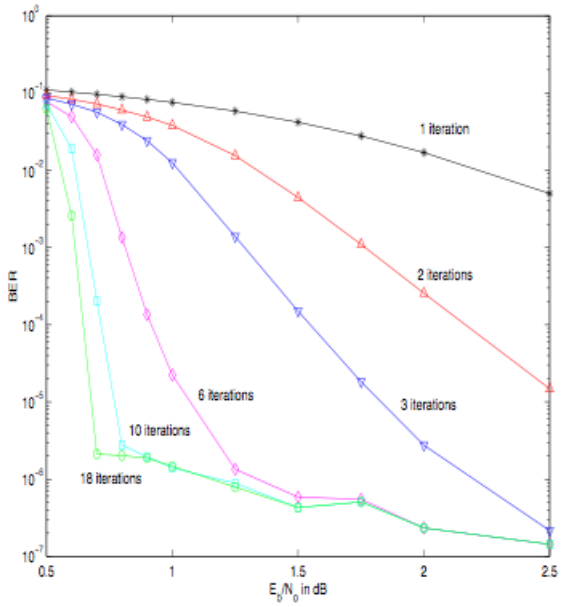
4.3 Iterativno odkodiranje

Iterativno odkodiranje je tako pomemben faktor, da daje turbo kodam njihovo ime. "Turbo"namreč ne prihaja iz njihove hitrosti ali uspešnost pri odpravljanju napak, temveč iz turbo efekta v iterativnem odkodiranju, ki spominja na ponovno uporabo izpuha v turbo motorju. Podobno kot pri motorju uporabimo izpušni plin, tu izrabimo izhodne informacije enega odkodirnika, da lahko drugi poda boljšo oceno, kot bi to storil samostojno.

Število iteracij Omenili smo že, da se kvaliteta ocene izboljšuje na vsaki iteraciji. Seveda pa obstaja spodnja meja in že po nekaj iteracijah je donos za ceno dodatnega porabljenega časa zanemarljiv. Pri izbiri števila iteracij torej nihamo med kvaliteto ocene in latenco, ki jo povzroči odkodiranje. Slika 5 prikazuje krivuljo napake turbo kode z $r = 1/2$, $k = 5$ in velikostjo bloka 64,536 bitov za različno število iteracij odkodiranja. Slika 5 izhaja iz [15].

Prekinitev odkodiranja Poleg vnaprej določenega števila iteracij, lahko poskušamo na podlagi rezultatov najti pravilni trenutek za prekinitev odkodiranja.

[1] boljše - stara in nova lokacija bita sta čim bolj narazen



Slika 5: Primerjava krivulje napake po različnem številu iteracij odkodiranja.

Najenostavnejši pogoj je lahko število različnih bitov, ki jih v isti iteraciji ocenita posamezna DEC_1 in DEC_2.

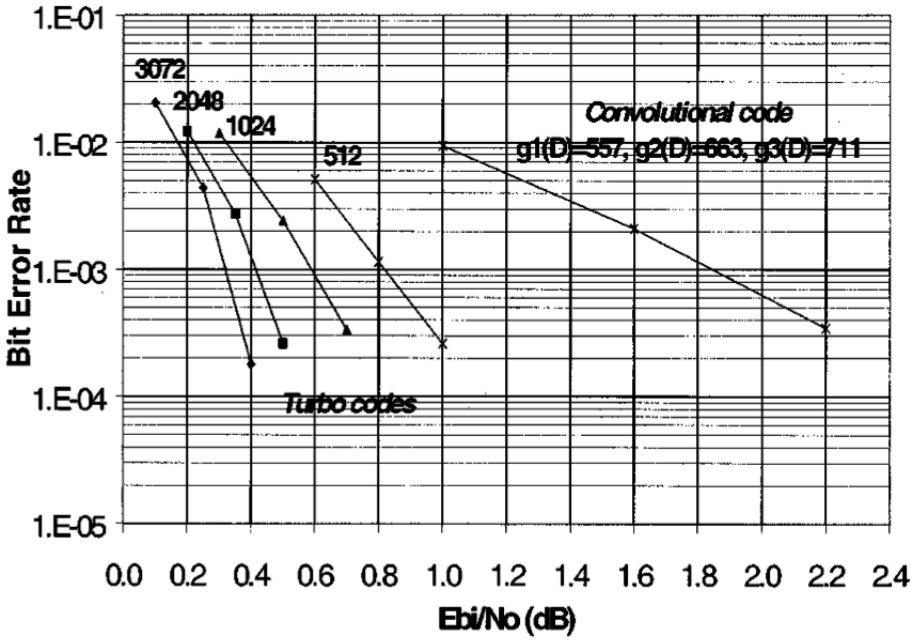
Poglejmo si še ustavitveni pogoj BHDA, ki ga predstavijo v [16]. BHDA sloni na BIP algoritmu, ki se uporablja za iskanje razlik v dveh binarnih nizih. BIP vrednosti izračunamo za iteracijo i in kodo dolžine n rekurzivno:

$$\begin{aligned} \mathrm{BIP}_m^{(i)}(j) &= 0, \quad j < 0 \\ \mathrm{BIP}_m^{(i)}(j) &= \mathrm{BIP}_m^{(i)}(j-1) + \hat{d}_k, \quad k \in [0..], \end{aligned}$$

kjer je $m = k \pmod{n}$, $j = \lceil k/n \rceil$. Če je $\mathrm{BIP}^{(i)} = \mathrm{BIP}^{(i-1)}$ za $i \geq 2$, potem iterativno odkodiranje ustavimo in vrnemo trenutno oceno.

4.4 RSC kode

Čeprav bi lahko v splošnem v zgradbi turbo kode uporabili poljubne kode za odpravljanje napak, se najpogosteje uporabljajo RSC, ker so zaradi rekurzivne strukture ključne za dobro učinkovitost turbo kod [10].



Slika 6: Primerjava (BER) učinkovitosti turbo kod in konvolucijski kod. Slika iz [4].

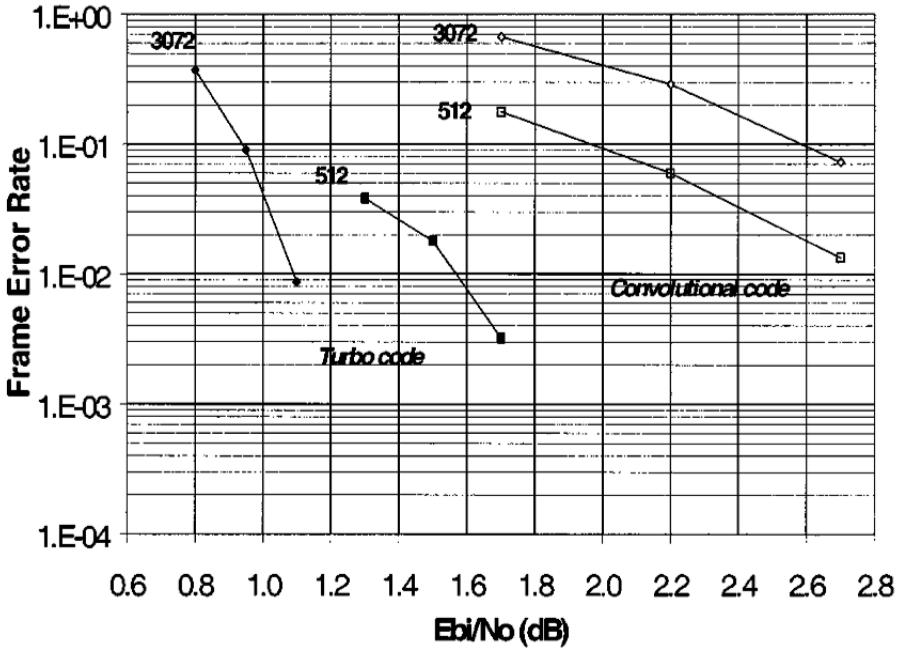
Število registrov Večanje števila registrov [8] izboljša učinkovitost turbo kode. Vendar kombinatorična kompleksnost odkodiranja hitro rase v odvisnosti od številom registrov, zato z registri ni smiselno pretiravati in jih običajno malo.

5 Uporaba

Turbo kode so zaradi svoje učinkovitosti našle pot v mnoga področja. Predvsem pa so pomembne v sistemih, ki zahtevajo majhno napako oz. majhno oddajno moč. Takšni sistemi so recimo komunikacija s sateliti, satelitske in brezžične ATM [14], brezžična omrežja [3] in s prihodom tretje generacije mobilne telefonije, tudi v mobilna omrežja.

5.1 UMTS in CDMA2000 mobilna omrežja

V mobilnih omrežjih je majhna oddajna moč uporabnikove naprave ključna, saj poveča zmogljivost celice [4] (več uporabnikov ima zagotovljeno boljšo kvaliteto



Slika 7: Primerjava (FER) učinkovitosti turbo kod in konvolucijski kod. Slika iz [4].

storitve). Zato je bilo smiselno dodati možnost uporabe turbo kod v UMTS in CDMA200 omrežjih.

Turbo kode so ena izmed možnih kod, ki jih lahko protokola uporabljata glede na razmere v omrežju. V primerjavi s konvolucijskimi kodami, so turbo kode opazno učinkovitejše (slika 6). V 3G omrežjih je zaradi samodejnega ponovnega prenosa okvirja v primeru napake smiselno vzeti v obzir napako celotnega okvirja (FER, frame erro rate) in ne BER. V slednjem primeru je izboljšanje učinkovitosti turbo kod napram konvolucijskim še večje [5]. Slednja primerjava je vidna na sliki 7.

6 Zaključek

V seminarski smo predstavili strukturo turbo kod, razloge za njihovo učinkovitost in uporabo. Pokažemo izjemno učinkovitost turbo kod in dejavnike za učinkovitost, ki predvsem izhajajo iz spretno sestavljenih rekurzivnih konvolucijskih kod in iz

inovativnega “turbo” iterativnega odkodiranja.

Turbo kode ohranjajo svojo pomembnost v okoljih, kjer je pomembno visoko učinkovito detektiranje in odpravljanje napak in iščejo svojo uporabo v vedno več aplikacijah.

Literatura

[1] Branka Vucetic, Yonghui Li, Lance C. Pérez and Fand Yiang. Recent Advances in Turbo Code Design and Theory, 2007.

[2] Claude Berrou, Alain Glavieux and Punya Thitimajshima. Near shannon limit error - correction coding and decoding : Turbo-codes, 1993.

[3] Jaão Martins, Alexandre Giulietti, Marius Strum. Performance comparison of convolutional and block turbo codes for WLAN applications, 2002.

[4] Lin-Nan Lee, A. Roger Hammons Jr, F.W. Sun, Mustafa Eroz. Application and standardization of turbo codes in third-generation high-speed wireless data services, 2000.

[5] Lin-Nan Lee, Mustafa Eroz, A. Roger Hammos Jr, K. Karimullah, F.W. Sun. Third generation mobile telephone systems and turbo codes, 1998.

[6] Matthew C. Valenti and Jian Sun. Turbo codes, Handbook of RF and Wireless Technologies, Chapter 12, 2002.

[7] Naotake Yamamoto, Tomoaki Ohtsuki. SOVA-base iterative decoding of turbo coded OOK and turbo coded BPMM.

[8] Raffi Achiba, Mehrnaz Mortazi and William Fizell. Turbo code performance and design trade-offs.

[9] Rejc, D. Konvolucijske kode. Master's thesis, Fakulteta za matematiko in fiziko, Univerza v Ljubljani, 2005.

[10] S. Benedeto and G Montorsi. Role of recursive convolutional codes in turbo codes.

[11] Shannon, C. E. A mathematical theory of communication, 1948.

[12] Steven S. Pietrobon. Implementation and Performance of a Turbo/MAP Decoder.

[13] Steven S. Pietrobon and Adrian S. Barbulescu. A simplification of the Modified Bahl Decoding Algorithm for Systematic Convolutional Codes, 1996.

[14] U. Vilaipornsawai and M.R. Soleymani. Turbo Codes for Satellite and Wireless ARM, 2001.

[15] Valenti, M. C. Turbo codes and iterative processing.

[16] Young-Sup Kim and Sung-Woong Ra. A Simple Efficient Stopping Criterion for Turbo Decoder, 2006.