Teorija informacij in sistemov, predavanje 8

Uroš Lotrič

Univerza v Ljubljani,
Fakulteta za računalništvo in informatiko

5.8.3 Polinom za preverjanje sodosti 1

- Zveza $g(p)h(p) = 0 \mod (p^n + 1)$ spominja na zvezo $\mathbf{GH}^T = \mathbf{0}$, ki velja za vse linearne bločne kode
- Izraz $x(p) = z(p)g(p)$ pomnožimo na obeh straneh s $h(p)$
- $x(p)h(p) = z(p)g(p)h(p) = z(p)\,0 = 0 \mod (p^n + 1)$
- Iz $x(p)h(p) = 0$ vidimo, da ima $h(p)$ podobno vlogo kot matrika $\mathbf{H}$
- Naredimo množenje bolj podrobno:
  - $$x(p)h(p) = \sum_{i=0}^{n-1} x_i p^i \sum_{l=0}^{k} h_l p^l = \sum_{i=0}^{n-1} x_i p^i \sum_{l=0}^{n-1} h_l p^l = 0$$
  - to smo lahko naredili, če vzamemo $h_{k+1} = \ldots = h_{n-1} = 0$. Vzemimo še $j = i + l$

$$x(p)h(p) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} x_i h_{j-i} p^j = 0$$

5.8.3 Polinom za preverjanje sodosti 2

- Za vsako stopnjo polinoma mora torej veljati $\sum_{i=0}^{n-1} x_i h_{j-i} = 0$
- V matrični obliki $(\mathbf{x}\mathbf{H}^T)^T = \mathbf{H}\mathbf{x}^T = \mathbf{0}$

$$\begin{pmatrix} h_0 & \dots & h_k & 0 & \dots & 0 & 0 \\ 0 & h_0 & \dots & h_k & 0 & \dots & 0 \\ \vdots & & & & & & \vdots \\ 0 & 0 & \dots & 0 & h_0 & \dots & h_k \end{pmatrix} \cdot \begin{pmatrix} x_{n-1} \\ \vdots \\ x_0 \end{pmatrix} = \mathbf{0}$$

- Primer: paritetni polinom za $g(p) = p^3 + p^2 + 1$

$$h(p) = p^7 + 1 : p^3 + p^2 + 1 = 1 + p^2 + p^3 + p^4$$

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

To je ciklični Hammingov kod, saj nastopajo vsi stolpci od 1 do $2^m - 1$ (gledano desetiško)

5.8.4 Kodiranje z množenjem

- Kodne zamenjave so večkratniki generatorskega polinoma.
- velja

$$x(p) = z(p) \cdot g(p) \mod (p^n + 1)$$

  $z(p)$ je polinom, ki ustreza podatkovnemu vektorju $\mathbf{z}$
- Kod, ki smo ga dobili z množenjem, ustreza generatorski matriki, ki ima v vrsticah koeficiente $p^{k-1}g(p), \ldots, pg(p), g(p)$, zato ni sistematičen.
- Primer: $g(p) = p^3 + p^2 + 1$, $\mathbf{z} = (1010)$, $\mathbf{x} = (1110010)$

5.8.4 Kodiranje z deljenjem 1

- Kodiranje na osnovi deljenja ustvari sistematičen cikličen kod
- Kodna zamenjava je zato sestavljena iz sporočila (podatkovnega bloka) in varnostnega bloka znakov, $\mathbf{x} = (\mathbf{z}|\mathbf{r})$
- Polinom podatkovnega bloka je $z(p) = z_{k-1}p^{k-1} + \ldots + z_1 p^1 + z_0 p^0$
- Če polinom pomnožimo s p^m, dobimo na desni m ničel $p^m z(p) = z_{k-1}p^{n-1} + \ldots + z_1 p^{m+1} + z_0 p^m$
- To ustreza bloku $\mathbf{z}$, premaknjenemu za m znakov v levo, $(z_{k-1} \ldots z_0 0 \ldots 0)$.
- Vzemimo, da je to naš nastavek. Če je to dobra kodna zamenjava, mora biti deljiva z generatorskim polinomom.
- V splošnem nastavek seveda ne bo deljiv, velja pa $p^m z(p) = g(p)t(p) + r(p)$, kjer je $t(p)$ količnik, $r(p)$ pa ostanek, s stopnjo manj on m

5.8.4 Kodiranje z deljenjem 2

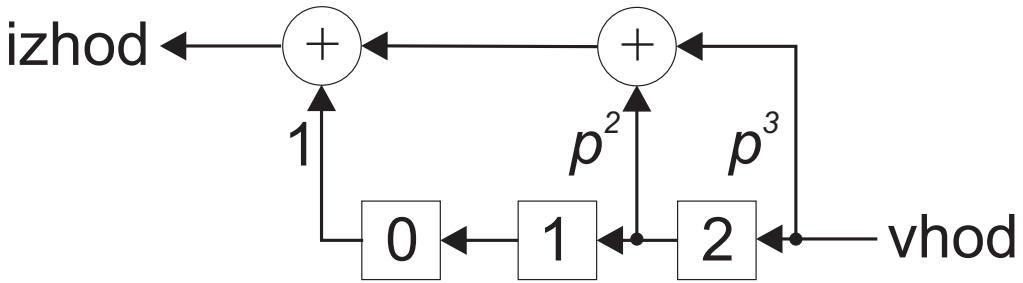
- Ostanek lahko zapišemo v obliki niza kot $(0 \dots 0 r_{m-1} \dots r_0)$
- Polinom $p^m z(p) + r(p) = g(p)t(p)$ je deljiv z $g(p)$ in je zato ustrezna kodna zamenjava.
- Kodno zamenjava tako dobimo, če ostanek deljenja z generatorskim polinomom prištejemo k osnovnemu nastavku, $(z_{k-1} \dots z_0 | r_{m-1} \dots r_0)$.
- Primer: $g(p) = p^3 + p^2 + 1$, $\mathbf{z} = (1010)$, $\mathbf{x} = (1010|001)$.

5.8.5 Strojna izvedba kodirnika 1

- Kodirnik lahko izvedemo kot enostaven končni avtomat, zasnovan kot premikalni register s povratnimi povezavami - kodirnik LFSR (ang. Linear Feedback Shift Register)
- Enostavna strojna izvedba je eden pomembnih razlogov za popularnost cikličnih kodov
- Uporabljeni trije tipi elementov: pomnilna celica tipa D, seštevalnik (XOR), množenje s konstanto (1=povezava, 0=ni povezave)
- Kodiranje na osnovi množenja, kodiranje na osnovi deljenja

5.8.5 Strojna izvedba kodirnika 2

- Kodiranje na osnovi množenja (za ogrevanje, to ni LFSR)
  - polinom $g(p) = p^3 + p^2 + 1$, niz $\mathbf{z} = (1010)$
  - vezje:

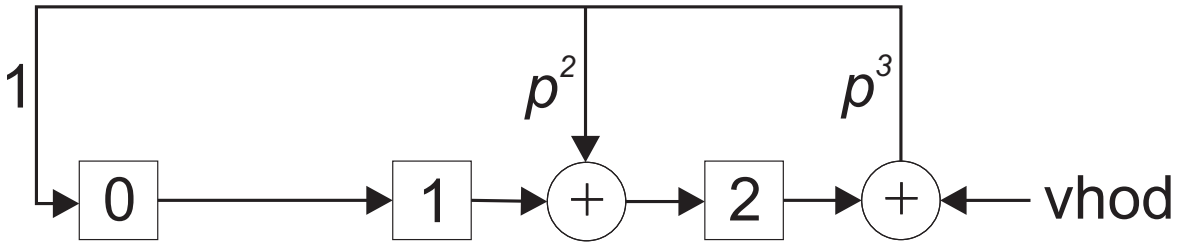


  - procesiranje:

korak	0	1	2	vhod	izhod
0	0	0	0	1	1
...					

5.8.5 Strojna izvedba kodirnika 3

- Kodiranje na osnovi deljenja:
  - polinom $g(p) = p^3 + p^2 + 1$, niz $\mathbf{z} = (1010)$
  - vezje:



  - procesiranje:

korak	0	1	2	vhod	p^3	izhdo
0	0	0	0	1	1	1
...						
4	**1**	**0**	**0**			0

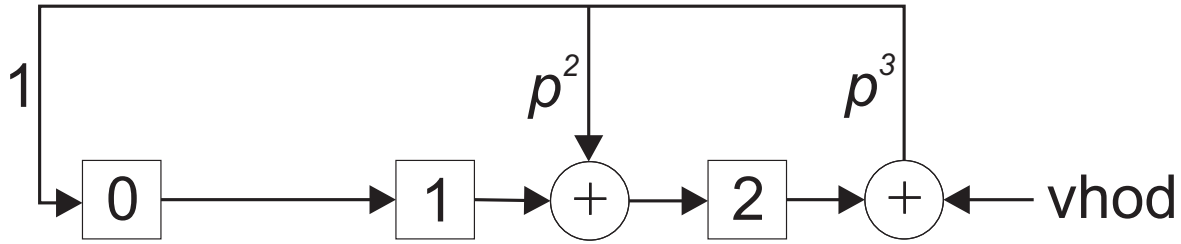
v prvih 4 korakih se na izhod pošiljajo kar vhodni znaki, v naslednjih 3 pa še vsebina pomnilnih celic od zadaj naprej

5.8.6 Dekodiranje

- Dekodiranje cikličnih kodov sloni na linearnih bločnih kodih.
- Vzemimo, da je pri prenosu prišlo do napake $\mathbf{y} = \mathbf{x} + \mathbf{e}$, ali polinomsko $y(p) = x(p) + e(p) = z(p)g(p) + e(p)$.
- Najprej izračunamo sindrom. Ekvivalent enačbe $\mathbf{s} = \mathbf{y}\mathbf{H}^T$ v polinomskem zapisu je $y(p) = q(p)g(p) + s(p)$ oziroma $s(p) = y(p) \mod g(p)$.
- Če je ostanek deljenja $y(p)$ z $g(p)$ različen od nič, je prišlo do napake.

5.8.6 Dekodiranje: odkrivanje napak

- Strojna izvedba dekodirnika z odkrivanjem napak
- Pri sistematičnih kodih uporabimo isto vezje
- Primer:
  - polinom $g(p) = p^3 + p^2 + 1$, niz $\mathbf{z} = (1010)$
  - vezje:



  - procesiranje:

korak	0	1	2	vhod	p^3
0	0	0	0	1	1
...					
7	**0/1**	**0/0**	**0/1**		

Če so na koncu v pomnilnih celicah same ničle, se je kodna zamenjava pravilno prenesla.

5.8.6 Dekodiranje: odpravljanje napak

- Iz $s(p) = y(p) \mod g(p)$ sledi, da je v primeru, ko je napaka na zadnjih m mestih, stopnja $e(p)$ manj kot m in velja kar $e(p) = s(p)$.
- Kaj pa ostale napake? Izkoristimo lahko cikličnost kodov.
- Naredimo trik: osnovno enačbo premaknemo za i mest, $p^i y(p) = p^i x(p) + p^i e(p)$
- Če najdemo pravi i, bo veljalo $p^i e(p) = s(p)$
- Kateri i je pravi? Igra verjetnosti pravi, da tisti, pri katerem bo imel $e(p)$ najmanj enic.
- To izkorišča algoritem za popravljanje napak na cikličnih kodih. Uporablja se zelo redko.

5.8.6 Dekodiranje: klasifikacija napak

- Napaki, ki se pojavi na izhodu odposlane kodne zamenjave neodvisno od morebitnih napak na sosednjih znakih, pravimo **posamična** ali **neodvisna** napaka.
- Do posamičnih napak pride zaradi motenj, ki so krajše od časa pošiljanja enega znaka.
- Povezanim napakam na več zaporednih znakih pravimo **izbruh**. Dolžina izbruha je število znakov med prvim in zadnjim napačno sprejetim znakom.
- Do izbruha pride, če je trajanje motenj daljše od časa pošiljanja enega znaka.
- **Ciklični kodi so posebej primerni za ugotavljanje izbruhov napak.**

5.8.7 Zmožnosti cikličnih kodov 1

- Odkrivanje napak s cikličnimi kodi $1 < \mathrm{st}(g(p)) < n$
  - Kod odkrije vsako posamično napako: $e(p) = p^i$ (pokaži)
  - Za določene generatorske polinome odkrije tudi dve posamični napaki do dolžine bloka $n = 2^m - 1$ (pokaži)
  - Odkrije poljubno število lihih napak, če $p + 1$ deli $g(p)$ (pokaži)
  - Odkrije vsak izbruh napak do dolžine m (pokaži)
  - Odkrije vse razen $2^{-(m-1)}$ izbruhov dolžine $m + 1$ (pokaži)
  - Kod odkrije tudi vse razen delež 2^{-m} izbruhov daljših od $m + 1$ (pokaži)
- Popravljanje napak s cikličnimi kodi $1 < \mathrm{st}(g(p)) < n$
  - Izračun sindroma
  - Ciklično prilagajanje sindroma prenesenemu bloku $\mathbf{y}$
  - Popravijo lahko do $e = \lfloor \frac{d-1}{2} \rfloor$ posamičnih napak, kjer je d Hammingova razdalja koda.
  - Popravijo lahko tudi izbruhe napak do dolžine $e = \lfloor \frac{m}{2} \rfloor$.

5.8.8 CRC 1

- CRC (ang. Cyclc Redundancy Check) temelji na cikličnih kodih.
- Dodatni triki, ki jih vključujejo standardi:
  - **registri v LFSR so na začetku nastavljeni na 1**; osnovni CRC ne loči sporočil, ki imajo različno število vodilnih ničel. Ta sprememba, ki je enaka negiranju prvih m bitov sporočila, to težavo odpravi.
  - **na koncu sporočila dodamo m-bitov** – odvisno od implementacije LFSR. Pri naši se to ne dela.
  - **operacija XOR na fiksnem vzorcu ostanka deljenja**; običajno je to kar negacija vseh bitov
  - **vrstni red bitov v bajtu** – nekateri serijski protokoli najprej oddajajo najmanj pomembne bite (najmanj pomembni bit ima najvišjo stopnjo polinoma)
  - **vrstni red bajtov** – pomnilniška organizacija računalnikov (little endian, big endian)

5.8.8 CRC 2

- Notacija CRC polinomov – biti označujejo prisotnost faktorja. Večkrat se izpušča eden od faktorjev - p^m ali 1.
- Primer CRC-16-ANSI: $g(p) = p^{16} + p^{15} + p^2 + 1$: z biti $= 1|1000|0000|0000|0101 =$ 0x18005 –> 0x8005
- Veliko uporabljani CRCji:

	CRC-16 (ANSI)	CRC-16 (CCITT)	CRC-32 (IEEE)
polinom	0x8005	0x1021	0x04C11DB7
registri	0x0000	0xFFFF	0xFFFFFFFF
XOR na CRC	0x0000	0x0000	0xFFFFFFFF
prezrcali bajt	da	ne	da
prezrcali CRC	da	ne	da
uporaba	USB	Bluetooth	Ethernet

5.8.8 CRC 3

- Omenjeni CRC-ji odkrijejo
  - enojne in dvojne posamične napake
  - liho število napak
  - vse izbruhe do dolžine 16/32
- Omenjeni CRCji ne odkrijejo:
  - $3{,}1 \cdot 10^{-5}/4{,}6 \cdot 10^{-10}$ 17/33-bitnih izbruhov
  - $1{,}5 \cdot 10^{-5}/2{,}3 \cdot 10^{-10}$ izbruhov dolžine 18/34 ali več
- Polinomi, uporabljeni v standardih izpred 30-50 let, niso najboljši. Pred 10 leti so z brute-force iskanjem našli polinome, ki dajo boljšo Hammingovo razdaljo koda.
- Enega od predlaganih polinomov uporablja naslednik TCP (Stream Control Transmission Protocol).
- Ciklični kodi so odlični za detekcijo napak. Za popravljanje napak danes obstajajo boljši kodi.

5.9 Prepletanje 1

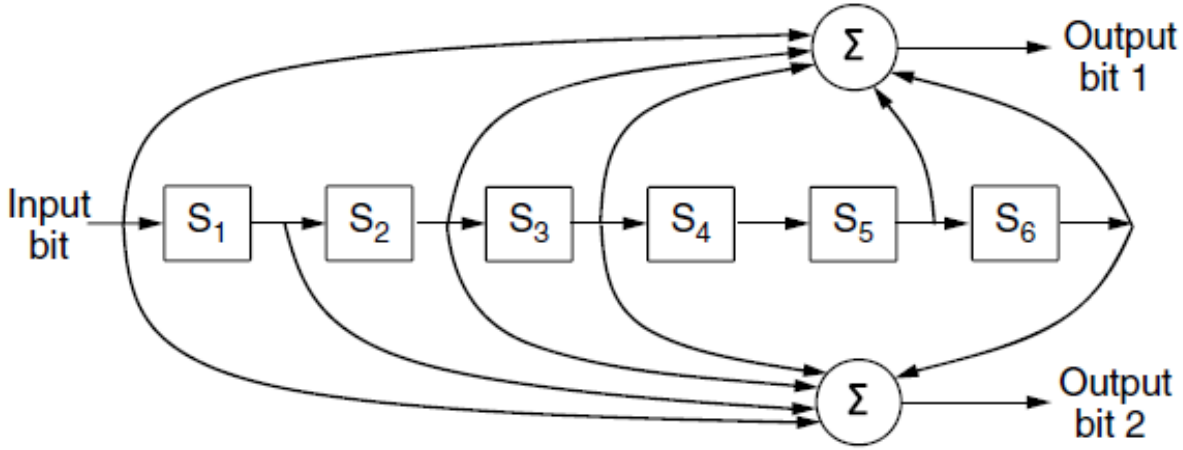
- Motnje so mnogokrat v obliki izbruhov. V takih primerih pride na določenih kodnih zamenjavah do velikega števila napak, na drugih pa napak ni.
- S prepletanjem bitov se da napake porazdeliti med več kodnih zamenjav.
- Enostavna rešitev: kodne zamenjave v kodirnik vpisujemo vrstico po vrstico, oddaja pa jih stolpec po stolpec. Obratno je na strani dekodirnika
- Načeloma je vzorec skoraj naključen. Matriko prepletanja poznata kodirnik in dekodirnik.
- Enostavna rešitev z zakasnitvijo: izmenično signali potujejo gor/dol, ena veja je zakasnjena.
- Dejanske rešitve bolj kompleksne: več vej, zakasnitve tudi do 20.

5.10 Konvolucijski kodi 1

- Primerni za popravljanje napak.
- Kovolucijske kode generiramo z linearnimi premikalnimi registri, ki so sestavljeni iz pomnilnih celic D in vrat XOR.
- Gre za nelinearne kode.
- Kovolucijski kodi so v bistvu avtomati stanj: izhod je odvisen od trenutnega stanja in vhoda - kodirnik ima spomin!
- Število pomnilnih bitov, od katerih je odvisen izhod, določa omejevalno dolžino koda (constraint length). Omejevalna dolžina koda je običajno za eno večja od števila pomnilnih bitov.
- Uporaba: GSM, komunikacija s sateliti, brezžična omrežja 802.11

5.10 Konvolucijski kodi 2

- Primer: NASA (Voyager 1977, zdaj 802.11), omejevalna dolžina 7, hitrost koda 1/2, kod ni sistematičen

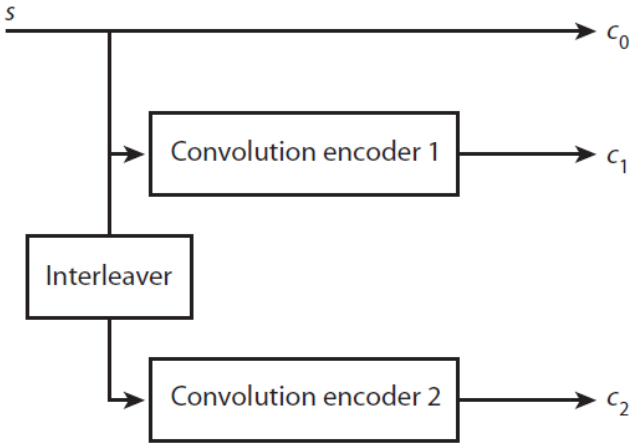


Primer: vhod: (111), notranje stanje: (100000), (110000), (111000), izhod: (11|10|01)

- Dekodiranje: kompleksno, za vsak korak in vsako notranje stanje ugotavlja kakšna vhodna kombinacija bi pripeljala do želenega zaporedja z najmanj napakami
- Dela lahko z verjetnostmi. Super v primerih, ko v kanalu logične vrednosti niso točno določene (0.9 V je zelo verjetno 1 V)

5.10 Turbo kodi

- Primerni za popravljanje napak.
- Zelo popularni pred 2000.
- Zgrajeni so tako, da posamezen znak vpliva na kodno zamenjavo na zelo širokem območju.
- Dekodiranje je zelo kompleksno, dostikrat vredno truda.
- Kombinacija prepletanja in kovolucijskih kodov (hitrost koda je 1/3)



- Dekodiranje: gradnja približnega sporočila, iterativno izboljševanje z večkratnim pošiljanjem popravljenega sporočila skozi kodirnik.

5.11 Kodi LDPC

- Kodi LDPC (ang. Low Density Parity Check) so linearni bločni kodi.
- Prvič omenjeni 1962, pozabljeni do 1995 zaradi kompleksnosti.
- Primerni za popravljanje napak.
- Vsak bit v kodni zamenjavi je zgrajen iz majhnega števila podatkovnih bitov. Kodi imajo zato zelo redko matriko za preverjanje sodosti (malo enic).
- Dekodiranje poteka z algoritmom, ki iterativno izboljšuje rešitev, da najde najustreznejšo kodno zamenjavo.
- So super za velike bloke podatkov, hitrost je blizu Shannonovi kapaciteti, zelo majhen delež neodkritih napak.
- Najdemo jih v 802.11, 10 Gbps Ethernet

5.11 Reed-Solomovi kodi 1

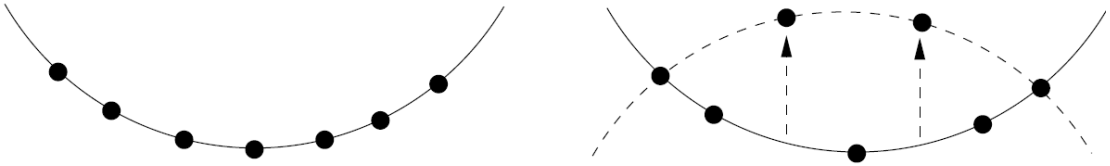
- Posplošitev linearnih cikličnih kodov: namesto posameznih bitov obravnavajo meta-znake sestavljene iz q osnovnih znakov
- Zelo kompleksna matematika Galousovih obsegov.
- So linearni, ciklični, lahko so sistematični.
- Kodne zamenjave so dolge $2^q - 1$ meta znakov.
- Trenutno najbolj vroči kodi.
- Uporaba: DSL (digital subscription line), satelitska komunikacija, CD, DVD, BlueRay

5.11 Reed-Solomovi kodi 2

- Kodiranje poteka tako, da skozi točke (meta-znake) potegnemo najboljši polinom. Polinom stopnje m je popolnoma podan z $m+1$ točko.

(a) Line (b) Parabola (c) Cubic

- Dodatne točke na krivulji (varnostni biti) so redundantne, kar lahko izkoristimo pri dekodiranju s popravljanjem napak.



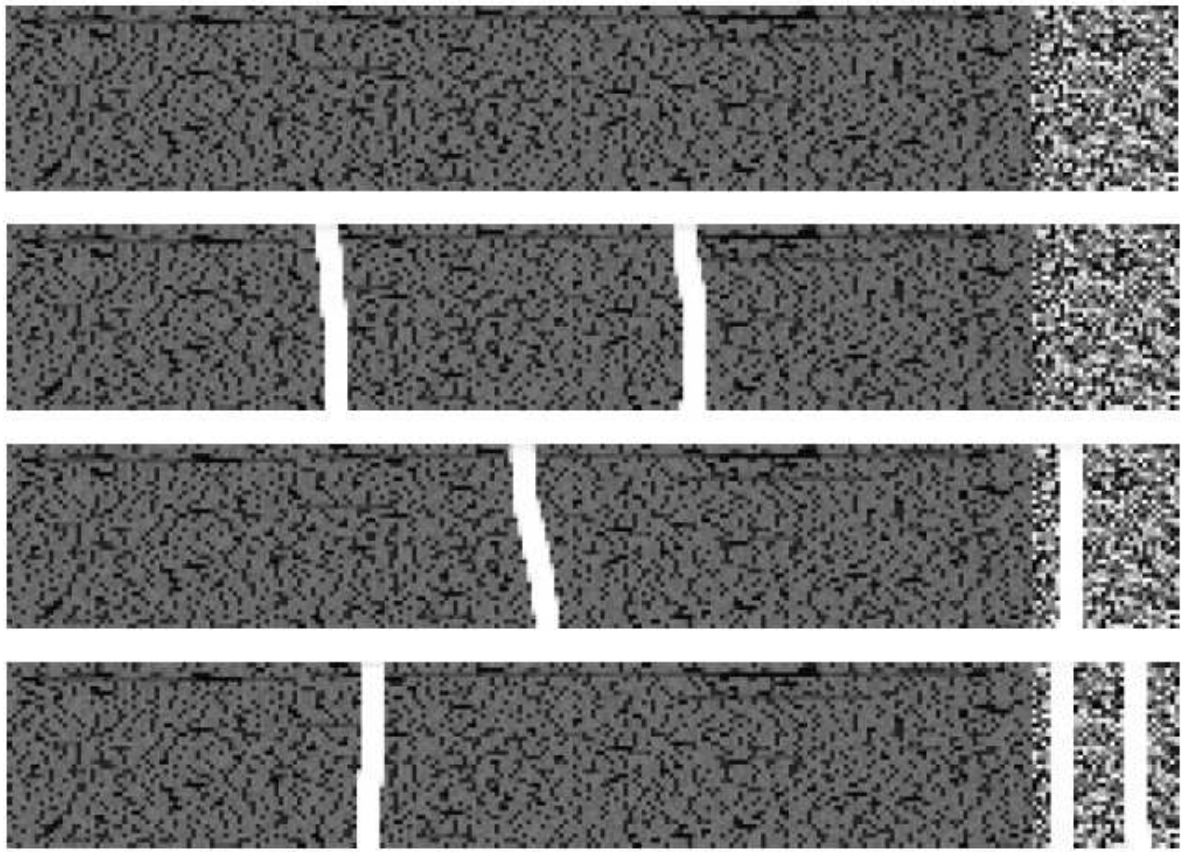
(a) Oversampled parabola (b) With 2 points moved

5.11 Reed-Solomovi kodi 3

- Tako kot ciklični kodi lahko popravijo izbruhe do dolžine $m/2$, kjer je m število varnostnih meta-znakov
- Ker dela z meta znaki dolžine q sta posamična napaka in izbruh dolžine q obravnavana enako
- Na nivoju bitov lahko kod popravi izbruhe do dolžine $m \cdot q$.
- Običajno je $q = 8$, tako da je 1 meta znak = 1 bajt. V tem primeru se najpogosteje uporablja kod $RS(255, 223)$ z 32 varnostnimi meta-znaki.
- Tak kod lahko odkrije in popravi izbruhe napak do dolžine $16 \cdot 8 = 128$ za binarni simetrični kanal oziroma dvakrat toliko za kanal z brisanjem
- Veliko se kombinirajo s konvolucijskimi kodi, ki so odlični za popravljanje posamičnih napak.

5.11 Reed-Solomovi kodi 4

- Primer: $RS(255, 223)$: vsak meta simbol je predstavljen kot točka v ustreznem odtenku sive, vsaka vrstica predstavlja eno kodno zamenjavo.



- Kljub napakam (črte dolžine 5 točk) kod pravilno popravi kodne zamenjave.