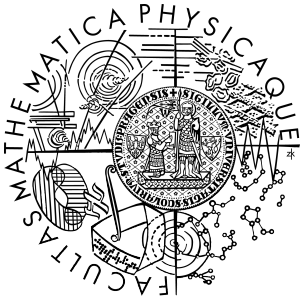
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jakub Skalický

Konvoluční kódy

Katedra algebry

Vedoucí bakalářské práce: Mgr. Libor Barto, PhD.
Studijní program: obecná matematika

2009

Rád bych na tomto místě poděkoval svému vedoucímu, Mgr. Liboru Bartovi, PhD za jeho neutuchající cenné připomínky, které mi byly při psaní této práce významnou pomocí.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 29. května 2009 Jakub Skalický

Obsah

1 Úvod **5**
1.1 Základní pojmy 5

2 Obecné vlastnosti konvolučních kódů **7**
2.1 Generující matice, vyjádření pomocí Laurentových řad . . . 9

3 Třídy konvolučních kódů **15**
3.1 Základní kodéry 16
3.2 Redukované kodéry 19
3.3 Kanonické kodéry 21
3.4 Katastrofické kodéry 23
3.5 Systematické kodéry 24
3.6 Příklady 26

4 Minimální vzdálenost konvolučních kódů **28**

5 Dekódování konvolučních kódů **31**
5.1 Viterbiův algoritmus 33
5.2 Sekvenciální dekódování 42
5.3 M-algoritmus 46
5.4 Porovnání jednotlivých algoritmů 46

6 Použití konvolučních kódů **48**

Literatura **49**

Název práce: Konvoluční kódy
Autor: Jakub Skalický
Katedra (ústav): Katedra algebry
Vedoucí bakalářské práce: Mgr. Libor Barto, PhD.
e-mail vedoucího: libor.barto@gmail.com

Abstrakt: V předložené práci studujeme konvoluční samoopravné kódy. Odvodíme jejich obecné vlastnosti a v několika obměnách ukážeme kódovací algoritmus. Následně popíšeme rozličné třídy těchto kódů a věnujeme se odhadu minimální vzdálenosti kódu. Největší část práce je věnována dekódování, Viterbiovu i sekvenciálním algoritmům a jejich srovnání.

Klíčová slova: konvoluční kódy, samoopravné kódy, Viterbiův algoritmus, sekvenciální dekódování

Title: Convolutional Codes
Author: Jakub Skalický
Department: Department of Algebra
Supervisor: Mgr. Libor Barto, PhD.
Supervisor's e-mail address: libor.barto@gmail.com

Abstract: In the present work we study convolutional error-correcting codes. We show their general properties and the encoding algorithm in several modifications. It is our goal then to describe various convolutional codes' classes and to obtain a bound on free distance. Most significant part of the work is dedicated to decoding, Viterbi and sequential algorithms and their comparison.

Keywords: convolutional codes, error-correcting codes, Viterbi algorithm, sequential decoding

Kapitola 1

Úvod

Historie konvolučních kódů se začala psát v roce 1955, kdy je P. Elias poprvé popsal ve své práci *Coding for noisy channels* [2]. Čtyři roky nato jej následoval D. W. Hagelbarger, jenž konvolučním kódům říkal rekurentní a popsal je ve svém příspěvku *Recurrent Codes: Easily mechanized, burst-correcting binary codes* [3]. V průběhu šedesátých let pak byla teorie konvolučních kódů obohacena o různé metody dekódování, z nichž nejdůležitější a nejpoužívanější — Viterbiův algoritmus — jeho autor publikoval v roce 1967 v článku nazvaném *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm* [10]. Konečně hlubšími algebraickými vlastnostmi konvolučních kódů se poprvé v první polovině sedmdesátých let zabýval G. D. Forney. V nedávné době se ukázalo, že při vhodném zkombinování konvolučních kódů s dalšími technikami vzniknou kódy, které se svými vlastnostmi blíží Shannonovu odhadu na ideální informační poměr. Tzv. *turbokódy* spatřily světlo světa v roce 1993 v článku francouzských elektroinženýrů okolo Claude Berroua nazvaném *Near Shannon Limit Error-correcting Coding and Decoding: Turbo-codes* [1].

V této práci se pokusím na základě knih [4], [5], [6], [7], [8] a [9] a článků [1], [2], [3] a [10] shrnout dosavadní výsledky bádání v oblasti konvolučních kódů, abych se pak zaměřil konkrétně na problém dekódování, používaných algoritmů a jejich srovnání.

1.1 Základní pojmy

Tradiční přístup k samoopravným kódům velí rozdělit předávanou zprávu na bloky stejné (obvykle velmi krátké) délky k, pak na každém bloku provést

kódovací algoritmus a získat sekvenci n bitů. Výsledkem je kódové slovo, které závisí pouze na jednom konkrétním bloku na vstupu a které se po přenosu dekóduje opět nezávisle na ostatních. Nastane-li při přenosu chyba, umí ji blokové kódy „opravit“ jen v rámci bloku, žádné další bloky na dekódování nemají vliv. Dostáváme tedy kódy s fixními parametry (jeden z nich je tzv. *informační poměr* — též nazývaný *nosnost* — $R = \frac{k}{n}$[1]).

Naopak konvoluční kódy na vstupu dostanou celou sekvenci bitů[2], přičemž výsledné bity získané z určitého bloku kódovacím algoritmem závisí i na určitém počtu bitů z *předchozích* bloků. Ty tedy nejsou nezávislé jako v prvním případě, právě naopak. Proto lze při dekódování opravit i velké množství chyb, pokud nejsou v jistém smyslu „příliš nahromaděny“ na jednom místě vysílaného slova. Označíme-li k počet vstupních bitů, ze kterých kodér v jednom kroku spočítá n výstupních bitů, opět dostáváme informační poměr $R = \frac{k}{n}$. Jak se však ukáže, konvoluční kódy mají obyčejně při stejném informačním poměru lepší parametry, zejména „snesou“ více chyb při přenosu.

V celé práci budeme značit vstupní vektor

$$\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots) = (v_0^{(1)}, v_0^{(2)}, \ldots, v_0^{(k)}, v_1^{(1)}, v_1^{(2)}, \ldots, v_1^{(k)}, \ldots),$$

a kódový vektor

$$\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1, \ldots) = (c_0^{(1)}, c_0^{(2)}, \ldots, c_0^{(n)}, c_1^{(1)}, c_1^{(2)}, \ldots, c_1^{(n)}, \ldots),$$

kde $\mathbf{v}_i$ a $\mathbf{c}_i$ jsou po řadě vstupní slovo délky k a kódové slovo délky n v čase $i = 0, 1, \ldots$. Pro vektory j-tých bitů z každého vstupního, respektive kódového slova užívám značení

$$\mathbf{v}^{(j)} = (v_0^{(j)},\ v_1^{(j)}, \ldots), \quad j = 1, \ldots, k,$$

$$\mathbf{c}^{(j)} = (c_0^{(j)},\ c_1^{(j)}, \ldots), \quad j = 1, \ldots, n.$$

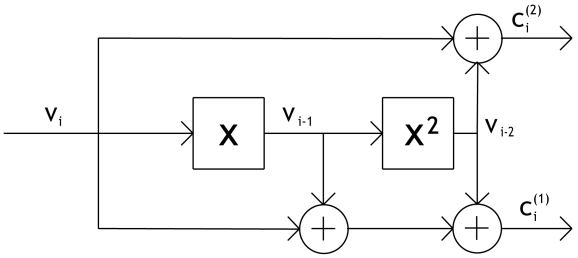
[1] Obecně je informační poměr kódu nad $\mathbb{Z}_p$ délky n s k kódovými slovy (tedy (n, k) kódu) definován jako $R = \frac{\log_p k}{n}$, ovšem můžeme se omezit na lineární kódy, tj. kódy s počtem slov p^k.

[2] V celé práci budu předpokládat práci nad tělesem $\mathbb{Z}_2$, i když teoreticky se dají konvoluční kódy vybudovat nad libovolným tělesem $\mathbb{Z}_p$. Je také možné (a pro porovnání s blokovými kódy i vhodné) si celou vstupní sekvenci představit jako sled bloků.

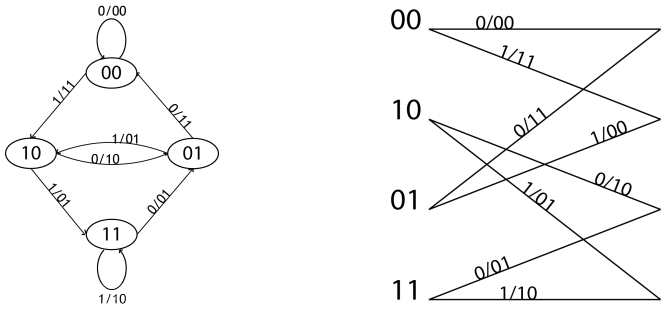
Kapitola 2

Obecné vlastnosti konvolučních kódů

Vysvětleme si nyní princip konvolučního kódování na příkladu $[2, 1]$ kódu, jehož kodér je vyobrazen na obrázku 2.1. Vstupní bit se (modulo 2) sečte s bitem v prvním a druhém registru a dá jeden výstup, v součtu s bitem ve druhém registru pak druhý výstup. Bit ve druhém registru se „zapomene“, do něj se posune bit z prvního registru a do toho se uloží vstup. Naskýtá se otázka, jak to vypadá, když se do registrů načítá první bit a když se již vstupní sekvence „vyčerpala“. Ve většině případů se předpokládá, že na počátku jsou všechny registry ve stavu 0 a na konci se vstupní sekvence doplní nulami tak, aby po posledním vstupu kodér zůstal v počátečním, tj.



Obrázek 2.1: konvoluční kodér $[2, 1]$ kódu



Obrázek 2.2: stavový stroj a mřížka našeho $[2, 1]$ kódu

nulovém stavu[1]. Pro náš $[2, 1]$ kód bude vstup $\mathbf{v}$ doplněn m nulami, kde m je počet registrů kodéru.

Na konvoluční kodér se také dá dívat jako na *stavový stroj*, kdy každou možnou posloupnost bitů uložených v registrech nazveme *stavem* — značíme $\mathbf{s} = \left(s^{(1)}, s^{(2)}, \ldots, s^{(m)}\right)$ — a budeme se zajímat o to, do jakých dalších stavů se z jednoho konkrétního může kodér dostat v závislosti na vstupu. Pro kodér o m registrech existuje přirozeně právě 2^m stavů, na obrázku 2.2 je vyobrazena taková reprezentace našeho kodéru z obrázku 2.1. Pro účely dekódování je ale nejvýhodnější reprezentace pomocí tzv. *mřížky*, jak vidíme na obrázku 2.2. Jeden krok kodéru je zobrazen jako orientovaný bipartitní graf, jehož vrcholy jsou vždy jednotlivé stavy a mezi dvěma stavy $\mathbf{s}_1$ a $\mathbf{s}_2$ vede hrana právě tehdy, když existuje vstup takový, že po něm kodér přejde ze stavu $\mathbf{s}_1$ do stavu $\mathbf{s}_2$. Zařadíme-li za sebe těchto kroků více, získáme právě mřížku — libovolná posloupnost orientovaných hran v ní nám pak odpovídá nějakému kódovému slovu.

Příklad 2.0.1. Nechť do našeho kodéru vstoupí sekvence $\mathbf{v} = (1, 1, 0, 0, 1, 0, 1)$, označme $c_i^{(j)}$, $j = 1, 2$, j-tý bit výstupu po i-tém vstupu. Spočítejme první dva kroky kódování:

Na začátku je kodér ve stavu $\mathbf{s}_0 = (0, 0)$, první bit vstupu je $v_0 = 1$. Tedy první bit výstupu spočítáme jako

$$c_0^{(1)} = v_0 + s_0^{(1)} + s_0^{(2)} = 1 + 0 + 0 \equiv 1 \pmod 2$$

a druhý jako

$$c_0^{(2)} = v_0 + s_0^{(2)} = 1 + 0 \equiv 1 \pmod 2.$$

[1]Toto se provádí zejména z praktických důvodů — při dekódování se hodí vědět, jak zhruba vstup končí. Více se dozvíme v kapitole o dekódování.

Aktualizujme i stav: $s_1^{(1)} = v_0$, $s_1^{(2)} = s_0^{(1)}$, tedy $\mathbf{s}_1 = (1, 0)$.

Druhý krok bude vypadat takto: na vstupu je $v_1 = 1$, tedy

$$c_1^{(1)} = v_1 + s_1^{(1)} + s_1^{(2)} = 1 + 1 + 0 \equiv 0 \pmod 2,$$

$$c_1^{(2)} = v_1 + s_1^{(2)} = 1 + 0 \equiv 1 \pmod 2.$$

Stav bude nyní zřejmě $\mathbf{s}_2 = (1, 1)$.

Budeme-li pokračovat takto dále (použijeme doplnění vstupního vektoru $\mathbf{v}$ dvěma nulami), vyjdou nám výstupy takto:

$$\mathbf{c}^{(1)} = (1, 0, 0, 1, 1, 1, 0, 1, 1) \qquad \text{a}$$

$$\mathbf{c}^{(2)} = (1, 1, 1, 1, 1, 0, 0, 0, 1),$$

které dají celkový výstup

$$\mathbf{c} = (11, 01, 01, 11, 11, 10, 00, 10, 11)\,,$$

kde čárky dávají dohromady výstupy ve stejném čase i.

2.1 Generující matice, vyjádření pomocí Laurentových řad

Pro sekvence bitů zvolme reprezentaci řadami v proměnné x, tedy sekvenci $\{\ldots v_{-2}, v_{-1}, v_0, v_1, v_2, \ldots\}$ zapíšeme jako formální Laurentovu řadu $v(x) = \sum_{l=-\infty}^{\infty} v_l x^l$. Množina všech Laurentových řad tvaru $\sum_{l\geq 0} a_l x^l$ [2] nad tělesem $\mathbb{Z}_2$ je obor integrity, který značíme $\mathbb{Z}_2[\![x]\!]$. Pokud chceme dostat těleso, musíme „povolit“ i záporný začátek řady: množina všech Laurentových řad $\sum_{l=k}^{\infty} a_l x^l$, kde $k \in \mathbb{Z}$ je libovolné a $a_l \in \mathbb{Z}_2$, je těleso, které značíme $\mathbb{Z}_2(\!(x)\!)$. Jako $\mathbb{Z}_2(x)$ označíme podtěleso $\mathbb{Z}_2(\!(x)\!)$, které je množinou všech *racionálních* Laurentových řad, tj. těch řad, které vzniknou rozvojem racionálních funkcí $\frac{P(x)}{Q(x)}$, kde $P(x), Q(x) \in \mathbb{Z}_2[x]$ a $Q(x) \neq 0$.

Definujme klíčové pojmy:

Definice 2.1.1. [5] $[n, k]$ *konvoluční* kód $\mathbf{C}$ je k-rozměrným podprostorem n-rozměrného prostoru Laurentových řad $\mathbb{Z}_2(\!(x)\!)^n$, jehož báze jsou výhradně vektory z tělesa racionálních Laurentových řad $\mathbb{Z}_2(x)^n$.

[2] Tedy vlastně mocninných řad.

Definice 2.1.2. Nechť vstupní sekvence je $\mathbf{v} = (v_0^{(1)},\ v_0^{(2)}, \ldots,\ v_0^{(k)}, v_1^{(1)}, \ldots,\ v_1^{(k)}, \ldots)$, označme $V^{(j)}(x)$ mocninnou řadu odpovídající vektoru $\mathbf{v}^{(j)} = (v_0^{(j)},\ v_1^{(j)}, \ldots)$, tedy

$$V^{(j)}(x) = \sum_{l=0}^{\infty} v_l^{(j)} x^l, \quad j = 1, \ldots, k.$$

Označ $V(x) = (V^{(1)}(x),\ V^{(2)}(x), \ldots,\ V^{(k)}(x))$, analogicky pro kódové sekvence: $C(x) = (C^{(1)}(x),\ C^{(2)}(x), \ldots,\ C^{(n)}(x))$, kde

$$C^{(j)}(x) = \sum_{l=0}^{\infty} c_l^{(j)} x^l, \quad j = 1, \ldots, n.$$

Řekneme, že $k \times n$ matice $G(x)$, $g_{ij}(x) \in \mathbb{Z}_2(x)$ je *generující maticí* $[n, k]$ konvolučního kódu $\mathbf{C}$, jestliže každá kódová sekvence $C(x) = (C^{(1)}(x),\ C^{(2)}(x), \ldots,\ C^{(n)}(x))$ lze vyjádřit jako lineární kombinace jejích řádků.

Poznámka 2.1.3. Budeme se zabývat výhradně tzv. *nezpožděnými* generujícími maticemi, tj. těmi, ze kterých nejde vytknout x^l pro nějaké $l > 0$.

Odvoďme generující matici pro náš $[2, 1]$ kód: z příkladu 2.0.1 je zřejmé, že dva bity výstupu v čase i (na rozdíl od blokových kódů) jsou funkcí jak i-tého vstupního bitu v_i, tak i vnitřního stavu $\mathbf{s}_i$ (to samé platí i pro stav $\mathbf{s}_{i+1}$). Obecně samozřejmě pro $[n, k]$ kód není vstup v čase i jediný bit, ale vektor $\mathbf{v}_i$ délky k; výstup je pak vektor $\mathbf{c}_i$ délky n a stav vektor $\mathbf{s}_i$ délky m. Vyjádřeme popisovaný vztah maticově:

$$\mathbf{s}_{i+1} = \mathbf{s}_i \mathcal{A} + \mathbf{v}_i \mathcal{B},$$

$$\mathbf{c}_i = \mathbf{s}_i \mathcal{C} + \mathbf{v}_i \mathcal{D},$$

kde $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ a $\mathcal{D}$ jsou matice nad $\mathbb{Z}_2$ o rozměrech

$$\mathcal{A} : m \times m, \qquad \mathcal{B} : k \times m,$$

$$\mathcal{C} : m \times n, \qquad \mathcal{D} : k \times n.$$

Číslo m se nazývá *stupeň kodéru*. Na tomto místě stojí za povšimnutí, že v případě, že by vnitřní stav kodéru měl délku 0 (tj. jednalo by se o kód tzv. *bez paměti*), rovnice by se redukovaly na $\mathbf{c}_i = \mathbf{v}_i \mathcal{D}$. To ovšem není nic jiného než blokový kód generovaný maticí $\mathcal{D}$; dostáváme jej tedy jako speciální případ konvolučního pro $m = 0$.

Ukažme nyní, jak lze od těchto čtyř matic přejít k jedné, generující: vynásobme obě rovnosti členem x^i a sečtěme přes všechna i (využijeme při tom faktu, že $\mathbf{s}_i$, $\mathbf{c}_i$ i $\mathbf{v}_i$ jsou nulové pro $i < 0$):

$$x^{-1}S(x) = S(x)\mathcal{A} + V(x)\mathcal{B},$$
$$C(x) = S(x)\mathcal{C} + V(x)\mathcal{D},$$

kde $V(x)$ a $C(x)$ jsou jako v definici 2.1.2 a podobně $S(x) = (S^{(1)}(x),\ S^{(2)}(x), \ldots,\ S^{(m)}(x))$, kde

$$S^{(j)}(x) = \sum_{l=0}^{\infty} s_l^{(j)} x^l, \quad j = 1, \ldots, m.$$

Upravujme první rovnici:

$$x^{-1}S(x) - S(x)\mathcal{A} = V(x)\mathcal{B}$$
$$S(x)(x^{-1}I_m - \mathcal{A}) = V(x)\mathcal{B}$$

Nyní je na místě si všimnout, že $x^{-1}I_m - \mathcal{A}$ je regulární matice typu $m \times m$ (protože $x^{-1}I_m$ je regulární a prvky $\mathcal{A}$ jsou skaláry tělesa $\mathbb{Z}_2$), tedy že k ní existuje inverzní. Proto

$$S(x) = V(x)\mathcal{B}(x^{-1}I_m - \mathcal{A})^{-1}.$$

Obdobně upravíme i druhou rovnici:

$$C(x) = S(x)\mathcal{C} + V(x)\mathcal{D}$$
$$C(x) = V(x)\mathcal{B}(x^{-1}I_m - \mathcal{A})^{-1}\mathcal{C} + V(x)\mathcal{D}$$
$$C(x) = V(x)(\mathcal{B}(x^{-1}I_m - \mathcal{A})^{-1}\mathcal{C} + \mathcal{D})$$

Dostali jsme tedy vyjádření výstupu $C(x)$ i stavů $S(x)$ v závislosti na vstupu $V(x)$:

$$S(x) = V(x)E(x),$$
$$C(x) = V(x)G(x),$$

kde matice $E(x)$ typu $k \times m$ a $G(x)$ typu $k \times n$ získáme jako

$$E(x) = \mathcal{B}(x^{-1}I_m - \mathcal{A})^{-1},$$
$$G(x) = \mathcal{D} + E(x)\mathcal{C} = \mathcal{D} + \mathcal{B}(x^{-1}I_m - \mathcal{A})^{-1}\mathcal{C}.$$

Konvoluční kód popsaný maticemi $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ a $\mathcal{D}$ je tedy vlastně množinou všech lineárních kombinací řádků matice $G(x)$ — proto je zcela legální nazvat matici $G(x)$ *generující* (viz definici 2.1.2), pro čtveřici ($\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$) se užívá označení *state-space realization* matice $G(x)$.

Příklad 2.1.4. Uveďme opět příklad „našeho“ kodéru, tentokrát ve formě čtyř matic. Z obrázku 2.1 je patrné, že

$$\mathbf{s}_{i+1} = \left(s_{i+1}^{(1)},\ s_{i+1}^{(2)}\right) = \left(v_i,\ s_i^{(1)}\right) \quad \text{a}$$

$$\mathbf{c}_i = \left(s_i^{(1)} + s_i^{(2)} + v_i,\ s_i^{(2)} + v_i\right).$$

Matice budou tedy vypadat takto:

$$\mathcal{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad \mathcal{B} = \begin{pmatrix} 1 & 0 \end{pmatrix},$$

$$\mathcal{C} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \qquad \mathcal{D} = \begin{pmatrix} 1 & 1 \end{pmatrix},$$

pak

$$E(x) = \mathcal{B}(x^{-1}I_m - \mathcal{A})^{-1} = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} x^{-1} & 1 \\ 0 & x^{-1} \end{pmatrix}^{-1} = \begin{pmatrix} x & x^2 \end{pmatrix},$$

a tedy

$$G(x) = \mathcal{D} + E(x)\mathcal{C} = \begin{pmatrix} 1 & 1 \end{pmatrix} + \begin{pmatrix} x & x^2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1+x+x^2 & 1+x^2 \end{pmatrix}.$$

Příklad 2.1.5. Když již máme odvozenou generující matici, vraťme se k reprezentaci sekvencí bitů jako Laurentových řad. V příkladu 2.1.4 jsme zjistili, že generující matice „našeho“ $[2, 1]$ kódu je $G(x) = \begin{pmatrix} 1+x+x^2 & 1+x^2 \end{pmatrix}$ a výstup se dá vyjádřit jako

$$C(x) = V(x)\begin{pmatrix} 1+x+x^2 & 1+x^2 \end{pmatrix}.$$

Označíme-li $g^{(1)}(x)$ a $g^{(2)}(x)$ prvky $G(x)$, máme $g^{(1)}(x) = 1 + x + x^2$ a $g^{(2)}(x) = 1 + x^2$. Interpretujme $V(x)$: protože máme $k = 1$, je $V(x) = V^{(1)}(x) = \sum_{i\geq 0} v_i^{(1)} x^i$. To ovšem neznamená nic jiného než že $V(x)$ je Laurentova řada, jejímiž koeficienty jsou vstupní bity. Naopak $C(x) = (C^{(1)}(x),\ C^{(2)}(x)) = \left(\sum_{i\geq 0} c_i^{(1)} x^i,\ \sum_{i\geq 0} c_i^{(2)} x^i\right)$, kde $c_i^{(j)}$ je j-tý bit výstupu v čase i. Pak z vyjádření výstupu platí, že $C^{(j)}(x) = V(x)g^{(j)}(x)$, $j = 1, 2$.

Vezměme si stejný vstup jako v příkladu 2.0.1: $\mathbf{v} = (1, 1, 0, 0, 1, 0, 1)$, tedy $V(x) = 1 + x + x^4 + x^6$. Pak

$$C^{(1)}(x) = V(x)g^{(1)}(x) = (1+x+x^4+x^6)(1+x+x^2) \equiv 1+x^3+x^4+x^5+x^7+x^8,$$

$C^{(2)}(x) = V(x)g^{(2)}(x) = (1+x+x^4+x^6)(1+x^2) \equiv 1+x+x^2+x^3+x^4+x^8.$

Po převedení z Laurentových řad zpět na vektory dostáváme

$$\mathbf{c}^{(1)} = (1,0,0,1,1,1,0,1,1)$$

$$\mathbf{c}^{(2)} = (1,1,1,1,1,0,0,0,1),$$

tedy ten samý výsledek jako v příkladu 2.0.1.

Funkce $g^{(1)}(x)$ a $g^{(2)}(x)$ z příkladu 2.1.5 se nazývají *transformující funkce*, matici $G(x)$ se proto někdy říká *transformující*.

Obecně pro $[n,k]$ kód vypadá uvedený postup následovně: Označme stejně jako v definici 2.1.2 $V^{(j)}(x) = \sum_{i\geq 0} v_i^{(j)} x^i$, $j = 1,2,\ldots,k$ Laurentovu řadu odpovídající j-tým bitům vstupu v čase i a $C^{(j)}(x) = \sum_{i\geq 0} c_i^{(j)} x^i$, $j = 1,2,\ldots,n$ Laurentovu řadu odpovídající j-tým bitům výstupu v čase i. Stejně jako dříve dostáváme $V(x) = \big(V^{(1)}(x),\ldots,\ V^{(k)}(x)\big)$ a $C(x) = \big(C^{(1)}(x),\ldots,\ C^{(n)}(x)\big)$. Obecná generující matice typu $k \times k$ má tvar

$$G(x) = \begin{pmatrix} g^{(1,1)}(x) & g^{(1,2)}(x) & \cdots & g^{(1,n)}(x) \\ g^{(2,1)}(x) & g^{(2,2)}(x) & \cdots & g^{(2,n)}(x) \\ \vdots & & \ddots & \\ g^{(k,1)}(x) & g^{(k,2)}(x) & \cdots & g^{(k,n)}(x) \end{pmatrix}.$$

Z rovnice $C(x) = V(x)G(x)$ získáváme obecný kódovací algoritmus, kdy jednoduše $C^{(j)}(x) = \sum_{l=1}^{k} V^{(l)}(x)g^{(l,j)}(x)$, $j = 1,2,\ldots,n$.

Jak je vidět, při reprezentaci vektorů pomocí Laurentových řad není třeba udržovat stavy (ty jsou již „zabudovány“ v generující matici, jak je zřejmé z jejího odvození), je to proto asi nejjednodušší způsob ručního počítání. V praxi se nicméně pro polynomiální $G(x)$ používá tzv. *impulse response*, což není nic jiného než vektor koeficientů polynomů v transformujících funkcích (seřazených od koeficientu u absolutního členu po vedoucí koeficient). Pro „naši“ matici $G(x)$ to bude vypadat takto: $\mathbf{q}^{(1)} = (1,1,1)$ a $\mathbf{q}^{(2)} = (1,0,1)$. Obecně pro *impulse response* $\mathbf{q}^{(j)} = (q_0^{(j)},\ q_1^{(j)},\ldots,\ q_m^{(j)})$ se j-tý bit výstupu kodéru v čase i dá vyjádřit jako

$$c_i^{(j)} = \sum_{l=0}^{m} v_{i-l} q_l^{(j)}. \quad {}^{3}$$

[3] Ve výrazu je nutno použít notaci přímo jednotlivých bitů vstupu, neboť obecně se v sumě mohou objevit bity vstupující do kodéru v různém čase.

To ovšem není nic jiného než operace diskrétní konvoluce $(\mathbf{v} * \mathbf{q}^{(j)})[i]$[4]. Tu lze zapsat i s použitím matic:

$$\mathbf{c} = [v_0, v_1, v_3 \ldots] \begin{pmatrix} q_0^{(j)} & q_1^{(j)} & q_2^{(j)} & \cdots & q_m^{(j)} & 0 & 0 & \cdots \\ 0 & q_0^{(j)} & q_1^{(j)} & \cdots & q_{m-1}^{(j)} & q_m^{(j)} & 0 & \cdots \\ 0 & 0 & q_0^{(j)} & \cdots & q_{m-2}^{(j)} & q_{m-1}^{(j)} & q_m^{(j)} & \cdots \\ \vdots & & & \ddots & & & & \ddots \end{pmatrix}.$$

Tento tvar generující matice se nazývá *Toeplitzův*.

Na závěr kapitoly se vraťme ke stupni kodéru m. Často se mu říká *paměť* kodéru, což má evokovat, kolik vstupních bitů si kodér ve svém stavu „pamatuje“, nicméně jak nyní uvidíme, tato interpretace není zcela přesná.

Příklad 2.1.6. Mějme $[2, 1]$ kód s maticemi

$$\mathcal{A}_{inf} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \mathcal{B}_{inf} = \begin{pmatrix} 1 & 0 \end{pmatrix},$$

$$\mathcal{C}_{inf} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \qquad \mathcal{D}_{inf} = \begin{pmatrix} 1 & 1 \end{pmatrix},$$

pak vyjdou tyto dvě rovnice:

$$\mathbf{s}_{i+1} = \left(s_i^{(1)} + s_i^{(2)} + v_i,\ s_i^{(1)} \right),$$

$$\mathbf{c}_{i+1} = \left(v_i,\ s_i^{(1)} + v_i + s_i^{(2)} + s_i^{(2)} \right) \equiv \left(v_i,\ s_i^{(1)} + v_i \right).$$

Přesně podle obecného postupu nastíněného výše chceme vyjádřit výstup $\mathbf{c}_i$, ovšem vyjde

$$c_i^{(1)} = v_i,$$

$$c_i^{(2)} = v_i + v_{i-1} + v_{i-2} + v_{i-3} + \cdots.$$

Kodér určený čtveřicí matic $(\mathcal{A}_{inf}, \mathcal{B}_{inf}, \mathcal{C}_{inf}, \mathcal{D}_{inf})$ z příkladu 2.1.6 tedy má trochu neintuitivně *nekonečnou* paměť. Z tohoto důvodu je vhodnější uvádět spíše stupeň kodéru než jeho paměť.

[4]Obecně je samozřejmě diskrétní konvoluce definována jako $(f * g)[n] = \sum_{m=-\infty}^{\infty} f(n-m)g(m)$, využijeme zřejmého faktu, že $q_i^{(j)} = 0\ \forall i \notin \{0,\ 1, \ldots,\ m\}$. Právě z tohoto vztahu pochází název konvolučních kódů.

Kapitola 3

Třídy konvolučních kódů

Uveďme nyní několik definic.

Definice 3.0.7. Nechť $G(x)$ je polynomiální generující matice $[n, k]$ konvolučního kódu. Pak definujme *vnitřní stupeň* $G(x)$ jako

$$\operatorname{intdeg} G(x) = \max_{D_k} \deg D_k,$$

kde D_k probíhá $k \times k$ subdeterminanty generující matice $G(x)$; *vnější stupeň* $G(x)$ je

$$\operatorname{extdeg} G(x) = \sum_{j=1}^{k} \deg \mathbf{g}_j,$$

kde $\deg \mathbf{g}_j$ je stupeň j-tého řádku $G(x)$, tj. maximální stupeň polynomů v j-tém řádku $G(x)$.

Definice 3.0.8. Nechť $\mathbb{F}$ je těleso, $G(x)$ matice s prvky $g_{ij}(x) \in \mathbb{F}[x]$. Řekneme, že $G(x)$ je *unimodulární*, jestliže $\det G(x) \in \mathbb{F} \setminus \{0\}$, tj. determinant je nenulový skalár z tělesa $\mathbb{F}$.

Poznámka 3.0.9. Protože pracujeme nad tělesem $\mathbb{Z}_2$, je podmínka unimodularity matice $G(x)$ ekvivalentní podmínce $\det G(x) = 1$. Stojí také za povšimnutí, že je-li $G(x)$ unimodulární, je čtvercová (typu $l \times l$ pro nějaké $l \in \mathbb{N}$) a invertibilní, tj. existuje polynomiální matice $G^{-1}(x)$ taková, že $G(x)G^{-1}(x) = I_m$. Existence polynomiální inverzní matice přitom plyne z adjungované matice a faktu, že $\det G(x)$ je skalár.

3.1 Základní kodéry

Definice 3.1.1. Řekneme, že dvě generující matice $G(x)$ a $G'(x)$ jsou *ekvivalentní*, jestliže generují stejný konvoluční kód, tedy jestliže existuje *invertibilní* matice $T(x)$ nad $\mathbb{Z}_2(x)$, že $G(x) = T(x)G'(x)$.

Definice 3.1.2. [5] Řekneme, že generující matice $G(x)$ typu $k \times n$ je *základní*, jestliže je polynomiální a mezi všemi polynomiálními generujícími maticemi ekvivalentními s $G(x)$ má minimální vnitřní stupeň.

Definice 3.1.3. [6] Nechť $k < n$ a $G(x)$ je matice typu $k \times n$, pak $G(x)$ nazveme *zprava invertibilní*, jestliže existuje (ne nutně polynomiální) matice $G^{-1}(x)$ typu $n \times k$ taková, že $G(x)G^{-1}(x) = I_k$, kde I_k je identická matice typu $k \times k$. Matici $G^{-1}(x)$ nazveme *zprava inverzní maticí* $G(x)$.

Poznámka 3.1.4. Všimněme si hned, že zprava inverzní matice může existovat pouze když $G(x)$ má hodnost k a nemusí být určena jednoznačně.

Věta 3.1.5. *Nechť $G(x)$ je polynomiální matice typu $k \times n$.*

(a) Jestliže $T(x)$ je libovolná invertibilní polynomiální matice typu $k \times k$, pak

$$\text{intdeg}\, T(x)G(x) = \text{intdeg}\, G(x) + \deg \det T(x),$$

speciálně intdeg $T(x)G(x) \geq$ intdeg $G(x)$*, přičemž rovnost nastane právě tehdy, když $T(x)$ je unimodulární.*

(b) intdeg $G(x) \leq$ extdeg $G(x)$.

Důkaz. (a) Podmatice $T(x)G(x)$ velikosti $k \times k$ jsou vlastně $k \times k$ podmatice $G(x)$ vynásobené $T(x)$. Tedy subdeterminanty $T(x)G(x)$ velikosti $k \times k$ jsou rovny $k \times k$ subdeterminantům $G(x)$, vynásobeným $\det T(x)$. Z definice vnitřního stupně nyní zřejmě plyne první část věty.

(b) Označme stupeň i-tého řádku $G(x)$ jako f_i — tj. $f_i = \max_{j=1,\ldots,n}\{\deg g_{ij}(x)\}$ — a D_k libovolný $k \times k$ subdeterminant $G(x)$. Z definice se D_k spočítá jako součet součinů k prvků $G(x)$, vždy po jednom z každého řádku (a každého z k sloupců). Protože ale každý prvek v i-tém řádku má stupeň $\leq f_i$, okamžitě vidíme, že $\deg D_k \leq f_1 + f_2 + \ldots + f_k =$ extdeg $G(x)$. □

Uveďme nyní charakterizaci základních kodérů:

Věta 3.1.6. *Nechť $G(x)$ je polynomiální generující matice typu $k \times n$. Pak $G(x)$ je základní, právě tehdy když platí některá z následujících podmínek:*

(a) invariantní faktory $G(x)$ jsou všechny 1, tj. ekvivalentně největší společný dělitel $k \times k$ subdeterminantů $G(x)$ je roven 1,

(b) Existuje polynomiální zprava inverzní matice k $G(x)$, tj. $\exists G^{-1}(x)$ typu $n \times k$ taková, že $G(x)G^{-1}(x) = I_k$,

(c) je-li $C(x) = V(x)G(x)$ a $C(x) \in \mathbb{Z}_2[x]^n$, pak $V(x) \in \mathbb{Z}_2[x]^k$ (jinými slovy, je-li kódové slovo polynom, pak i vstup byl polynom).

Důkaz. Jako první krok nepřímo dokážeme, že z předpokladu $G(x)$ je základní plyne (a): nechť $\Gamma(x)$ je invariantní matice k $G(x)$ typu $k \times n$, tedy

$$\Gamma(x) = \begin{pmatrix} \gamma_1 & & & & & \\ & \gamma_2 & & & & \\ & & \gamma_3 & & & \mathbf{0}_{k,n-k} \\ & & & \ddots & & \\ & & & & \gamma_k & \end{pmatrix},$$

kde γ_i je invariantní faktor matice, tj. $\gamma_i = \frac{\Delta_i}{\Delta_{i-1}}$, kde Δ_i je největší společný dělitel všech subdeterminantů $G(x)$ typu $i \times i$ a $\mathbf{0}_{k,n-k}$ je matice typu $k \times (n-k)$, která obsahuje samé nuly. Pak existují unimodulární matice $X(x)$ typu $k \times k$ a $Y(x)$ typu $n \times n$ takové, že

$$X(x)G(x)Y(x) = \Gamma(x).$$

Označme $\Gamma_k(x)$ matici tvořenou prvními k sloupci matice $\Gamma(x)$, pak matice $G'(x) = \Gamma_k^{-1}(x)X(x)G(x)$ je polynomiální matice ekvivalentní $G(x)$ (roli $T(x)$ z definice ekvivalentních generujících matic hraje $\Gamma_k^{-1}(x)X(x)$). Podívejme se na determinant:

$$\det(\Gamma_k^{-1}(x)X(x)) = \det\Gamma_k^{-1}(x) \cdot \det X(x) = (\det\Gamma_k(x))^{-1} \cdot \det X(x) = s(\gamma_1\gamma_2\cdots\gamma_k)^{-1},$$

kde $s = \det X(x)$ je nenulový skalár. Podle věty 3.1.5 (a) ale platí, že intdeg $T(x)G(x) =$ intdeg $G(x) + \deg\det T(x)$, což v tomto případě znamená, že jestliže všechny invariantní faktory $G(x)$ nejsou rovny 1, $G'(x)$ má menší vnitřní stupeň než $G(x)$, ergo $G(x)$ není základní.

Ekvivalence obou tvrzení v bodu (a) je zřejmá:

$$\gamma_1\gamma_2 \cdot \gamma_k = \frac{\Delta_1}{\Delta_0} \cdot \frac{\Delta_2}{\Delta_1} \cdots \frac{\Delta_k}{\Delta_{k-1}} = \frac{\Delta_k}{\Delta_0} = \Delta_k,$$

kde definitoricky je $\Delta_0 = 1$. Ovšem Δ_k není nic jiného než největší společný dělitel všech $k \times k$ subdeterminantů $G(x)$.

Dokažme implikaci $(a) \Rightarrow (b)$. Nechť $\Delta_k = 1$ a označme $D_{k,\nu}(x)$ jednotlivé subdeterminanty typu $k \times k$, $\nu = 1, 2, \ldots, \binom{n}{k}$. Pak z Cramerova pravidla pro každé ν existuje matice $H_\nu(x)$ typu $n \times k$ taková, že $G(x)H_\nu(x) = D_{k,\nu}(x)I_k$. Protože $\Delta_k = 1$, existuje lineární kombinace s polynomiálními koeficienty $\lambda_\nu(x)$ taková, že $\sum_\nu \lambda_\nu(x)D_{k,\nu}(x) = 1$. Z toho plyne, že $G^{-1}(x) = \sum_\nu \lambda_\nu(x)H_\nu(x)$ je hledaná polynomiální inverzní matice k $G(x)$.

K důkazu $(b) \Rightarrow (c)$ předpokládejme, že $G(x)$ má inverzní polynomiální matici $G^{-1}(x)$ typu $n \times k$ a že $V(x)$ je vektor délky k, jehož prvky jsou racionální funkce. Navíc předpokládejme, že $C(x) = V(x)G(x)$ je vektor polynomů délky n. Vynásobme poslední rovnost zprava maticí $G^{-1}(x)$: dostaneme $C(x)G^{-1}(x) = V(x)$, a protože $G^{-1}(x)$ i $C(x)$ jsou polynomiální, musí být nutně polynomiální i $V(x)$.

K dokončení důkazu nám zbývá ukázat, že (c) implikuje $G(x)$ základní. Nechť $T(x)$ je libovolná racionální invertibilní matice typu $k \times k$ taková, že $G'(x) = T(x)G(x)$ je polynomiální. Z předpokladu vlastnosti (c) ale plyne, že $T(x)$ je ve skutečnosti polynomiální (za $C(x)$ dosaďme postupně řádky $G'(x)$ a za $V(x)$ řádky $T(x)$). Pak ale z věty 3.1.5 (a) máme, že intdeg $G'(x) \geq$ intdeg $G(x)$, tedy $G(x)$ je základní. □

Základní matice mají jednu zajímavou vlastnost, a to takovou, že existují pro libovolný konvoluční kodér a libovolnou generující matici lze na ně převést:

Věta 3.1.7. *Každá racionální generující matice má ekvivalentní základní generující matici.*

Důkaz. Mějme $G(x)$ polynomiální matici typu $k \times n$. Pak platí

$$G(x) = A(x)\Gamma(x)B(x),$$

kde $A(x)$ a $B(x)$ jsou polynomiální matice typu $k \times k$, resp. $n \times n$, pro které platí $\det(A(x)) = \det(B(x)) = 1$ (a tedy jsou unimodulární) a $\Gamma(x)$ je diagonální matice typu $k \times n$

$$\Gamma(x) = \begin{pmatrix} \gamma_1(x) & & & & & \\ & \gamma_2(x) & & & & \\ & & \gamma_3(x) & & & \mathbf{0}_{k,n-k} \\ & & & \ddots & & \\ & & & & \gamma_k(x) & \end{pmatrix},$$

kde $\mathbf{0}_{k,n-k}$ je matice typu $k \times (n-k)$, jejíž prvky jsou samé nuly, a $\gamma_i(x)$ jsou nenulové invariantní faktory, které splňují $\gamma_i(x) \mid \gamma_{i+1}(x)$. Pokud tento fakt rozšíříme na racionální matice $G(x)$, změní se pouze členy $\Gamma(x)$ na $\gamma_i(x) = \frac{\alpha_i(x)}{\beta_i(x)}$, kde $\alpha_i(x) \mid \alpha_{i+1}(x)$ a $\beta_{i+1}(x) \mid \beta_i(x)$.

Označme nyní $B(x) = \begin{pmatrix} G'(x) \\ B_2(x) \end{pmatrix}$, kde $G'(x)$ je typu $k \times n$ (a tedy $B_2(x)$ typu $(n-k) \times n$). Protože posledních k sloupců $\Gamma(x)$ je nulových, můžeme psát

$$G(x) = A(x) \begin{pmatrix} \frac{\alpha_1(x)}{\beta_1(x)} & & & & \\ & \frac{\alpha_2(x)}{\beta_2(x)} & & & \\ & & \frac{\alpha_3(x)}{\beta_3(x)} & & \\ & & & \ddots & \\ & & & & \frac{\alpha_k(x)}{\beta_k(x)} \end{pmatrix} G'(x) = A(x)\Gamma'(x)G'(x).$$

Matice $A(x)$ a $\Gamma'(x)$ ale jsou obě invertibilní typu $k \times k$, tedy $G(x)$ a $G'(x)$ jsou ekvivalentní dle definice. Nyní potřebujeme ještě ukázat, že $G'(x)$ je základní. To je ovšem jednoduché, neboť $G'(x)$ je částí polynomiální matice $B(x)$, ergo je taktéž polynomiální. Protože $B(x)$ je unimodulární, má inverzní matici, a tedy i $G'(x)$ má zprava inverzní polynomiální matici. Tedy $G'(x)$ je základní generující matice ekvivalentní $G(x)$ a důkaz je u konce. □

3.2 Redukované kodéry

Definice 3.2.1. Řekneme, že polynomiální generující matice $G(x)$ typu $k \times n$ je *redukovaná*, pokud mezi všemi maticemi tvaru $T(x)G(x)$, kde $T(x)$ je unimodulární matice řádu k, má $G(x)$ minimální možný vnější stupeň.

Připomeňme *stupně řádků* generující matice $G(x)$, definované jako $f_i = \max_{j=1,\dots,n}\{\deg g_{ij}(x)\}$. Uvedeme charakteristiku redukovaných generujících matic:

Věta 3.2.2. *Nechť $G(x)$ je polynomiální generující matice typu $k \times n$. Pak $G(x)$ je redukovaná, právě tehdy když platí některá z následujících ekvivalentních podmínek:*

(a) definujeme-li „matici incidence“ $\overline{H}$ členů s nejvyšším stupněm v řádku $G(x)$ jako

$$\overline{h_{ij}} = \begin{cases} 0, & \text{pokud } \deg g_{ij}(x) < f_i \\ 1, & \text{pokud } \deg g_{ij}(x) = f_i. \end{cases}$$

kde f_i je stupeň i-tého řádku $G(x)$, pak $\overline{H}$ má hodnost k;

(b) $\operatorname{intdeg} G(x) = \operatorname{extdeg} G(x)$*;*

(c) pro libovolný k-rozměrný vektor polynomů $\mathbf{u}(x) = (u_1(x), \ldots, u_k(x)) \in \mathbb{Z}_2[x]^k$ *platí*

$$\deg\left(\mathbf{u}(x)G(x)\right) = \max_{1\le i\le k}\left(\deg u_i(x) + \deg \mathbf{g}_i(x)\right),$$

kde $\mathbf{g}_i(x)$ značí i-tý řádek $G(x)$.

Důkaz. V důkazu postupně dokážeme (redukovanost) $\Rightarrow$ (a) $\Rightarrow$ (b) $\Rightarrow$ (redukovanost) a pak $(a) \Leftrightarrow (c)$.

- Implikaci (*redukovanost*) $\Rightarrow$ (a) dokážeme nepřímo: nechť podmínka (a) neplatí, tedy existuje nenulový vektor délky k nad tělesem $\mathbb{Z}_2$, řekněme $\alpha = (\alpha_1, \ldots, \alpha_k)$, takový, že $\alpha\overline{H} = 0$. Označme $G(x) = (\mathbf{g}_1(x), \ldots, \mathbf{g}_k(x))$ řádky $G(x)$, přičemž $\deg \mathbf{g}_i(x) = f_i$ a $f_1 \le f_2 \le \cdots \le f_k$. Pak z $\alpha\overline{H} = 0$ plyne, že koeficient u x^{f_k} v lineární kombinaci

  $$\mathbf{g'}_k(x) = \alpha_1 x^{f_k - f_1}\mathbf{g}_1(x) + \alpha_2 x^{f_k - f_2}\mathbf{g}_2(x) + \cdots + \alpha_k x^{f_k - f_k}\mathbf{g}_k(x)$$

  je 0, takže transformace, která řádek $\mathbf{g}_k(x)$ nahradí $\mathbf{g'}_k(x)$, sníží vnější stupeň $G(x)$. Protože matice této transformace je unimodulární, získali jsme generující matici s menším vnějším stupněm než má $G(x)$, ergo $G(x)$ není redukovaná.

- $(a) \Rightarrow (b)$: Předpokládejme, že $\overline{H}$ má hodnost k a označme $\overline{H}_\nu$ podmatice $\overline{H}$ typu $k \times k$, $\nu = 1, 2, \ldots, \binom{n}{k}$. Pak protože hodnost matice $\overline{H}$ je k, existuje alespoň jedno ν_0 takové, že $\det \overline{H}_{\nu_0} \neq 0$. Jsou-li stupně řádků $G(x)$ $f_1, \ldots, f_k$, pak koeficient u $x^{f_1+\cdots+f_k}$ v $\det G_{\nu_0}$ je $\det \overline{H}_{\nu_0} \neq 0$. Tedy $\operatorname{intdeg} G(x) \ge \operatorname{extdeg} G(x)$, a protože opačná nerovnost platí vždy (viz věta 3.1.5 (b)), dostáváme žádanou rovnost $\operatorname{intdeg} G(x) = \operatorname{extdeg} G(x)$.

- $(b) \Rightarrow$ (*redukovanost*): Nechť $\operatorname{intdeg} G(x) = \operatorname{extdeg} G(x)$ a $T(x)$ je libovolná unimodulární matice typu $k \times k$. Potom

  $$\operatorname{extdeg} T(x)G(x) \ge \operatorname{intdeg} T(x)G(x) = \operatorname{intdeg} G(x) = \operatorname{extdeg} G(x),$$

  kde první nerovnost máme z věty 3.1.5 (b), druhou rovnost z věty 3.1.5 (a) a třetí z předpokladu. Celkem jsme tedy dokázali, že $\operatorname{extdeg} T(x)G(x) \ge \operatorname{extdeg} G(x)$, a tedy $G(x)$ je redukovaná.

- Na závěr ukažme ekvivalenci bodu (a) s (c): nechť $\mathbf{u}(x) = (u_1(x), \ldots, u_k(x))$ je vektor polynomů a nechť $\mathbf{w}(x) = (w_1(x), \ldots, w_n(x))$ je vektor polynomů takový, že $\mathbf{w}(x) = \mathbf{u}(x)G(x)$. Při uvedeném značení řádků matice $G(x)$ tedy máme, že

$$\mathbf{w}(x) = u_1(x)\mathbf{g}_1(x) + \cdots + u_k(x)\mathbf{g}_k(x).$$

  Označme b_i stupeň polynomu $u_i(x)$, $i = 1, \ldots, k$ a f_i tradičně stupeň řádku $\mathbf{g}_i(x)$. Potom z uvedené rovnice plyne, že stupeň $\mathbf{w}(x)$ je nanejvýš $b = \max_i (b_i + f_i)$. Zajímat nás bude, za jakých okolností přechází nerovnost v rovnost. Označme α_i koeficient x^{b-f_i} v $u_i(x)$. Pak vektor koeficientů x^b v $\mathbf{w}(x)$ bude $\alpha = (\alpha_1, \ldots, \alpha_k)\overline{H}$, neboť alespoň pro jedno i je $\alpha_i \neq 0$ ($b_i + f_i = b$ musí platit alespoň pro jedno i). Ovšem $\alpha\overline{H} \neq 0$ platí pro všechny nenulové α právě tehdy, když hodnost $\overline{H}$ je k, a tedy rovnost (kterou jsme chtěli dokázat) platí právě tehdy, má-li $\overline{H}$ hodnost k.

□

3.3 Kanonické kodéry

Definice 3.3.1. Řekneme, že matice $G(x)$ typu $k \times n$ generující $[n, k]$ kód $\mathbf{C}$ je *kanonická*, jestliže je polynomiální a má minimální možný vnější stupeň mezi všemi polynomiálními ekvivalentními generujícími maticemi. Tento stupeň se nazývá *stupeň kódu* $\mathbf{C}$, značíme $\deg \mathbf{C}$ nebo t; $[n, k]$ konvoluční kód stupně t značíme také jako $[n, k, t]$ kód.

Poznámka 3.3.2. Velice často se lze v popisu konvolučních kódů setkat s pojmem *constraint length*, který nicméně není rigorózně definován, resp. jeho definice se v různých pracech liší. Jedna z nich odpovídá i před okamžikem zavedenému stupni kódu t.

Pro polynomiální generující matice existuje kritérium, pomocí kterého lze zjistit, zda je matice kanonická:

Věta 3.3.3. *Polynomiální generující matice $G(x)$ konvolučního kódu $\mathbf{C}$ je kanonická právě tehdy, když je základní a současně redukovaná.*

Důkaz. Nejprve dokážeme implikaci $\Rightarrow$. Označme h_0 vnitřní stupeň základních generujících matic kódu $\mathbf{C}$, z těchto matic vyberme tu s nejmenším

vnějším stupněm a označme ji $G_0(x)$. Pak $G_0(x)$ je redukovaná, protože máme-li libovolnou unimodulární matici $T(x)$, tak podle věty 3.1.5 jest intdeg $T(x)G_0(x)$ = intdeg $G_0(x)$ = h_0, a tedy z definice $G_0(x)$ platí extdeg $T(x)G_0(x) \geq$ extdeg $G_0(x)$. Nyní nechť je $G(x)$ kanonická generující matice. Pak musí platit intdeg $G_0(x) \leq$ intdeg $G(x) \leq$ extdeg $G(x) \leq$ extdeg $G_0(x)$. Ovšem $G_0(x)$ je redukovaná, takže intdeg $G_0(x) =$ extdeg $G_0(x)$, tedy ve výše uvedených nerovnostech všude platí rovnosti. Speciálně tedy intdeg $G(x) =$ intdeg $G_0(x) = h_0$, tedy $G(x)$ je základní, a intdeg $G(x) =$ extdeg $G(x)$, tedy $G(x)$ je redukovaná.

Naopak předpokládejme, že $G(x)$ je základní a redukovaná a nechť $G'(x)$ je libovolná polynomiální generující matice kódu **C**. Dostáváme vztahy

$$\text{extdeg } G'(x) \geq \text{intdeg } G'(x) \geq \text{intdeg } G(x) = \text{extdeg } G(x),$$

které plynou po řadě z věty 3.1.5 o vnitřním a vnějším stupni, definice základní matice 3.1.2 a věty 3.2.2 o redukovaných maticích. Tedy celkem extdeg $G'(x) \geq$ extdeg $G(x)$, z čehož plyne, že $G(x)$ je kanonická. □

Kanonické generující matice nejsou nutně jednoznačně určené, nicméně následující věta říká, že si jsou v jistém smyslu velmi podobné:

Věta 3.3.4. *Pro každý konvoluční kód* **C** *je množina stupňů řádků jeho libovolné kanonické generující matice stejná.*

Důkaz. Označme $e_1 \leq e_2 \leq \cdots \leq e_k$ a $f_1 \leq f_2 \leq \cdots \leq f_k$ stupně řádků různých kanonických generujících matic $G(x)$ a $G'(x)$ kódu **C**. Pokud by neplatilo $e_i \leq f_i \ \forall i = 1, \ldots, k$, existoval by index $j < k$ takový, že $e_1 \leq f_1, \ldots, e_j \leq f_j$, ale $e_{j+1} > f_{j+1}$. $G(x)$ je kanonická, tedy i základní a redukovaná a z jejich charakterizací — věta 3.1.6 (e) a věta 3.2.2 (c) — plyne, že by prvních $j+1$ řádků $G'(x)$ muselo být lineární kombinací prvních j řádků $G(x)$, což je ve sporu s lineární nezávislostí řádků $G'(x)$. □

Definice 3.3.5. Stupně řádků $e_1, \ldots, e_k$ z důkazu věty 3.3.4 se nazývají *Forneyho indexy* kódu **C**. Maximální Forneyho index se nazývá *paměť kódu*.

Poznámka 3.3.6. Povšimněme si, že součet Forneyho indexů $e_1 + e_2 + \cdots + e_k$ je roven stupni kódu **C** z definice 3.3.1, hovoříme o $[n, k, t]$ kódech. Jak to je v našem případě? Matice $G(x)$ je zřejmě základní, neboť NSD$(1+x+x^2,\ 1+x^2) = 1$ a také redukovaná, neboť extdeg $G(x) = 2 =$ intdeg $G(x)$. Podle charakterizace kanonických matic je tedy $G(x)$ kanonická, a tedy stupně jejích řádků jsou Forneyho indexy kódu **C**. Zřejmě $e_1 = 2$, a tedy $t = 2$. „Náš“ kód má tedy parametry $[2, 1, 2]$.

3.4 Katastrofické kodéry

V této sekci si ukážeme, že ne všechny kodéry mají příznivé vlastnosti.

Příklad 3.4.1. Vezměme si generující matici

$$G_{kat}(x) = \begin{pmatrix} 1+x & 1+x^2 & x \\ x+x^2 & 1+x & 0 \end{pmatrix}$$

a vstup $V(x) = \left(0,\ \frac{1}{1+x}\right)$, což je slovo nekonečné váhy, neboť

$$\frac{1}{1+x} = 1 + x + x^2 + x^3 + \dots$$

Po zakódování dostaneme

$$C(x) = V(x)G_{kat}(x) = (x,\ 1,\ 0)\,,$$

což je sekvence o Hammingově váze 2. Předpokládejme, že při průchodu kanálem nastaly chyby na nenulových pozicích a na druhé straně bylo přijato právě nulové slovo. Dá se vcelku rozumně předpokládat, že dekódování nulového slova skončí nulovým výsledkem, tedy že $v_{dek}(x) = (0, 0)$, což ovšem znamená, že konečný počet chyb při přenosu způsobí nekonečně mnoho chyb při dekódování — označení dekódovací katastrofa je plně na místě.

Definice 3.4.2. [6] Řekneme, že konvoluční kodér s maticí $G(x)$ je *katastrofický*, jestliže existuje slovo nekonečné Hammingovy váhy, které po zakódování má váhu konečnou.

Zdůrazněme nyní, že označení „katastrofický“ padá na vrub kodéru, nikoliv kódu jako takového. Konvoluční kód může totiž být generován různými kodéry (resp. generujícími maticemi) s diametrálně odlišnými vlastnostmi. K popisu situací, v nichž je či není kodér katastrofický, se nám bude hodit definice zprava invertibilní matice 3.1.3.

Věta 3.4.3. [6] *Kodér s generující maticí $G(x)$ není katastrofický, právě když existuje zprava inverzní matice $G^{-1}(x)$, která má za prvky pouze polynomy.*

Důkaz. Nejprve dokážeme implikaci $\Leftarrow$. Máme tedy, že $\exists G^{-1}(x)$. Teoreticky (za předpokladu, že při přenosu nedošlo k žádným chybám) lze určit původní zprávu jako

$$C(x)G^{-1}(x) = V(x)G(x)G^{-1}(x) = V(x).$$

Nechť nyní $V(x)$ má nekonečnou Hammingovu váhu, pak (protože $G^{-1}(x)$ je polynomiální, a tedy její členy mají konečný počet koeficientů) i $C(x)$ má nekonečnou Hammingovu váhu a kodér není katastrofický.

Implikaci $\Rightarrow$ dokážeme nepřímo. Nechť $\exists G^{-1}(x)$, jejíž právě jeden člen je racionální funkce $\frac{p(x)}{q(x)}$, pak pro $c(x)$ takové, že $q(x) \nmid c(x)$, platí, že váha $m(x)$ je rovna ∞, a tedy kodér je katastrofický. Případ, kdy v $G^{-1}(x)$ je více než jeden člen racionální funkce, je nyní zřejmý. Pokud $G^{-1}(x)$ neexistuje, matice $G(x)$ nemá hodnost k, a tedy není generující maticí $[n, k]$ kódu. □

3.5 Systematické kodéry

Jak jsme viděli v části věnované základním kodérům, můžeme libovolnou generující matici převést na základní. Mohlo by se zdát, že se lze omezit pouze na polynomiální generující matice, ovšem v této části ukážeme, že takové zjednodušení není na místě. Právě naopak, v některých případech je výrazně výhodnější využít matic s racionálními členy, pokud jsou tzv. *systematické*. Tato třída generujících matic je z praktického hlediska velmi významná, neboť je základem již zmíněných turbokódů, které se svými vlastnostmi blíží ideálnímu informačnímu poměru.

Definice 3.5.1. Řekneme, že generující matice $G(x)$ typu $k \times n$ je systematická, jestliže (po případném přeházení řádků a sloupců) obsahuje jednotkovou matici I_k.

Jak je vidět z definice, v kódovém slově se tedy v nějakém pořadí (ne nutně identickém) objeví vstupní slovo. Speciálně tedy systematické kodéry nemohou být katastrofické — je-li vstupní slovo nekonečné váhy, je nutně nekonečné váhy i kódové slovo. Nyní ukážeme, že každý kód má systematický kodér:

Věta 3.5.2. *Pro každý konvoluční kód existuje systematický kodér.*

Důkaz. Bez újmy na obecnosti předpokládejme, že máme $G(x)$ základní generující matici[1], tedy existuje polynomiální pravý inverz $G^{-1}(x)$. Pro něj platí

$$G^{-1}(x) = (A(x)\Gamma(x)B(x))^{-1} = B^{-1}(x)\Gamma^{-1}(x)A^{-1}(x)$$

[1]Pokud máme racionální nebo polynomiální bez pravého inverzu, podle věty 3.1.7 ji můžeme na základní převést.

a protože $G^{-1}(x)$, $A^{-1}(x)$ i $B^{-1}(x)$ jsou polynomiální, musí být polynomiální i $\Gamma^{-1}(x)$. Z toho ovšem plyne, že $\gamma_i(x) = 1\ \forall i = 1, 2, \ldots, k$. Tedy největší společný dělitel všech subdeterminantů $G(x)$ velikosti $k \times k$ je roven 1, z čehož vyplývá, že existuje subdeterminant D_k velikosti $k \times k$, který není polynom v x (jinak by největší společný dělitel byl dělitelný x).

Přeházejme sloupce $G(x)$ tak, aby na prvních k pozicích vznikla matice $T(x)$, jejímž determinantem je D_k. Protože $T(x)$ je invertibilní (je typu $k \times k$ a $\det(T(x)) \neq 0$), můžeme napsat

$$G_{sys}(x) = T^{-1}(x)G(x) = (\ I_k \quad P(x)\ ),$$

kde $P(x)$ se nazývá *paritní* matice typu $(n-k) \times k$ (ne nutně polynomiální). $G_{sys}(x)$ je již zřejmě systematickou maticí ekvivalentní s $G(x)$. □

Síla a užitečnost systematických kódů spočívá v jejich dalším použití v turbokódech, samy o sobě nemusí nabízet nejlepší výkon. Z [6] pochází tabulka 3.1, kde $L = \max_{i,j} \deg(g_{ij}(x)) + 1$ je největší stupeň transformujících funkcí z $G(x)$ zvětšený o 1[2]. Porovnávají se v ní hodnoty minimální vzdálenosti[3] u polynomiálních systematických a nesystematických kodérů pro $[3, 1]$ kódy. Ze stejného zdroje máme i obdobnou tabulku 3.2 pro $[2, 1]$ kódy.

Tabulka 3.1: srovnání minimální vzdálenosti systematických a nesystematických polynomiálních $[3, 1]$ kódů

L	d_{sys}	d_{nesys}
2	3	3
3	4	5
4	4	6
5	5	7
6	6	8
7	6	8
8	7	10

Z tabulek jasně plyne, že se zvětšujícím se stupněm kodéru rostou rozdíly mezi systematickými a nesystematickými kodéry ve prospěch druhých jmenovaných, je proto nevýhodné používat samotné systematické kodéry.

[2]Tato hodnota se v mnoha pracech nazývá taktéž *constraint length*.

[3]Pro definici a některé vlastnosti viz kapitola 4.

Tabulka 3.2: srovnání minimální vzdálenosti systematických a nesystematických polynomiálních $[2, 1]$ kódů

L	d_{sys}	d_{nesys}
2	5	5
3	6	8
4	8	10
5	9	12
6	10	13
7	12	15
8	12	16

3.6 Příklady

V této sekci aplikujeme právě nabyté znalosti na konkrétních příkladech. Následujících osm matic $G_1(x)$ až $G_8(x)$ generuje ten samý $[4, 2]$ konvoluční kód (každá z matic je ekvivalentní $G_1(x)$, jak je ukázáno):

$$G_1(x) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1+x & x & 1 \end{pmatrix},$$

$$G_2(x) = \begin{pmatrix} 1+x & 0 & 1 & x \\ 1 & x & 1+x & 0 \end{pmatrix} = \begin{pmatrix} 1+x & 1 \\ 1 & 1 \end{pmatrix} G_1(x),$$

$$G_3(x) = \begin{pmatrix} 1 & 1+x+x^2 & 1+x^2 & 1+x \\ x & 1+x+x^2 & x^2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & x \\ x & 1+x \end{pmatrix} G_1(x),$$

$$G_4(x) = \begin{pmatrix} \frac{1}{1+x+x^2} & 1 & \frac{1+x^2}{1+x+x^2} & \frac{1+x}{1+x+x^2} \\ 1 & \frac{1+x+x^2}{x} & x & \frac{1}{x} \end{pmatrix} = \begin{pmatrix} \frac{1}{1+x+x^2} & \frac{x}{1+x+x^2} \\ 1 & \frac{1+x}{x} \end{pmatrix} G_1(x),$$

$$G_5(x) = \begin{pmatrix} 1 & 1+x+x^2 & 1+x^2 & 1+x \\ 0 & 1+x & x & 1 \end{pmatrix} = \begin{pmatrix} 1 & x \\ 0 & 1 \end{pmatrix} G_1(x),$$

$$G_6(x) = \begin{pmatrix} 1 & x & 1+x & 0 \\ 0 & 1+x & x & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} G_1(x),$$

$$G_7(x) = \begin{pmatrix} 1+x & 0 & 1 & x \\ x & 1+x+x^2 & x^2 & 1 \end{pmatrix} = \begin{pmatrix} 1+x & 1 \\ x & 1+x \end{pmatrix} G_1(x),$$

$$G_8(x) = \begin{pmatrix} 1 & 0 & \frac{1}{1+x} & \frac{x}{1+x} \\ 0 & 1 & \frac{x}{1+x} & \frac{1}{1+x} \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{1+x} \\ 0 & \frac{1}{1+x} \end{pmatrix} G_1(x).$$

Budeme se zajímat o to, jaký je vnitřní a vnější stupeň každé matice a jestli náleží do některé z výše uvedených tříd konvolučních kódů. Jednotlivé údaje (založené na teoretických výsledcích této kapitoly) jsou uvedeny v tabulce 3.3.

Tabulka 3.3:

matice	intdeg	extdeg	základní?	redukovaná?	kanonická?	systematická?
$G_1(x)$	1	1	ano	ano	ano	ne
$G_2(x)$	2	2	ne	ano	ne	ne
$G_3(x)$	3	4	ne	ne	ne	ne
$G_4(x)$	–	–	–	–	–	ne
$G_5(x)$	1	3	ano	ne	ne	ne
$G_6(x)$	1	2	ano	ne	ne	ano
$G_7(x)$	3	3	ne	ano	ne	ne
$G_8(x)$	–	–	–	–	–	ano

Kapitola 4

Minimální vzdálenost konvolučních kódů

V této části se budeme snažit pomocí Hilbertových řad odvodit odhady na minimální vzdálenost konvolučního $[n, k]$ kódu, zavedeme také třídu optimálních konvolučních kódů jakožto kódů s největší minimální vzdáleností (a tedy největší schopností opravit chyby) při zadaném informačním poměru.

Definice 4.0.1. Nechť $\mathbf{u}$ a $\mathbf{v}$ jsou vektory nad tělesem $\mathbb{Z}_2$, pak přirozeně definujeme jejich Hammingovu vzdálenost $d_H(\mathbf{u}, \mathbf{v})$ jako počet míst, na kterých se tyto vektory liší.

Definice 4.0.2. Mějme $[n, k, t]$ konvoluční kód $\mathbf{C}$. Jeho minimální vzdálenost d definujeme jako $d = \min\{d_H(\mathbf{u}, \mathbf{v}) | \mathbf{u} \neq \mathbf{v} \in \mathbf{C}\}$, hovoříme o $[n, k, t, d]$ kódu.

Mějme kód $\mathbf{C}$, označme $\mathbf{C}_L$ množinu všech polynomiálních kódových slov $\mathbf{C}$ stupně $\leq L$ (tj. množinu všech kódových slov, které obsahují pouze polynomy stupně $\leq L$ — *polynomiální podkód* kódu $\mathbf{C}$). Pak $\mathbf{C}_L$ je podmnožinou množiny všech polynomiálních vektorů stupně $\leq L$, ale také vektorový prostor nad $\mathbb{Z}_2$ — označme proto δ_L jeho dimenzi nad dvouprvkovým tělesem. Ta jde spočítat z Forneyho indexů kódu $\mathbf{C}$:

Věta 4.0.3. *Nechť $\mathbf{C}$ je $[n, k]$ konvoluční kód s Forneyho indexy $(e_1, \ldots, e_k)$ a dimenzemi polynomiálních podkódů δ_L. Pak*

$$\sum_{L \geq 0} \delta_L z^L = \frac{z^{e_1} + \cdots + z^{e_k}}{(1 - z)^2}.$$

Důkaz. Nechť $G(x)$ je kanonická generující matice kódu $\mathbf{C}$, jejíž stupně řádků jsou Forneyho indexy $e_1 \leq e_2 \leq \cdots \leq e_k$, dále nechť $\mathbf{g}_1(x), \ldots, \mathbf{g}_k(x)$ jsou řádky $G(x)$ a $C(x)$ libovolné polynomiální kódové slovo stupně $\leq L$. Pak z charakterizace základních generujících matic (věta 3.1.6 (e)) plyne, že $C(x) = V(x)G(x)$, kde $V(x) = (v^{(1)}(x), \ldots, v^{(k)}(x))$ je taktéž vektor polynomů. Z charakterizace redukovaných generujících matic (věta 3.2.2 (c)) máme, že $\deg v^{(i)}(x) + e_i \leq L$ pro $i = 1, \ldots, k$. Tedy báze $\mathbf{C}_L$ nad $\mathbb{Z}_2$ je rovna $\{x^j \mathbf{g}_i(x) | j + e_i \leq L\}$. Proto

$$\sum_{L\geq 0} \delta_L z^L = \sum_{i=1}^{k} \sum_{j\geq 0} (z^{e_i+j} + z^{e_i+j+1} + \cdots) = \sum_{i=1}^{k} \sum_{j\geq 0} \frac{z^{e_i+j}}{1-z} = \sum_{i=1}^{k} \frac{z^{e_i}}{(1-z)^2}.$$

□

Poznámka 4.0.4. Řada $\sum_{L\geq 0} \delta_L z^L$ z věty 4.0.3 se nazývá *Hilbertova řada* kódu $\mathbf{C}$.

Exaktní vzorec na výpočet δ_L zní takto:

Důsledek 4.0.5. *Za předpokladů předcházející věty platí:*

$$\delta_L = \sum_{i=1}^{k} \max\{L + 1 - e_i,\ 0\}.$$

Důkaz. V důkazu důsledku využijeme faktu, že $(1-z)^{-2} = \sum_{j\geq 0}(j+1)z^j$, což se dá snadno ověřit: označme $F'(z) = \sum_{j\geq 0}(j+1)z^j$, pak $F(z) = \sum_{j\geq 0} z^{j+1} = \frac{z}{1-z}$. Proto $F'(z) = \left(\frac{z}{1-z}\right)' = \frac{1}{(1-z)^2}$. Po dosazení do věty 4.0.3 dostáváme následující:

$$\sum_{L\geq 0} \delta_L z^L = \sum_{i=1}^{k} \frac{z^{e_i}}{(1-z)^2} = \sum_{i=1}^{k} \sum_{j\geq 0} (j+1) z^{e_i+j} = \sum_{i=1}^{k} \sum_{j\geq e_i} (j+1-e_i) z^j.$$

Tedy koeficient u z^L v Hilbertově řadě kódu $\mathbf{C}$ je $\sum_{i=1}^{k} \max\{L+1-e_i, 0\}$, což jsme chtěli dokázat.

□

Skutečný odhad minimální vzdálenosti nám pak dává tato věta:

Věta 4.0.6. *Nechť* $\mathbf{C}$ *je* $[n, k, t]$ *konvoluční kód nad tělesem* $\mathbb{Z}_q$ *s Forneyho indexy* $(e_1, \ldots, e_k)$*, pak*

$$d \leq \min_{L \geq 0} \{\Delta_q \left(n(L+1),\ k(L+1) - t\right)\},$$

kde $\Delta_q(n, k)$ *značí největší možnou minimální vzdálenost* $[n, k]_q$ *lineárního blokového kódu.*

Definice 4.0.7. Řekneme, že $[n, k, t, d]$ konvoluční kód je *optimální*, jestliže má mezi všemi $[n, k, t]$ konvolučními kódy největší minimální vzdálenost d.

Definice 4.0.8. Řekneme, že $[n, k, t]$ konvoluční kód je *kompaktní*, jestliže pro všechna $L \geq 0$ platí $\delta_L = \max\{(L+1)k - t,\ 0\}$.

Příklad 4.0.9. Vraťme se zpět k našemu $[2, 1, 2]$ kódu. Protože $k = 1$, jest $\delta_L = \max\{L + 1 - e_1,\ 0\}$ a $t = e_1$. Tedy kód je podle definice kompaktní, navíc $\delta_0 = 0$ a $\delta_L = L - 1$ pro $L \geq 1$, tedy můžeme odvodit minimální vzdálenost jako

$$d \leq \min\{\Delta(2, 0), \Delta(4, 0), \Delta(6, 1), \Delta(8, 2), \Delta(10, 3), \ldots\},$$

po dosazení

$$d = \min\{\infty, \infty, 6, 5, 5, \ldots\} = 5.$$

Každý $[2, 1, 2]$ kód tedy má minimální vzdálenost nejvýše 5. Je známo, že v našem případě je $d = 5$, a tedy jedná se o optimální $[2, 1, 2]$ kód. Ověření probíhá následovně: chceme dvě cesty v mřížce kódu, které se liší v nejmenším počtu hran. Nechť máme dvě cesty, které se v nějakém uzlu rozdělí, tedy na vstup přišly rozdílné bity. Protože stav kodéru je vektor délky 2, tyto cesty se mohou setkat nejdříve za další dva kroky (za předpokladu, že nadále na vstupu budou ty samé bity). Máme celkem 16 možností takovýchto rozdělení dvou cest (4 možnosti stavu, za kterého se cesty rozdělily, a 4 možnosti následného vstupu délky 2) — když všechny vypíšeme, zjistíme, že skutečně se takto rozdělené cesty liší na pěti pozicích.

Nabízí se otázka, zda pro každou trojici parametrů $[n, k, t]$ existuje optimální konvoluční kód — odpověď je záporná, nicméně její důkaz je nad rámec této práce.

Kapitola 5

Dekódování konvolučních kódů

V předchozích kapitolách jsme si vysvětlili princip kódování, odvodili vlastnosti různých tříd konvolučních kódů v závislosti na tvaru generující matice a odvodili odhad na minimální vzdálenost, přeneseně tedy schopnost kódu opravovat chyby vzniklé při přenosu. V této části se zaměříme na to, jak z přijatého vektoru $\mathbf{r}$ získat zpět vyslané kódové slovo $\mathbf{c}$.

Použijeme následující model: vstupní vektor $\mathbf{v}$ se zprava doplní nulami tak, aby po posledním vstupu byly v registrech kodéru samé nuly, čímž vznikne vektor $\mathbf{w}$, zakódováním dostaneme $\mathbf{c}$ a po přenosu $\mathbf{r}$ s případnými chybami[1]. Zopakujme značení:

- Vstupní slovo v čase i o k bitech označme $\mathbf{v}_t = \left(v_t^{(1)}, \ldots, v_t^{(k)}\right)$, analogicky vektory s nulami navíc jsou označeny $\mathbf{w}_t$. Sekvenci L bloků na vstupu označme $\mathbf{w} = (\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{L-1})$.

- Stejně jako doposud označíme i příslušné kódové bity $c_i^{(j)}$, $i = 1, 2, \ldots, n$ tvořící vektor $\mathbf{c}_i$. Celá výstupní sekvence pak je $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{L-1})$. Přijaté slovo ma totožnou notaci:

$$\mathbf{r} = (\mathbf{r}_0, \ldots, \mathbf{r}_{L-1}) = (r_0^{(1)}, r_0^{(2)}, \ldots, r_0^{(n)}, \ldots, r_{L-1}^{(1)}, \ldots, r_{L-1}^{(n)}).$$

- Jako přenosový kanál zvolme *binární symetrický kanál (BSC)*, tj. binární kanál, jež má pravděpodobnost chyby při přenosu p (typicky se

[1]Mezi $\mathbf{c}$ a $\mathbf{r}$ je obvykle ještě jedna fáze, tzv. modulace, kdy se kódové slovo mapuje na signál vhodný k přenosu; z matematického hlediska je však nezajímavá a můžeme ji proto vypustit. V našem případě, kdy uvažujeme binární symetrický kanál, je navíc tato modulace identitou.

předpokládá, že $p < \frac{1}{2}$). Chybovou sekvenci označme $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_{L-1})$, pak přijaté bity budou odpovídat

$$r_i^{(j)} = c_i^{(j)} \oplus a_i^{(j)},$$

kde $\oplus$ značí sčítání modulo 2 a $a_i^{(j)} \sim \mathcal{B}(p)$, kde $\mathcal{B}(p)$ je Bernoulliova náhodná veličina. Navíc pro jednoduchost předpokládejme, že $a_i^{(j)}$ jsou nezávislé (tedy že kanál je tzv. *bez paměti*).

Potřebujeme najít nějaké kódové slovo $\mathbf{c_m} \in \mathbf{C}$ takové, že jeho Hammingova vzdálenost od přijatého slova $\mathbf{r}$ je nejmenší možná. Chceme tedy navrhnout algoritmus, který by v celém kódu $\mathbf{C}$ našel slovo $\mathbf{c_m}$, jež by bylo nejbližší přijatému slovu $\mathbf{r}$. První, naivní myšlenka velí spočítat Hammingovu vzdálenost pro všechna slova kódu $\mathbf{C}$ a pak vybrat žádané, již na první pohled je ale jasné, že to je výpočetně prakticky nemožné. V tuto chvíli nastupuje Viterbiův algoritmus.

Poznámka 5.0.10. Nemusí být na první pohled jasné, proč chceme minimalizovat právě Hammingovu vzdálenost. Označme $P(\mathbf{r}_i|\mathbf{c}_i)$ podmíněnou pravděpodobnost, že bylo přijato slovo $\mathbf{r}_i$ za předpokladu, že bylo vysláno slovo $\mathbf{c}_i$ a $d_H(\mathbf{r}_i, \mathbf{c}_i)$ Hammingovu vzdálenost vektorů $\mathbf{r}_i$ a $\mathbf{c}_i$. Pak můžeme psát

$$P(\mathbf{r}_i|\mathbf{c}_i) = p^{d_H(\mathbf{r}_i,\mathbf{c}_i)}(1-p)^{n-d_H(\mathbf{r}_i,\mathbf{c}_i)} = \left(\frac{p}{1-p}\right)^{d_H(\mathbf{r}_i,\mathbf{c}_i)} (1-p)^n.$$

Nyní rozšíříme myšlenku na celé sekvence, neboť nestačí maximalizovat pravděpodobnost pro jednotlivá slova — vzpomeňme, že kódová slova závisejí i na vnitřním stavu kodéru, který pro změnu závisí i na předchozích vstupech. Tedy

$$P(\mathbf{r}|\mathbf{c}) = P(\mathbf{r}_0,\ \mathbf{r}_1, \ldots,\ \mathbf{r}_{L-1}|\mathbf{c}_0,\ \mathbf{c}_1, \ldots,\ \mathbf{c}_{L-1}) = \prod_{i=0}^{L-1} f(\mathbf{r}_i|\mathbf{c}_i),$$

kde jsme v poslední rovnosti použili nezávislost chyb.

Podobně jako u blokových kódů se realizuje myšlenka hledání maximálně věrohodného kódového slova, tj. slova $\mathbf{c_m}$ s co nejvyšší pravděpodobností $P(\mathbf{r}|\mathbf{c_m})$. Ale protože $p < \frac{1}{2}$, máme

$$\frac{p}{1-p} < \frac{\frac{1}{2}}{\frac{1}{2}} < 1,$$

tedy funkce $\left(\frac{p}{1-p}\right)^k$ je klesající pro rostoucí $k \in \mathbb{N}$, a tedy podmíněná pravděpodobnost $P(\mathbf{r}_i|\mathbf{c}_i) = \left(\frac{p}{1-p}\right)^{d_H(\mathbf{r}_i,\mathbf{c}_i)}(1-p)^n$ klesá pro rostoucí Hammingovu vzdálenost kódového a přijatého slova $d_H(\mathbf{r}_i, \mathbf{c}_i)$. Maximalizace $P(\mathbf{r}|\mathbf{c_m})$ je tak ekvivalentní minimalizaci $d_H(\mathbf{r},\ \mathbf{c_m})$.

5.1 Viterbiův algoritmus

Viterbiův algoritmus je založen na jednoduché myšlence: počítání věrohodnostní funkce všech kódových slov obnáší vlastně projití všech možných cest v mřížce. Je ale zjevné, že mnoho cest bude procházeno zbytečně, neboť bude zřejmé, že pro některé z nich bude funkce již po několika krocích výrazně větší než u jiných. Definujme nyní *metriku* cesty a větve:

Definice 5.1.1. Nechť $\hat{\mathbf{w}}_0^{i-1} = (\hat{\mathbf{w}}_0, \ldots,\ \hat{\mathbf{w}}_{i-1})$ je sekvence vstupních slov, která definuje cestu mřížkou $\Pi_i = \{\mathbf{s}_0,\ \mathbf{s}_1, \ldots,\ \mathbf{s}_t\}$ a která „zanechá“ kodér ve stavu $\mathbf{s}_i$. Nechť jí odpovídá $\hat{\mathbf{c}}_0^{i-1} = (\hat{\mathbf{c}}_0, \ldots,\ \hat{\mathbf{c}}_{i-1})$ sekvence kódových slov. Pak definujeme *metriku cesty* Π_i jako

$$M_{i-1}(\mathbf{s}_i) = -\log P(\mathbf{r}_0,\ \mathbf{r}_1, \ldots,\ \mathbf{r}_{i-1}|\hat{\mathbf{c}}_0,\ \hat{\mathbf{c}}_1, \ldots,\ \hat{\mathbf{c}}_{i-1}).$$

Definice 5.1.2. Řekneme, že hodnota $\mu_i(\mathbf{r}_i,\ \hat{\mathbf{c}}_i)$, definovaná jako $\mu_i(\mathbf{r}_i,\ \hat{\mathbf{c}}_i) = -\log P(\mathbf{r}_i|\hat{\mathbf{c}}_i)$, se nazývá *metrika větve.*

Je zjevné, že metrika celé cesty je součtem jednotlivých metrik větví:

$$M_i(\mathbf{s}_{i+1}) = \sum_{j=0}^{i} \log P(\mathbf{r}_j|\hat{\mathbf{c}}_j) = \sum_{j=0}^{i} \mu_j(\mathbf{r}_j|\hat{\mathbf{c}}_j) = M_{i-1}(\mathbf{s}_i) + \mu_i(\mathbf{r}_i|\hat{\mathbf{c}}^{\mathbf{s}_i,\mathbf{s}_{i+1}}),$$

kde $\hat{\mathbf{c}}^{(\mathbf{s}_i,\mathbf{s}_{i+1})}$ je kódové slovo příslušné vstupnímu $\hat{\mathbf{w}}^{(\mathbf{s}_i,\mathbf{s}_{i+1})}$, které kodér ze stavu $\mathbf{s}_i$ posune do stavu $\mathbf{s}_{i+1}$. Prodloužení cesty o jeden krok v mřížce tedy vlastně znamená k dosavadní metrice cesty přičíst metriku nové větve.

Poznámka 5.1.3. Takto zřejmě převedeme úlohu maximalizace věrohodnostní funkce na minimalizaci metriky cesty. Jak jsme však ukázali výše, v našem jednoduchém případě binárního symetrického kanálu nám stačí minimalizovat Hammingovu vzdálenost — obě definice jsou zde uvedeny pro srovnání s dalšími dekódovacími algoritmy, kde již podobné zjednodušení nebude možné. Hammingova vzdálenost je navíc metrika, proto v dalším textu popisujícím Viterbiův algoritmus můžeme uvažovat pouze ji a pod pojmem

metrika větve rozumět $\mu_i(\mathbf{r}_i|\hat{\mathbf{c}}_i) = d_H(\mathbf{r}_i,\ \hat{\mathbf{c}}_i)$ a pod *metrikou cesty* příslušnou sumu.

Na pozorování, že metrika cesty je součtem metrik větví, je založen Viterbiův algoritmus: začneme ve stavu $\mathbf{s}_0$, o kterém je známo, jak vypadá (je to výchozí stav kodéru, tedy většinou samé nuly), projdeme každou hranu, která z $\mathbf{s}_0$ vychází, spočítáme pro ni metriku větve a do cílového stavu $\mathbf{s}_1$ ji uložíme.

Ve druhém kroku z každého stavu opět projdeme všechny hrany, spočítáme jejich metriky, přičteme k metrikám, které již cesta před prodloužením o jednu hranu měla, a výsledek uložíme do cílového stavu. Takto projdeme postupně všechny stavy a na konci se podíváme, který z nich má nakumulovanou nejmenší metriku — ta bude odpovídat nejpravděpodobnějšímu slovu $\mathbf{c_m}$. Tímto postupem sice najdeme pouze metriku, nikoliv příslušné kódové slovo, nicméně budeme-li si při procházení kromě metriky pamatovat i příslušnou cestu, můžeme na konci kódové slovo zrekonstruovat.

Hlavní myšlenka Viterbiova algoritmu spočívá v řešení situace, kdy se dvě cesty setkají, což je v mřížce konvolučního kódu velmi častý jev. Nechť tedy máme dva různé stavy $\mathbf{s}_{i_1}$ a $\mathbf{s}_{i_2}$, v nichž končí cesty Π_{i_1} a Π_{i_2} s metrikami $M_{i-1}(\mathbf{s}_{i_1})$ a $M_{i-1}(\mathbf{s}_{i_2})$. Myšlenka je jednoduchá: dekódujeme-li metodou maximální věrohodnosti (a tedy minimalizujeme metriku cesty), musíme v každém kroku zachovat pouze ty cesty, jejichž metrika je nejmenší možná. Tedy v každém kroku si v každém uzlu ponecháme pouze tu cestu (a její metriku), která je nejpravděpodobnější. Tedy

$$M_i(\mathbf{s}_{i+1}) = \min\{M_{i-1}(\mathbf{s}_{i_1}) + \mu_i(\mathbf{r}_i|\hat{\mathbf{c}}^{(s_{i_1},\mathbf{s}_{i+1})}), M_{i-1}(\mathbf{s}_{i_2}) + \mu_i(\mathbf{r}_i|\hat{\mathbf{c}}^{(\mathbf{s}_{i_2},\mathbf{s}_{i+1})})\}$$

a cesta s menší metrikou (tzv. *přeživší cesta*) se zachová, kdežto ta druhá se zapomene. Takto si algoritmus v každém kroku pamatuje pouze 2^m metrik a příslušných cest a přitom neztratí žádné optimální řešení. Může nastat ještě jeden případ: obě dvě metriky vedoucí do jednoho uzlu jsou shodné, pak se může buď jedna z nich náhodně zvolit (na maximální věrohodnosti dekódovaného slova to nic nezmění), nebo si kodér může někde do statické paměti uložit informaci, že do daného stavu vedla ještě jedna cesta a v případě, že uzel byl součástí dekódovaného slova, nabídnout obě varianty.

Viterbiův algoritmus by se dal zformulovat takto:

(i) Inicializace: $i = 0$.

(ii) V čase $i + 1$ pro stav kodéru $\mathbf{s}_{i+1}$ najdi všechny stavy $\mathbf{s}_i^{(j)}$ v čase i, ze

kterých vede do $\mathbf{s}_{i+1}$ hrana. Pro každý z nich spočítej metriku cesty jako $M_{i-1}(\mathbf{s}_i^{(j)}) + \mu_i(\mathbf{r}_i, \hat{\mathbf{c}}^{(\mathbf{s}_i^{(j)}, \mathbf{s}_{i+1})})$.

(iii) Přeživší cestu do stavu $\mathbf{s}_{i+1}$ vyber jako cestu s nejnižší metrikou mezi všemi z bodu (ii). Do stavu $\mathbf{s}_{i+1}$ ulož přeživší cestu a její metriku.

(iv) Opakuj pro všechny stavy $\mathbf{s}_{i+1}$.

(v) Zvětši i o 1 a celé opakuj, dokud neprojdeš celou mřížku.

(vi) Na konci vyber stav s nejnižší akumulovanou metrikou a příslušnou cestu v mřížce prohlaš za výsledek.

Příklad 5.1.4. Vraťme se k našemu $[2, 1, 2]$ kódu s generující maticí $G(x) = (\,1 + x + x^2 \quad 1 + x^2\,)$ a ilustrujme na něm dekódování pomocí Viterbiova algoritmu. Nechť vstup je $\mathbf{v} = (1, 0, 0, 1, 1, 0, 1)$, pak výstup je (k ručnímu výpočtu použijeme reprezentaci pomocí Laurentových řad)

$$\mathbf{c}^{(1)} = (1, 1, 1, 1, 0, 0, 0, 1, 1)$$

$$\mathbf{c}^{(2)} = (1, 0, 1, 1, 1, 1, 0, 0, 1),$$

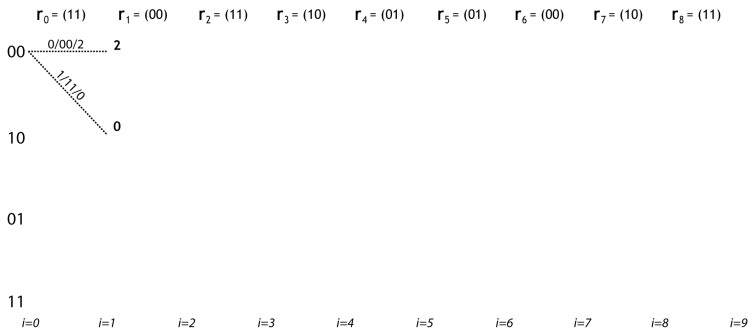
tedy celkem

$$\mathbf{c} = (11, 10, 11, 11, 01, 01, 00, 10, 11).$$

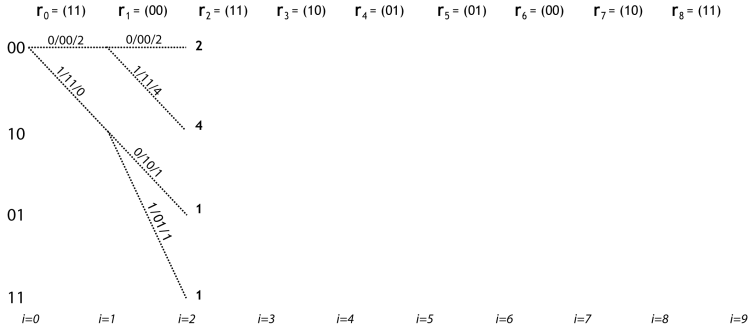
Necháme kódové slovo „projít“ binárním symetrickým kanálem a na druhé straně „přijmeme“ $\mathbf{r} = (11, \underline{0}0, 11, 1\underline{0}, 01, 01, 00, 10, 11)$, kde podtržené jsou bity, na nichž došlo k chybám.

V prvním kroku dekodéru (viz obrázek 5.1) se pouze prodlouží cesty ze stavu 00. Nad každou cestu si zaznačíme příslušný bit vstupu, bity výstupu a celkovou naakumulovanou metriku. Ve druhém kroku (obrázek 5.2) ještě stále každá cesta vede do jiného vrcholu a k jejich křížení nedochází. Ve třetím kroku (obrázek 5.3) už se ale cesty setkávají, na řadu tedy přichází těžiště Viterbiova algoritmu: ponechání pouze té nejlepší cesty, co do uzlu vede — výsledek po odstranění špatných cest je vidět na obrázku 5.4. Jedna cesta zůstala „slepá“, tj. její naakumulovaná metrika byla natolik veliká, že nebylo výhodné prodloužit ji do nějakého stavu v čase $i = 3$. V dalším kroku (obrázek 5.5) ji tedy můžeme smazat.

Postupně takto procházíme celou mřížku (v případech, kdy se potkají dvě cesty s totožnou metrikou, volíme mezi nimi náhodně), jak je naznačeno na obrázcích 5.6 až 5.11. V posledním kroku se podíváme, ve kterém uzlu je



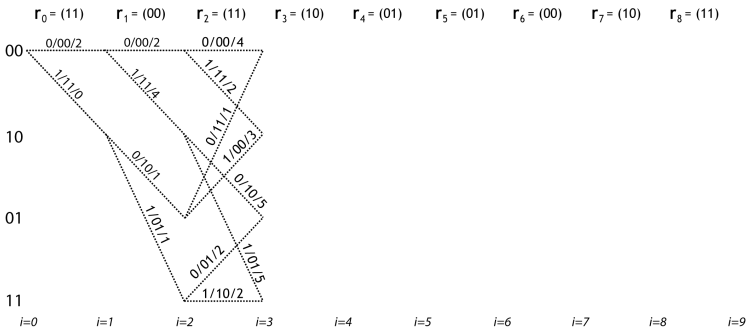
Obrázek 5.1:



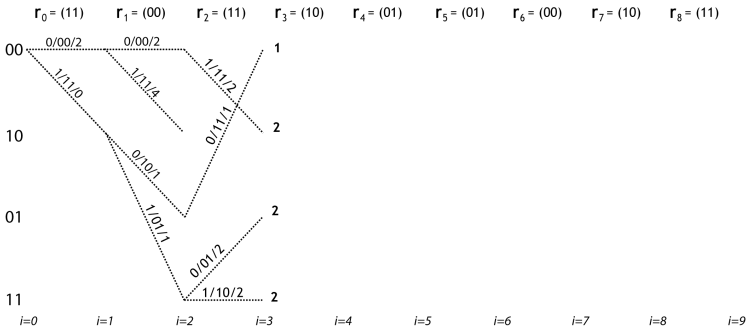
Obrázek 5.2:

naakumulována nejmenší metrika a za výsledek prohlásíme cestu, která do něj vede, jak je ukázáno na obrázku 5.12. Přečteme-li kódové bity na této cestě, dostaneme $\mathbf{c_m} = (11, 10, 11, 11, 01, 01, 00, 10, 11)$, tedy $\mathbf{c_m} = \mathbf{c}$ a dekódování úspěšně opravilo obě dvě chyby; přečteme-li vstupní bity, dostaneme $\mathbf{v_m} = (1, 0, 0, 1, 1, 0, 1, 0, 0)$ — stačí si odmyslet poslední dvě nuly, kterými jsme vstupní vektor $\mathbf{v}$ doplnili, a dostáváme přesně vstup $\mathbf{v}$.

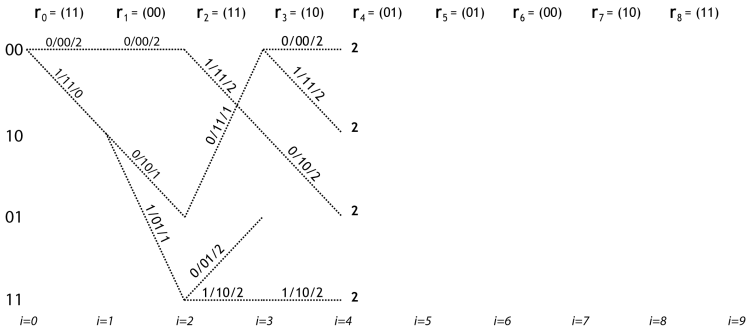
V praxi se ještě Viterbiův algoritmus vylepšuje několika způsoby. Při přijímání velmi dlouhých slov by mezi příjmem a výstupem z dekodéru vznikalo velké zpoždění (neboť dekodér napřed musí dojít do konečného stavu, aby mohl rozhodnout, která cesta je výstup), navíc by tím vznikaly vysoké nároky na paměť dekodéru. Při pohledu na mřížku ve výše uvedeném příkladě



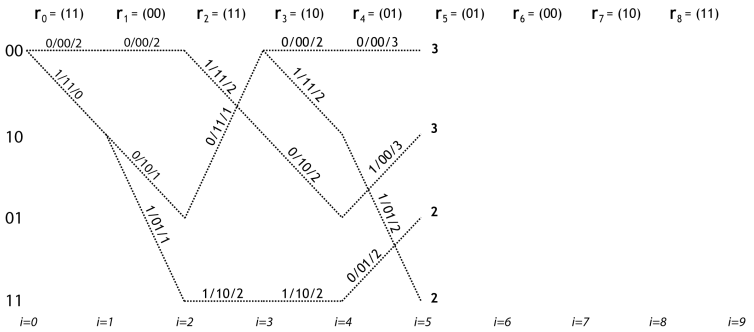
Obrázek 5.3: třetí krok krok dekodéru — první křížení cest



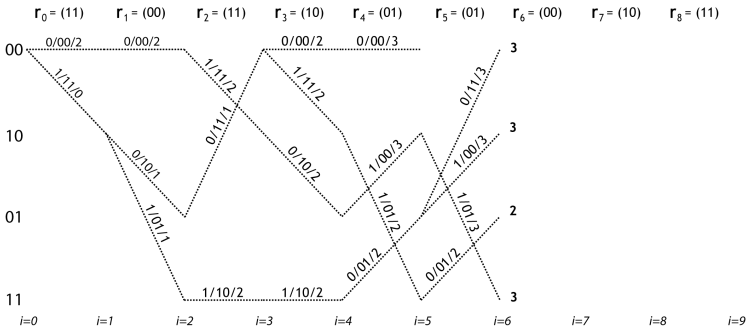
Obrázek 5.4: třetí krok krok dekodéru — výsledek po ponechání nejlepších cest



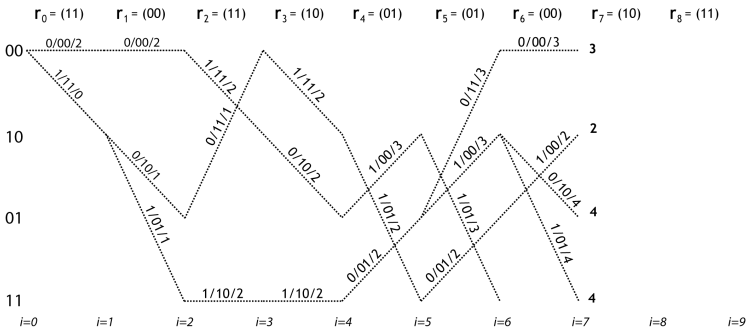
Obrázek 5.5:



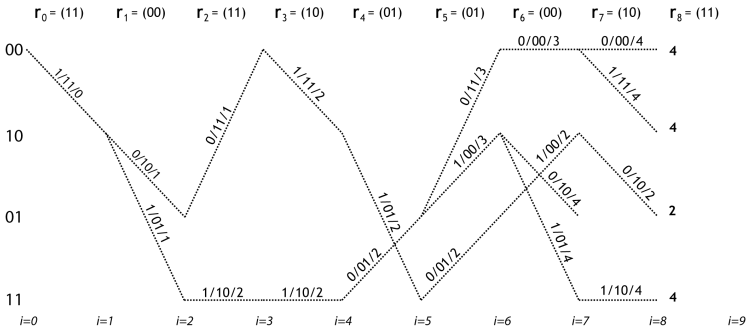
Obrázek 5.6:



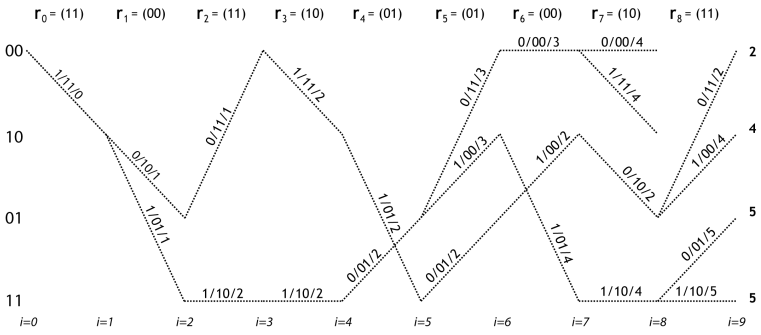
Obrázek 5.7:



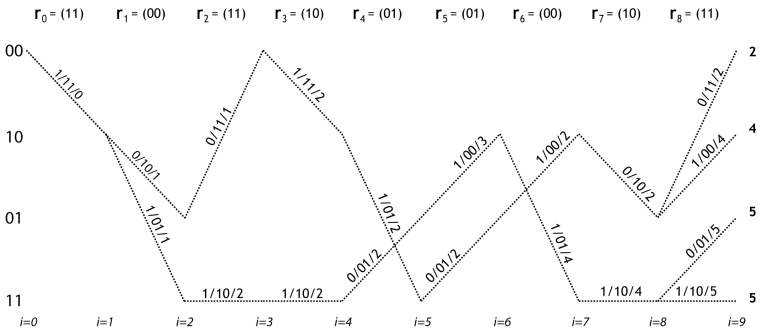
Obrázek 5.8:



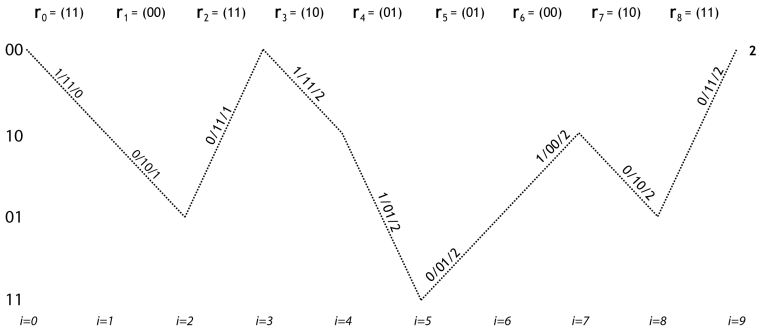
Obrázek 5.9:



Obrázek 5.10:



Obrázek 5.11:



Obrázek 5.12: výsledek Viterbiova algoritmu — cesta s minimální metrikou

je jasné, že toho není třeba. Při dekódování totiž pravidelně v dostatečně velké vzdálenosti před aktuálně procházenými stavy typicky „přežije“ pouze jediná cesta a nezávisle na tom, jak se dekodér rozhodne dále, zůstane na výstupu tak jako tak. V našem případě to bylo v čase $i = 8$, kdy mezi $i = 0$ a $i = 1$ přežila pouze jedna cesta (viz obrázek 5.9). Je proto možné si místo celé mřížky pamatovat pouze její část, jakési posunovací okno a průběžně posílat na výstup ty bity, jejichž přítomnost v dekódovaném slově je již jistá. Velikost tohoto okna, *dekódovací hloubka*, se značí Γ, tedy v čase i dekodér pošle na výstup bit $\mathbf{c}_{\mathbf{m}i-\Gamma}$. Je samozřejmě možné, že v době, kdy dekodér bude chtít na výstup poslat bit zpřed Γ kroků, ještě v čase $i-\Gamma$ nebude jediná přeživší cesta, ale volbou dostatečně velké dekódovací hloubky (zhruba pěti- až desetinásobek stupně kódu t) lze pravděpodobnost, že se tomu tak stane, snížit na minimum a přitom zachovat nově nabytou výkonnost dekodéru.

Druhé vylepšení (či spíše řešení technického problému) spočívá v tom, že je-li přijaté slovo dlouhé, v jednotlivých uzlech mřížky se mohou akumulovat vysoké hodnoty metrik — abychom zabránili příliš velkým paměťovým nárokům, stačí pravidelně od každé metriky odečíst stejnou hodnotu (na konci sice nezískáme přesnou hodnotu metriky nejpravděpodobnější cesty, ale metoda jako hledání takové cesty bude fungovat) a tím držet velikost ukládaných údajů v určitých mezích.

5.2 Sekvenciální dekódování

Viterbiův algoritmus je optimální ve smyslu, že vždy najde maximálně věrohodné kódové slovo. Na druhou stranu v každém z l kroků (kde l je délka přijímaného slova $\mathbf{r}$) se musí v každém z 2^k stavů spočítat a porovnat metriky cest do nich mířící. Spousta výpočtů bude prováděna zcela zbytečně, neboť ještě před tím, než by byly cesty s vysokou metrikou jako „mrtvé“ smazány, několikrát se prodlouží (a spotřebují část výpočetního výkonu). Ulehčení nepřichází, ani když z nějakého důvodu víme, že nenastalo mnoho chyb. V této části si ukážeme algoritmy, které za určitých okolností mohou být výrazně výkonnější než univerzálně použitelný Viterbiův.

Místo mřížky budeme kód $\mathbf{C}$ reprezentovat stromem, kdy kořen bude odpovídat počátku kódování. Ukažme si to na našem $[2, 1]$ kódu: z každého kořene povedou dvě větve, přičemž horní z nich značí 0 na vstupu a spodní 1. Do uzlu si uložíme příslušný výstup $\mathbf{c}_i$, každá posloupnost vrcholů stromu od kořene k nějakému listu pak odpovídá kódovému slovu.

Projití všech větví stromu by přirozeně mělo exponenciální složitost,

proto se na základě metriky vybere cesta (potažmo kódové slovo $\mathbf{c_m}$), která nejvíc odpovídá přijatému slovu $\mathbf{r}$. K tomu budeme potřebovat nově zavedenou metriku, neboť potřebujeme porovnávat slova různé délky (představme si situaci, kdy porovnáváme dvě větve délky např. 2 a 14, kdy v první je jedna chyba a ve druhé dvě: standardní metrika by vybrala první, ovšem je rozumné předpokládat, že dva chybné bity ze čtrnácti je lepší výsledek než jeden ze dvou).

Nechť tedy $\tilde{\mathbf{c}}^{(i)} = (\mathbf{c}_0^{(i)},\ \mathbf{c}_1^{(i)}, \ldots,\ \mathbf{c}_{n_i-1}^{(i)})$ je část kódového slova délky n_i, kde každé $\mathbf{c}_j^{(i)}$ je vektor n bitů (tedy výsledek jednoho kroku kódování). Vektor $\tilde{\mathbf{c}}^{(i)}$ má tedy celkem nn_i bitů. Předpokládejme, že bity vstupního slova jsou náhodné veličiny s 0-1 rozdělením a $p = \frac{1}{2}$, pak pravděpodobnost, že z kodéru vystoupí $\tilde{\mathbf{c}}^{(i)}$, je přesně

$$P(\tilde{\mathbf{c}}^{(i)}) = (2^{-k})^{n_i} = 2^{-Rnn_i},$$

kde $R = \frac{k}{n}$ značí informační poměr $[n, k]$ kódu.

Označme $\mathcal{X} = \{\tilde{\mathbf{c}}^{(1)}, \ldots,\ \tilde{\mathbf{c}}^{(L)}\}$ množinu všech částečných sekvencí, mezi kterými se chceme rozhodnout, a $n_{\max} = \max\{n_1, n_2, \ldots, n_L\}$ jako největší délku ze všech těchto sekvencí. Z přijatého slova $\mathbf{r}$ si vezmeme právě tak dlouhou částečnou sekvenci: $\tilde{\mathbf{r}} = (\mathbf{r}_0,\ \mathbf{r}_1, \ldots,\ \mathbf{r}_{n_{\max}-1}) = (r_0,\ r_1,\ \ldots,\ r_{nn_{max}-1})$. Za optimální výsledek pak považujeme vybrání sekvence $\tilde{\mathbf{c}}^{(i)}$ takové, že je maximalizována $P(\tilde{\mathbf{c}}^{(i)}|\tilde{\mathbf{r}})$.

Úpravami jako v poznámce 5.0.10 lze docílit tohoto vzorce:

$$\log P(\tilde{\mathbf{c}}^{(i)}|\tilde{\mathbf{r}}) = \sum_{j=0}^{nn_i-1} \Big(P(r_j|c_j^{(i)}) - P(r_j)\Big) - \log_2 Rnn_i.$$

Definice 5.2.1. Fanova metrika cesty je definována jako

$$M_F(\tilde{\mathbf{c}}^{(i)}|\tilde{\mathbf{r}}) = \log P(\tilde{\mathbf{c}}^{(i)}|\tilde{\mathbf{r}}),$$

Fanovu metriku větve definujeme takto:

$$\mu(\mathbf{r}_j, \mathbf{c}_j^{(i)}) = \sum_{l=1}^{n} \Big(\log P(r_j^{(l)}|c_j^{(i,l)}) - \log P(r_j) - R\Big),$$

kde $r_j^{(l)}$ značí l-tý bit $\mathbf{r}_j$ a $c_j^{(i,l)}$ l-tý bit $\mathbf{c}_j^{(i)}$.

Všimněme si, že Fanova metrika je vlastně zobecněním metriky, jež jsme doposud používali[2]. První člen sumy v metrice větve zajišťuje, že věrohodnostní funkce se maximalizuje, druhé dva dávají do souvislosti i délky vektorů. Pokud bychom je měli všechny stejně dlouhé (jako v konvenčním případě), nemuseli bychom tyto dva členy vůbec uvažovat a dostali bychom klasický případ Viterbiova dekódování.

Příklad 5.2.2. Odvoďme přesné hodnoty Fanovy metriky větve pro BSC, $p = 0,1$ a $[2,1]$ kód (tedy $R = \frac{1}{2}$). Máme z definice

$$\mu(\mathbf{r}_j, \mathbf{c}_j^{(i)}) = \sum_{l=1}^{2} \left(\log_2 P(r_j^{(l)} | c_j^{(i,l)}) - \log_2 P(r_j^{(l)}) - R \right).$$

Rozdělme sumu na dva sčítance, pak každý z nich bude roven

$$\begin{cases} \log_2(1-p) - \log_2 \frac{1}{2} - R = \log_2 2(1-p) - R = 0,348 & \text{je-li } r_j^{(l)} = c_j^{(i,l)}, \\ \log_2 p - \log_2 \frac{1}{2} - R = \log_2 2p - R = -2,82 & \text{je-li } r_j^{(l)} \neq c_j^{(i,l)}. \end{cases}$$

Po znormování (vydělení obou hodnot číslem $0,348$) dostáváme, že každý sčítanec je

$$\begin{cases} 1 & \text{je-li } r_j^{(l)} = c_j^{(i,l)}, \\ -8 & \text{je-li } r_j^{(l)} \neq c_j^{(i,l)}. \end{cases}$$

Tedy je-li cesta algoritmu správná, Fanova metrika cesty by měla mírně narůstat, naopak pokud dojde k chybě, strmě klesne dolů. Konkrétně ve **Fanově algoritmu** je vše realizováno následovně: v paměti je uchovávána cesta z kořenu binárního stromu až do aktuálního uzlu a její metrika, plus speciální hodnota, tzv. *práh (threshold)* T. Algoritmus se v každém kroku podívá do následujících uzlů a vybere ten s největší metrikou cesty N_A: pokud ta je větší než aktuální metrika N, prodlouží aktuální cestu o daný uzel a vše opakuje. Pokud je $N_A < N$, algoritmus se podívá na metriku v předchozím uzlu N_B a porovná ji s T. V případě, že $N_B > T$, aktuální uzel se zavrhne a algoritmus se vrátí do stavu s metrikou N_B a odtam opět pokračuje výběrem dalších možných uzlů (pokud jsou již všechny vyčerpané, opět se o krok vrátí atd.). Pokud je ovšem $N_B < T$, zmenší T o Δ a algoritmus se přenese do následujícího uzlu. Pokud algoritmus nějaký uzel navštíví poprvé, odečte od T násobek Δ tak, aby $T < N$.

[2]S jediným rozdílem, a to že nyní chceme mít metriku co největší.

Role Δ nemusí být na první pohled zcela jasná, proto jí věnujme následující odstavec. Do hry Δ vstupuje pouze tehdy, když jsme v uzlu, jehož metrika je větší než nejlepší metrika jeho následovníků — tzn. buď děláme chybu při dekódování, nebo naopak chybu při přenosu opravujeme. Právě tuto hranici (skrze T) určuje Δ — čím větší je Δ, tím méně je třeba početních operací (ale také tím déle bude algoritmus považovat špatná řešení za správná; nesmíme také zapomenout, že Δ je shora omezené pravděpodobností maximálně věrohodné cesty); naopak čím menší Δ, tím častěji se algoritmus bude vracet o krok zpět. Je proto více než vhodné předem na počítači ověřit výkon algoritmu s pevným Δ.

Druhým algoritmem spadajícím do škatulky „sekvenciální“ je tzv. **zásobníkový (Zigangirovův) algoritmus**. Jak již název napovídá, datová reprezentace bude odpovídat zásobníku, jehož prvky budou dvojice $\{M_F(\tilde{\mathbf{c}}^{(i)}|\tilde{\mathbf{r}}),\ \tilde{\mathbf{c}}^{(i)}\}$ seřazené podle Fanovy metriky $M_F(\tilde{\mathbf{c}}^{(i)}|\tilde{\mathbf{r}})$ od největší po nejmenší. Jeden krok algoritmu vypadá tak, že se vezme prvek s největší (a tedy nejlepší) metrikou, příslušná cesta se o jeden krok prodlouží všemi 2^k způsoby, pro každé prodloužení se spočítá nová metrika a všechny výsledky se uloží do zásobníku, který je pak opětovně setříděn.

Algoritmus funguje, neboť pokud je navrchu cesta vedoucí z počátečního do koncového stavu, jedná se o výsledek (protože je nahoře, má největší metriku a tedy jeho věrohodnostní funkce je maximální mezi všemi kódovými slovy). V každém kroku vždy jednu cestu prodloužím, po konečném počtu kroků se tedy algoritmus zastaví. Trochu jiná otázka už je jeho časová složitost, neboť v každém kroku z jedné cesty získám 2^k nových, které potřebuji zatřídit do současných.

Tento problém se pokusil vyřešit F. Jelinek zajímavým vylepšením původního Zigangirovova algoritmu — interval možných hodnot Fanovy metriky se předem rozdělí na menší části, z nichž každá má pevnou velikost přiřazené paměti (tzv. *bucket*). Algoritmus vezme vrchní prvek z intervalu s nejvyššími hodnotami, který není prázdný, vymaže jej ze zásobníku a prodloužené cesty uloží navrch příslušného bucketu, ty už dále netřídí. Tím sice ušetří čas potřebný k běhu algoritmu, nicméně je celkem zřejmé, že výstup nebude nejlepší, ale jen „velmi dobré“ kódové slovo. Nicméně vhodnou volbou hustého rozdělení intervalů a za předpokladu učiněného na začátku podkapitoly o sekvenciálních algoritmech, totiž že nenastalo mnoho chyb, je ztráta výkonu původního algoritmu minimální.

5.3 M-algoritmus

Variací na Viterbiův algoritmus je tzv. *M-algoritmus*, který se snaží vypořádat s výpočetní složitostí pro větší stupně kodéru. V mřížce kódu máme jen jedno správné řešení, přitom v každém kroku uchováváme 2^m cest a počítáme 2^{mk} metrik větví, drtivou většinu z nich zcela zbytečně. Proto mějme algoritmus, který si v každém kroku pamatuje pouze M nejlepších cest, tedy při prodlužování počítáme pouze $M2^k$ metrik. Nepoužíváme navíc „slévání“ cest v uzlech jako u Viterbiova algoritmu, v přeživších M cestách mohou být dvě různé vedoucí do jednoho uzlu — proto je lepší použít k reprezentaci strom, ne mřížku.

Pokud zvolíme $M = 2^m$, dostáváme algoritmus velmi podobný tomu Viterbiovu, nicméně ne shodný (některé ponechané cesty mohou končit ve stejném uzlu); přitom lze M volit výrazně menší a ztratit pouze malou část výkonu dekódování. M-algoritmus tedy také zařadíme do rodiny suboptimálních algoritmů, vhodných k použití za určitých podmínek.

Velmi podobnou myšlenku používá i tzv. T-algoritmus, který místo ponechání M nejlepších cest ponechá všechny cesty, jejichž metrika není horší o více jak T než metrika nejlepší z nich.

5.4 Porovnání jednotlivých algoritmů

Srovnejme si nyní jednotlivé algoritmy:

- Zásobníkový algoritmus navštěvuje každý uzel grafu maximálně jednou (a proto každou metriku cesty počítá jen jednou), zatímco Fanův algoritmus používá backtracking, a proto se může stát, že některé uzly navštíví i několikrát za sebou — z toho plyne několikeré počítání metriky stejné cesty (což se sice dá odstranit ukládáním všech již spočítaných metrik, nicméně toto řešení zase vyžaduje dodatečnou paměť). Viterbiův algoritmus každý uzel navštíví také několikrát, nicméně počet návštěv je konstantní (do každého uzlu se „podívá“ 2^k-krát, když rozhoduje, jaká v něm bude metrika).

- Fanův algoritmus je z hlediska paměti méně náročný, neboť si pamatuje vždy jen aktuální cestu a její metriku (případně i všechny spočítané), zatímco zásobníkový algoritmus musí uchovávat v paměti všechny spočítané cesty (resp. nějakou jejich přeživší část). Viterbiův algoritmus

je na paměť také náročný: v každém ze 2^m uzlů si pamatuje jak nakumulovanou metriku, tak i příslušnou cestu.

- Co se týče časové složitosti, zde je situace nejednoznačná. Náročnost Viterbiova algoritmu nezávisí na počtu chyb při přenosu, všechny výpočty se dělají stále stejně, pro kódy většího stupně m je tato náročnost vysoká; naproti tomu rychlost sekvenciálních algoritmů je do velké míry ovlivněna chybovostí přenosu. Pro malý počet chyb (a tedy i malý počet vracení se Fanova nebo rychlé prodlužování nejlepší cesty zásobníkového algoritmu) platí, že sekvenciální dekódování je rychlejší než Viterbiovo (a Fanovo trochu rychlejší než zásobníkové), nicméně pro větší počet chyb již tyto algoritmy rychle ztrácejí dech (a situace se obrací, díky většímu množství kroků zpět Fanova algoritmu).

  Sekvenciální algoritmy byly původně v 60. letech XX. století navrženy pro kódy velkého stupně, pro které Viterbiův algoritmus (exponenciálně závislý na m a k) byl (na svou dobu) již příliš pomalý; za pomoci moderní techniky již není větší problém dekódovat Viterbiovým algoritmem i tyto kódy.

- Podíváme-li se na princip a myšlenku algoritmů, tak Fanův je v podstatě hledání do hloubky, Viterbiův hledání do šířky a zásobníkový kombinace obou dvou.

- Nejdůležitější rozdíl mezi Viterbiovým algoritmem a dalšími (ať už sekvenciálními nebo M-algoritmem) spočívá v tom, že prvně jmenovaný **vždy** dá správný (maximálně věrohodný) výsledek, kdežto podobnou jistotu u jiných algoritmů nemáme (pouze víme, že řešení je „velmi dobré“).

Kapitola 6

Použití konvolučních kódů

Konvoluční kódy byly již krátce po svém objevení a popsání „nasazeny“ na nejkritičtější místa: v roce 1977 byly vypuštěny dvě vesmírné sondy *Voyager 1* a *Voyager 2*, které k přenosu signálu zpět na Zemi používaly Reed-Solomonovy kódy kombinované s $[2, 1]$ konvolučním kódem s generující maticí

$$G(x) = (\, 1 + x + x^2 + x^3 + x^6 \quad 1 + x^2 + x^5 + x^6 \,) \,.$$

Pro nové mise (jmenovitě dvojice „vozítek“ *Mars Rover* a *Mars Exploration Rover* či sonda *Cassini* letící k Saturnu) se již používá výkonnějších $[6, 1]$ konvolučních kódů o $m = 14$. Samozřejmě, dekódování je příslušně složitější (není složité ověřit, že právě $256\times$), nicméně v tomto případě jistě nevadí prodleva mezi přijetím zprávy a jejím dekódování.

Konvoluční kódy se uplatňují i v satelitní komunikaci, nejrozšířenější sítě používají ty samé kódy jako program *Voyager*, moderní hardware pokročil natolik, že jejich nasazení do provozu v reálném čase není problém.

Obecně se konvoluční kódy používají v menších stupních, jejich náročnost na dekódování pro větší stupně je činí nevhodnými pro široké použití. Ovšem v situaci, kdy záleží hlavně na předání zprávy bez chyb, jsou konvoluční kódy (případně v kombinaci s Reed-Solomonovými) vynikajícím pomocníkem. Sice je v současné době z pozice nejvýkonnějších známých kódů vytlačily turbokódy a LDPC kódy (*Low-Density Parity Check*), nicméně základem prvně jmenovaných jsou kódy konvoluční, takže se o jejich budoucnost obávat nemusíme.

Literatura

[1] Berrou C., Glavieux A., Thitimajshima P.: *Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes*, in: Proc. 1993 IEEE International Conference on Communications, s. 1064–1070, Geneva, 1993.

[2] Elias P.: *Coding for Noisy Channels*, in: IRE Conv. Rept. Pt. 4, s. 37–47, 1955.

[3] Hagelbarger D. W.: *Reccurent Codes: Easily Mechanized, Burst-Correcting Binary Codes*, in: Bell Syst. Tech. J. 38, s. 969–984, 1959.

[4] MacKay D. J. C.: *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, Cambridge, 2003.

[5] McEliece R. J.: *The Algebraic Theory of Convolutional Codes*, Handbook of Coding Theory, pp. 1065–1138, 1998.

[6] Moon T. K.: *Error Correction Coding: Mathematical Methods and Algorithms*, John Wiley & Sons, New Jersey, 2005.

[7] Morelos-Zaragoza R. H.: *The Art of Error Correction Coding*, John Wiley & Sons, Chichester, 2006.

[8] Purser M.: *Introduction to Error-Correcting Codes*, Artech House, Norwood, 1995.

[9] Sweeney P.: *Error Control Coding: From Theory to Practise*, John Wiley & Sons, Chichester, 2002.

[10] Viterbi A. J.: *Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm*, in: IEEE Trans. Information Theory, vol. 13, s. 260–269, 1967.