Západočeská univerzita v Plzni
Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Využití samoopravného kódu v bezdrátovém přenosu

Diplomová práce

Autor práce: Bc. Jan Broulím
Vedoucí práce: doc. Dr. Ing. Vjačeslav Georgiev

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan BROULÍM**
Osobní číslo:	**E10N0042P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Název tématu:	**Využití samoopravného kódu v bezdrátovém přenosu**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Prostudujte fyzickou a linkovou komunikační vrstvu protokolu ISO modelu OSI s ohledem na vhodnou implementaci samoopravných kódů.
2. S ohledem na prostředí, délku zpráv, fyzickou vrstvu a další parametry mající vliv na dosah přenosu navrhněte vhodný samoopravný kód.
3. Zvolte vhodnou obvodovou a programovou realizaci vámi navrženého kódu.
4. Proveďte měření dosahu a porovnejte parametry přenosu s parametry bez samoopravného kódu. Dosažené výsledky prezentujte.

Rozsah grafických prací:	**podle doporučení vedoucího**
Rozsah pracovní zprávy:	**30 - 40 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

1. **User guides Texas Instrumnets - Software & IEEE 802.15.4**
2. **Model OSI**
3. **Todd K. Moon, Error Correction Coding**

Vedoucí diplomové práce:	**Doc. Dr. Ing. Vjačeslav Georgiev** Katedra aplikované elektroniky a telekomunikací
Konzultant diplomové práce:	**Ing. Aleš Krutina** Katedra elektroenergetiky a ekologie
Ostatní konzultanti:	**Doc. Dr. Ing. Vjačeslav Georgiev** Katedra aplikované elektroniky a telekomunikací

Datum zadání diplomové práce:	**17. října 2011**
Termín odevzdání diplomové práce:	**11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 17. října 2011

Abstrakt

Předkládaná diplomová práce se zabývá návrhem, simulací a implementací samoopravných kódů, především rodiny LDPC. Kódy z této rodiny jsou stručně představeny, podstatnou částí této práce je pak naprogramování softwarových knihoven pro návrh a simulaci opravných vlastností LDPC kódů. Vybraný kód byl testován měřením spolehlivosti přenosu v bezdrátové komunikaci.

Klíčová slova

LDPC, opravné kódy, bitová chybovost, minimální vzdálenost.

Abstract

This diploma thesis deals with design, simulation and implementation of error correcting codes, especially with the LDPC family. The LDPC codes are briefly introduced. Substantial part of this work is devoted to developing of software libraries for design and simulation of the LDPC codes. A chosen code was tested in wireless data transfer.

Key Words

LDPC, error correcting codes, bit error rate, minimum distance.

Prohlášení

Předkládám tímto k posouzení a následné obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni. Prohlašuji, že jsem zadanou diplomovou práci zpracoval zcela samostatně, pouze s použitím literatury a pramenů, jejichž úplný seznam je součástí.

V Plzni, dne .....

.....

Podpis

Poděkování

Chtěl bych tímto poděkovat doc. Dr. Ing. Vjačeslavu Georgievovi, jehož cenné a profesionální rady výrazně přispěly ke konečné podobě této práce.

Další poděkování patří Ing. Aleši Krutinovi za zapůjčení vývojového kitu pro bezdrátovou komunikaci.

Také velmi oceňuji přístup výpočetnímu vybavení a datovým uložištím vlastněném institucemi, které jsou zapojené do projektu Národní Gridové Infrastruktury MetaCentrum.

The access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005) is highly appreciated.

Obsah

Seznam obrázků

Seznam algoritmů

Seznam tabulek

Seznam použitých zkratek

AWGN Additive White Gaussian Noise

BEC Binary Erasure Channel

BER Bit Error Rate

BFS Breadth-First Search

BP Belief Propagation

BPSK Binary Phase Shift Keying

BSC Binary Symmetric Channel

BSEC Binary Symmetric Erasure Channel

ECC Error Correcting / Correction Code

FEC Forward Error Correction

FER Frame Error Rate

FFT Fast Fourier Transform

GF Galois Field

HAL Hardware Abstraction Layer

IEEE Institute of Electrical and Electronics Engineers

ISM Industrial, Scientific and Medical

ISO International Standard Organisation

LDPC Low-Density Parity-Check

MA Moving Average

NF Noise Figure

OSI Open Systems Interconnection

RF Radio Frequency

RSSI Received Signal Strength Indication

QAM Quadrature Amplitude Modulation

QPSK Quadrature Phase Shift Keying

SP Sum-Product

UART Universal Asynchronous Receiver / Transmitter

USCI Universal Serial Communication Interface

1 Úvod

Tato práce se zabývá návrhem, simulací a implementací samoopravných kódů. Na doporučení vedoucího práce je pozornost věnována rodině LDPC kódů, které se dnes řadí mezi nejmodernější kódy používané v datových komunikacích. Práce si klade za cíl především vyřešení návrhu LDPC kódů, jejich implementaci v jednočipovém mikrokontroléru a zkoumání spolehlivosti přenosu v bezdrátové komunikaci, zejména její zlepšení při použití vhodného kódu. Výstupem jsou softwarové knihovny pro návrh a simulaci LDPC kódů, kodér a dekodér pro nasazení v jednočipovém mikrokontroléru, ukázková aplikace použití LDPC kódů v bezdrátovém přenosu a několik navržených kódů, které byly generovány a simulovány s využitím Národní Gridové Infrastruktury MetaCentrum. Jedním z řešených problémů je také vyhodnocování naměřené spolehlivosti přenosu.

Text je členěn do několika na sebe navazujících kapitol. V první kapitole jsou popsány dva abstraktní modely užívané v datových komunikacích - Shannonův a vrstvený ISO/OSI. Na to navazují tři kapitoly, které se postupně zabývají teorií kanálového kódování, některými vlastnostmi opravných kódů a detailnějším popisem LDPC kódů. Následuje dokumentace k naprogramovaným knihovnám a kapitola věnující se implementaci vybraného kódu v bezdrátovém přenosu. Na závěr jsou prezentovány výsledky měření vybraného kódu a porovnány dosažené spolehlivosti přenosu bez použití kódu a při použití opravného kódu. Některé z dalších navržených kódů jsou uvedeny v příloze.

2 Digitální komunikace

2.1 Množství informace a entropie

Množství informace I jevu x_i ze souboru vzájemně vylučujících se jevů X je kvantifikovatelnou veličinou. Dnes používanou jednotkou je [Shannon], dříve [bit]. Platí rovnost:

$$I(x_i) = \log_2 \frac{1}{P(x_i)} = -\log_2 P(x_i)\,[\mathrm{Sh}], \tag{2.1}$$

kde $P(x_i)$ označuje pravděpodobnost daného jevu.

Entropie H slouží k charakterizování souboru jevů jako celku a vyjadřuje střední hodnotu množství informace.

$$H(X) = -\sum P(x_i) \log_2 P(x_i)\,[\mathrm{Sh}]. \tag{2.2}$$

Příklad: Zdroj vysílá nuly s pravděpodobností 20% a jedničky s pravděpodobností 80%. Množství informace při vyslání nuly bude

$$I(0) = -\log_2 0{,}2 \doteq 2{,}32\,\mathrm{Sh},$$

a při vyslání jedničky

$$I(1) = -\log_2 0{,}8 \doteq 0{,}32\,\mathrm{Sh}.$$

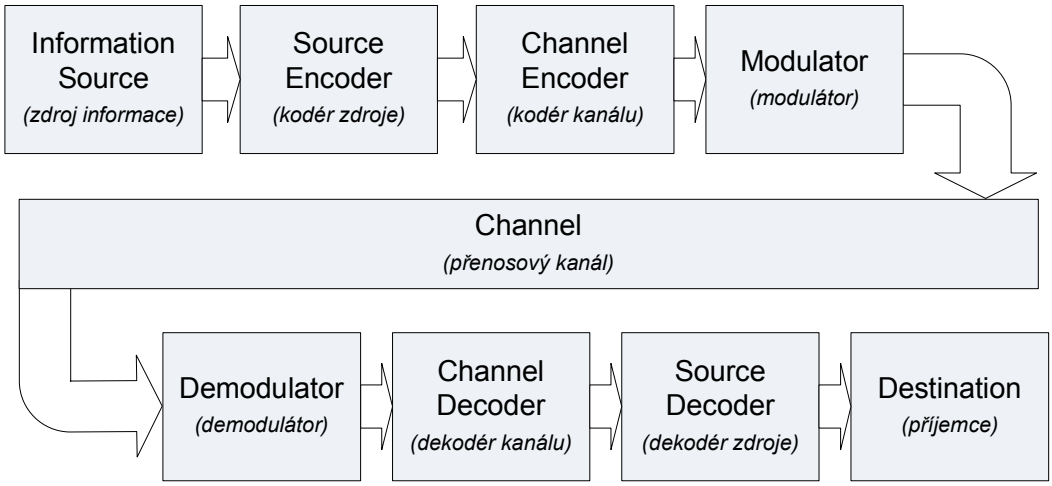
2.2 Schéma komunikačního systému podle Shannona

Shannon na přelomu 40. a 50. let zobecnil schéma datových komunikací do řetězce znázorněného na obr. 2.1. Řetězec rozděluje do samostatně funkčních bloků. Jejich výčet je následující:

- zdroj informace - generuje zprávy formou posloupnosti jevů, které lze kvantifikovat entropií a zároveň mají *sémantický* význam,
- kodér zdroje - jeho úkolem je transformace zpráv zdroje do podoby zpracovatelné kodérem kanálu, přičemž redundance výstupní posloupnosti by měla být co nejmenší (tzn. co nejvyšší entropie),
- kodér kanálu - zabezpečuje přenos přidáním redundantní informace,
- modulátor - provádí transformaci do podoby přenositelné kanálem,
- přenosový kanál - médium, kde dochází k poruchám přenosu,

- demodulátor - transformuje posloupnost přijatou z kanálu do podoby zpracovatelné dekodérem kanálu,
- dekodér kanálu - opravuje chyby vzniklé při přenosu za pomoci redundantní informace a tuto redundanci odstraní,
- dekodér zdroje - převádí tok do podoby srozumitelné příjemcem.

Takové systémy jsou zabezpečeny pomocí opravného kódu (ECC, *Error Correcting / Correction Code*) a jelikož se zde nevyskytuje zpětná vazba, označují se jako FEC (*Forward Error Correction*). Modulátor se často chápe jako součást kanálu.



Obr. 2.1: Schéma komunikačního řetězce podle Shannona

2.2.1 Modely kanálu

Modely kanálu můžeme rozdělit na *diskrétní* a *spojité*. Mezi nejpoužívanější diskrétní kanály patří:

- BSC (*Binary Symmetric Channel*) - kanál bez paměti, charakterizován pravděpodobností chyby v přenosu p, přičemž chybou se rozumí negace bitu,
- BEC (*Binary Erasure Channel*) - charakterizován pravděpodobností výmazu p_e, tzn. když na přijímací straně nelze stanovit odeslaný symbol,
- BSEC - kombinace dvou předchozích, dána chybovost i pravděpodobnost výmazu.

Mezi typické zástupce spojitých kanálu řadíme kanál AWGN (*Additive White Gaussian Noise*), vycházející z normálního rozdělení amplitudy šumu. Kanál lze popsat například rozptylem σ^2. Dále do této skupiny patří různé druhy *fading* kanálů.

2.2.2 Kapacita kanálu

Kapacita diskrétního kanálu je definována jako maximální množství informace, které lze kanálem přenést. Vztahuje se na jeden přenášený symbol.

Pro BSC kanál platí rovnost:

$$C = 1 - H = 1 + p \log_2 p + (1 - p) \log_2 (1 - p) \,[\text{Sh}]. \tag{2.3}$$

Známe-li délku symbolu T_s, můžeme stanovit přenosovou rychlost:

$$C' = \frac{C}{T_s} \,[\text{bit/s, Sh/s}]. \tag{2.4}$$

V souvislosti s AWGN kanálem kapacitou rozumíme *Shannon-Hartleyův vztah*:

$$C = B \log_2 \left(1 + \frac{P_S}{P_N}\right) \,[\text{bit/s}], \tag{2.5}$$

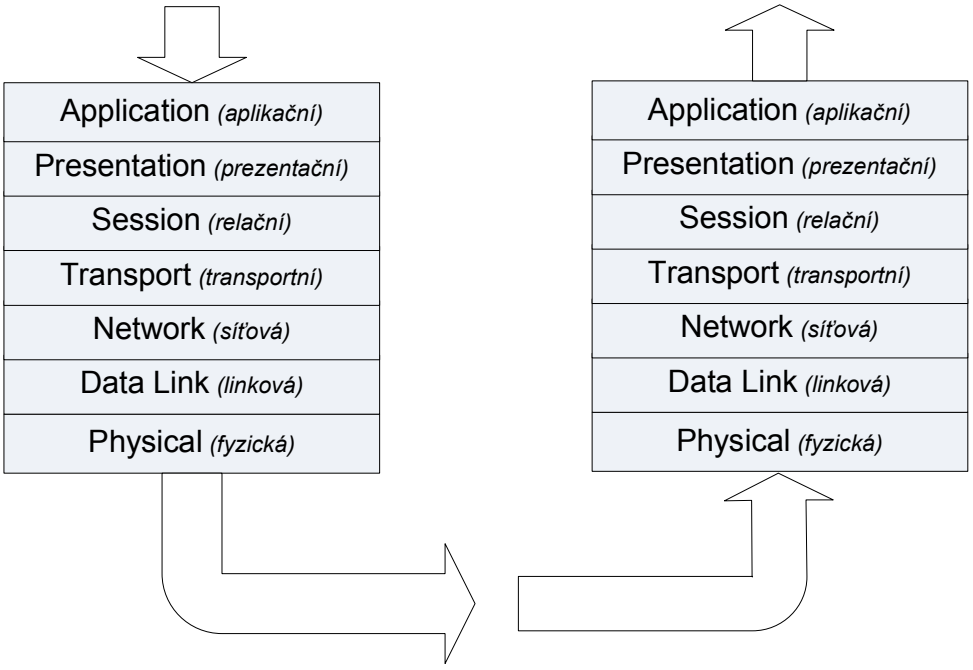
kde B [Hz] značí šířku pásma, P_S [W] výkon signálu a P_N [W] výkon šumu.

2.3 ISO/OSI model

Model byl uveden v roce 1984 jako mezinárodní norma ISO 7498 především za účelem standardizace počítačových sítí. Přináší jiný pohled na datovou komunikaci rozdělením na sedm abstraktních vrstev (obr. 2.2), čímž umožňuje modulární řešení komunikačních systémů. Jednotlivé vrstvy vždy využívají služeb vrstev nižších a poskytují rozhraní vrstvám vyšším.

Uvedeme-li ho do souvislosti se Shannonovým modelem, tak kanál obvykle chápeme jako součást fyzické vrstvy. Zdrojem i příjemcem informace však může být uživatel nad aplikační vrstvou, směrovací protokoly třetí vrstvy či dokonce *autonegotiation* protokol vrstvy fyzické. Můžeme tak hovořit o komunikaci uživatelské a komunikaci stroj-stroj (napříč vrstvami).

Kanálové kódování (a dekódování), kterým se tato práce zabývá, by se mělo odehrávat přímo na fyzické vrstvě. Důvodem je zabezpečení celé posloupnosti přenášené kanálem.



Obr. 2.2: ISO/OSI model

3 Vybrané pojmy z teorie kanálového kódování

3.1 Syntaktická abeceda

Syntaktická abeceda $\mathcal{A}$ je množina všech možných symbolů v přenášené posloupnosti. Počet prvků této množiny se značí obvykle písmenem q.

Příklad: Pro binární abecedu platí

$$\mathcal{A} = \{0, 1\}, \quad q = 2.$$

3.2 Hammingova váha a Hammingova vzdálenost

Hammingova váha $w(\mathbf{v})$ označuje počet nenulových prvků ve vektoru, Hammingova vzdálenost $d(\mathbf{v_1}, \mathbf{v_2})$ pak počet symbolů, kde se dva vektory liší.

Příklad: $\mathbf{v_1} = [1, 0, 0, 1]$, $\mathbf{v_2} = [0, 1, 1, 1]$,

$$\Rightarrow w(\mathbf{v_1}) = 2,\ w(\mathbf{v_2}) = 3,\ d(\mathbf{v_1}, \mathbf{v_2}) = 3.$$

3.3 Kanálový kodér a dekodér

Kanálový *kodér* je zařízení, které realizuje injektivní zobrazení mezi množinou zpráv a množinou kódových slov podle daného předpisu. Ke každému injektivnímu zobrazení existuje zobrazení inverzní a zařízení, které ho realizuje, se nazývá *dekodér*.

Vstupem kanálového kodéru je informační vektor $\mathbf{m}$, jeho výstupem kódový vektor $\mathbf{c}$.

3.4 Blokový kód

Blokový kód $\mathcal{C} =(n, k)$ je definován jako množina kódových slov délky n o počtu prvků q^k.

3.5 Lineární kód

Blokový kód (n, k) je lineární, pokud všechna jeho kódová slova tvoří podprostor lineárního vektorového prostoru $\mathcal{F}_q^n$ nad tělesem $\mathcal{F}_q$ a dimenze tohoto podprostoru je právě k.

Podstatným důsledkem této definice je, že libovolná lineární kombinace dvou kódových slov opět tvoří kódové slovo. Dále v takovém kódu existuje báze, což je lineárně nezávislá množina vektorů, která prostor kódových slov generuje. Příkladem lineárního

kódu je Hammingův kód (tab. 3.1), naopak typickým zástupcem nelineární skupiny je kódování 4B/5B (tab. 3.2).

Příklad: Hammingův kód (tab. 3.1) tvoří vektorový prostor $\mathcal{F}_2^4 \subset \mathcal{F}_2^7$ nad tělesem $\mathcal{F}_2$.

3.6 Informační poměr a redundance kódu

Informační poměr kódu udává množství neredundantní informace v kódovém slově. Obvykle se značí písmeny R nebo Q.

Platí rovnost:

$$R = \frac{\log_q |\mathcal{C}|}{n}, \tag{3.1}$$

kde q je počet prvků abecedy a n délka kódového slova.

Pro blokový kód $\mathcal{C} =(n, k)$ dostaneme:

$$R = \frac{\log_q |q^k|}{n} = \frac{k}{n}, \tag{3.2}$$

kde k je počet prvků informační zprávy.

Redundance r blokového kódu $\mathcal{C} =(n, k)$ označuje rozdíl délky kódu n a počtu informačních symbolů k.

$$r = n - k. \tag{3.3}$$

Příklad: Pro Hammingův kód (7,4), uvedený v tab. 3.1, platí

$$R = \frac{4}{7}, \quad r = 7 - 4 = 3.$$

3.7 Generující matice

Každý lineární blokový kód lze popsat pomocí generující matice $\mathbb{G}$. Řádky generující matice tvoří bázi daného kódu a generují tak prostor všech kódových slov. Každé kódové slovo lze vyjádřit jako lineární kombinaci těchto bázových vektorů. Matice je tedy tvořena k řádky a n sloupci.

Mezi kódovým a informačním vektorem platí vztah:

$$\mathbf{c} = \mathbf{m}\mathbb{G}. \tag{3.4}$$

Matice $\mathbb{G}$ dává informaci jak o podobě lineárního blokového kódu (generuje prostor kódových slov), tak o funkci kodéru (definuje zobrazení do množiny kódových slov).

Tab. 3.1: Převodní tabulka z binárního kódu na Hammingův kód (7,4)

zpráva **m**				kódové slovo **c**						
m_3	m_2	m_1	m_0	c_6	c_5	c_4	c_3	c_2	c_1	c_0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1	1	1
0	0	1	0	0	0	1	0	1	1	0
0	0	1	1	0	0	1	1	0	0	1
0	1	0	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	0	0	1	1
0	1	1	1	0	1	1	1	1	0	0
1	0	0	0	1	0	0	0	0	1	1
1	0	0	1	1	0	0	1	1	0	0
1	0	1	0	1	0	1	0	1	0	1
1	0	1	1	1	0	1	1	0	1	0
1	1	0	0	1	1	0	0	1	1	0
1	1	0	1	1	1	0	1	0	0	1
1	1	1	0	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1

Tab. 3.2: Kódování 4B/5B

zpráva **m**				kódové slovo **c**				
m_3	m_2	m_1	m_0	c_4	c_3	c_2	c_1	c_0
0	0	0	0	1	1	1	1	0
0	0	0	1	0	1	0	0	1
0	0	1	0	1	0	1	0	0
0	0	1	1	1	0	1	0	1
0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	0	1	1
0	1	1	0	0	1	1	1	0
0	1	1	1	0	1	1	1	1
1	0	0	0	1	0	0	1	0
1	0	0	1	1	0	0	1	1
1	0	1	0	1	0	1	1	0
1	0	1	1	1	0	1	1	1
1	1	0	0	1	1	0	1	0
1	1	0	1	1	1	0	1	1
1	1	1	0	1	1	1	0	0
1	1	1	1	1	1	1	0	1

Příklad: Hammingův kód dle tab. 3.1 lze popsat generující maticí

$$\mathbb{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Dále máme zadaný informační vektor $\mathbf{m} = [\ 1 \quad 1 \quad 0 \quad 0\ ]$ a požadujeme výpočet kódového slova. To získáme maticovým násobením:

$$\mathbf{c} = \mathbf{m}\mathbb{G} = [\ 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0\ ].$$

3.8 Kontrolní matice

Matice $\mathbb{H}$ slouží k ověření, zda je libovolný vektor $\mathbf{v}$ (přijatý z kanálu) kódovým slovem. Vynásobením vektoru $\mathbf{v}$ a transponované kontrolní matice získáme tzv. *syndrom* $\mathbf{s}$. Je-li *syndrom* roven nulovému vektoru, jedná se o kódové slovo. Podobně jako generující matice, kontrolní matice $\mathbb{H}$ existuje pro každý lineární blokový kód.

$$\mathbf{s} = \mathbf{v}\mathbb{H}^{\mathrm{T}} = \mathbf{0}, \quad \text{pouze pokud } \mathbf{v} \in \mathcal{C}. \tag{3.5}$$

Mezi generující a kontrolní maticí platí vztah:

$$\mathbb{G}\mathbb{H}^{\mathrm{T}} = \mathbf{0}. \tag{3.6}$$

Příklad: Jako ukázku lze uvést odpovídající matici pro Hammingův kód[1] (7,4) zadaný tab. 3.1,

$$\mathbb{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{3.7}$$

[1]Hammingův kód je takový kód, jehož kontrolní matice je složena ze sloupcových vektorů představující čísla $1 \ldots n$.

Pomocí této matice můžeme ověřit, zda zadané vektory $\mathbf{v_1}$ a $\mathbf{v_1}$ jsou kódovými slovy:

$$\mathbf{v_1} = [\ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ ] \Rightarrow \mathbf{s_1} = \mathbf{v_1}\mathbb{H}^{\mathrm{T}} = [\ 0\ \ 0\ \ 0\ ] \Rightarrow \mathbf{v_1} \in \mathcal{C},$$
$$\mathbf{v_2} = [\ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ ] \Rightarrow \mathbf{s_2} = \mathbf{v_2}\mathbb{H}^{\mathrm{T}} = [\ 1\ \ 0\ \ 1\ ] \Rightarrow \mathbf{v_2} \notin \mathcal{C}.$$

V případě vektoru $\mathbf{v_2}$ lze detekovat a opravit chybu - syndrom odpovídá druhému sloupci $\mathbb{H}$, což znamená, že chyba nastala ve druhém symbolu. Toto neplatí obecně, ale jedná se o vlastnost zmiňovaného kódu. Ve většině případů je dekódování mnohem náročnější proces než kódování.

3.9 Systematické kódování

Kódování je systematické, pokud informační symboly lze nalézt v kódovém slově v nezměněné podobě. Dodržení pořadí není v obecném případě nutné. Zde je vhodné poznamenat, že při definování kódu jako množiny kódových slov (viz kap. 3.4) není systematický kód, ale kódování.

Pro lineární blokové kódování se často používá přísnější podmínka ve vztahu ke generující matici. Aby takové kódování bylo systematické, generující matice spřažená s kódem a kodérem musí mít následující podobu:

$$\mathbb{G} = [\mathbb{I}\,|\,\mathbb{P}] \ \vee \ \mathbb{G} = [\mathbb{P}\,|\,\mathbb{I}]\,, \text{ kde } \mathbb{I} \text{ je jednotková matice.} \tag{3.8}$$

3.10 Nebinární kódy

Dosud byly uvažovány pouze binární kódy. Problematiku však lze zobecnit pro počítání v libovolném konečném tělese.

Těleso je definováno jako množina se dvěma binárními operacemi, která musí splňovat několik vlastností (komutativita, distributivita, existence nulového a neutrálního prvku, atd.). Přesnou definici lze nalézt v dostupné literatuře. Je-li množina konečná, je konečné i těleso.

Nejjednodušší cestou je počítání v aritmetice $\bmod p$. Je-li p prvočíslo, jsou splněny vlastnosti tělesa.

Příklad: Tabulky 3.3 a 3.4 ukazují aritmetiky $\bmod 5$ a $\bmod 4$. Aritmetika $\bmod 5$ generuje těleso, jelikož 5 je prvočíslo, zatímco $\bmod 4$ těleso negeneruje (problém existence inverzního prvku v operaci násobení).

Často požadujeme těleso o jiném počtu prvků než prvočíslo. Pokud bychom náhodně vyplnili tabulky operací (např. symetricky podle diagonály pro zajištění komutativity), obvykle bude porušen zákon distributivity a o těleso se jednat nebude. Tento problém

Tab. 3.3: Aritmetika mod 5

$\oplus$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\otimes$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Tab. 3.4: Aritmetika mod 4

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\otimes$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

lze vyřešit rozšířením existujícího tělesa *iredubicilním polynomem*[2]. Násobení a sčítání v takovém tělese provádíme na úrovni polynomů modulo zmiňovaným iredubicilním polynomem.

Příklad: Číselná reprezentace binárního polynomu $x^5 + x^3 + 1$ je 10101_{BIN} nebo 41_{DEC}.

Příklad: Polynom $x^5 + x^3 + 1$ je iredubicilní. Chceme vypočítat součin $(x^3 + x^1 + 1)\, x^2$ v tělese GF(2^5), tzn. že těleso GF(2) rozšíříme tímto polynomem. Budeme postupovat následovně:

$$\left(x^3 + x^1 + 1\right) x^2 = x^5 + x^3 + x^2,$$

$$\begin{aligned} &\left(x^5 + x^3 + x^2\right) : \left(x^5 + x^3 + 1\right) = 1, \\ &\underline{\oplus \left(x^5 + x^3 + 1\right)} \\ &\qquad\qquad x^2 + 1 \end{aligned}$$

$$\Rightarrow \text{GF}(2^5) : \left(x^3 + x^1 + 1\right) \otimes x^2 = x^2 + 1.$$

Dekadicky pak v takovém tělese platí:

$$11 \otimes 4 = 5.$$

[2] *Iredubicilní polynom* je takový polynom, který není rozložitelný na součin polynomů nižšího stupně.

Příklad: Tělesa GF(4), GF(8) vygenerovaná rozšířením tělesa GF(2) iredubicilními polynomy $x^2 + x + 1 = 7_{\mathrm{DEC}}$ a $x^3 + x + 1 = 11_{\mathrm{DEC}}$.

Tab. 3.5: Počítání v tělese GF(4)

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

$\otimes$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Tab. 3.6: Počítání v tělese GF(8)

$\oplus$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

$\otimes$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

4 Zkoumání vlastností opravných kódů

4.1 Minimální vzdálenost

4.1.1 Definice minimální vzdálenosti

Minimální vzdálenost d_{min} kódu $\mathcal{C}$ je definována jako minimální Hammingova vzdálenost mezi všemi dvojicemi kódových slov. Výrazně ovlivňuje detekční a opravné schopnosti daného kódu.

$$d_{min} = \min_{\mathbf{c_1},\mathbf{c_2} \in \mathcal{C} \wedge \mathbf{c_1} \neq \mathbf{c_2}} d(\mathbf{c_1}, \mathbf{c_2}), \tag{4.1}$$

kde d označuje Hammingovu vzdálenost.

Příklad:

$$\mathbb{G} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \; q = 2 \Rightarrow k = 2, \; n = 3,$$

$$\Rightarrow \mathcal{C} = \{\mathbf{c_0}, \mathbf{c_1}, \mathbf{c_2}, \mathbf{c_3}\} = \{[0,0]\mathbb{G}, \; [0,1]\mathbb{G}, \; [1,0]\mathbb{G}, [1,1]\mathbb{G}\},$$
$$\mathcal{C} = \{[0,0,0], \; [0,1,1], \; [1,0,1], \; [1,1,0]\},$$

$$\Rightarrow d_{min} = \min\left(d(\mathbf{c_0},\mathbf{c_1}), d(\mathbf{c_0},\mathbf{c_2}), d(\mathbf{c_0},\mathbf{c_3}), d(\mathbf{c_1},\mathbf{c_2}), d(\mathbf{c_1},\mathbf{c_3})\right),$$
$$d_{min} = \min\left(2,2,2,2\right) = 2.$$

4.1.2 Vliv na opravné schopnosti kódu

Má-li kód $\mathcal{C}$ minimální vzdálenost d_{min}, je schopen garantovat:

- detekci chyb o násobnosti $t = d_{min} - 1$,
- korekci chyb o násobnosti $t = \dfrac{d_{min} - 1}{2}$.

4.1.3 Singletonova mez

Pro blokový kód $\mathcal{C} = (n, k)$ lze dokázat nerovnost

$$|\mathcal{C}| = q^k \leq q^{n - d_{min} + 1}, \tag{4.2}$$

po úpravě

$$d_{min} \leq n - k + 1. \tag{4.3}$$

4.1.4 Výpočet pomocí Hammingových vah kódových slov

Pro výpočet minimální vzdálenost obecného kódu musíme určit minimum z Hammingových vzdáleností všech kódových dvojic. Pro *lineární* kódy lze výpočet významně zjednodušit následujícím způsobem:

$$\forall \mathbf{c_1}, \mathbf{c_2} \in \mathcal{C} :$$
$$d(\mathbf{c_1}, \mathbf{c_2}) = d(\mathbf{c_1} - \mathbf{c_2}, \mathbf{c_2} - \mathbf{c_2}) = d(\mathbf{c_1} - \mathbf{c_2}, \mathbf{0}).$$

Protože jakákoliv lineární kombinace dvou kódových slov tvoří jiné slovo z vektorového prostoru, platí:

$$d_{min} = \min\left(w\left(\mathbf{c_0}\right), w\left(\mathbf{c_2}\right), \ldots, w\left(\mathbf{c_{q^k-1}}\right)\right). \tag{4.4}$$

Minimální vzdálenost je tedy rovna minimální váze kódového slova (kromě nulového).

Příklad: Určení minimální vzdálenosti d_{min} Hammingova kódu (7,4) zadaného převodní tabulkou 3.1:

$$d_{min} = \min\left(w\left(\mathbf{c_0}\right), \ldots, w\left(\mathbf{c_{15}}\right)\right),$$
$$d_{min} = \min\left(4,3,3,3,3,4,4,3,3,4,4,4,4,3,7\right) = 3.$$

4.1.5 Určení z kontrolní matice

Minimální vzdálenost kódu d_{min} je také možné určit z kontrolní matice $\mathbb{H}$. Platí tvrzení, že minimální vzdálenost kódu je rovna nejmenšímu počtu sloupcových vektorů, jejichž kombinace je lineárně závislá.

Příklad: Opět bude uvažován Hammingův kód (7,4). Kód je zadaný kontrolní maticí 3.7, přičemž vektory $\mathbf{h_i}$ značí sloupce této matice. Výpočet minimální vzdálenosti lze provést v následujících krocích:

1. $\nexists \mathbf{h_i}, \mathbf{h_j} : \mathbf{h_i} \neq \mathbf{h_j} \ \wedge \ \mathbf{h_i} \oplus \mathbf{h_j} = \mathbf{0}$,
2. $\exists \mathbf{h_i}, \mathbf{h_j}, \mathbf{h_k} : \mathbf{h_i} \neq \mathbf{h_j} \neq \mathbf{h_k} \ \wedge \ \mathbf{h_i} \oplus \mathbf{h_j} \oplus \mathbf{h_k} = \mathbf{0} \quad \Rightarrow d_{min} = 3.$

Protože existuje alespoň jedna trojice sloupcových vektorů, jejíž exkluzivní součet dá nulový vektor (trojice je lineárně závislá), konkrétně jsou to kombinace vektorů

$$\{\mathbf{h_0}, \mathbf{h_1}, \mathbf{h_2}\}, \{\mathbf{h_0}, \mathbf{h_3}, \mathbf{h_4}\}, \{\mathbf{h_0}, \mathbf{h_5}, \mathbf{h_6}\}, \{\mathbf{h_1}, \mathbf{h_3}, \mathbf{h_5}\},$$
$$\{\mathbf{h_1}, \mathbf{h_4}, \mathbf{h_6}\}, \{\mathbf{h_2}, \mathbf{h_3}, \mathbf{h_6}\}, \{\mathbf{h_2}, \mathbf{h_4}, \mathbf{h_5}\},$$

a neexistuje kombinace kratší délky, která by tuto podmínku splňovala, minimální vzdálenost Hammingova kódu (7,4) je 3.

4.2 Simulace chybovosti v AWGN kanálu

4.2.1 Praktické použití modelu kanálu

Opravné vlastnosti se často testují pomocí simulace chybovosti v AWGN kanálu. Pod pojmem chybovost je v tomto případě myšlena pravděpodobnost výskytu chyby při přenosu bitu P_b (*BER, Bit Error Rate*). Informační slovo nejprve zakódujeme (kap. 3.7). Získané kódové slovo modulujeme, přidáme aditivní šum, demodulujeme, dekódujeme a upravíme opět na informační slovo. Demodulace v simulačních účelech může být součástí dekódování (kap. 5.3). Porovnáním původního a dekódovaného informačního slova již můžeme stanovit hodnotu BER.

Chybovost se testuje postupně v závislosti na poměru E_b/N_0. E_b vyjadřuje energii vztaženou na bit a má rozměr [J], někdy se zapisuje [J/bit]. N_0 má význam spektrální výkonové hustoty šumu. Spektrální výkonová hustota N_0 se nejčastěji udává v jednotkách [W/Hz], vyjadřuje však také energii a fyzikální rozměr N_0 i E_b je stejný. Zmiňovaný poměr lze pak vyjádřit v [dB].

Mezi rozptylem a spektrální výkonovou hustotou šumu v AWGN kanálu platí vztah:

$$\sigma^2 = \frac{N_0}{2}. \tag{4.5}$$

V této souvislosti se také používá hodnota FER (*Frame Error Rate*), která znamená pravděpodobnost chyby při přenosu celého slova.

Příklad: Zadán odstup signál/šum na bit $10 \log E_b/N_0 = 3\text{dB}$, kódové slovo $\mathbf{c} = [\ 0 \quad 1 \quad 0 \quad 0 \quad 1\ ]$ a informační poměr kódu $R = 0{,}75$. Použita BPSK modulace s energií na bit $E_b = 1\,\text{J}$. Úkolem je určení vektoru po průchodu AWGN kanálem, tzn. posloupnosti přijaté přijímačem.

Po modulaci se zadanou energií získáme vektor:

$$\mathbf{c_{BPSK}} = |E_b = 1| = [\ -1 \quad 1 \quad -1 \quad -1 \quad 1\ ].$$

Ze zadaného odstupu signál/šum určíme spektrální hustotu výkonu šumu a rozptyl kanálu. Hodnota spektrální výkonové hustoty šumu je rovna:

$$N_0 = 10^{-3/10} \doteq 0{,}501,$$

po přepočtení na kódové slovo

$$N_{0c} = \frac{N_0}{R} \doteq 0{,}668.$$

Většina generátorů náhodných čísel dle normálního rozdělení pracuje s jednotkovým rozptylem. Takto vygenerovaný vektor může vypadat např. následovně:

$$\mathbf{r} = [\ 0,67 \quad -1,20 \quad 0,71 \quad 2,76 \quad 0,48\ ].$$

Odmocnina z rozptylu kanálu, potřebná pro výpočet zašuměného vektoru je:

$$\sigma = \sqrt{\frac{N_{0c}}{2}} \doteq 0{,}578.$$

Nyní již můžeme vyjádřit zašuměný vektor po průchodu kanálem:

$$\mathbf{y} = \mathbf{c_{BPSK}} + \sigma\mathbf{r} = [\ -0{,}61 \quad 0{,}31 \quad -0{,}59 \quad 0{,}60 \quad 1{,}28\ ].$$

4.2.2 Chybovost modulace bez kódu

Pravděpodobnost chyby po průchodu AWGN kanálem závisí na typu modulace a stanovuje se pomocí tzv. chybové funkce (erfc) nebo pomocí Q-funkce. Obě tyto funkce vycházejí z normálního rozdělení pravděpodobnosti a je nutné je vyčíslit numericky. Ucelený přehled modulací a chybovosti je uveden v [42].

Pravděpodobnost chyby vztažená na symbol je při použití BPSK modulace:

$$P_{s,\mathrm{BPSK}} = \frac{1}{2}\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right) = \mathrm{Q}\left(\sqrt{\frac{2E_b}{N_0}}\right), \tag{4.6}$$

a při modulace s více stavy (MPSK):

$$P_{s,\mathrm{MPSK}} = \mathrm{erfc}\left(\sqrt{\log_2(M)\frac{E_b}{N_0}}\sin\left(\frac{\pi}{M}\right)\right), \tag{4.7}$$

kde M je počet stavů modulace.

Pravděpobnost chyby vztaženou na bit (BER) pak určíme:

$$BER_{\mathrm{BPSK}} = P_{b,\mathrm{BPSK}} = P_{s,\mathrm{BPSK}} = \frac{1}{2}\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right), \tag{4.8}$$

$$BER_{\mathrm{MPSK}} = P_{b,\mathrm{MPSK}} = \frac{P_{s,\mathrm{MPSK}}}{\log_2(M)} = \frac{1}{\log_2(M)}\mathrm{erfc}\left(\sqrt{\log_2(M)\frac{E_b}{N_0}}\sin\left(\frac{\pi}{M}\right)\right). \tag{4.9}$$

Příklad: Použita modulace 8PSK, chybovost při použití AWGN kanálu bude rovna:

$$BER_{\mathrm{8PSK}} = P_{b,\mathrm{8PSK}} = \frac{1}{3}\mathrm{erfc}\left(\sqrt{\frac{3E_b}{N_0}}\sin\left(\frac{\pi}{8}\right)\right). \tag{4.10}$$

Pro doplnění je možné dále uvést:

$$BER_{4\text{QAM}} = P_{b,4,\text{QAM}} = \frac{1}{2}\text{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right), \tag{4.11}$$

$$BER_{16\text{QAM}} = P_{b,16\text{QAM}} = \frac{3}{8}\text{erfc}\left(\sqrt{\frac{4}{10}\frac{E_b}{N_0}}\right). \tag{4.12}$$

4.2.3 Limit chybovosti s kódem

Na základě vztahů 2.3 a 4.6 lze určit limit chybovosti AWGN kanálu při použití BPSK modulace (binární AWGN kanál) [41]. Výslednou rovnici je nutno řešit numericky. Tento limit závisí na informačním poměru kódu R je znázorněn na obr. 4.2.

Asymptoty těchto křivek jsou:

$$R = 0{,}25 : \lim_{P_b \to 0} \frac{E_b}{N_0} \doteq -0{,}794\,\text{dB}, \tag{4.13}$$

$$R = 0{,}5 : \lim_{P_b \to 0} \frac{E_b}{N_0} \doteq -0{,}187\,\text{dB}, \tag{4.14}$$

$$R = 0{,}75 : \lim_{P_b \to 0} \frac{E_b}{N_0} \doteq 1{,}626\,\text{dB}, \tag{4.15}$$

$$R = 5/6 : \lim_{P_b \to 0} \frac{E_b}{N_0} \doteq 2{,}362\,\text{dB}, \tag{4.16}$$

$$R = 0{,}9 : \lim_{P_b \to 0} \frac{E_b}{N_0} \doteq 3{,}198\,\text{dB}. \tag{4.17}$$

4.2.4 Ultimátní Shannonův limit

Nejprve vyjádříme poměr signál / šum pomocí spektrální účinnosti modulace a poměru signál šum vztaženého na bit. Dostaneme vztah:

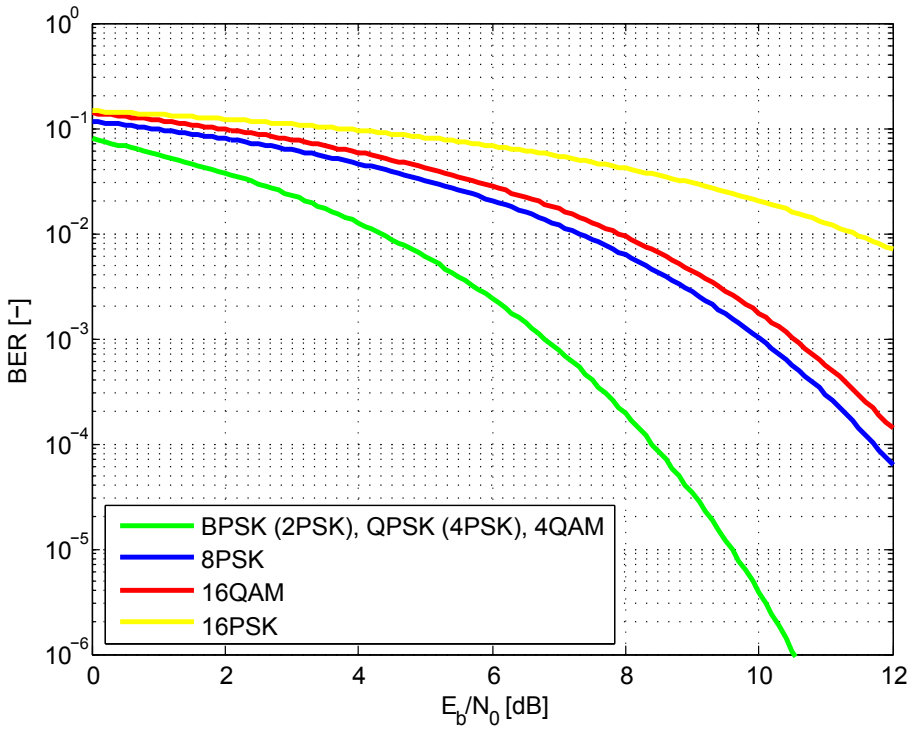
$$\frac{P_S\,[\text{W}]}{P_N\,[\text{W}]} = \frac{RE_b}{BN_0} = \eta\frac{E_b}{N_0}, \tag{4.18}$$

kde R značí bitovou rychlost (bit rate) [bit/s] a B šířku pásma (*bandwidth*, počet vzorků za sekundu) [Hz] a η spektrální účinnost modulace [-].

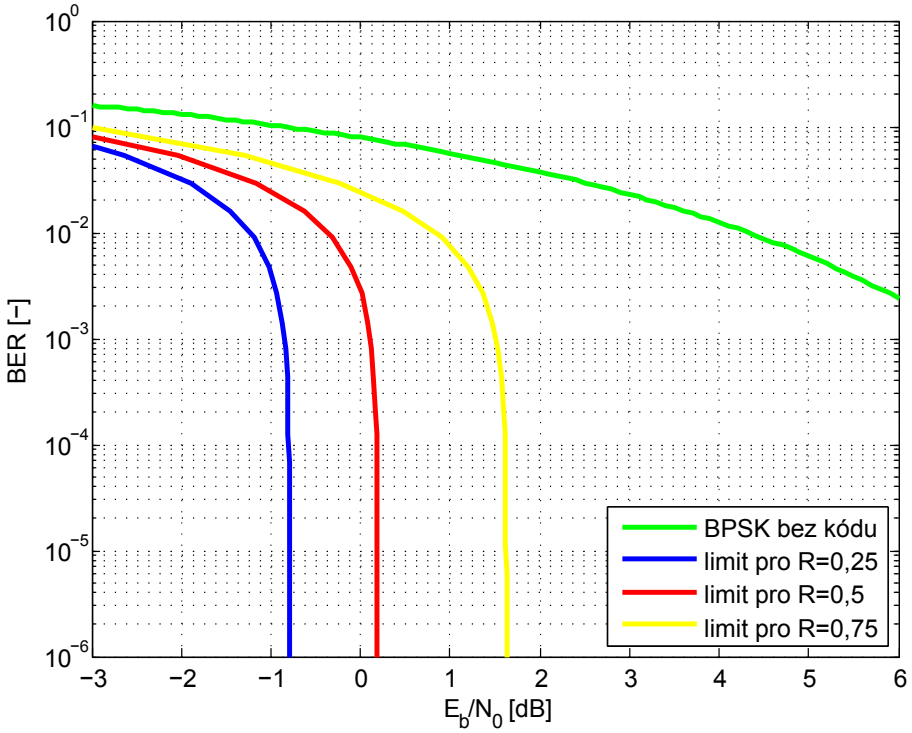
Příklad: Pro BPSK modulaci platí:

$$R = B\log_2 M = B\log_2 2 \Rightarrow R = B.$$

M má význam počtu stavů modulace.



Obr. 4.1: Chybovost modulace v AWGN kanálu



Obr. 4.2: Limit chybovosti binárního AWGN kanálu

Spektrální účinnost BPSK modulace je:

$$\eta = \frac{R}{B} = 1.$$

Příklad: Pro QPSK modulaci platí:

$$R = B \log_2 4 \Rightarrow R = 2B.$$

Dosadíme-li poměr 4.18 do vztahu 2.5, pro dosažitelnou bitovou rychlost R dostaneme:

$$R < B \log_2 \left(1 + \frac{R}{B}\frac{E_b}{N_0}\right), \tag{4.19}$$

Po úpravách a zavedení spektrální účinnosti modulace η pak vyjádříme nerovnost:

$$2^\eta < 1 + \eta\frac{E_b}{N_0},$$

$$\frac{E_b}{N_0} > \frac{2^\eta - 1}{\eta}. \tag{4.20}$$

Pravá strana nerovnosti je rostoucí funkcí a pro spektrální účinnost vždy platí, že $\eta > 0$. Funkce tedy nabývá minima, pokud se η limitně zprava blíží nule.

$$\min \frac{2^\eta - 1}{\eta} = \lim_{\eta \to 0^+} \frac{2^\eta - 1}{\eta} = 2^\eta \ln 2 = \ln 2.$$

Pro zrealizování datového přenosu tedy potřebujeme minimální odstup E_b/N_0:

$$\frac{E_b}{N_0} > \ln 2 \doteq 0{,}693 \doteq -1{,}592\,\text{dB}. \tag{4.21}$$

Při přenosu s kódem přejde spektrální účinnost η na hodnotu, kterou označíme η_c. Platí:

$$\eta_c = \frac{k}{n}\eta,$$

kde k je počet informačních symbolů a n počet kódových symbolů.

Tato spektrální účinnost se limitně blíží nule, právě když se informační poměr kódu $R = k/n$ se blíží nule. Označením R se nyní nemyslí bitová rychlost, ikdyž se často používá stejná symbolika.

Dáme-li vztah 4.21 do souvislosti s předchozí kapitolou a grafem na obr. 4.2, dostaneme

$$R \to 0 : \lim_{P_b \to 0} \frac{E_b}{N_0} \doteq -1{,}592\,\text{dB}. \tag{4.22}$$

4.3 Opravné kódy v bezdrátovém přenosu

Uvažujme vysílací a přijímací anténu. Výkonová hustota v místě příjmu je rovna:

$$\rho = \frac{P_{TX}G_{TX}}{S} = \frac{P_{TX}G_{TX}}{4\pi d^2}, \tag{4.23}$$

kde P_{TX} značí výkon vysílací antény [W], G_{TX} zisk přijímací antény [-] a S má význam kulové plochy [m^2].

Vztah mezi výkonovou hustotou v místě příjmu a výkonem na přijímací anténě určuje apertura antény A. Platí rovnost:

$$P_{RX} = \rho A, \tag{4.24}$$

a lze udvodit, že:

$$A = \frac{\lambda^2}{4\pi}G_{RX}\ [\mathrm{m}^2], \tag{4.25}$$

kde λ je vlnová délka [m].

Na základě předchozích vztahů lze určit výkon na přijímací anténě v závislosti na útlumu volného prostoru. Tento vztah je známý jako *Friisova rovnice*.

$$P_{RX} = \rho A = P_{TX}\frac{1}{(4\pi d)^2}\lambda^2 G_{TX}G_{RX}\ [\mathrm{W}], \tag{4.26}$$

Pro zjednodušení dalších výpočtů budeme uvažovat všesměrové antény. Definujeme

$$G_{TX} \triangleq 1,\ G_{RX} \triangleq 1.$$

V následujících výpočtech jsem se inspiroval dokumentem [21], který se zabývá porovnáním přenosu s kódem a bez kódu z hlediska energie.

Jednou z vlastností přijímače je parametr *NF (Noise Figure)* [dB]. Pomocí tohoto parametru můžeme určit výkon šumu na přijímači. Jinými slovy, čím vyšší je tato hodnota, tím více klesne odstup signál / šum na přijímači. Mezi parametrem NF a výkonem šumu platí vztah:

$$P_N = 10^{NF/10}kTB, \tag{4.27}$$

kde k značí Boltzmannovu konstantu, T termodynamickou teplotu a B šumovou šířku pásma (nerovná se šířce pásma ve vztahu 4.18).

Poměr výkonu signálu na přijímací anténě a šumu na přijímači se musí rovnat vztahu 4.18:

$$\frac{P_S}{P_N} = \frac{P_{RX}}{10^{NF/10}kTB} = \eta\frac{E_b}{N_0}. \tag{4.28}$$

Dosadíme do *Friisovy rovnice* (4.26):

$$P_{TX} = \left(\frac{4\pi}{\lambda}\right)^2 d^2 P_{RX} = \left(\frac{4\pi}{\lambda}\right)^2 d^2 \eta \frac{E_b}{N_0} 10^{NF/10} kTB, \tag{4.29}$$

a můžeme určit vzdálenost přenosu:

$$d = \sqrt{P_{TX} \left(\frac{\lambda}{4\pi}\right)^2 \frac{1}{kTB} \frac{1}{\eta} \frac{1}{10^{NF/10}} \frac{1}{10^{\frac{Eb}{N_0}[dB]/10}}}. \tag{4.30}$$

Chceme-li zjistit maximální teoretické zlepšení dosahu přenosu s kódem vůči přenosu bez kódu, dáme do poměru dvě vzdálenosti - s kódem a bez kódu. Jako proměnné zde budou vystupovat spektrální účinnost bez kódu η, spektrální účinnost s kódem η_c, poměr signálu a šumu vztažený na bit samotné modulace bez použití kódu E_b/N_0 a poměr při použití kódu $(E_b/N_0)_c$. Ostatní veličiny jsou z tohoto pohledu konstanty.

Zmiňovaný poměr vzdáleností pak bude roven:

$$\frac{d_c}{d} = \sqrt{\frac{\eta}{\eta_c} \frac{10^{\frac{Eb}{N_0}[dB]/10}}{10^{\left(\frac{Eb}{N_0}\right)_c[dB]/10}}} = \sqrt{\frac{n}{k} \frac{10^{\frac{Eb}{N_0}[dB]/10}}{10^{\left(\frac{Eb}{N_0}\right)_c[dB]/10}}}. \tag{4.31}$$

Příklad: BPSK modulace potřebuje pro dosažení chybovosti $BER = 10^{-6}$ odstup $E_b/N_0 = 10{,}5\,\text{dB}$. Kódem, který je charakterizován informačním poměrem $k/n = 0{,}5$, tuto hodnotu snížíme na $E_b/N_0 = 6\,\text{dB}$ (*kódový zisk* pro danou chybovost je 4,5 dB). Maximální zlepšení dosahu při použití tohoto kódu bude:

$$\frac{d_c}{d} = \sqrt{\frac{1}{0{,}5} 10^{4{,}5/10}} \doteq 2{,}37.$$

Toto je samozřejmě teoretická hodnota při uvažování útlumu volného prostoru bez dalších rušení a za předpokladu dokonalé synchronizace. Je také předpokládáno, že šum přijímače neznemožní přenos.

5 LDPC kódy

5.1 Úvod a historie

LDPC (*Low Density Parity Check*) jsou rodinou kódů původně představenou R.G. Gallagerem na začátku 60.let [12],[13] a jak již název říká, jedná se o kódy s řídkou kontrolní maticí, tzn. s maticí $\mathbb{H}$, kde počet jedniček je menší než počet nul.

Významným přínosem problematiky byla práce R.M. Tannera [57], který mimo jiné představil bipartitní grafy pro $\mathbb{H}$ matici, dnes známé jako *Tannerovy grafy*. Na těch je založen dekódovací algoritmus. LDPC kódy však zůstávali dlouhou dobu bez povšimnutí a dostatečného prozkoumání, zastíněné jinými moderními kódy (Reed-Solomonovi kódy apod.). Jedním z celkem logických důvodů byla výpočetně náročnější implementace LDPC kódů, především jejich dekódování. Obrat nastal zhruba v polovině 90.let, kdy hovoří se o znovuobjevení těchto kódů Davidem J.C. MacKayem [36],[37]. V současné době dochází ke značné popularizaci LDPC kódů a jsou nasazovány v mnoha moderních aplikacích, jako je například televizní vysílání (DVB-S2, DVB-T2, DVB-C2), desetigigabitový ethernet (IEEE 802.3, 10GBase-T), WiFi bezdrátová komunikace (IEEE 802.11n) a mnoho dalších.

5.2 Tannerův graf

Tannerův graf kódu $\mathcal{C}$ je neorientovaný graf, jehož část matice sousednosti tvoří matice $\mathbb{H}$, množiny vrcholů příslušící řádkům a sloupcům matice $\mathbb{H}$ jsou navzájem disjunktní a v grafu se nenalézají jiné hrany než ty, které jsou zobrazeny v $\mathbb{H}$ matici. Jinak řečeno, neexistuje společný vrchol, který by náležel zároveň sloupci i řádku kontrolní matice, a v grafu se nacházejí pouze hrany uvedené v $\mathbb{H}$ matici.

Uzly, které přísluší řádkům kontrolní matice se obvykle nazývají *check nodes*, uzly příslušící sloupcům kontrolní matice se označují jako *variable* nebo *bit nodes*. Terminologie plyne z funkce kontrolní matice, kde každý řádek matice reprezentuje jednu lineární rovnici (vztah 3.5).

Mezi těmito dvěma množinami nevede hrana a graf je tedy bipartitní. Jednou z důležitých vlastností takových grafů je, že zde neexistuje cyklus liché délky.

Příklad: Pro názornost bude opět uvažován Hammingův kód $(7, 4)$ zadaný maticí 5.1. Tannerův graf takového kódu obsahuje 2 množiny vrcholů $\{v_0, v_1, \ldots, v_6\}$ a $\{c_0, c_1, c_2\}$. Z vrcholu v_0 vede hrana do vrcholů c_1 a c_2, z vrcholu v_1 vede hrada do vrcholů c_0 a c_2 atd.

$$\mathbb{H} = \begin{array}{c} \\ c_0 \\ c_1 \\ c_2 \end{array} \begin{array}{c} \begin{array}{ccccccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \left[\begin{array}{ccccccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \end{array}. \tag{5.1}$$

Příklad: Kompletní matice sousednosti Tannerova grafu pro zmiňovaný kód je čtvercová a má podobu

$$\mathbb{A} = \begin{bmatrix} 0 & \mathbb{H} \\ \mathbb{H}^{\mathrm{T}} & 0 \end{bmatrix} = \begin{array}{c} \\ c_0 \\ c_1 \\ c_2 \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccccccc} c_0 & c_1 & c_2 & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \left[\begin{array}{cccccccccc} & & & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ \ldots & 0 & \ldots & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ & & & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & & & & & & & \\ 1 & 0 & 1 & & & & & & & \\ 1 & 1 & 0 & & & & & & & \\ 1 & 1 & 1 & & & \ldots & 0 & \ldots & & \\ 1 & 0 & 0 & & & & & & & \\ 0 & 1 & 0 & & & & & & & \\ 0 & 0 & 1 & & & & & & & \end{array}\right] \end{array}.$$

Nyní můžeme definovat množinu hran E, obsahující neuspořádané dvojice vrcholů

$$E \triangleq \{\{c_i, v_j\},\, i, j : \mathbb{H}_{i,j} = 1\}\,, \tag{5.2}$$

v tomto konkrétním případě je rovna:

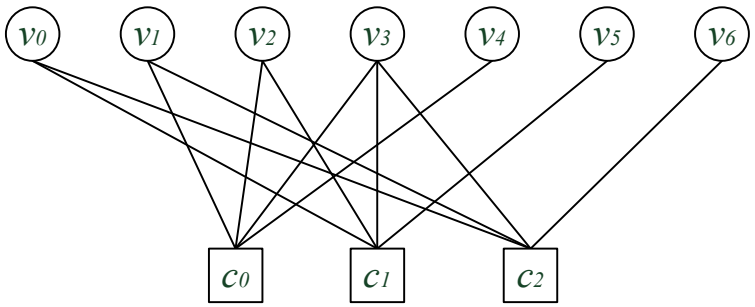
$$E = \{\{c_0, v_1\}, \{c_0, v_2\}, \{c_0, v_3\}, \{c_0, v_4\}, \{c_1, v_0\}, \{c_1, v_2\},$$
$$\{c_1, v_3\}, \{c_1, v_5\}, \{c_2, v_0\}, \{c_2, v_1\}, \{c_2, v_3\}, \{c_2, v_6\}\}.$$

Pro zjednodušení a budoucí vysvětlení dekódovacího algoritmu dále definujeme množinu uspořádaných dvojic vrcholových indexů (stále se však jedná o neorientovaný graf).

$$\mathcal{E} \triangleq \{(i, j),\, i, j : \{c_i, v_j\} \in E\}. \tag{5.3}$$

V případě uvažovaného grafu bude množina vypadat následovně:

$$\mathcal{E} = \{(0,1)\,, (0,2)\,, (0,3)\,, (0,4)\,, (1,0)\,, (1,2)\,,$$
$$(1,3)\,, (1,5)\,, (2,0)\,, (2,1)\,, (2,3)\,, (2,6)\}.$$



Obr. 5.1: Tannerův graf

5.3 Dekódování LDPC kódů

Dekódování LDPC kódů se obvykle provádí iteračně předáváním zpráv po hranách Tannerova grafu matice $\mathbb{H}$. Tyto algoritmy se souhrně označují jako BP (*Belief Propagation*) a protože se zde pracuje s pravděpodobnostmi, jedná se o tzv. *soft* dekódování. Algoritmů je několik a liší především ve způsobu výpočtu předávaných zpráv a následném provádění odhadu správného kódového slova. Zde budou uvedeny dva často používané.

5.3.1 SP algoritmus

SP (*Sum-Product*) algoritmus je prvním ze zmiňovaných způsobů dekódování. Pro jeho popis nejdříve definujeme množiny uzlových indexů $\mathcal{M}$ a $\mathcal{N}$. Množina $\mathcal{M}$ obsahuje všechny indexy *check* uzlů, ze kterých vede hrana do *variable* uzlu j, množina $\mathcal{N}$ pak všechny indexy *variable* uzlů, z nichž vede hrana do *check* uzlu i. Ve vztahu kontrolní matici, $\mathcal{M}$ přísluší sloupci j a $\mathcal{N}$ řádku i.

$$\begin{aligned} \mathcal{M}_j &\triangleq \{i \,:\, (i,j) \in \mathcal{E}\}, \\ \mathcal{N}_i &\triangleq \{j \,:\, (i,j) \in \mathcal{E}\}. \end{aligned} \tag{5.4}$$

Prvním krokem dekódovacího procesu je inicializace dle přijatého kódového slova z přenosového kanálu. Provede se výpočet podmíněných pravděpodobností $P(c_j \,|\, y_j)$ a první polovina iterace, kde jsou tyto pravděpodobnosti poslány směrem k *check* uzlům jako zprávy q_{ij} po hranách Tannerova grafu. Označení y_i představuje symbol přijatý z kanálu a $P(c_j \,|\, y_j)$ má význam podmíněné pravděpodobnosti, že byl vyslán symbol c_j, když byl přijat symbol y_j. Tuto pravděpodobnost určíme z modelu kanálu. Index j je zvolen z důvodu příslušnosti sloupci kontrolní matice.

Definujeme

$$p_j^{(0)} \triangleq P(c_j = 0 \,|\, y_j), \tag{5.5}$$

$$p_j^{(1)} \triangleq P(c_j = 1 \,|\, y_j), \tag{5.6}$$

a odešleme

$$q_{ij}^{(0)} = p_j^{(0)},\ q_{ij}^{(1)} = p_j^{(1)}. \tag{5.7}$$

Pro kanál BSC dostáváme hodnoty

$$\begin{aligned} p_j^{(0)} &= |\mathrm{BSC}, y_j \in \{0,1\}\,| = \begin{cases} 1-p, & y_j = 0 \\ p, & y_j = 1 \end{cases}, \\ p_j^{(1)} &= 1 - p_j^{(0)}| = \begin{cases} p, & y_j = 0 \\ 1-p, & y_j = 1 \end{cases}, \end{aligned} \tag{5.8}$$

kde p je z intervalu <0,1> a značí aktuální chybovost kanálu.
Pro kanál s výmazem (BEC) jsou tyto pravděpodobnosti rovny

$$\begin{aligned} p_j^{(0)} &= |\mathrm{BEC}, y_j \in \{0,1,\mathrm{e}\}\,| = \begin{cases} 0, & y_j = 1 \\ 1, & y_j = 0 \\ 1/2\ , & y_j = \mathrm{e} \end{cases}, \\ p_j^{(1)} &= \begin{cases} 1, & y_j = 1 \\ 0, & y_j = 0 \\ 1/2\ , & y_j = \mathrm{e} \end{cases}. \end{aligned} \tag{5.9}$$

Rozšíříme-li vztah 5.8 o výmaz, získáme tak kanál BSEC

$$\begin{aligned} p_j^{(0)} &= |\mathrm{BSEC}, y_j \in \{0,1,\mathrm{e}\}\,| = \begin{cases} 1-p, & y_j = 0 \\ p, & y_j = 1 \\ 1/2 & y_j = \mathrm{e} \end{cases}, \\ p_j^{(1)} &= 1 - p_j^{(0)}| = \begin{cases} p, & y_j = 0 \\ 1-p, & y_j = 1 \\ 1/2 & y_j = \mathrm{e} \end{cases}. \end{aligned} \tag{5.10}$$

V simulacích pak často pracujeme s kanálem AWGN a BPSK modulací:

$$\begin{aligned} p_j^{(0)} &= |\mathrm{AWGN}, y_j \in \mathbb{R}| = \frac{1}{1 + \mathrm{e}^{2y_j/\sigma^2}}, \\ p_j^{(1)} &= 1 - p_j^{(0)} = \frac{1}{1 + \mathrm{e}^{-2y_j/\sigma^2}}. \end{aligned} \tag{5.11}$$

Příklad: Z kanálu BSC byl přijatý symbol 1, chybovost kanálu je 90%. Inicializační pravděpodobnosti budou

$$p^{(0)} = 1/10,\ p^{(1)} = 9/10.$$

Příklad: Z AWGN kanálu s rozptylem $\sigma^2 = 2$ byla přijata hodnota $-1{,}86$. Použita BPSK modulace.

$$p^{(1)} = \frac{1}{1+\mathrm{e}^{-2y_j/\sigma^2}} = \frac{1}{1+\mathrm{e}^{1{,}86}} = 0{,}13,\ p^{(0)} = 1 - p_j^{(1)} = 0{,}87.$$

Po inicializaci podmíněných pravděpodobností p_j a předání zpráv q_{ij} dokončíme iteraci zpětným předáním r_{ij} a realizací aktuálního odhadu. Hodnoty r_{ij}^a vyjadřují pravděpodobnost, že je splněna parita i-tého řádku kontrolní matice pro kódové slovo se symbolem a na pozici j. Určíme je na základě zpráv přijatých od *variable* uzlů, přičemž výpočtu se účastní všechny zprávy q_{ij} přijaté od *variable* uzlů kromě zprávy odeslané z uzlu, který má být nyní příjemcem.

$$r_{ij}^{(0)} = \frac{1}{2} + \frac{1}{2} \prod_{j' \in \{\mathcal{N}_i \setminus j\}} \left(1 - 2q_{ij'}^{(1)}\right), \tag{5.12}$$

$$r_{ij}{}^{(1)} = 1 - r_{ij}^{(0)}. \tag{5.13}$$

Příklad: Uvedený příklad se vztahuje ke kompletní ukázce dekódování, která je zařazená v příloze (kap. 10.1). Uvažujme, že *check* uzel č. 4 je propojen hranami s *variable* uzly č. 8, 9, 10 a 16. Tento uzel přijal zprávy

$$q_{4,8}^{(1)} = 0{,}744,\ q_{4,9}^{(1)} = 0{,}024,\ q_{4,10}^{(1)} = 0{,}965,\ q_{4,16}^{(1)} = 0{,}996.$$

Variable uzlu č. 9 budou předány:

$$\begin{aligned} r_{4,9}^{(0)} &= \frac{1}{2} + \frac{1}{2} \prod_{j' \in \{\{8,9,10,16\} \setminus 9\}} \left(1 - 2q_{ij'}^{(1)}\right), \\ &= \frac{1}{2} + \frac{1}{2} (1 - 0{,}744)(1 - 0{,}965)(1 - 0{,}996) \doteq 0{,}275, \\ r_{4,9}^{(1)} &= 1 - r_{4,9}^{(0)} \doteq 0{,}725. \end{aligned}$$

Po přijetí zpráv r_{ij} vypočteme nové aktuální pravděpodobnosti Q_j jednotlivých symbolů a provedeme odhad vyslaných symbolů $\hat{\mathbf{c}}$. Je-li tento odhad prvkem množiny kódových slov, dekódování končí. Pokud potřebujeme znát skutečnou hodnotu

pravděpodobnosti, normujeme hodnoty Q_j pomocí konstanty K_j na jednotkový součet. Pro dekódování není normování nutné.

$$Q_j^{(0)} = \mathrm{K_j} p^{(0)} \prod_{i \in \mathcal{M}_j} r_{ij}^{(0)}, \tag{5.14}$$

$$Q_j^{(1)} = \mathrm{K_j} p^{(1)} \prod_{i \in \mathcal{M}_j} r_{ij}^{(1)}, \tag{5.15}$$

$$\mathrm{K_j} : Q_j^{(0)} + Q_j^{(1)} = 1, \tag{5.16}$$

$$\hat{c_j} = \begin{cases} 1, \ Q_j^{(1)} > Q_j^{(0)} \\ 0, \ \text{jinak} \end{cases} . \tag{5.17}$$

Příslušnost do množiny kódových slov ověříme platností rovnice 3.5:

$$\hat{\mathbf{c}} \mathbb{H}^{\mathrm{T}} = \mathbf{0}. \tag{5.18}$$

Příklad: Variable uzel č. 0 obdržel při dekódování od *check* uzlů 6 a 10 zprávy

$$r_{6,0}^{(1)} = 0{,}099, \ r_{10,0}^{(1)} = 0{,}426,$$

$$\Rightarrow r_{6,0}^{(0)} = 0{,}901, \ r_{10,0}^{(0)} = 0{,}574.$$

Přijatý symbol z kanálu je 0 a chybovost kanálu je 10%. Této chybovosti odpovídají pravděpodobnosti

$$p^{(1)} = 0{,}1, \ p^{(0)} = 0{,}9.$$

Aktuální odhad dekódovaného symbolu č. 0 bude:

$$\begin{aligned} Q^{(0)} &= \mathrm{K_0} \cdot 0{,}9 \cdot 0{,}901 \cdot 0{,}574 \doteq 0{,}465 \cdot \mathrm{K_0}, \\ Q^{(1)} &= \mathrm{K_0} \cdot 0{,}1 \cdot 0{,}099 \cdot 0{,}426 \doteq 0{,}0042 \cdot \mathrm{K_0}, \\ \Rightarrow K_0 &\doteq 2{,}129, \\ &\Rightarrow Q^{(0)} \doteq 0{,}991, \ Q^{(1)} \doteq 0{,}009, \\ Q^{(0)} > Q^{(1)} &\Rightarrow \hat{c_0} = 0. \end{aligned}$$

Pokud nepotřebujeme přesné vyčíslení pravděpodobností pro jiné účely, násobení konstantou $\mathrm{K_j}$ není nutné. Výsledek to v tomto případě samozřejmě nijak neovlivní.

Jestliže ani po první iteraci není zpráva dekódována, pokračujeme iterací druhou

a případně dalšími. Vypočteme hodnoty q_{ij} (vztahy 5.19 až 5.21), přičemž opět využijeme zprávy přijaté po hranách grafu kromě té, kterou odeslal příjemce právě počítané q_{ij}. Provedeme normování na jednotkový součet, odešleme směrem k *check* uzlům a dokončíme iteraci pomocí kroků 5.12 až 5.17, případně začneme iteraci další krokem 5.19. Pokud se zprávu nepodaří dekódovat po určitém počtu iterací (např. 50), dekódovací proces končí jako neúspěšný.

$$q_{ij}^{(0)} = \mathrm{K_{ij}} p^{(0)} \prod_{i' \in \{\mathcal{M}_j \backslash i\}} r_{i'j}^{(0)}, \tag{5.19}$$

$$q_{ij}^{(1)} = \mathrm{K_{ij}} p^{(1)} \prod_{i' \in \{\mathcal{M}_j \backslash i\}} r_{i'j}^{(1)}, \tag{5.20}$$

$$\mathrm{K_{ij}}: \; q_{ij}^{(0)} + q_{ij}^{(1)} = 1. \tag{5.21}$$

Příklad *Variable* uzel č.0 přijal zprávy

$$r_{6,0}^{(1)} = 0{,}099,\ r_{10,0}^{(1)} = 0{,}426,\ r_{6,0}^{(0)} = 0{,}901,\ r_{10,0}^{(0)} = 0{,}574.$$

Výpočet zpráv pro *check* uzel č.7:

$$q_{6,0}^{(0)} = \mathrm{K_{6,0}} p^{(0)} \prod_{i' \in \{\{6,10\} \backslash 6\}} r_{10,0}^{(0)} = 0{,}9 \cdot 0{,}574 \cdot \mathrm{K_{6,0}} = 0{,}5166 \cdot \mathrm{K_{6,0}},$$

$$q_{6,0}^{(1)} = \mathrm{K_{6,0}} p^{(1)} \prod_{i' \in \{\{6,10\} \backslash 6\}} r_{10,0}^{(1)} = 0{,}1 \cdot 0{,}426 \cdot \mathrm{K_{6,0}} = 0{,}0426 \cdot \mathrm{K_{6,0}},$$

$$\Rightarrow \mathrm{K_{6,0}} \doteq 1{,}788.$$

$$\Rightarrow q_{6,0}^{(0)} \doteq 0{,}924,\ q_{6,0}^{(1)} \doteq 0{,}076.$$

Lze namítnout, že za velmi nepříznivých okolností může proces konvergovat k jinému slovu, než bylo původně vysláno, a přesto dekódování skončí úspěšně. Zabezpečení tohoto jevu je však úkolem vyšších protokolových vrstev.

5.3.2 SP v logaritmickém zobrazení a min-sum algoritmus

Protože násobení je výpočetně a časově mnohem náročnější operace než sčítání, zavádí se modifikace výše popsaného algoritmu. Ta je založena na vlastnostech logaritmování součinu a podílu. Předpokládejme surjektivní zobrazení L,

$$L: \; L(x, y) = \ln \frac{x}{y}, \tag{5.22}$$

a definujme počáteční pravděpodobnosti a zprávy předávané po hranách grafu v množině obrazů

$$L(p_j) \triangleq L(p_j^{(0)}, p_j^{(1)}) = \ln \frac{p_j^{(0)}}{p_j^{(1)}}, \tag{5.23}$$

$$L(r_{ij}) \triangleq L(r_{ij}^{(0)}, r_{ij}^{(1)}) = \ln \frac{r_{ij}^{(0)}}{r_{ij}^{(1)}}, \tag{5.24}$$

$$L(q_{ij}) \triangleq L(q_{ij}^{(0)}, q_{ij}^{(1)}) = \ln \frac{q_{ij}^{(0)}}{q_{ij}^{(1)}}. \tag{5.25}$$

Pracujeme-li s AWGN kanálem, platí rovnost

$$L(p_j) = |\mathrm{AWGN}| = -\frac{2y_i}{\sigma^2}. \tag{5.26}$$

Je možné odvodit, že ve zprávách $L(r_{ij})$ se vyskytuje opakující se funkce, často označovaná jako ϕ. Spolu s touto funkcí mají zprávy předávané směrem k *variable* uzlům následující podobu:

$$\phi(x) \triangleq -\ln\left(\tanh\frac{x}{2}\right) = \ln\frac{\mathrm{e}^x+1}{\mathrm{e}^x-1}, \tag{5.27}$$

$$L(r_{ij}) = \prod_{j'\in\{\mathcal{N}_i\setminus j\}} \mathrm{sign}\left(L(q_{ij'})\right)\cdot\phi\left(\sum_{j'\in\{\mathcal{N}_i\setminus j\}}\phi\left|L(q_{ij'})\right|\right). \tag{5.28}$$

Přesné vyčíslování funkce ϕ v reálném čase může být problematické a lze ho řešit např. vyhledávací tabulkou. Další a hojně využívaný způsob je založen na přibližném vyjádření druhého činitele ve vztahu 5.28, čímž je vyčíslování funkce ϕ úplně eliminováno. Takový algoritmus se pak obvykle označuje jako *min-sum*.

$$L(r_{ij}) \approx \prod_{j'\in\{\mathcal{N}_i\setminus j\}} \mathrm{sign}\left(L(q_{ij'})\right)\cdot\min\left|L(q_{ij'})\right|. \tag{5.29}$$

Zprávy $L(q_{ij})$ směrem k *check* uzlům se vyjádří prostým součtem

$$L(q_{ij}) = L(p_j) + \sum_{i'\in\{\mathcal{M}_j\setminus i\}} L(r_{ij}). \tag{5.30}$$

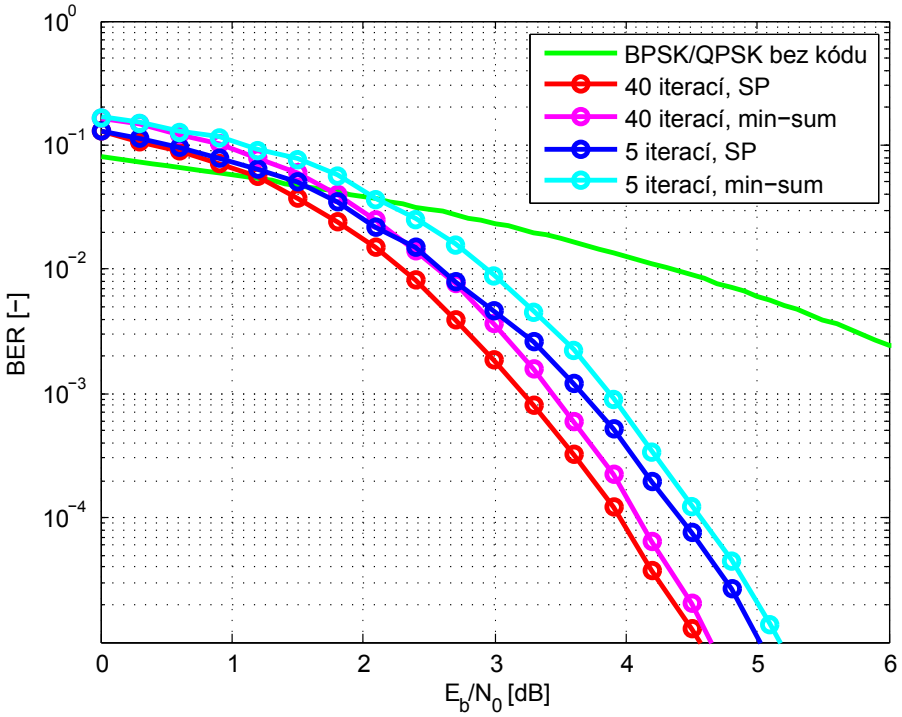
Po každém výpočtu a předání $L(r_{ij})$ vypočteme aktuální odhad kódového slova

$$L(Q_j) = L(p_j) + \sum_{i\in\mathcal{M}_j} L(r_{ij}), \tag{5.31}$$

$$\hat{c}_j = \begin{cases} 1, & L(Q_j) < 0 \\ 0, & \text{jinak} \end{cases}, \tag{5.32}$$

a otestujeme, zda se syndrom rovná nulovému vektoru. Pokud ano, dekódování skončí, stejně jako v případě SP algoritmu.

Porovnání SP a min-sum algoritmu ukázán v simulaci na obr. 5.2. Pro návrh a simulaci kódu byly využity naprogramované třídy popsané v kap. 6.



Obr. 5.2: Porovnání algoritmů pro kód LDPC (128,64)

Příklad: Mějme LDPC kód zadaný kontrolní maticí $\mathbb{H}$ a uvažujme vyslané kódové slovo $\mathbf{c}$, chybový vektor $\mathbf{e}$ a přijaté slovo $\mathbf{c}'$,

$$\mathbf{c}' = \mathbf{c} \oplus \mathbf{e}. \tag{5.33}$$

$$\mathbb{H}=\begin{bmatrix}
0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&1&0&0&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&1&0\\
0&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0\\
0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0
\end{bmatrix},$$

$$\mathbf{c} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$
$$\mathbf{e} = [\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ ],$$
$$\mathbf{c}' = [\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$

Odhady při použití SP algoritmu:

$$\hat{\mathbf{c}}|_{\text{it.0}} = [\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.1}} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.2}} = [\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.3}} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$

Odhady při použití min-sum algoritmu:

$$\hat{\mathbf{c}}|_{\text{it.0}} = [\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ]$$
$$\hat{\mathbf{c}}|_{\text{it.1}} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.2}} = [\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.3}} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.4}} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ].$$

Kompletní ukázka dekódování tohoto vektoru včetně předávaných zpráv uvedena v příloze (kap. 10.1).

5.3.3 Dekódování nebinárních LDPC kódů

Dosud byly popsány algoritmy pro dekódování v binární abecedě. Je však možné provést zobecnění pro libovolné konečné těleso. Dekódování je založené na stejném principu jako pro binární abecedu, tzn. předáváním hodnot q_{ij} a r_{ij} po hranách Tannerova grafu. Pouze jejich výpočet je potřeba modifikovat.

Poměrně komplikovaným problémem je výpočet zpráv $r_{ij}^{(a)}$. Platí rovnost

$$r_{ij}^{(a)} = \sum_{\mathbf{c}\,:\,c_j=a \,\wedge\, \mathbf{ch_i'}=\mathbf{0}} \left(\prod_{j' \in \{\mathcal{N}_i \setminus j\}} \left(q_{ij'}^{(c_{j'})} \right) \right), \mathbf{c} \in \mathcal{C}. \tag{5.34}$$

Provádí součet přes všechna kódová slova $\mathbf{c}$ se symbolem a na pozici i, která navíc vyhovují i-té rovnici kontrolní matice $\mathbb{H}$. Zjišťuje se tak pravděbodobnost, že symbol na pozici j je roven právě a. Vektor $\mathbf{h_i'}$ označuje i-tý řádek matice $\mathbb{H}$.

V dostupných materiálech se splnění i-té rovnice vyjadřuje jako podmíněná pravděpobnost se zavedením další proměnné z_i (uvedeno níže). Definice této proměnné bohužel často chybí a měla by vyjadřovat splnění rovnice, z čehož pak plyne, že hodnota pravděpodobnosti je prvkem množiny $\{0, 1\}$. Dle mého názoru může být tento způsob, zejména pro rychlé pochopení algoritmu, mírně matoucí. Výrazy 5.34 a 5.35 však vyjadřují totéž.

$$r_{ij}^{(a)} = \sum_{\mathbf{c}\,:\,x_j=a} p(z_i|\mathbf{c}) \left(\prod_{j' \in \{\mathcal{N}_i \setminus j\}} \left(q_{ij'}^{(c_{j'})} \right) \right). \tag{5.35}$$

Příklad: Dána kontrolní matice v tělese mod 3.

$$\mathbb{H} = \begin{array}{c} i \\ \\ \\ \end{array} \begin{bmatrix} & & \cdots & & & \\ 0 & 1 & 1 & 0 & 2 & 0 \\ & & \cdots & & & \end{bmatrix},$$

Pro výpočet $r_{i,1}^{(1)}$ (i značí index zadaného řádku matice, na pozici 1 symbol 1) je potřeba udělat součet přes tato kódová slova:

$$\Big\{ [\ c_0 \quad 1 \quad 0 \quad c_3 \quad 1 \quad c_5\ ], [\ c_0 \quad 1 \quad 1 \quad c_3 \quad 2 \quad c_5\ ], [\ c_0 \quad 1 \quad 2 \quad c_3 \quad 0 \quad c_5\ ],\ c_k \in \mathcal{A} \Big\}.$$

Na symbolech na pozicích 0, 3, 5 tedy nezáleží. Na těchto pozicích jsou nuly v kontrolní matici a v příslušném grafu zde nevedou hrany. Hodnotu $r_{i,1}^{(1)}$ můžeme tedy vyčíslit takto:

$$r_{i,1}^{(1)} = \sum_{\mathbf{c}\,:\,c_j=a \,\wedge\, \mathbf{ch_i'}=\mathbf{0}} \left(\prod_{j' \in \{\{1,2,4\}\setminus 1\}} \left(q_{ij'}^{(c_{j'})} \right) \right) = 3^3 q_{i,2}^{(0)} q_{i,4}^{(1)} + 3^3 q_{i,2}^{(1)} q_{i,4}^{(2)} + 3^3 q_{i,2}^{(2)} q_{i,4}^{(0)}.$$

Vztahy pro zprávy předávané směrem od *variable* k *check* uzlům jsou obdobné jako pro binární abecedu. Výpočet $q_{ij}^{(a)}$ je tedy mnohem jednodušší než výpočet $r_{ij}^{(a)}$.

$$q_{ij}^{(a)} = \mathrm{K_{ij}} p^{(a)} \prod_{i' \in \{\mathcal{M}_j \setminus i\}} r_{i'j}^{(a)}, \tag{5.36}$$

$$\mathrm{K_{ij}} : \sum_a q_{ij}^{(a)} = 1. \tag{5.37}$$

Odhad realizujeme jako

$$\hat{x_j} = \underset{a}{\operatorname{argmax}} \left(p^{(a)} \prod_{i \in \{\mathcal{M}_j\}} r_{ij}^{(a)} \right). \tag{5.38}$$

Liší se i výpočet inicializačních pravděpodobností, pracujeme-li s AWGN kanálem. Ty se udávají proporciálně k vlastnostem kanálu. Pro BPSK modulaci je to:

$$p_j^{(a)} = p(y_j \,|\, c_j = a)\,|_{GF} \;\propto\; \mathrm{e}^{\dfrac{((y_j\,\mathrm{BIN})_k \pm 1)^2}{2\sigma^2}}. \tag{5.39}$$

Příklad: Pomocí BPSK modulace s energií 1J na bit byl odeslán do AWGN kanálu, popsaného rozptylem $\sigma^2 = 1$, symbol $2 = 10_{\mathrm{BIN}} \leftrightarrow [\ 1 \quad -1\ ]_{\mathrm{BPSK}}$. Na příjmu jsou hodnoty [1,2 $\quad -0{,}9$]. Inicializační pravděpodobnosti pro dekódování v tělese GF(4) vypadají následovně:

$$\begin{aligned}
p_j^{(0)} &= \mathrm{K_j e}^{\frac{(1{,}2-1)^2}{2\sigma^2}} \mathrm{e}^{\frac{(-0{,}9-1)^2}{2\sigma^2}} = 6{,}203\mathrm{K_j}, \\
p_j^{(1)} &= \mathrm{K_j e}^{\frac{(1{,}2-1)^2}{2\sigma^2}} \mathrm{e}^{\frac{(-0{,}9+1)^2}{2\sigma^2}} = 1{,}025\mathrm{K_j}, \\
p_j^{(2)} &= \mathrm{K_j e}^{\frac{(1{,}2+1)^2}{2\sigma^2}} \mathrm{e}^{\frac{(-0{,}9-1)^2}{2\sigma^2}} = 68{,}375\mathrm{K_j}, \\
p_j^{(3)} &= \mathrm{K_j e}^{\frac{(1{,}2+1)^2}{2\sigma^2}} \mathrm{e}^{\frac{(-0{,}9+1)^2}{2\sigma^2}} = 11{,}302\mathrm{K_j}, \\
&\Rightarrow \mathrm{K_j} = 0{,}0115, \\
&\Rightarrow p_j^{(0)} = 0{,}07,\ p_j^{(1)} = 0{,}01,\ p_j^{(2)} = 0{,}79,\ p_j^{(3)} = 0{,}13.
\end{aligned}$$

5.4 Kódování

Získání kódového slova je možné pomocí násobení informačního vektoru a generující matice kódu (vztah 3.4). V porovnání s dekódováním se jedná o méně komplikovaný proces. LDPC kódy jsou však zpravidla zadány pouze kontrolní maticí, generující matici neznáme a musíme ji určit.

5.4.1 Převod kontrolní matice na generující

Mezi generucící a kontrolní maticí v systematické podobě platí vztah:

$$\mathbb{G} = \left[\begin{array}{c|c} \mathbb{I} & -\mathbb{P}^{\mathrm{T}} \end{array} \right] \Longleftrightarrow \mathbb{H} = \left[\begin{array}{c|c} \mathbb{P} & \mathbb{I} \end{array} \right], \tag{5.40}$$

kde $\mathbb{I}$ označuje jednotkovou matici.

Problém získání generující matice LDPC kódu je tedy především problémem převodu kontrolní matice $\mathbb{H}$ do systematické podoby. K tomu využijeme Gaussovu eliminační metodu. Její součástí jsou operace:

1. Násobení řádku konstantou.
2. Přičtení řádku násobeného konstantou k jinému řádku matice.
3. Záměna dvou řádků.

Tyto operace obecně nestačí k převodu na požadovaný tvar a musíme využívat operaci další - záměny sloupců. Zatímco předchozí dvě nemají vliv na soustavu rovnic, kterou matice $\mathbb{H}$ reprezentuje, záměna sloupců již ano.

Po úpravě kontrolní matice do systematické podoby získáme *generující* matici podle vztahu 5.40. Označíme ji jako $\mathbb{G}_{\mathrm{syst}}$. Takto získaná generující matice však neodpovídá původní matici $\mathbb{H}$ a abychom získali matici $\mathbb{G}$, která přísluší původní matici $\mathbb{H}$, musíme provést stejné záměny sloupců $\mathbb{G}_{\mathrm{syst}}$ jako v eliminační metodě, ale v *opačném* pořadí.

Příklad: Následující příklad je ukázkou, jak ke kontrolní matice LDPC kódu získat matici generující. Pro názornost je zvoleno těleso mod 3.

$$\mathbb{H} = \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 & 1 & 0 \end{bmatrix} \underset{\text{zám. } \mathbf{h_7}, \mathbf{h_8}}{\sim} \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \sim$$

$$\sim \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \\ 1 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \underset{\text{zám. } \mathbf{h_6}, \mathbf{h_7}}{\sim} \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 2 & 1 & 0 \\ 1 & 0 & 0 & 0 & 2 & 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \sim$$

$$\sim \begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \underset{\text{zám. } \mathbf{h_2}, \mathbf{h_4}}{\sim} \begin{bmatrix} 1 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \sim$$

$$\sim \begin{bmatrix} 1 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 1 & 2 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \mathbb{H}_{\text{syst}}.$$

$$\mathbb{G}_{\text{syst}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 0 & 0 & 2 & 2 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 & 0 & 1 & 2 \end{bmatrix} \underset{\substack{\text{zám. } \mathbf{g_2}, \mathbf{g_4} \\ \text{zám. } \mathbf{g_6}, \mathbf{g_7}}}{\sim} \begin{bmatrix} 1 & 0 & 2 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 & 1 & 0 & 2 \end{bmatrix} \sim$$

$$\underset{\text{zám. } \mathbf{g_7}, \mathbf{g_8}}{\sim} \begin{bmatrix} 1 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 & 1 & 2 & 0 \end{bmatrix} = \mathbb{G}.$$

5.5 Komplexní řešení kodéru a dekodéru

Popsán byl výpočet generující matice, pomocí které se určí kódové slovo přenášené kanálem. Při přenosu vznikají chyby a tyto chyby se pokouší opravit dekodér. Jinak řečeno, pokouší se získat původní kódový vektor. Výstupem dekodéru musí však být informační vektor, takže je nutné odstranit paritní symboly a získat symboly informační ve správném pořadí, což se zajistí zohledněním záměn sloupců v eliminačního procesu $\mathbb{H}$ matice. Pokud provedeme záměny symbolů v přijatém kódovém vektoru ve *stejném*

pořadí jako při eliminaci, dostaneme vektor rozdělený na informační a paritní část. Paritní část se ořízne a zbyde požadované informační slovo.

Příklad: Předpokládejme stejný kód jako v předchozí ukázce a uvažujme informační vektor

$$\mathbf{m_{TX}} = [\ 1 \quad 1 \quad 2 \quad 0\ ],$$

výstupem kodéru potom je kódový vektor

$$\mathbf{c} = \mathbf{m_{TX}}\mathbb{G} = [\ 1 \quad 1 \quad 2 \quad 0\ ] \begin{bmatrix} 1 & 0 & 2 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 2 \\ 0 & 0 & 0 & 1 & 0 & 2 & 1 & 2 & 0 \end{bmatrix} =$$

$$= [\ 1 \quad 1 \quad 0 \quad 0 \quad 2 \quad 1 \quad 0 \quad 2 \quad 2\ ].$$

Kódový vektor je poslán informačním kanálem, kde může dojít k výskytu chyb (sečtení s chybovým vektorem). Úkolem dekodéru je tyto chyby eliminovat, čímž se získá původní kódový vektor, a převést zpět na informační vektor.

$$\mathbf{c} \sim \begin{vmatrix} \text{zám. } c_7, c_8 \\ \text{zám. } c_6, c_7 \\ \text{zám. } c_2, c_4 \end{vmatrix} = [\ 1 \quad 1 \quad 2 \quad 0 \quad 0 \quad 1 \quad 2 \quad 0 \quad 2\ ],$$

$$\Rightarrow \mathbf{m_{RX}} = [\ 1 \quad 1 \quad 2 \quad 0\ ].$$

5.6 Rozdělení LDPC kódů

LDPC kódy se dělí na *regulární* a *iregulární*. Regulární kód je takový, má-li jeho kontrolní matice konstantní počet jedniček ve všech řádcích a všech sloupcích (mají-li všechny řádkové i sloupcové vektory konstantní Hammingovu váhu). Iregulární je pak takový kód, který tuto podmínku nesplňuje.

5.7 Návrh LDPC kódů

5.7.1 Principy návrhu LDPC kódů

Návrh LDPC kódu obvykle spočívá v návrhu kontrolní matice určitým algoritmem. Generující matice se získá následně eliminační metodou.

Kritickým faktorem pro dekódování je výskyt smyček v Tannerově grafu, které mají výrazný vliv na konvergenci dekódovacího procesu. Lze říci, že čím kratší smyčky se v grafu vyskytují, tím horší jsou dekódovací vlastnosti. Omezení však platí i z druhé

strany. Eliminace smyček má za následek velmi řídkou kontrolní matici, čímž se opět snižují opravné schopnosti daného kódu. Úkolem návrhového algoritmu je tedy najít určitý kompromis.

Příklad: Výskyt smyčky délky 4 v Tannerově grafu odpovídá kontrolní matici:

$$\mathbb{H} = \begin{bmatrix} & \vdots & & \vdots & \\ \dots & 1 & \dots & 1 & \dots \\ & \vdots & & \vdots & \\ \dots & 1 & \dots & 1 & \dots \\ & \vdots & & \vdots & \end{bmatrix}.$$

Zajímavý a snadno implementovatelný algoritmus, založený na vygenerování základní matice a následné expanzi (náhrada jedniček náhodnou sloupcovou permutací jednotkové matice) na požadovanou velikost, je popsán v [10]. Jiné algoritmy využívají náhodné generování kontrolní matice a odstraňováním krátkých smyček, případně odstraňování uzlů s hodností 1. Dobrých výsledků dosahují tzv. PEG algoritmy (*Progressive Edge Growth*), kde se generují hrany pomocí stromové struktury [22]. Různé algoritmy jsou uvedeny například v [19], [24], [30], [62] a přehled vybraných kódů v [35].

5.7.2 Návrhový informační poměr

Návrhový informační poměr je dán vahami řádků a sloupců kontrolní matice. Platí rovnost:

$$R = 1 - \frac{w_c}{w_r}. \tag{5.41}$$

Příklad Kódu s 50% redundancí ($R = 0{,}5$) odpovídá váha sloupcového vektoru kontrolní matice $w_c = 3$ a váha řádkového vektoru kontrolní matice $w_r = 6$.

5.7.3 Gallagerův algoritmus

Gallager již na začátku 60. let popsal jednoduchý algoritmus pro návrh regulárních kódů [13], proto jsou tyto kódy někdy nazývány *Gallagerovy*. Písmenem k označuje váhu řádkového vektoru, písmenem j váhu sloupcového vektoru a n značí délku kódu.

Nejprve se vytvoří základní matice $\mathbb{A}_0$:

$$\mathbb{A}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & & & & & & & & & & & & \\ & & & & 1 & 1 & 1 & 1 & & & & & & & & \\ & & & & & & & & 1 & 1 & 1 & 1 & & & \\ & & & & & & & & & & & & 1 & 1 & 1 & 1 \end{bmatrix} = (n/k \times n)\,.$$

Kontrolní matice se složí z náhodných sloupcových permutací π_i matice $\mathbb{A}_0$:

$$\mathbb{H} = \begin{bmatrix} \pi_1(\mathbb{A}_0) \\ \pi_2(\mathbb{A}_0) \\ \vdots \\ \pi_j(\mathbb{A}_0) \end{bmatrix} = (nj/k \times n)\,. \tag{5.42}$$

6 Implementace softwarového návrhu LDPC kódů

Pro účely návrhu opravného kódu jsem vytvořil několik softwarových balíčků v jazyce Java. Ty jsou rozděleny podle funkčnosti na:

- mathematics - jedná se matematickou podporu pro výpočty s lineráními kódy,
- tanner_graph - obsahuje třídy pro práci s Tannerovým grafem,
- ldpc - implementuje třídy pro kódování, dekódování a návrh LDPC kódů,
- code_simulator - programová realizace simulátoru chybovosti LDPC kódů,
- minimum_distance - implementace paralelního výpočtu minimální vzdálenosti lineárních blokových kódů.

6.1 Balíček mathematics

6.1.1 Třída GFMath

Jedná se o podporu práce s Galoisovy tělesy. Obsahuje statické metody pro počítání v tělesech vyšších než GF(2), testování iredubicilních polynomů a generování vyhledávacích tabulek. Vybrané metody z této třídy:

- isIrreducible(polynomial : int) : boolean - testuje, zda je polynom iredubicilní v GF(2),
- moduloPol(pol : int, mod : int) : int - počítá zbytek po dělení polynomu polynomem,
- initPol(modPol : int) : void - vytvoří vyhledávací tabulky pro počítání v tělese, které je generováno rozšířením GF(2) polynomem z parametru,
- initModulo(p : int) : void - vygeneruje vyhledávací tabulky pro počítání v aritmetice mod p,
- sum(a : int, b : int) : int - vrátí součet dvou čísel ve vygenerovaném tělese dle vyhledávací tabulky,
- multiply(a : int, b : int) : int - vrátí součin dvou čísel ve vygenerovaném tělese dle vyhledávací tabulky.

Příklady použití: Následovat bude několik ukázek práce s touto třídou.

Zjištění, zda je zadaný polynom iredubucilní v GF(2), inicializace vyhledávacích tabulek rozšířením GF(2) tímto polynomem a operace nad vygenerovanými vyhledávacími tabulkami:

```
1 // zjisteni, zda je polynom iredubicilni
2 boolean b = GFMath.isIrreducible(11);
3 // naplneni LUT tabulek,
4 // rozsireni GF2 polynomem 11 = x^3 + x + 1
5 GFMath.initPol(11);
6 // vypocty pomoci LUT tabulek
7 int p = GFMath.multiply(GFMath.sum(4,6),5);
```

Inicializace tělesa modulo 3 a násobení v tomto tělese:

```
1 // naplneni LUT tabulek pro aritmetiku modulo 3
2 GFMath.initModulo(3);
3 // nasobeni pomocí LUT tabulky
4 int n = GFMath.multiply(2,2);
```

Operace bez vyhledávacích tabulek:

```
1 // vynasobeni binarnich polynomu
2 int p1 = multiplyPol(4,5)
3 // modulo polynom 3
4 int p2 = moduloPol(p1,3)
```

6.1.2 Třída MatrixMath

Obsahuje statické a často přetížené metody pro práci s vektory a maticemi za účelem dalšího použití v souvislosti s lineárními kódy. Vybrané metody z této třídy:

- isZeroVector(int[]) : boolean - testuje, zda se se jedná o nulový vektor,
- hammingDistance(byte[], byte[]) : int - počítá Hammingovu vzdálenost mezi dvěma binárními vektory,

- hammingWeight(byte[] vector) : int - počítá Hammingovu váhu binárního vektoru,
- randomBinVector(length : int) : byte[] - generuje náhodný binární vektor,
- decToBin(decVector : int[], binBlockLength : int) : byte[] - převádí vektor dekadických čísel na vektor binární,
- binToDec(binVector : byte[], binBlockLength : int) : int[] - převádí binární vektor na vektor dekadických čísel, v případě nesoudělnosti délky binárního bloku s vektorem vyhazuje výjimku WrongBlockLengthException
- swapColumns(matrix : int[][], index1 : int, index2 : int) : void - provede záměnu sloupců v matici,
- swapRows(matrix : int[][], index1 : int, index2 : int) : void - provede záměnu prvků ve vektoru,
- rearrangeVector(vector : int[], posVector : int[]) : int[] - uspořádá prvky v daném vektoru dle vektoru pozic,
- rearrangeColumns(matrix : int[][], posVector : int[]) : int[][] - uspořádá sloupce matice dle vektoru pozic,
- generateAscendingVector(dimension : int) : int[] - vygeneruje vzestupný vektor dané délky,
- randomRowsAndCols(matrix : int[][] ,colSwaps : int, rowSwaps : int) - náhodné záměny řádek a sloupců v matici,
- printMarix(matrix : int[][]) - vypíše matici na standardní výstup.

Příklady použití: Změna uspořádání sloupců v matici:

```
1 int[][] a = {{1,2,3},{4,5,6},{7,8,9}};
2  // preusporadani sloupcu matice
3 int[][] b = MatrixMath.rearrangeColumns(
4                     a, new int[]{1,2,0});
5 MatrixMath.printMatrix(b);
```

Výstup programu:

```
[2,3,1;
5,6,4;
8,9,7;]
```

Výpočet Hammingovy váhy a převod binárního vektoru na dekadický:

```
1 byte[] bin = {0,1,1,1,0,0};
2 int w = MatrixMath.hammingWeight(bin);
3 System.out.println("weight: " + w);
4 int[] dec = MatrixMath.binToDec(bin, 2);
5 System.out.println(Arrays.toString(dec));
```

Výstup programu:

```
weight: 3
[1, 3, 0]
```

6.1.3 Třída VectorGenerator

Slouží pro postupné generování posloupnosti vektorů zadané délky. Posloupnost má charakter čítače. Třída se před použitím instancuje.

Obsahuje tyto instanční metody:

- next() : boolean - vygenerování dalšího binárního vektoru; pokud byla dosažena maximální hodnota, vrací false,
- next(gfSize) : boolean - vygenerování dalšího binárního vektoru pro počítání v libovolném tělese,
- getLength() : int - vrací délku generovaného vektoru,
- reset() : void - vynulování čítače,
- getVector() : int[] - vrací vygenerovaný vektor.

Příklad použití: Vygenerování binárních a nebinárních vektorů dané délky:

```
1  int len = 2; int mod = 3;
2  VectorGenerator vg = new VectorGenerator(len);
3   // binarni vektory
4  do {
5          System.out.print(
6                  Arrays.toString(vg.getVector())+ "␣");
7  } while (vg.next());
8  System.out.println();
9  vg.reset();
10  // nebinarni vektory
11 do {
12         System.out.print(
13                 Arrays.toString(vg.getVector())+ "␣");
14 } while (vg.next(mod));
```

Výstup programu:

```
[0, 0] [0, 1] [1, 0] [1, 1]
[0, 0] [0, 1] [0, 2] [1, 0] [1, 1] [1, 2] [2, 0] [2, 1] [2, 2]
```

6.1.4 Třída CombinationGenerator

Třída pro generování kombinací o daném počtu prvků, kterou jsem následně využíval pro výpočet minimální vzdálenosti kódu. Vlastní generování realizuje zásobník předplněný rostoucí posloupností. Po naplnění zásobníku rostoucí posloupností se po krocích provádí inkrementace od vrchních hodnot, jak ukazuje algoritmus č. 1. Využití předprogramovaných knihovních tříd (Collections framework v Javě) pro realizaci zásobníku by vedlo ke značnému zpomalení generování. Zde plně postačuje pole celočíselných hodnot s celočíselným ukazatelem konce zásobníku.
Třída implementuje přetížený konstruktor:

- CombinationGenerator(setSize : int, kComb : int) - pro klasické generování všech kombinací,
- CombinationGenerator(setSize : int, kComb : int, start : int, limit : int) - pro limitaci prvního prvku do zadaného intervalu,

a metodu next(), která vrací pole následující kombinace. Skutečná implementace doplňuje algoritmus č. 1 o limitaci prvního prvku do intervalu. Tuto limitaci jsem využíval pro paralelizaci výpočtu minimální vzdálenosti.

Algoritmus 1 Generátor všech kombinací dle zadané třídy a počtu prvků.

```
// setSize - velikost cele mnoziny (n)
// kComb - trida kombinace (k)
// stack - pole predplnene hodnotami 0,1,2,..,k-1
// pos - ukazatel pozice zasobniku, pri inicializaci na k-1
public int[] next() {
    if ((kComb == 1) && (stack[0] >= setSize-1))     return null;
    while (true) {
        if (finished) {
            finished = false;
            return stack;
        }
        finished = true;
        i = stack[pos]; // pop
        pos--;
        if (i == setSize-1 ) {
            i = stack[pos];
            pos--;
        }
        i++; // inkrementace posledniho vybraneho prvku (123->124)
        if (pos == -1 && i == setSize-1)     break;
        pos++;
        stack[pos] = i; // vraceni zpet
        while (pos+1 != kComb) {
            i++;
            if (i > setSize-1) {
                finished = false; // je nutna dalsi iterace
                break;
            }
            pos++; // push
            stack[pos] = i;
        }
    }
    if (kComb == 1) {
        if (stack[0] < setSize) {
            stack[0]++;
            return stack;
        }
    }
    return null;
}
```

Příklad použití: Generování kombinací třetí třídy z pěti prvků a ukázka limitace prvního prvku do intervalu $< 1, 2)$

```
1  CombinationGenerator comb
2          = new CombinationGenerator(5,3);
3  int[] array;
4  while ( (array=comb.next()) != null)
5          System.out.print(Arrays.toString(array) + "; ");
6
7  System.out.println("\nlimited");
8  Combinations comb2
9          = new CombinationGenerator(5,3,1,2);
10 while ( (array=comb2.next()) != null)
11         System.out.print(Arrays.toString(array) + "; ");
```

Výstup programu:

```
[0, 1, 2]; [0, 1, 3]; [0, 1, 4]; [0, 2, 3]; [0, 2, 4];
[0, 3, 4]; [1, 2, 3]; [1, 2, 4]; [1, 3, 4]; [2, 3, 4];
limited
[1, 2, 3]; [1, 2, 4]; [1, 3, 4];
```

6.1.5 Třída LinearCodes

Třída se statickými metodami pro práci s generující a kontrolní maticí lineárních blokových kódů. Metody jsou často přetížené pro více datových typů.

Vybrané metody z této třídy:

- convertParityCheckToSystematic(hMatrixNS : int[][], posVector1 : int[], posVector2 : int[]) : int[][] - převede binární kontrolní matici do systematické podoby, před zavoláním je nutné vytvořit instance vektorů o délce kódového slova (počet sloupců kontrolní matice) a naplnit hodnotami $0, 1, \ldots, n-1$; metoda implementuje eliminační metodu popsanou v kapitole 5.4.1,
- convertParityCheckToSystematicGF(hMatrixNS : int[][], posVector1 : int[], posVector2 : int[]) : int[][] - zobecnění předchozí metody pro počítání v předgenerovaném konečném tělese, využívá třídu GFMath (kap. 6.1.1)
- parityCheckNSToGeneratorNS(hMatrixNonSyst : int[][]) : int[][] - převede libovolnou binární kontrolní matici na matici generující, generování vektoru pozic a záměny sloupců řeší interně,

- parityCheckSystToGeneratorSyst(hMatrixSyst : int[][]) : int[][] - převod binární kontrolní matice v systematickém tvaru na generující matici v systematickém tvaru; implementuje větu 5.40,
- parityCheckSystToGeneratorSystGF(int[][] hMatrixSyst) : int[][] - zobecnění předchozí metody pro libovolné konečné těleso,
- checkSyndrome(int[][] hMatrix, int[] codeVector) : boolean - provede násobení kódového vektoru s transponovanou kontrolní maticicí a vrací true, pokud produkt tohoto násobení je roven nulovému vektoru,
- encodeWord(int[][] genMatrix, int[] infoVector) : int[] - provede vynásobení binárního informačního vektoru s generující maticí,
- encodeWordGF(int[][] genMatrix, int[] infoVector) : int[] - zobecnění předchozí metody pro libovolné konečné těleso,
- encodeWord(int[][] genMatrixSyst, int[] infoVector, int[] positionVector) : int[] - vynásobení informačního vektoru s generující a záměny sloupců dle vektoru pozic.

Příklad použití: Ukázka převodu kontrolní kontrolní matice Hammingova kódu (zadaného maticí 3.7) na matici generující. Kontrolní matice je již v systematické podobě, takže není nutno provádět eliminační metodu.

```
1 int[][] hSyst = {{0,1,1,1,1,0,0},{1,0,1,1,0,1,0},
2                  {1,1,0,1,0,0,1}};
3 int[][] gSyst2 =
4         LinearCodes.parityCheckSystToGeneratorSyst(hSyst);
5 MatrixMath.printMatrix2(gSyst);
```

Výstup programu:

```
{{1,0,0,0,0,1,1},
{0,1,0,0,1,0,1},
{0,0,1,0,1,1,0},
{0,0,0,1,1,1,1},
}
```

Příklad použití: Převod obecné kontrolní matice na generující s využitím aritmetiky konečných těles a kódování informačního vektoru. Jedná se o implementaci příkladů z kapitol 5.4.1 a 5.5.

```
int[][] hMatrix = {{1,2,1,0,0,0,0,0,0},{0,0,0,2,2,2,0,0,0},
        {0,0,0,0,0,0,1,1,2},{1,0,0,0,2,0,0,0,2},
        {0,0,2,1,0,0,2,0,0},{0,2,0,0,0,2,0,1,0}};
int[] posVect1 = MatrixMath.generateAscendingVector(
                                    hMatrix[0].length);
int[] posVect2 = MatrixMath.generateAscendingVector(
                                    hMatrix[0].length);
System.out.println(Arrays.toString(posVect1));
GFMath.initModulo(3);
int[][] hSyst =
        LinearCodes.convertParityCheckToSystematicGF(
                        hMatrix, posVect1, posVect2);
System.out.println("Pos. vector for encoding = "
                        + Arrays.toString(posVect1));
System.out.println("Pos. vector for decoding = "
                        + Arrays.toString(posVect2));
System.out.println(Arrays.toString(posVect2));
System.out.println("Hyst =");
MatrixMath.printMatrix(hSyst);
int[][] genNS = MatrixMath.rearrangeColumns(
        LinearCodes.parityCheckSystToGeneratorSystGF(hSyst),
        posVect1);
System.out.println("Generator =");
MatrixMath.printMatrix(genNS);
int[] encoded = LinearCodes.encodeWordGF(
        genNS, new int[]{1,1,2,0});
System.out.println("Codeword = "
        + Arrays.toString(encoded));
```

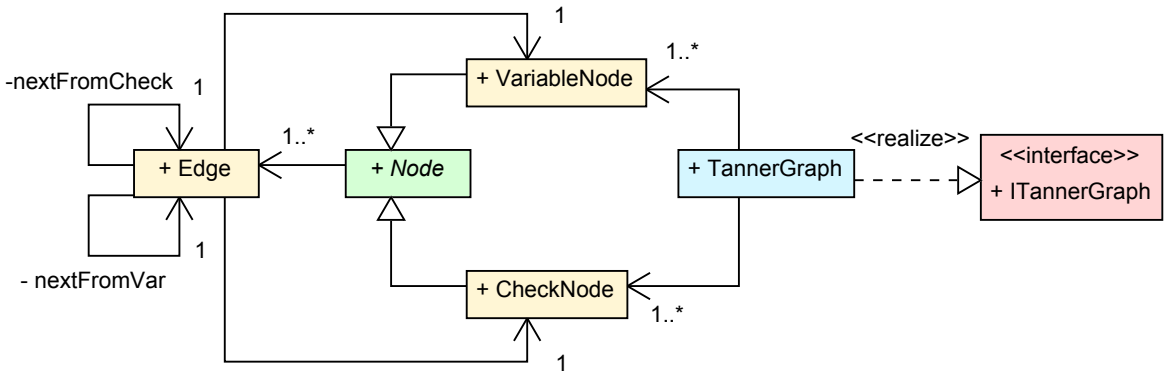
Výstup programu:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
Pos. vector for encoding = [0, 1, 4, 3, 2, 5, 7, 8, 6]
Pos. vector for decoding = [0, 1, 4, 3, 2, 5, 8, 6, 7]
Hyst =
[1,2,0,0,1,0,0,0,0;
0,0,1,1,0,1,0,0,0;
2,0,1,0,0,0,1,0,0;
2,1,0,2,0,0,0,1,0;
```

```
0,2,1,1,0,0,0,0,1;
]
Generator =
[1,0,2,0,0,0,1,0,1;
0,1,1,0,0,0,2,1,0;
0,0,0,0,1,2,0,2,2;
0,0,0,1,0,2,1,2,0;
]
Codeword = [1, 1, 0, 0, 2, 1, 0, 2, 2]
```

6.2 Balíček tanner_graph

Tento balíček obsahuje třídy pro práci s *Tannerovým grafem*. Každý uzel a každá hrana jsou reprezentovány instancemi příslušných objektů a graf je vytvořen pomocí referencí do obdoby vázaného seznamu. Práce s grafem, jako je přidávání nových hran, předávání zpráv při dekódování či vyhledávání cyklů, se děje na úrovni instančních metod příslušných objektů a bude dále předvedena.



Obr. 6.1: Zestručněný UML diagram tříd balíčku tanner_graph

6.2.1 Třída Node

Jedná se o abstraktní třídu. Zapouzdřuje referenci na objekt hrany a tyto privátní atributy:

- index : int - index uzlu,
- edgeCount - počet připojených hran,
- distance : int - vzdálenost od jiného uzlu; používá se při vyhledávání cyklů,
- cycleLength : int - délka nejmenšího cyklu, který uzlem prochází,

- state : enum {FRESH, OPEN, CLOSED} - nastavení stavu uzlu při průchodu grafu během hledání cyklů.

Zmiňované atributy jsou přístupné pomocí tzv. *gettrů* a *settrů*. Třída dále obsahuje dvě metody, které souvisí s objektem hrany, pro umožnění iterování hran z *variable* i *check* uzlu. Jedná se o tyto metody:

- addEdgeFromVar(newEdge : Edge) - přidá hranu do seznamu začínajícího ve *variable* uzlu,
- addEdgeFromCheck(newEdge : Edge) - přidá hranu do seznamu začínajícího v *check* uzlu.

6.2.2 Třída Edge

Reprezentuje jednu hranu v grafu. Obsahuje reference na instance dvou uzlů, reference da další hrany v seznamech pro iteraci z *variable* a *check* uzlů a atributy pro předávání zpráv po těchto hranách při dekódování. Instanční atributy této třídy:

- nextNodeFromVar : Edge, nextNodeFromCheck : Edge - reference pro zmiňované seznamy,
- valueQ0 : double, valueQ1 : double, valueR0 : double, valueR1 : double, valueQgf : double[], valuePgf : double[] - předávané hodnoty,
- coef : int - konstanta v kontrolní matici.

6.2.3 Třída CheckNode

Třída neobsahuje žádné instanční atributy. Slouží pouze pro odlišení množiny uzlů.

6.2.4 Třída VariableNode

Třída pro reprezentaci *variable* uzlu. Zapouzdřuje atributy pro zaznamenání inicializačních pravděpodobností. Jsou to:

- p0 : double, p1 : double, pGF : double[].

6.2.5 Třída TannerGraph

Jedná se o reprezentaci *Tannerova grafu*. Třída implementuje rozhraní ITannerGraph, které zpřístupňuje metody pro práci s grafem a zároveň umožňuje rozšiřitelnost nebo případné úpravy této třídy.

Příklad použití: Základní použití této třídy zřejmě nejlépe osvětlí praktická ukázka. Jedná se o ruční vytvoření Tannerova grafu ke kontrolní matici o rozměrech 2×4, kde jsou hrany určeny množinou $\{\{0,1\},\{1,2\}\}$. Ukázána je také změna konstanty v kontrolní matici.

```
1  ITannerGraph graph = new TannerGraph();
2  graph.setCheckNodes(new CheckNode[2]);
3  graph.setVariableNodes(new VariableNode[4]);
4  for (int i = 0; i < graph.getCheckNodes().length; i++) {
5          checkNodes[i] = new CheckNode();
6          checkNodes[i].setIndex(i);
7  }
8  for (int i = 0; i < getVariableNodes().length; i++) {
9          variableNodes[i] = new VariableNode();
10         variableNodes[i].setIndex(i);
11 }
12 Edge e1 = new Edge(checkNodes[0], variableNodes[1]);
13 checkNodes[0].addEdgeFromCheck(e1);
14 variableNodes[1].addEdgeFromVar(e1);
15 e1.setCoef(2); // ukazka zmeny nasobiciho koeficientu
16 Edge e2 = new Edge(checkNodes[1], variableNodes[2]);
17 checkNodes[1].addEdgeFromCheck(e2);
18 variableNodes[2].addEdgeFromVar(e2);
19  // vypis kontrolni matice
20 MatrixMath.printMatrix2(graph.toHMatrixInt());
```

Výstup programu:

```
{{0,2,0,0},
{0,0,1,0},
}
```

Třída dále implementuje metody pro práci s grafem během dekódování.

Inicializaci přijatého vektoru zajišťují metody:

- setInitEstimatesSPA(vector : int[], prob0 : double, prob1 : double) : void,
- setInitEstimatesSPA(vector : byte[], prob0 : double, prob1 : double) : void,
- setInitEstimatesLog(vector : int[], prob0 : double, prob1 : double) : void,
- setInitEstimatesLog(vector : byte[], prob0 : double, prob1 : double) : void,

- setInitEstimatesGF(vector : int[], gfSize : int, prob : double[]) : void,
- setInitEstimatesAWGN(vector : double[], N0 : double) :void,
- setInitEstimatesAWGNGF(binVector : double[], N0 : double, gfSize : int, binBlockLength : int) : void,
- setInitEstimatesAWGNLog(vector : double[], N0 : double) : void.

Proměnná prob0 má význam podmíněné pravděpodobnosti příjmu 0, pokud byla vyslána 0. Obdobně proměnná prob1 znamená podmíněnou pravděpodobnost příjmu 1, pokud byla vyslána 1. Pracujeme-li se symetrickým kanálem, jsou tyto hodnoty stejné.

Používaný parametr gfSize označuje počet prvků abecedy při dekódování nebinárních kódů. Je nutný pro instancování polí inicializační pravděpodobnosti a předávaných hodnot. K tomu je použit tzv. *lazy loading*[3] při volání metod pro nastavení těchto hodnot.

Odeslání inicializačních pravděpodobností jako zpráv q_{ij} se realizujuje metodami:

- firstMessagesSPA() : void,
- firstMessagesLog() : void,
- firstMessagesSPAGF(int gfSize) : void.

Metody pro výpočet hodnot q_{ij} a r_{ij} včetně jejich předání na hrany:

- iterateToCheckLog() : void,
- iterateToCheckSPA() : void,
- iterateToCheckSPAGF(gfSize : int) : void,
- iterateToVariablesLog(minsum : boolean) : void,
- iterateToVariablesSPAGF(gfSize : int) : void,
- iterateToVariablesSPA() : void.

Metody pro realizaci aktuálního odhadu:

- decisionLogInt() : int[],
- decisionLogByte() : byte[],
- decisionSPAInt() : int[],

[3]Jedná se návrhový vzor, kde objekty jsou vytvářeny až v momentě, kdy je potřeba s nimi pracovat.

- desicionSPAByte() : byte[],
- decisionSPAGF(gfSize : int) : int[].

Metody pro ověření, zda vektor patří do množiny kódových slov:

- checkSyndrome(decodedVector : int[]) : booelan,
- checkSyndromeGF(decodedVector : int[]) : boolean.

Syndromem se rozumí produkt násobení s transponovanou kontrolní maticí (viz 3.5). Násobení se neprovádí na úrovni vektoru a matice (n^2 operací), ale procházením seznamu hran připojených do *check* uzlu. Každá hrana představuje nenulový prvek v kontrolní matici a vzhledem k tomu, že v případě LDPC kódů je tato matice velmi řídká, je takto realizované násobení mnohem rychlejší než klasický součin vektoru a matice. Zrychlení výrazně závisí na hustotě a velikosti matice. U krátkých kódů ($n \approx 100$) se dle měření doby výpočtu jedná o zrychlení zhruba desetinásobné, u delších kódů ($n \approx 1000$) může být až tisícinásobné.

Příklad použití: Ukázka dekódování přijatého binárního vektoru *sum-product* algoritmem.

```
1  int[][][] hMatrix = ...
2  ITannerGraph tg = TannerGraph.createTannerGraph(hMatrix);
3  int[] receivedVector = ... // prijate kodove slovo
4  double rel = 0.9;
5  int iterations = 30;
6  tg.setInitEstimatesSPA(receivedVector, rel, rel);
7  tg.firstMessagesSPA();
8  tg.iterateToVariablesSPA();
9  for (int i = 0; i < iterations; i++) {
10         tg.iterateToCheckSPA();
11         tg.iterateToVariablesSPA();
12         if (tg.checkSyndrome(decodedVector)) {
13                 break;
14         }
15 }
```

Dosud bylo popsáno dekódování pomocí třídy TannerGraph. Tato třída implementuje i další metody pro práci s LDPC kódy. Jedná se především o vyhledávání cyklů (smyček) v grafu.

Vyhledávání cyklů jsem řešil pomocí průchodu grafu do šířky, známého jako *BFS* (*Breadth-First Search*)[4]. Algoritmus využívá značkování uzlů do tří stavů podle často používané terminologie (FRESH, OPEN, CLOSED), aby mohl být vlastní průchod realizován. Princip tohoto algoritmu si lze představit jako prohledávání stromu po patrech od počátečního (kořenového) uzlu, přičemž postupné procházení po patrech zajišťuje paměť typu FIFO. Do fronty jsou ukládány a z fronty jsou vybírány uzly při běhu algoritmu. Dosud neprohledaný uzel v daném čase je označen jako FRESH, uzel čekající ve frontě je ve stavu OPEN a uzlu, který byl již nalezen, se nastaví značka CLOSED. Při běhu pak každý aktuálně vybraný uzel z fronty prohledá všechny své sousedy se stavem FRESH, nastaví jim značku OPEN, uloží je do fronty a sebe přepne do stavu CLOSED. Prázdná fronta indikuje průchod souvislé části daného grafu.

Algoritmus jsem doplnil o počítání vzdálenosti, zastavení při dosažení parametrizovatelné maximální vzdálenosti, zastavení při nalezení požadovaného uzlu a zákaz procházení přes zadanou hranu pro budoucí vyhledávání cyklů. Bylo také nutno vyřešit bipartitní povahu Tannerova grafu, kde se rozlišují dvě množiny uzlů. Zápis jako metodu v jazyce Java ukazuje algoritmus č. 2. Metoda vrací dosaženou vzdálenost při zastavení průchodu.

Zmiňovaný algoritmus již lze využít pro vyhledávání smyček (cyklů), které jsem řešil v jiné veřejné metodě. Uzel, kde hledáme cykly, postupně prochází všechny své sousedy a volá BFS prohledávání s vyloučením hrany k sousednímu uzlu. Pokud prohledávání narazí na tento sousední uzel, znamená to nalezení cyklu a prohledávání je ukončeno. Vyjádření pomocí metody ukazuje algoritmus č. 3. Metoda vrací nejkratší délku cyklu, který prochází zadaným uzlem.

Veřejné metody zpřístupněné přes rozhraní pro průchod grafu a hledání smyček:

- bfsTraverse(nodeStart : Node, toCheck : boolean, maxDist : int) : void - prohledá graf a označí vzdálenosti uzlů; nejprve nastaví všechny uzly v grafu na maximální vzdálenost a poté volá privátní metodu z algoritmu č. 2, přičemž jako koncový uzel a zakázaná hrana je předána hodnota null,
- bfsSearchCycles(nodeStart : Node, toCheck : boolean) : int - implementace algoritmu č. 3; hledá nejkratší smyčku procházející uzlem,
- minimumCycleLengthVarNodes() : int - provede hledání cyklů z množiny všech *variable* uzlů a vrací délku nejkratšího cyklu,

[4]Druhou možností prohledávání grafu je algoritmus *DFS* (*Depth-First Search*). Jeho využití pro hledání nejkratších cyklů je však velmi problematické

Algoritmus 2 Průchod Tannerova grafu do šířky.

```
private int bfsTraverse2(Node nodeStart, Edge closed,
 Node endNode, boolean toCheck, int maxDist) {
   freshNodes(); // stavy vsech uzlu na FRESH
   Queue<Node> q = new LinkedList<Node>();
   nodeStart.setDistance(0);
   q.add(nodeStart);
   boolean check = toCheck;
   while (!q.isEmpty()) { // vlastni BFS pruchod
      Node node = q.poll();
      if (maxDist != 0 && node.getDistance() > maxDist)
         return node.getDistance();
      if (node.getDistance() % 2 == 0) check = toCheck;
      else  check = !toCheck;
      if (node == endNode) return node.getDistance() + 1;
      Edge edge = node.getEdges();
      while (edge == closed) {
         if (edge == null) break;
         // dalsi hrana z vazaneho seznamu
         // prom.check je true pro iteraci z variable uzlu
         edge = nextEdge(edge, check);
         if (edge == null) break;
      }
      while (edge != null) {
         // protejsi uzel
         Node neighb = nextNode(edge, check);
         if (neighb.getState() == State.FRESH) {
            neighb.setState(State.OPEN);
            neighb.setDistance(node.getDistance() + 1);
            q.add(neighb);
         }
         while (true) {
            edge = nextEdge(edge, check);
            if (edge == null) break;
            if (edge == closed)  continue;
            break;
         }
      }
      node.setState(State.CLOSED);
   }
   return 0;
}
```

Algoritmus 3 Hledání nejkratší smyčky, která prochází daným uzlem.

```
public int bfsSearchCycles(Node nodeStart, boolean toCheck) {
        Edge e = nodeStart.getEdges();
        int min = Integer.MAX_VALUE;
        while (e != null) {
                Node neighb = nextNode(e, toCheck);
                int i = bfsTraverse2(nodeStart, e,
                        neighb, toCheck, Integer.MAX_VALUE);
                if (i < min && i != 0) {
                        min = i;
                }
                nodeStart.setCycleLength(min);
                e = nextEdge(e, toCheck);
        }
        return min;
}
```

- minimumCycleLengthCheckNodes() : int - provede hledání cyklů z množiny všech *check* uzlů a vrací délku nejkratšího cyklu,
- noDegreOne() : boolean - vrací true, pokud hodnost každého z uzlů je větší nebo rovna dvěma.

Metody pro tisk a ladění, případně jako pomocné pro návrh kódů:

- printEdgesFromCheck(indexCheck : int) : void - vypíše všechny hrany (indexy koncových uzlů) připojené do *check* uzlu,
- printEdgesFromVariable(indexVar : int) : void - vypíše všechny hrany (indexy koncových uzlů) připojené do *variable* uzlu,
- printEdges() : void - vypíše všechny hrany grafu jako množinu uzlových indexů dle definice 5.2,
- printCycles() : void - vypíše nejkratší délky cyklů postupně pro všechny uzly,
- printDegrees() : void - vypíše hodnosti všech uzlů (počty hran do nich připojených).

Metody pro vygenerování kontrolní matice z existující instance grafu:

- toHMatrixInt() : int[][] - převede graf na kontrolní matici jako pole proměnných typu int,

- toHMatrixByte() : byte[][][] - převede graf na kontrolní matici jako pole proměnných typu byte,

Metoda pro změnu do požadovaného konečného tělesa:

- changeGF(gfSize : int) : void - náhodně změní násobící koeficienty dle dimenze požadovaného tělesa; všechny koeficienty zůstanou nenulové.

6.3 Balíček ldpc

6.3.1 Třída LDPCGenerator

Třída obsahuje implementaci generátoru LDPC kódů. Generátorů LDPC kódů jsem postupně vytvořil několik, většina však nevedla k úspěchu v podobě dobrých výsledků při simulaci chybovosti. Po několika neúspěšných pokusech s náhodným rozmisťováním jedniček do kontrolní matice, což téměř vždy vedlo na vytvoření cyklů délky 4, jsem se rozhodl graf generovat přidáváním objektů hran a záměrným vytvářením dlouhých cyklů. Zkoušel jsem vygenerování cyklů konstantní délky do podoby řetízku a doplnění hran na požadovanou Hammingovu váhu řádků a sloupců kontrolní matice (hodnosti uzlů), ale ani tento způsob nevedl k algoritmu, který by byl použitelný pro různou délku kódu.

Úspěch přinesl až algoritmus s vytvářením velmi dlouhých cyklů a postupným snižováním právě generované délky. Zpočátku samozřejmě nelze cykly vytvářet kvůli nedostatku hran. Algoritmus pracuje s hodnotou minimální vzdálenosti dvou uzlů, mezi kterými bude vytvářena hrana. Je-li tato vzdálenost větší nebo rovna minimální délce cyklu a hodnosti uzlů jsou menší než požadované, je možné přidat novou hranu do grafu. Vzdálenost je nastavena na relativně vysokou hodnotu sudého čísla a postupně snižována až do zadané minimální délky cyklu. Nejprve se tedy jedná o prosté přidávání hran a k vytváření smyček algoritmus plynule přechází. Poměrně zajímavých výsledků jsem dosáhl náhodným promícháváním řádků a sloupců kontrolní matice a zašuměním návrhových vah řádků a sloupců kontrolní matice s určitým rozptylem. Počátek hrany je střídavě umisťován do množiny *variable* a do množiny *check* uzlů. Po dokončení generování jsou volitelně zkontrolovány hodnosti uzlů. Vedou-li do uzlu méně než dvě hrany, nemůže během iteračního procesu dekódování předávat novou informaci. Pokud je takový uzel nalezen, je algoritmem hledán nejvzdálenější uzel od tohoto a je mezi nimi přidána nová hrana. Toto může mít za následek nižší délky cyklů v grafu, než je požadovaná hodnota. Je-li však tato hodnota délky minimálních cyklů dostatečně nízká, k dodatečnému přidávání hran vůbec nedojde. Protože váhy řádkových a sloupcových vektorů nejsou konstantní, ale jsou generovány s určitým rozptylem, výstupem jsou kódy *iregulární*.

Generování ukazuje algoritmus č. 4, odstranění výskytu uzlů s hodností jedna algoritmus č. 5. Oba tvoří dohromady metodu:

- generateLDPC(checkNodes : int, varNodes : int, startingCycleLen : int, endCycleLen : int, randomSwaps : boolean, designWr : int, designWc : int, sigmaWr : double, sigmaWc : double, gfSize : int) : TannerGraph.

Parametry mají význam:

- checkNodes a varNodes - určují počty uzlů (rozměry kontrolní matice),
- startingCycleLen a endCycleLen - interval délky generovaných cyklů,
- randomSwaps - při hodnotě true budou promíchávány sloupce a řádky kontrolní matice,
- designWr, designWc - návrhová váha řádkových a sloupcových vektorů kontrolní matice (hodností uzlů),
- sigmaWr, sigmaWc - určují rozptyl návrhových vah,
- gfSize - počet prvků abecedy, pro binární aritmetiku nastavit na hodnotu 2.

Algoritmy, které předcházely tomuto, jsem vzhledem výsledkům simulací do finální podoby práce nezařadil.

Příklad použití: Vygenerování LDPC kódu, jehož kontrolní matice bude mít rozměry 256×512, minimální délku cyklů 10 a informační poměr 0,5. Zadaného informačnímu poměru odpovídají váhy $w_r = 6$, $w_c = 3$ (viz vztah 5.41).

```
1 TannerGraph tg = LDPCGenerator.generateLDPC(
2         256, 512, 50, 10, true, 6,3, 1.0, 1.0, 2, true);
3 MatrixMath.printMatrix(tg.toHMatrixInt());
```

6.3.2 Třída GallagersParityCheckGenerator

Třída implementuje Gallagerův algoritmus (kap. 5.7.3) pro generování *regulárních* kódů podle vztahu 5.42. Třídu je nutno instancovat, přičemž konstruktor obsahuje tři po sobě jdoucí parametry: n, w_c, w_r. Váha řádkových vektorů w_r musí být soudělná s délkou kódu n.

Veřejné instanční metody této třídy:

- next() : void - generace dalšího náhodného kódu,

Algoritmus 4 Vlastní algoritmus generování LDPC kódů.

```
public static TannerGraph generateLDPC(
   int checkNodes, int varNodes, int startingCycleLen, int endCycleLen,
   boolean randomSwaps, int designWr, int designWc,
   double sigmaWr, double sigmaWc, int gfSize, boolean noDegOne) {

   Random rand = new Random();    TannerGraph tg = new TannerGraph();
   tg.createNodes(checkNodes,varNodes);
   boolean varcheck = false;
   for (int cycle = startingCycleLen; cycle >= endCycleLen; cycle-=2) {
      int wc = designWc + (int)Math.round(rand.nextGaussian()*sigmaWc);
      int wr = designWr + (int)Math.round(rand.nextGaussian()*sigmaWr);
      if (wc < 2) wc = 2;
      if (wr < 2) wr = 2;
      boolean finished = false;
      while (!finished) {
         if (randomSwaps) tg = TannerGraph.randomTanner(tg, 50000, 5000);
         finished = true;  varcheck = !varcheck;
         if (varcheck) {
            for (int i = 0; i < tg.getVariableNodes().length; i++) {
               VariableNode vn = tg.getVariableNodes()[i];
               if (vn.getEdgeCount() < wc) {
                  tg.bfsTraverse(vn, true, cycle);
                  for (CheckNode cn : tg.getCheckNodes()) {
                     if (cn.getEdgeCount() >= designWr)  continue;
                     if (cn.getDistance() >= cycle - 1) {
                        finished = false; Edge e = new Edge(cn,vn);
                        cn.addEdgeFromCheck(e); vn.addEdgeFromVar(e);
                        e.setCoef(rand.nextInt(gfSize-1)+1);   break;
            } } } }
         } else {
            for (int i = 0; i < tg.getCheckNodes().length; i++) {
               CheckNode cn = tg.getCheckNodes()[i];
               if (cn.getEdgeCount() < wr) {
                  tg.bfsTraverse(cn, false, cycle);
                  for (VariableNode vn : tg.getVariableNodes()) {
                     if (vn.getEdgeCount() >= designWc)  continue;
                     if (vn.getDistance() >= cycle - 1) {
                        finished = false; Edge e = new Edge(cn,vn);
                        cn.addEdgeFromCheck(e); vn.addEdgeFromVar(e);
                        e.setCoef(rand.nextInt(gfSize-1)+1);   break;
         } } } } }
      } // konec cyklu while
   } // konci for, design hotov
if (!noDegOne) return tg;
// metoda pokracuje doplnenim hran pro odstraneni hodnosti mensi nez 2
```

Algoritmus 5 Odstranění výskytu uzlů s hodností 2 v Tannerově grafu.

```
// dokonceni metody pro generovani LDPC kodu
   for (int i = 0; i < tg.getVariableNodes().length; i++) {
      VariableNode vn = tg.getVariableNodes()[i];
      if (vn.getEdgeCount() < 2) {
         tg.bfsTraverse(vn, true, startingCycleLen);
         int max = Integer.MIN_VALUE;
         CheckNode nodemax = null;
         for (CheckNode cn : tg.getCheckNodes()) {
            if (cn.getDistance() > max) {
               nodemax = cn;
               max = cn.getDistance();
            }
         }
         System.out.println("deg. 1, var node");
         Edge e = new Edge(nodemax,vn);
         e.setCoef(rand.nextInt(gfSize-1) + 1);
         nodemax.addEdgeFromCheck(e);
         vn.addEdgeFromVar(e);
      }
   }
   for (CheckNode cn : tg.getCheckNodes()) {
      if (cn.getEdgeCount() < 2) {
         tg.bfsTraverse(cn, true, startingCycleLen);
         int max = Integer.MIN_VALUE;
         VariableNode nodemax = null;
         for (VariableNode vn : tg.getVariableNodes()) {
            if (vn.getDistance() > max) {
               nodemax = vn;
               max = vn.getDistance();
            }
         }
         Edge e = new Edge(cn, nodemax);
         e.setCoef(rand.nextInt(gfSize-1) + 1);
         nodemax.addEdgeFromVar(e);
         cn.addEdgeFromCheck(e);
      }
   }
// navratova hodnota metody, ktera zacina jiz v predchozim algoritmu
return tg;
```

- getHMatrix() : int[][] - vrací vygenerovanou matici.

Velkou nevýhodou tohoto algoritmu je, že obvykle vede na výskyt smyček délky 4. Při hledání dobrého kódu je tedy potřeba prozkoumat velkou množinu vygenerovaných kontrolních matic.

Příklad použití: Ukázka generování regulárního kódu.

```
1 GallagersParityCheckGenerator galag
2         = new GallagersParityCheckGenerator(240,3,6);
3 // prvni kod, vzdy stejny
4 MatrixMath.printMatrix(galag.getHMatrix());
5 galag.next();
6 // dalsi po vytvoreni nahodnych sloupcových permutaci
7 MatrixMath.printMatrix(galag.getHMatrix());
```

6.3.3 Třída LDPCEncoderDecoder

Komplexní implementace kodéru a dekodéru binárních LDPC kódů. Konstruktor obsahuje referenci na objekt implementující rozhraní ITannerGraph. Tato třída pak implementuje rozhraní ILDPCEncoderDecoder.

Veřejné instanční metody, jejichž význam je zřejmý z názvu:

- encodeWord(infoVector : byte[]) : byte[],
- decode(iterations : int, vector : byte[], prob0 : double, prob1 : double) : byte[],
- decode(iterations : int, vector : int[], prob0 : double, prob1 : double) : int[],
- decodeLog(iterations : int, vector : byte[], prob0 : double, prob1 : double, minsum : boolean) : byte[],
- decodeLog(iterations : int, vector : int[], prob0 : double, prob1 : double, minsum : boolean) : int[],
- decodeAwgnInt(iterations : int, vector : double[], N0 : double) : int[],
- decodeAwgnByte(iterations : int, vector : double[], N0 : double) : byte[] ,
- decodeAwgnLogByte(iterations : int, vector : double[], N0 : double, minsum : boolean) : byte[],

- decodeAwgnLogInt(iterations : int, vector : double[], N0 : double, minsum : boolean) : int[],
- getInfoBits() : int, getCodeBits() : int,
- getGenMatrix() : byte[][], getParityCheckMatrix() : byte[][],
- getPosVector1() : int[], getPosVector2() : int[].

Příklad použití: Následující kód je ukázkou použití popisované třídy.

```
1 ITannerGraph tg = TannerGraph.createTannerGraph(hMatrix);
2 ILDPCEncoderDecoder ed = new LDPCEncoderDecoder(tg);
3 int[] codeword = ed.encodeWord(...);
4 codeword = ... // zasumeni
5 int iterations = 10;
6 int[] decoded = ed.decode(iterations, codeword,
7         new int[]{0.9,0.9});
```

6.3.4 Třída LDPCEncoderDecoderGF

Jedná se o obdobu předchozí třídy se zobecněním pro libovolné konečné těleso. Před využíváním je nutná generace vyhledávacích tabulek třídou GFMath.

Veřejné metody přístupné přes rozhraní ILDPCEncoderDecoderGF:

- encodeWord(infoVector : int[]) : int[],
- decode(iterations : int, vector : int[], prob : double[]) : int[],
- decodeAwgn(iterations : int, vector : double[], N0 : double) : int[],
- getInfoSymbols() : int, getCodeSymbols() : int,
- getGenMatrix() : int[][], getParityCheckMatrix() : int[][],
- getPosVector1() : int[], getPosVector2() : int[].

Příklad použití: Ukázka kódování a dekódování. Kontrolní matice shodná s maticí použitou v kapitole 5.4.1.

```
1  int[][][] hMatrix = {
2          {1,2,1,0,0,0,0,0,0}, {0,0,0,2,2,2,0,0,0},
3          {0,0,0,0,0,0,1,1,2}, {1,0,0,0,2,0,0,0,2},
4          {0,0,2,1,0,0,2,0,0}, {0,2,0,0,0,2,0,1,0}};
5   // inicializace na aritmetiku mod 3,
6   // pro telesa vznikla rozsirenim iredubic.
7   // polynomem pouzit metodu initPol(..)
8  GFMath.initModulo(3);
9  ITannerGraph tg = TannerGraph.createTannerGraph(hMatrix);
10 ILDPCEncoderDecoderGF ed = new LDPCEncoderDecoderGF(tg);
11 System.out.println("Generator matrix: ");
12 MatrixMath.printMatrix(ed.getGenMatrix());
13 System.out.println("Info symbols: " + ed.getInfoSymbols());
14 System.out.println("Codeword length: "
15         + ed.getCodeSymbols());
16 int[] codeword = ed.encodeWord(new int[]{1,1,2,0});
17 System.out.println("Codeword: "
18         + Arrays.toString(codeword));
19  // vytvoreni chyb v kodovem slove
20 codeword[0]+=2; codeword[0]%=3;
21 codeword[2]+=2; codeword[2]%=3;
22 System.out.println("With errors: "
23         + Arrays.toString(codeword));
24 double[] prob = {0.8,0.8,0.8};
25 int[] decoded =  ed.decode(10, codeword, prob);
26 System.out.println("Decoded infoword: "
27         + Arrays.toString(decoded));
```

Výstup programu:

```
Generator matrix:
[1,0,2,0,0,0,1,0,1;
0,1,1,0,0,0,2,1,0;
0,0,0,0,1,2,0,2,2;
0,0,0,1,0,2,1,2,0;
]
Info symbols: 4
Codeword length: 9
Codeword: [1, 1, 0, 0, 2, 1, 0, 2, 2]
```

```
With errors: [0, 1, 2, 0, 2, 1, 0, 2, 2]
Decoded infoword: [1, 1, 2, 0]
```

6.4 Balíček code_simulator

6.4.1 Třída BERSimulator

Třída implementuje rozhraní IBERSimulator a před použitím je nutno ji instancovat. Simulaci je možné provádět těmito metodami:

- berTest(algorithm : Algorithm, start : double, step : double, berStop : double, tg : ITannerGraph, maximumBlockError : int, iterations : int, ctrMin : long) : ArrayList<Double>,
- berTest(algorithm : Algorithm, start : double, step : double, berStop : double, tg : ITannerGraph, maximumBlockError : int, iterations : int) : ArrayList<Double>,
- berTestGF(start : double, step : double, berStop : double, tg : ITannerGraph, maximumBlockError : int, iterations : int, ctrMin : long) : ArrayList<Double>,
- berTestGF(start : double, step : double, berStop : double, tg : ITannerGraph, maximumBlockError : int, iterations : int) : ArrayList<Double>,
- getBerResults() : double[],
- getEbN0Values() : double[].

Význam použitých parametrů je následující:

- algorithm - použití algoritmus, enum {SP, LOG, MINSUM},
- start - počáteční hodnota E_b/N_0; zadáno v [dB],
- step - krok poměru E_b/N_0, po kterém se prování simulace chybovosti; zadáno v [dB],
- berStop - požadovaná hodnota chybovosti, kde se simulace zastaví,
- tg - reference na vytvořený objekt třídy TannerGraph, viz kap. 6.2.5,
- maximumBlockErrors - počet chybných kódových slov, které se nepodaří dekódovat a ze kterých se počítá chybovost; čím vyšší hodnota, tím přesnější simulace,
- iterations - počet dekódovacích iterací,

- ctrMin - při použití metod s tímto parametrem je chybovost počítána, pokud počet pokusů o dekódování dosáhne hodnoty ctrMin, čili dříve, než počet chybových slov dosáhne hodnoty maximumBlockErrors; lze použít pro urychlení simulace.

Příklad Generování LDPC kódu, jeho simulace, výpis nejkratší délky cyklů, které prochází uzly, a výpis hodností uzlů.

```
1  ITannerGraph tg = LDPCGenerator.generateLDPC(
2          96*2, 128*2, 50, 10, true, 4, 3, 2.0, 2.0, 2, true)
3  double step = 0.3;       double berStop = 1e-4;
4  int maximumBlockErrors = 100;    int iterations = 30;
5  IBERSimulator simulator = new BERSimulator();
6  simulator.berTest(
7          Algorithm.SP, step, berStop, tg,
8          maximumBlockErrors, iterations);
9  System.out.println("dB");
10 System.out.println(
11         Arrays.toString(simulator.getEbN0Values()));
12 System.out.println("BER");
13 System.out.println(
14         Arrays.toString(simulator.getBerResults()));
15  // nejkratsi cykly check a variable uzly
16 tg.printCycles();
17  // hodnosti check a variable uzlu
18 tg.printDegrees();
```

Jeden z možných výstupů programu (zkrácen):

```
dB
[0.0, 0.3, 0.6, 0.8999999999999999, 1.2, 1.5, 1.8, 2.1, 2.4,
2.6999999999999997, 2.9999999999999996, 3.2999999999999994,
3.599999999999999, 3.899999999999999]
BER
[0.1723090277777778, 0.1500984251968504, 0.13058035714285715,
0.10208333333333333, 0.07854540358744394, 0.05248015873015873,
0.03118070953436807, 0.016347543352601156, 0.010154540481400437,
0.005706142970927917, 0.0020325203252032522, 6.443795958245488E-4,
2.462898524863193E-4, 6.203549618124573E-5]
```

```
cycles
8 8 8 10 8 10 10 8 8 10 8 8 8 8 10 10 8 10 10 8 8 8 8 10 10 8 10
10 10 8 10 8 10 10 10 ...
10 10 10 10 8 8 10 10 10 8 10 10 10 8 10 10 8 8 10 10 8 8 8 10
10 10 10 10 8 10 8 10 ...
degrees
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 4 4 4 4 4 4 4 4 4 5 4
4 4 4 4 4 4 4 4 4 4 4 4 ...
3 3 3 3 3 4 3 3 3 2 2 4 3 3 2 3 3 3 3 3 2 4 3 3 3 2 3 4 3 5 3 3 4
4 3 3 3 4 3 4 3 3 3 3 4 ...
```

6.5 Balíček minimum_distance

Minimální vzdálenost je jedním ze základních parametrů, který charakterizuje opravný kód. Je žádoucí, aby byla, pokud možno, co největší. U lineárních blokových kódů můžeme její výpočet provádět buď hledáním kódového slova s nejmenší Hammingovou váhou (kap. 4.1.4), nebo hledáním nejmenšího počtu lineárně závislých sloupcových vektorů v kontrolní matici (kap. 4.1.5). Oba způsoby jsou však velmi časově náročné. Pro nelineární kódy navíc tato tvrzení neplatí a její výpočet je ještě komplikovanější.

Pro realizaci hledání kódového slova s nejmenší váhou bylo nutné vyřešit postupné generování všech vektorů informačních slov (třída VectorGenerator, kap. 6.1.3). Každé informačních slovo je za pomoci generující matice $\mathbb{G}$ kódováno a z těchto kódových slov počítána Hammingova váha. Výběrem minima z těchto vah (nebo jeho průběžným ukládáním) je získána minimální vzdálenost.

Uvažujme binární kód. Má-li informační vektor k bitů a kódové slovo n bitů, je potřeba přesně $(2^k - 1)kn$ násobení, $(2^k - 1)(k - 1)n$ operací exkluzivních součtů (pro kódování) a $(2^k - 1)$ krát vypočítat Hammingovu váhu. Tyto operace je nutné provést vždy, doba výpočtu tedy nezávisí na d_{min}.

Naopak, hledáme-li nejmenší počet lineárně závislých sloupců kontrolní matice, začínáme na dvou sloupcích a postupně zvyšujeme třídu kombinací, dokud není nalezena lineárně závislá kombinace. Počet prvků této kombinace (třída kombinace) se rovná hodnotě d_{min}. Čím je tato hodnota vyšší, tím déle trvá výpočet.

Příklad: Uvažujme binární matici $\mathbb{H}$ o 15 řádcích a 20 sloupcích. Kódové slovo má tedy 20 bitů. Při výpočtu minimální vzdálenosti takového kódu je nutné provést následující počet (označen jako m) exkluzivních součtů sloupcových vektorů:

$$\binom{20}{2} + \binom{20}{3} + \ldots + \binom{20}{d_{min} - 1} < m \leq \binom{20}{2} + \binom{20}{3} + \ldots + \binom{20}{d_{min}}.$$

Pro minimální vzdálenost kódu rovnou 4 je počet těchto operací větší než 1330, pro $d_{min} = 6$ je to 21679 a pro $d_{min} = 8$ již 137959. Tento fakt dostatečně ilustruje obrovskou výpočetní náročnost celého procesu, zejména pro dlouhé kódy. Abychom získali skutečný počet provedených exkluzivních součtů mezi bity, je nutné uvažovat velikost sloupcového vektoru (což je v tomto případě číslo 15) a počet prvků právě počítané kombinace. Přesto je tento způsob mnohem rychlejší než první zmiňovaný a je jádrem naprogramovaného algoritmu.

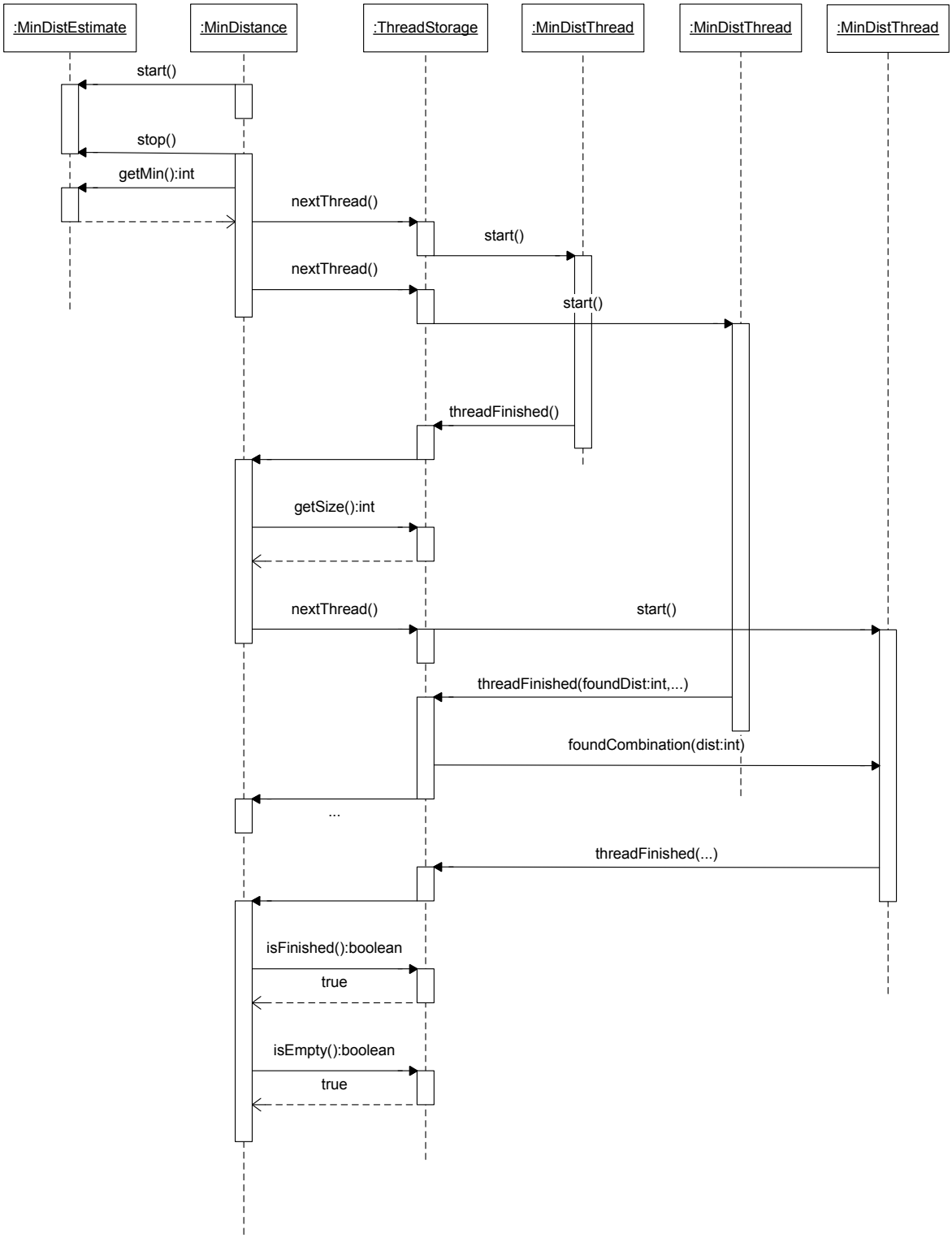
Výpočet minimální vzdálenosti jsem realizoval s maximálním využitím paralelního běhu vláken. Paralelizace probíhá na úrovni generátoru kombinací (viz třída CombinationGenerator, kap. 6.1.4) pomocí limitace prvního prvku. Na každou hodnotu tohoto prvku připadá jedno běžící vlákno, přičemž maximální počet běžících vláken je nastavitelný. Nalezne-li jedno vlákno lineárně závislou sloupcovou kombinaci, jsou ukončena všechna vlákna hledající kombinaci stejné a vyšší třídy a je zakázáno spouštění vláken nových. Přesná hodnota minimální vzdálenosti je stanovena až po skončení běhu všech vláken.

Pro zrychlení jsem využil způsob s hledáním minimální váhy kódového slova. Před spouštěním paralelních vláken je provedeno po určitou časovou dobu (např. v jednotkách sekund) právě hledání minimální váhy kódového slova. Nalezená hodnota je větší nebo rovna skutečné minimální vzdálenosti kódu. Tímto je zajištěno, že nebudou spouštěna vlákna pro třídu kombinací větší nebo rovnou této hodnotě, a navíc tato hodnota poskytuje orientační údaj, pokud se skutečná minimální vzdálenost nepodaří kvůli výpočetní náročnosti stanovit. Odhad lze využít i nezávisle na paralelním výpočtu.

Balíček obsahuje třídy:

- MinDistance - hlavní vlákno, které spouští vlákna pro paralelizovaný výpočet voláním metod z třídy ThreadStorage,
- MinDistEstimate - realizace odhadu minimální vzdálenosti (viz úvod této kapitoly),
- MinDistThread - jedno běžící vlákno výpočtu,
- ThreadStorage - obsahuje uložiště běžících vláken a slouží jako prostředník pro komunikaci mezi třídami MinDistance a MinDistThread.

Třídy MinDistThread a ThreadStorage slouží pouze pro interní komunikaci pomocí synchronizovaných metod. Pro praktické nasazení nejsou příliš podstatné a nebudou dále popisovány.



Obr. 6.2: Sekvenční diagram paralelního výpočtu minimální vzdálenosti

6.5.1 Třída MinDistEstimate

Třída provádí kódování posloupnosti všech informačních slov a implementuje větu z kap. 4.1.4. Výpočet probíhá pouze zadanou dobu a může sloužit jako odhad minimální vzdálenosti. Skutečná minimální vzdálenost je menší nebo rovna tomuto odhadu.

Kontrolní matice se předává přes konstruktor. Instanční metody pro praktickou realizaci výpočtu jsou:

- estimate(milies : long) : int - počítá odhad po zadanou dobu v milisekundách a tento odhad je návratovou hodnotou metody,
- getMin() : int - vrací poslední hodnotu počítaného odhadu.

Příklad použití: Odhad minimální vzdálenosti kódu. Výpočet bude běžet deset sekund.

```
1 ITannerGraph tg = ...
2 long milies = 10000;
3 MinDistEstimate mde
4         = new MinDistEstimate(tg.toHMatrixInt());
5 int dmin = mde.estimate(milies);
6 System.out.println("estimate of minimum distance: " + dmin)
```

6.5.2 Třída MinimumDistance

Hlavní vlákno. Instanční metody této třídy:

- setEstimateMilies(estimateMilies : long) - nastavení času pro prvotní realizaci odhadu,
- setBreakValue(breakValue : int) - slouží pro omezení dlouhých výpočtů, při této hodnotě minimální vzdálenosti je výpočet zastaven,
- computeMinDistance(withEstimate : boolean, threads : int) : void - výpočet minimální vzdálenosti, výsledek vypsán na standardní výstup.

Příklad použití Výpočet minimální vzdálenosti s pětisekundovým odhadem a paralelním výpočtem do 1000 vláken.

```
1  ITannerGraph tg = ...
2  int threads = 1000;
3  MinDistance md = new MinDistance(tg.toHMatrixInt());
4   // maximalni hodnota pocitane minimalni vzdalenosti
5   // pri dosazeni teho hodnoty se vypocet zastavi
6  md.setBreakValue(16);
7   // doba odhadu
8  md.setEstimateMilies(5000);
9   // vypocet dmin, vysledek vypsan do standardniho vystupu
10 md.computeMinDistance(true,threads);
```

Jeden z možných výstupů programu (zkrácen):

```
start computing estimate..
minimum: 24
minimum: 16
minimum: 8
..time expired
estimating finished
new thread for dist/start: 2/0
new thread for dist/start: 2/1
new thread for dist/start: 2/2
...
found dist 6
...
running threads: 1
exitting
running threads: 0
break
minimum distance is 6
time: 150115 ms
```

6.5.3 Třída MinDistanceStatic

Třída se dvěma statickými metodami pro výpočet minimální vzdálenosti (bez paralelizace). Jsou to:

- minimumDistanceParityCheck(hMatrix : int[][]) : int - výpočet z kontrolní matice,
- minimumDistanceHammingWeight(hMatrix : int[][]) : int - výpočet pomocí nejmenší Hammingovy váhy kódového slova.

7 Implementace LDPC kódů v hardwaru a podpora měření chybovosti

7.1 Programová realizace kodéru a dekodéru

Pro nasazení LDPC kódu v jednočipovém mikropočítači jsem napsal kodér a dekodér v jazyku C. Kódování jsem realizoval násobením informačního vektoru s generující maticí, dekodér pomocí vytvoření reprezentace Tannerova grafu. Realizace Tannerova grafu je obdobná popsané hierarchii v kap. 6.2, pouze přizbůsobená použitému jazyku a implementaci. Objekty jsou v tomto případě nahrazeny strukturami. Uzly a hrany reprezentují struktury node, varnode, checknode a edge. Při inicializaci se vytvoří seznamy hran a nastaví příslušné ukazatele podle Tannerova grafu, který je zadán komprimovaným tvarem kontrolní matice (množina uzlových indexů). Pro dosažení maximální rychlosti jsem implementovat *min-sum* algoritmus a výpočty realizoval *fixed-point* aritmetikou. Logaritmování, které se provádí při stanovení inicializačních hodnot při příjmu z kanálu (viz kap. 5.3.2), jsem řešil vyhledávací tabulkou. Nutné bylo také vyřešit převod vektorů do binární podoby a zpět.

Implementace kodéru a dekodéru zahrnuje tyto soubory:

- code.h - definuje konstanty pro počet hran a rozměry matic jako direktivy preprocesoru,
- convertor.h, convertor.c - funkce pro převod informačních a kódových vektorů do binární podoby a naopak,
- decoder_ldpc.h, decoder_ldpc.c - implementace LDPC dekodéru min-sum algoritmem (viz kap. 5.3.2),
- encoder_ldpc.h, encoder_ldpc.c - implementace lineárního kodéru násobením informačního vektoru s generující maticí,
- gen_matrix.h, gen_matrix.c - zadání generující matice jako dvou rozměrného pole,
- h_matrix.h, h_matrix.c - zadání kontrolní matice jako seznamu hran a vektoru pozic pro dekódování (více o vektoru pozic v kap. 5.5 a 6.1.5),
- types.h - definuje datové typy (int8, uint8, atd.).

Z hlediska praktického nasazení pro kódování a dekódování vektoru po bajtech jsou významné tyto funkce:

- encodeByteVector(infoword : uint8[]) : uint8* - kódování informačního vektoru, který je zadán jako pole bajtů,

- initDecoder(void) : void - inicializace dekodéru (vytvoření Tannerova grafu),
- decodeByteVector(prob : uint8, byteVector[] : uint8, iterations : uint8) : int8 - dekódování vektoru po bajtech, vrací 1 při úspěchu,
- generateInfoMessage(void): void - vygeneruje informační vektor v bitech podle vektoru pozic a převede ho na pole bajtů,
- getInfoMessageBytes(void) : uint8* - vrací ukazatel na pole bajtů dekódovaného informačního vektoru.

Ukázka praktického nasazení: Popsán bude příklad vygenerování LDPC kódu s využitím naprogramovaných knihoven z kap. 6 a jeho nasazení pro využití v jednočipovém mikropočítači.

Vygenerování kódu:

```
ITannerGraph tg =
 LDPCGenerator.generateLDPC(96, 128, 50, 8,
        true, 4, 3, 2.0, 2.0, 2, true);
ILDPCEncoderDecoder enc = new LDPCEncoderDecoder(tg);
tg.printEdges(); // vypis seznamu hran
System.out.println("pos vector:");
System.out.println(Arrays.toString(enc.getPosVector2()));
System.out.println("info bits: "+ enc.getInfoBits());
System.out.println("code bits: "+ enc.getCodeBits());
 // generujici matice do souboru
MatrixParser.saveToFile("e:\\g.txt", enc.getGenMatrix());
```

Jeden z možných výstupů programu (zkrácen):

```
{0,87},{0,40},{0,30},{0,20},
{1,108},{1,99},...
edges: 398
pos. vector: [0, 1, 2, 3, ...
info bits: 32
code bits: 128
```

Seznam hran a vektor pozic ze standardního výstupu zkopírujeme do inicializovaných polí hMatrix a posVector v souboru h_matrix.c. V hlavičkovém souboru code.h přepíšeme hodnoty konstant EDGES, VAR_NODES, CHECK_NODES, CODE_BYTES, INFO_BYTES a INFO_BITS. V tomto případě platí, že EDGES

= 398, VAR_NODES = 128, CHECK_NODES = 96, CODE_BYTES = 16, INFO_BYTES = 4, INFO_BITS = 32. Počet kódových bitů se vždy rovná počtu *check* uzlů, ale počet informačních bitů je dán rozměrem generující matice a proto je nutné ho uvést zvlášť.

Dále přepíšeme generující matici v souboru gen_matrix.c. Tato obecně není řídká jako matice kontrolní a zadává v podobě klasického dvourozměrného pole. Obě pole (generující matice i seznam hran kontrolní matice) jsou zadány s klíčovým slovem `const` za účelem jejich umístění do FLASH paměti (pokud je linker takto nastaven).

Kódování informačního vektoru:

```
1 #include "ldpc/encoder_ldpc.h"
2 ...
3 uint8 infoword[6] = {0xff,0x81,0x01,0xcd,0xfe,0xab};
4 txData = encodeByteVector(infoword);
```

Dekódování vektoru a generování informačního vektoru:

```
1 #include "ldpc/decoder_ldpc.h"
2 ...
3 initDecoder(); // volat pouze jednou
4 ...
5 uint8 success = decodeByteVector(90, rxData, 15);
6 generateInfoMessage();
7 infoBytes = getInfoMessageBytes();
```

7.2 Použitý hardware

Pro účely implementace kódu v hardwaru jsem použil zapůjčený vývojový kit CC2250DK od firmy Texas Instruments. Součástí tohoto kitu jsou víceúčelové desky SmartRF®05EB, moduly CC2520EM, antény, procesorová deska CCMSP-EM430F2618 a programátor MSP-FET430UIF. V této práci jsem využíval dva moduly CC2520EM (jeden jako vysílač, druhý jako přijímač) osazené anténami, dvě procesorové desky a samozřejmě programátor. Více o tomto kitu v [75].

Zmiňované moduly jsou osazeny obvodem CC2520. Tento obvod pracuje jako RF transceiver v nelicencovaném ISM (industrial, scientific and medical) pásmu 2,4GHz a primárně slouží pro ZigBee a IEEE 802.15.4 aplikace. Více v [73].

Jádrem procesorové desky je mikrokontrolér MSP430F1618 [76],[77],[78], který s RF

transceiverem komunikuje po SPI sběrnici. Firma Texas Instruments nabízí pro práci s transeiverem volně ke stažení knihovny Basic RF API a HAL RF API [74]. Mikrokontrolér MSP430F2618 disponuje pamětí 116KB FLASH a 8KB RAM.

7.3 Přenos přijatých dat do PC

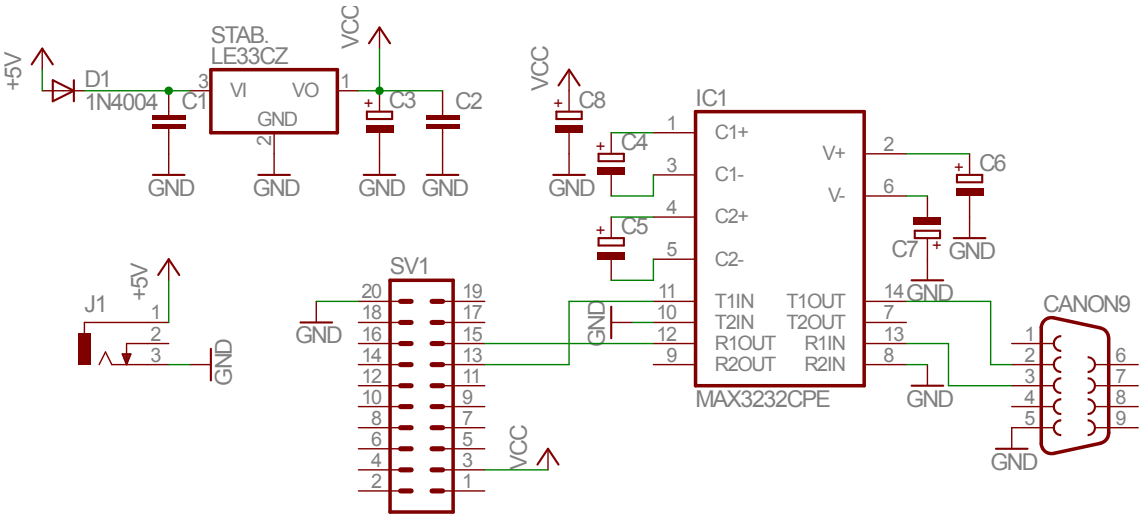
Pro měření chybovosti bylo nutné řešit vyhodnocení přijatých paketů před dekódováním a po dekódování. Po zvážení okolností (především výpočetní náročnost dekodéru) jsem se rozhodl veškeré vyhodnocování chybovosti provádět v PC. Mikrokontrolér odešle, zda se syndrom rovná nulovému vektoru (viz věta 3.5), a informační vektor před a po dekódování do PC, kde se provede vlastní vyhodnocení chybovosti a zobrazení v grafickém rozhraní.

Komunikaci je pouze jednosměrná a provádí se po sběrnici RS232. Jelikož mikrokontrolér pracuje s logikou 3V CMOS, bylo nutné řešit převod logických úrovní. Pro tento účel jsem navrhl zapojení s obvodem MAX232CPE [71] a stabilizátorem LE33CZ [72]. Schéma je uvedeno na obr. 7.1, deska plošného spoje na obr. 7.2 a osazovací výkres na obr. 7.3. Hodnoty kondenzátorů jsou $C_1 = 0{,}1\,\mu F$, $C_2 = 2{,}2\,\mu F$, $C_3 = 0{,}1\,\mu F$ a $C_{4..7} = 0{,}1\,\mu F$.

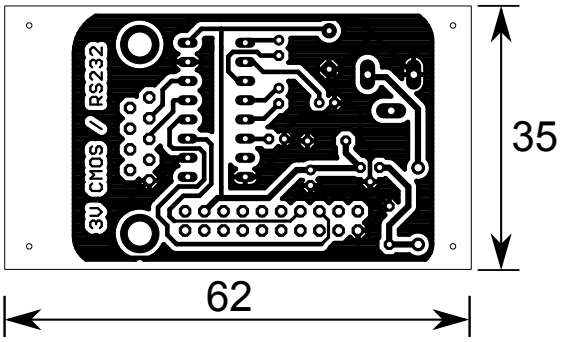
Pro přenos dat z procesoru se využívá rozhraní USCI (Universal Serial Communication Interface), které je nastavené do UART (Universal Asynchronous Receiver / Transmitter) režimu. Hodinový signál zprostředkovává interní časovač SMCLK.

7.4 Program pro měření chybovosti

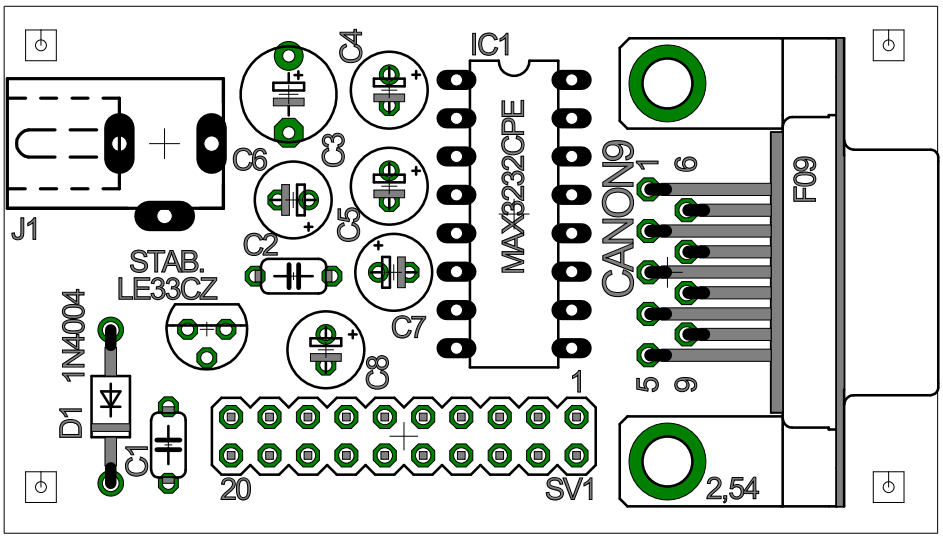
Pro vyhodnocení přijatých dat jsem vytvořil aplikaci v jazyce Java. Pro komunikaci se sériovým portem jsem využíval knihovny RXTX [25] a pro napsání grafického rozhraní knihovny SWT [70]. Plnění streamu hodnot a vyhodnocení se provádějí v samostatných vláknech. Informace o změně hodnot je posílána grafickému rozhraní pod vzoru *observer-observable* a následně je asynchronním voláním provedena změna zobrazovaných hodnot. Protože jsou data posílána periodicky s konstantním časovým odstupem, je možné detekovat ztrátu synchronizace. To zajišťuje další vlákno, které se uspává na dobu delší než je perioda vysílání paketů.



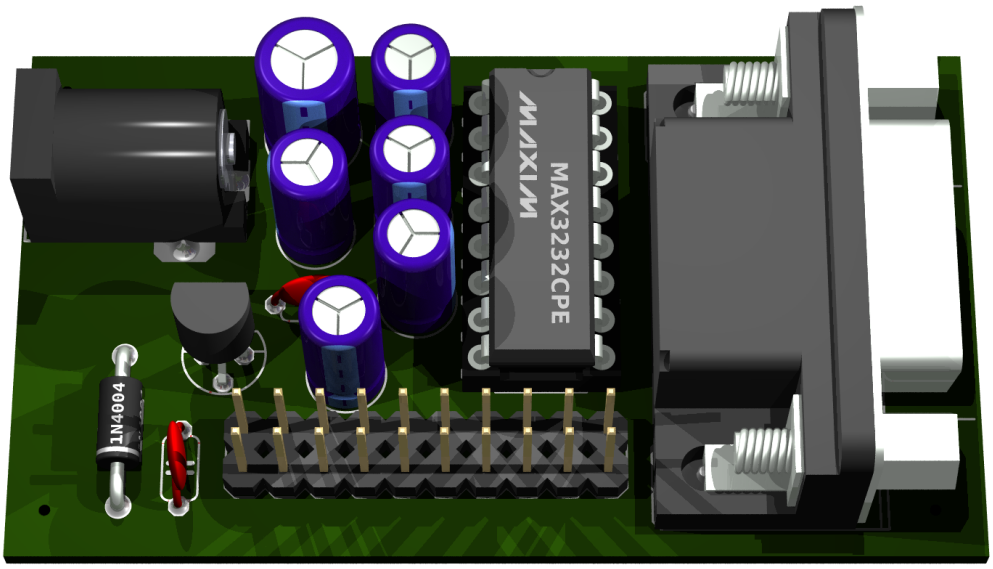
Obr. 7.1: Schéma zapojení převodníku úrovní 3V/RS232



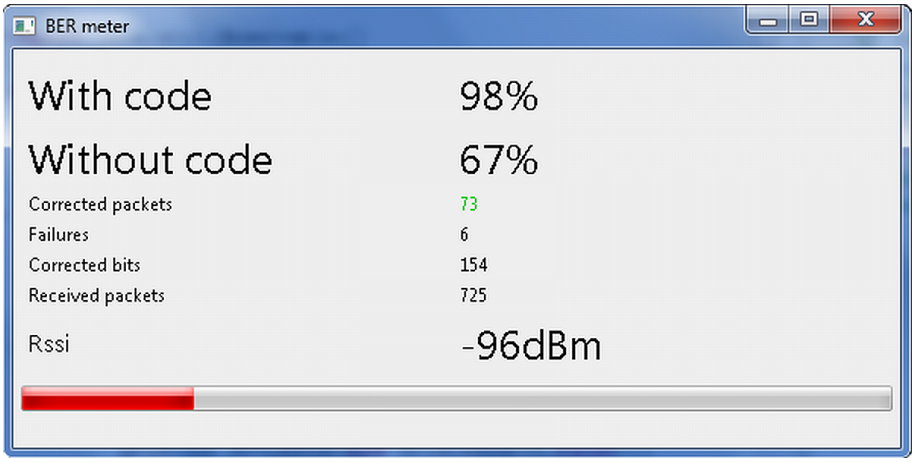
Obr. 7.2: Deska plošného spoje pro převodník úrovní



Obr. 7.3: Osazovací výkres převodníku úrovní



Obr. 7.4: Render osazené desky

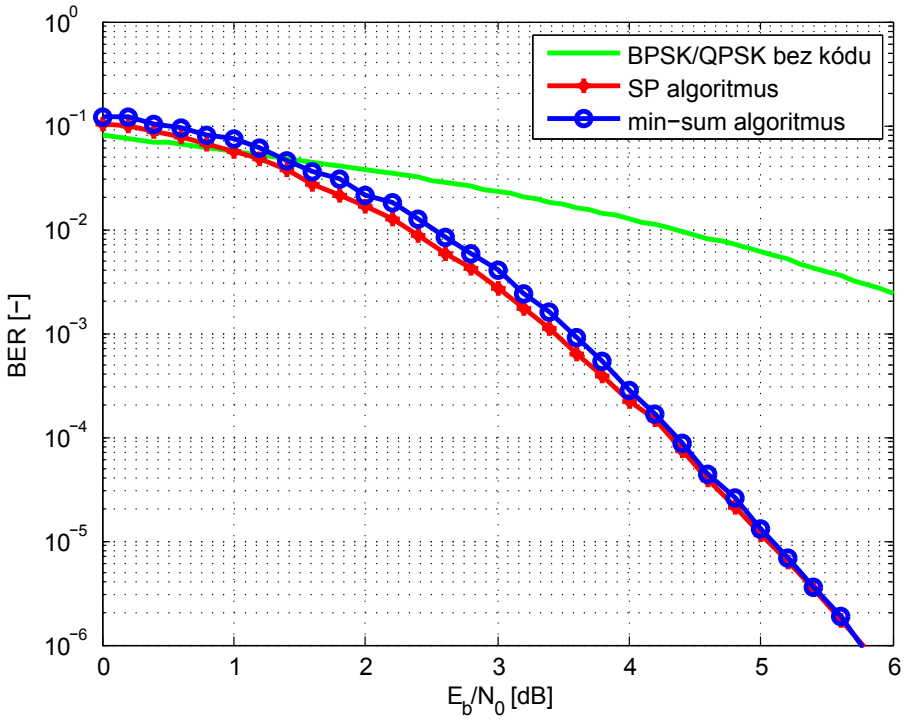


Obr. 7.5: Program pro měření chybovosti

8 Měření dosahu a spolehlivosti přenosu

8.1 Použitý kód

Volil jsem *iregulární* binární kód s 50% redundancí a s využitím naprogramovaných balíčků z (kap. 6) jsem provedl návrh. Váhu řádkových vektorů kontrolní matice (hodnosti *check* uzlů) jsem nastavil na hodnotu 6, váhu sloupcových vektoru (hodnosti *variable* uzlů) na hodnotu 3 a rozptyl těchto hodnot na hodnotu 1. Způsob nastavení těchto parametrů je popsán v kapitole 6.3.1. Na gridové síti MetaCentra jsem navrhl a simuloval vždy 100 různých kódů pro konstantní návrhové parametry a vybíral takový, který dosáhl hodnoty $BER = 10^{-4}$ při nejmenším odstupu E_b/N_0. Jako velmi výrazná limitace pro výběr kódu se projevila velikost RAM paměti procesoru. Nejdelším kódem, který se mi podařilo implementovat s ohledem na velikost dostupné paměti, je kód LDPC (96,48) - 96 bitů kódových a 48 informačních. Tzn. 6 zabezpečených bajtů. Výsledky simulace tohoto kódu ukazuje obr. 8.1.



Obr. 8.1: Simulace kódu LDPC (96,48) použitého při měření

8.2 Měření v EMC komoře

Měření chybovosti (spolehlivosti) jsme prováděli ve stíněné EMC komoře, která je součástí školních laboratoří. Původním předpokladem bylo, že bychom kalibrovanou anténou měřili intenzitu pole v místě příjmu. To se dostupnou technikou bohužel nepodařilo. Důvodem byl velmi malý vyzařovaný výkon a impulsní charakter vysílání. Čip CC2520 však umožňuje orientační přijatého měření přijatého signálu - hodnoty RSSI, kterou jsem pomocí mikrokontroléru posílal sériovou komunikací do PC a která byla zobrazována v grafickém rozhraní. I s touto hodnotou je výsledek měření poměrně zajímavý.

Vzhledem k rozměrům komory jsme vysílací modul odstínili krabicí z Fe plechu tloušťky 1 mm o rozměrech 20×14×10 cm. Měření probíhalo při vysílacím výkonu −4 dBm, což je minimální nastavitelná hodnota.

Výsledky měření uvedeny v tab. 8.1 a na obr. 8.2 a 8.3.

Tab. 8.1: Měření v EMC komoře, spolehlivost s kódem a bez kódu

vzdálenost. [m]	RSSI [dBm]	spolehlivost s kódem [%]	spolehlivost bez kódu [%]	synchr.
7,78	-	0	0	-
7,77	-105	51	13	*výpadky*
7,76	-104	81	25	*výpadky*
7,75	-103	96	64	
7,74	-102	100	94	
7,73	-101,5	100	99,5	
7,72	-101	100	100	

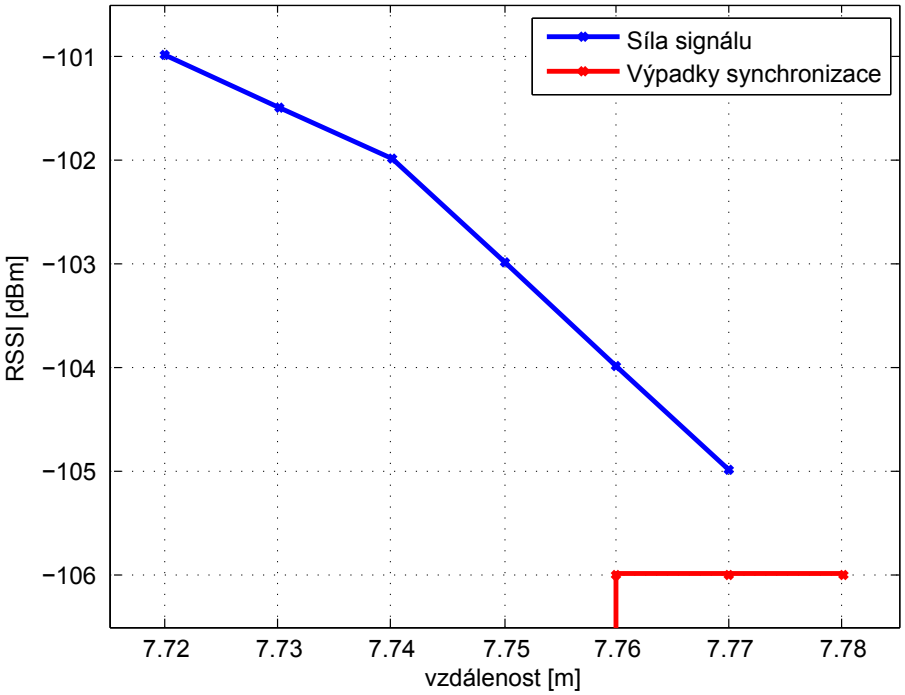
8.3 Měření ve volném prostoru

Další měření, zaměřené na dosah, proběhlo v místě bez zarušení signálem WIFI. Nejsou však vyloučeny odrazy od překážek. Vysílací výkon je stejný s předchozím případem, (−4 dBm). Výsledky uvedeny v tab. 8.2 a v grafech na obr. 8.8 a 8.9.

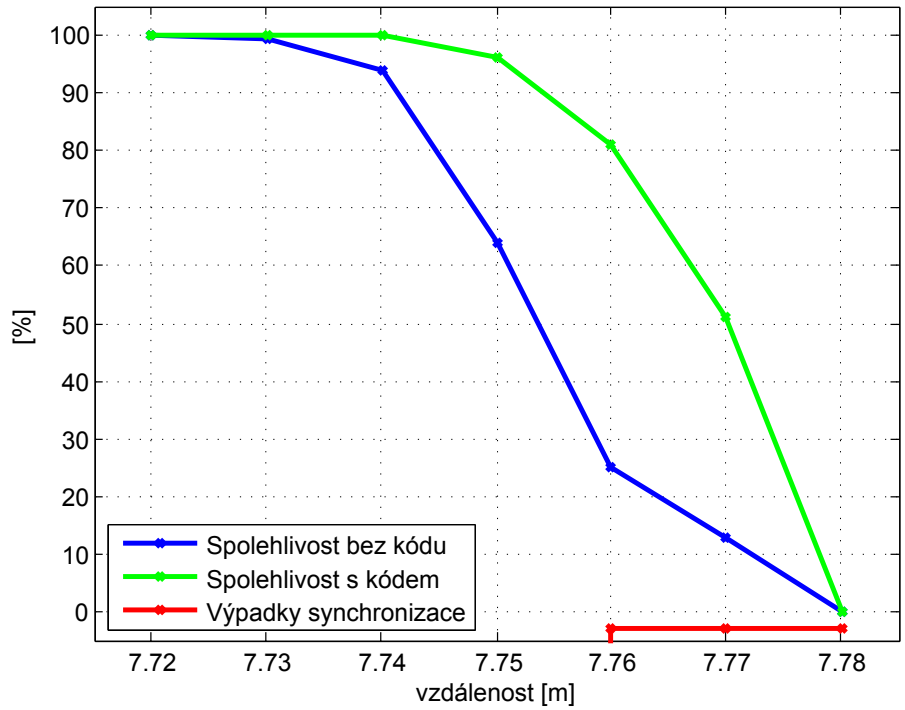
Pro názornější interpretaci výsledků je ukázána spolehlivost bez kódu a s kódem po filtraci klouzavým průměrem (MA, *Moving Average*). Impulsní odezva použitého filtru je:

$$\mathbf{h} = \frac{1}{10}\left[1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1\right].$$

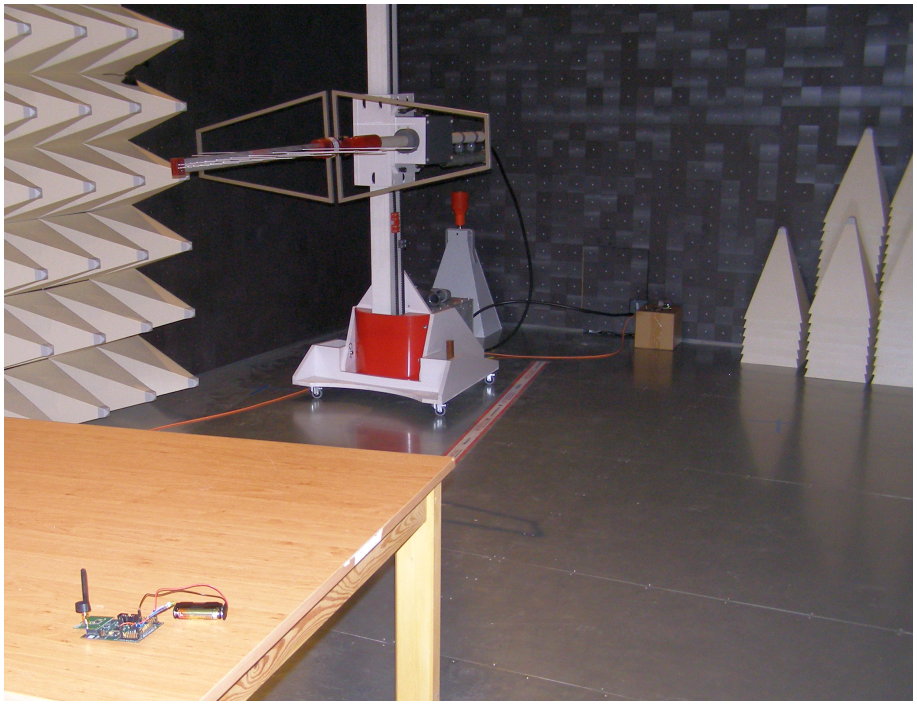
Graf na obr. 8.10 ukazuje filtrované výsledky s uvažováním ztrát synchronizace jako 0% spolehlivosti, graf obr. 8.11 ukazuje spolehlivosti bez uvažování těchto ztrát.



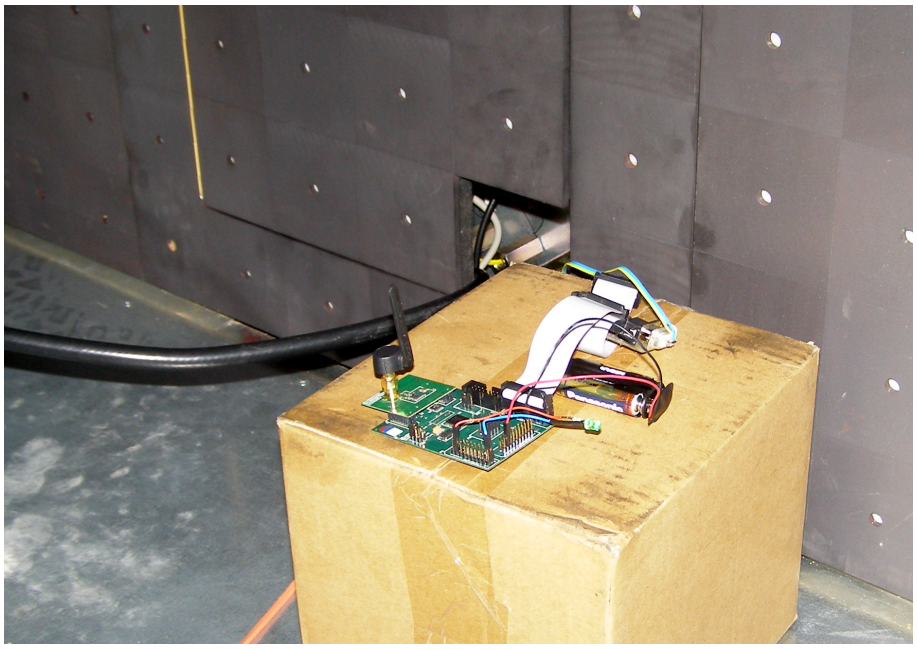
Obr. 8.2: Měření síly signálu v EMC komoře



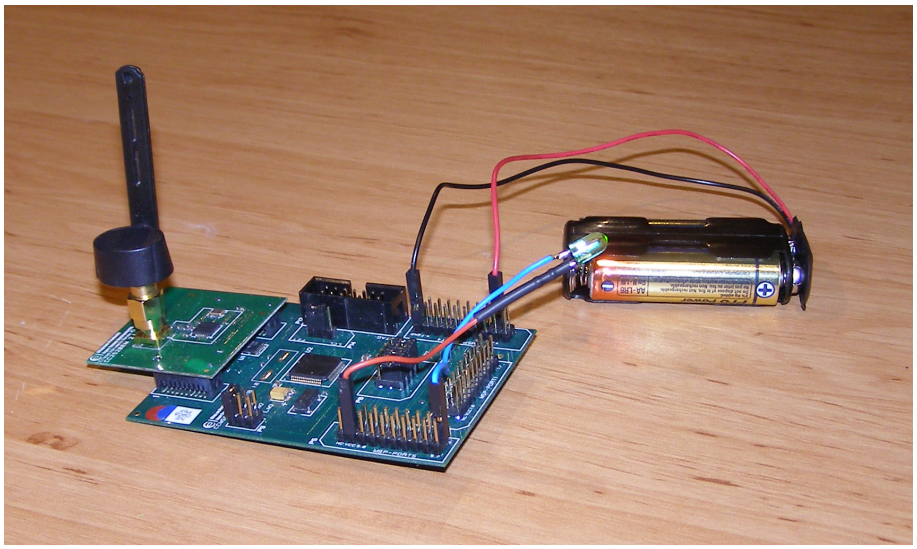
Obr. 8.3: Měření spolehlivosti přenosu v EMC komoře



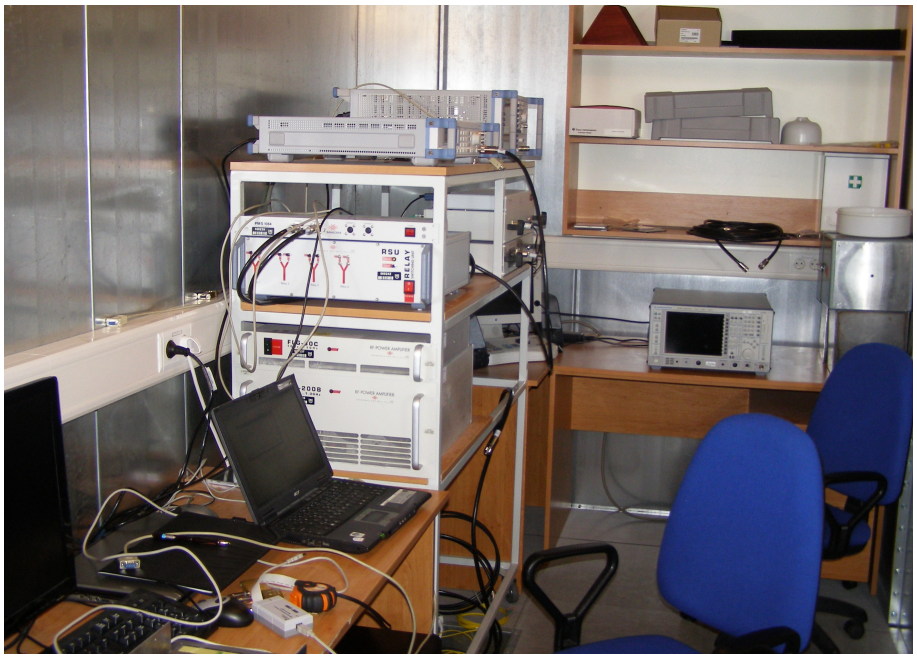
Obr. 8.4: EMC komora



Obr. 8.5: Přijímací modul v EMC komoře



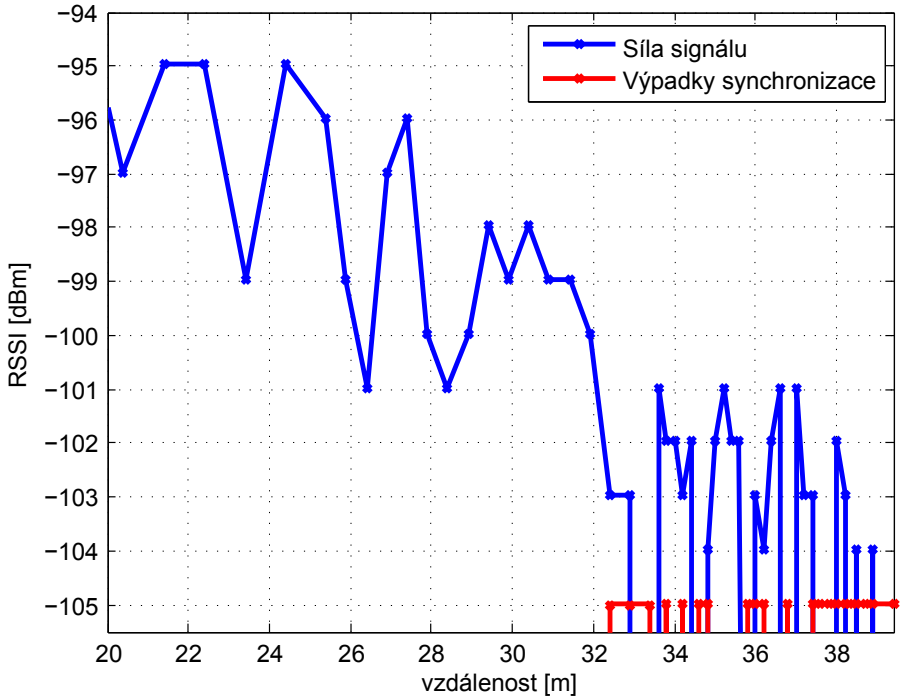
Obr. 8.6: Vysílací modul



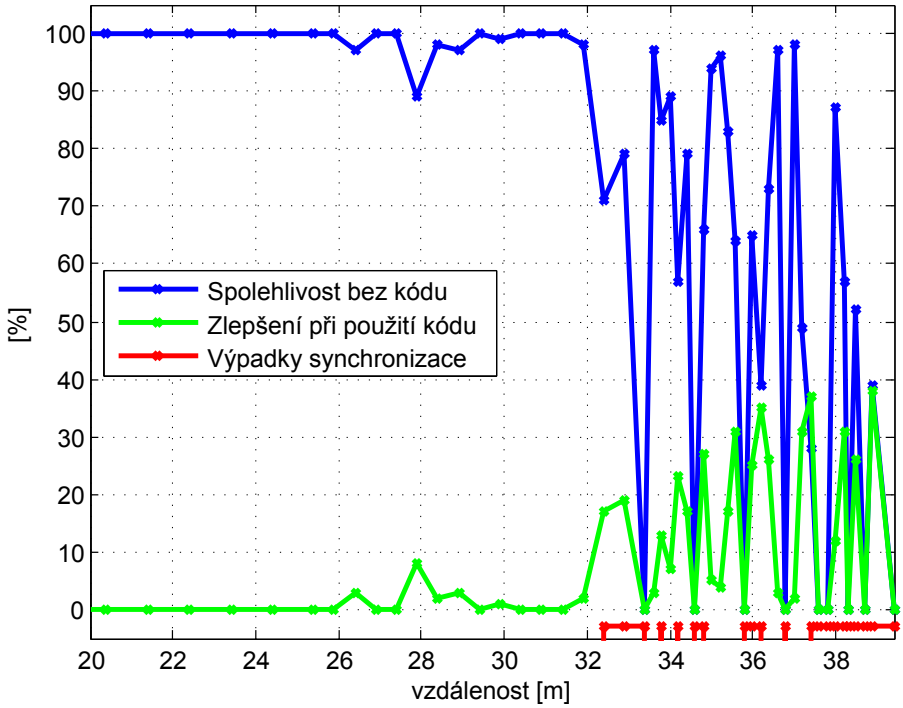
Obr. 8.7: Velitelské stanoviště

Tab. 8.2: Měření na volném prostranství, spolehlivost s kódem a bez kódu

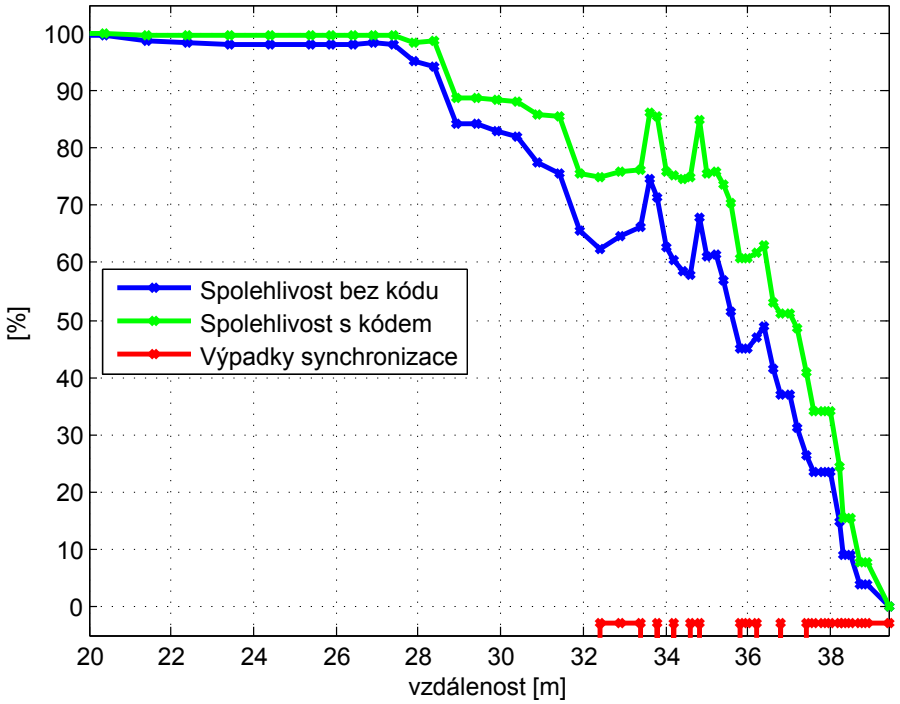
vzd. [m]	RSSI [dBm]	[%] kód	[%] bez kódu	sync.	vzd. [m]	RSSI [dBm]	[%] kód	[%] bez kódu	sync.
39,4	-	0	0	-	34	-102	96	89	
38,9	-104	77	39		33,8	-102	98	85	*výpadky*
38,7	-	0	0	-	33,6	-101	100	97	
38,5	-104	78	52	*výpadky*	33,4	-	0	0	-
38,3	-	0	0	-	32,9	-103	98	79	*výpadky*
38,2	-103	88	57	*výpadky*	32,4	-103	88	71	*výpadky*
38	-102	99	87	*výpadky*	31,9	-100	100	98	
37,8	-	0	0	-	31,4	-99	100	100	
37,6	-	0	0	-	30,9	-99	100	100	
37,4	-103	65	28	*výpadky*	30,4	-98	100	100	
37,2	-103	80	49		29,9	-99	100	99	
37	-101	100	98		29,4	-98	100	100	
36,8	-	0	0	-	28,9	-100	100	97	
36,6	-101	100	97		28,4	-101	100	98	
36,4	-102	99	73		27,9	-100	97	89	
36,2	-104	74	39	*výpadky*	27,4	-96	100	100	
36	-103	90	65	*výpadky*	26,9	-97	100	100	
35,8	-	0	0	-	26,4	-101	100	97	
35,6	-102	95	64		25,9	-99	100	100	
35,4	-102	100	83		25,4	-96	100	100	
35,2	-101	100	96		24,4	-95	100	100	
35	-102	99	94		23,4	-99	100	100	
34,8	-104	93	66	*výpadky*	22,4	-95	100	100	
34,6	-	0	0	-	21,4	-95	100	100	
34,4	-102	96	79		20,4	-97	100	100	
34,2	-103	80	57	*výpadky*	19,4	-94	100	100	



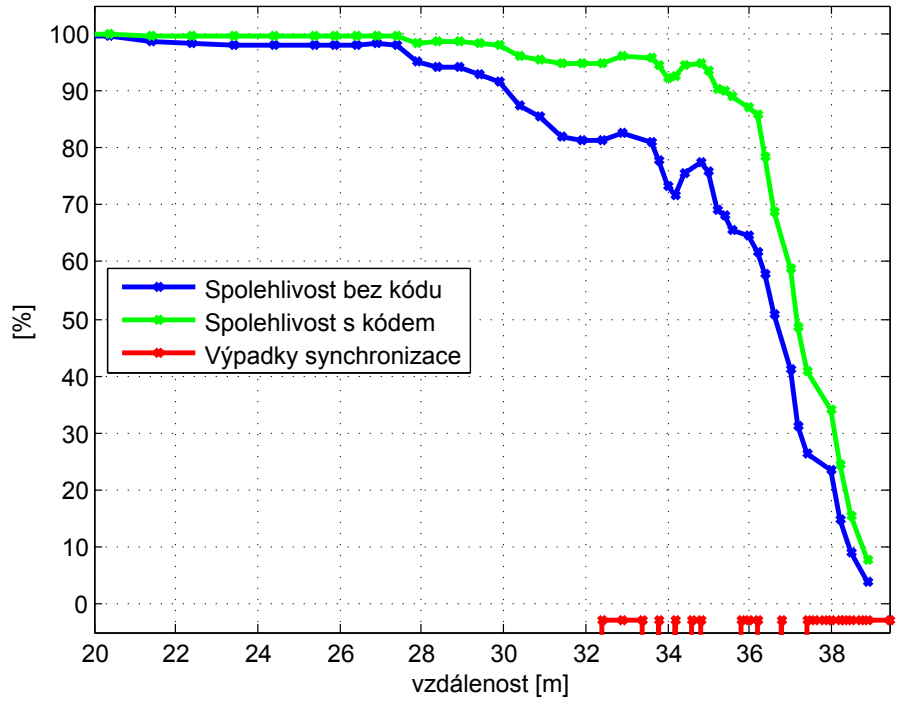
Obr. 8.8: Měření síly signálu na volném prostranství



Obr. 8.9: Měření spolehlivosti přenosu na volném prostranství



Obr. 8.10: Měření na volném prostranství po filtraci



Obr. 8.11: Měření na volném prostranství po filtraci, bez uvažování ztráty synchronizace

9 Závěr

V rámci této práce byl vytvořen software pro převod kontrolní matice na generující, parametrizovatelný návrh LDPC kódů různé délky, simulaci jejich chybovosti a paralelní výpočet minimální vzdálenosti lineárních blokových kódů. Vybraný kód byl implementován v bezdrátové komunikaci a měřena byla spolehlivost přenosu.

LDPC kódy lze popsat kontrolní maticí. Ke každé kontrolní matici existuje reprezentace v podobě *Tannerova grafu*, na kterém je založeno jejich dekódování. Kritickým faktorem pro dekódovací algoritmus je především výskyt krátkých cyklů ve zmiňovaném grafu. Vytvořený software pro návrh LDPC kódů pracuje na principu přidávání nových hran do grafu a vytváření dlouhých cyklů v grafu, přičemž minimální délka cyklů je nastavitelná. Dále umožňuje parametrizovat rozptyl hodností uzlů v tomto grafu (vytváří se tak kódy *iregulární*) a náhodné promíchávání těchto uzů v průběhu generování.

Simulace chybovosti LDPC kódů se provádí pomocí zašumění BPSK modulovaného vektoru v AWGN kanálu. Informační slovo se zakóduje, zašumí a dekóduje. Dekódovaný vektor se pak porovnává s původním informačním slovem. Výstupem simulace je křivka chybovosti v závislosti na poměru E_b/N_0. Podporována je i práce s nebinárními kódy.

Software také umožňuje paralelní výpočet minimální vzdálenosti lineárních blokových kódů. Čím větší je její hodnota, tím lepší opravné schopnosti má kód. Tab. 9.1 ukazuje porovnání doby výpočtu minimální vzdálenosti kódu LDPC (96,48) pomocí programu Matlab a pomocí vytvořeného softwaru. Měření bylo provedeno na počítači s dvoujádrovým procesorem Intel Core 2 Duo 1,8 GHz a uvedené časy jsou průměrem z několika měření. Minimální vzdálenost zmiňovaného kódu je rovna 6.

Součástí práce je dále kodér a dekodér pro implementaci v hardwaru a ukázková aplikace s vývojovým kitem CC2250DK. Dekodér je založený na rychlém *min-sum* algoritmu a pracuje s *fixed-point* aritmetikou.

Navržený kód LDPC (96,48) byl testován v bezdrátové komunikaci, kde měření jednoznačně prokázalo zlepšení spolehlivosti přenosu. Ve výsledcích měření můžeme pozorovat zlepšení spolehlivosti až v řádu desítek procent. Je-li přenos charakterizován zhruba devadesátiprocentní spolehlivostí, při použití zmiňovaného kódu dosáhne spolehlivost sta procent (chybovost je neměřitelná). Je-li přenos charakterizován spolehlivostí kolem 60%, při použití kódou stoupne nad 90%. Největší relativní zlepšení spolehlivosti (dle měření v EMC komoře) je z 25% na 85% při použití kódu. Zde však již docházelo k výpadkům synchronizace, které do této hodnoty nejsou započítávány. Při výpadku synchronizace nejsou přijata žádná data a není co dekódovat.

Pozorovat můžeme také zlepšení dosahu na volném prostranství. K prvním chybám v přenosu došlo ve vzdálenosti 26,4 m, které kód úspěšně opravil na 100% a tuto spolehlivost udržel (kromě jedné výjimky) až do vzdálenosti 31,9 m. Při

dalším vzdalování docházelo vlivem odrazů již k výpadkům synchronizace a spolehlivost přenosu značně kolísala. Úplná ztráta synchronizace, kde již přenos nebyl obnoven, nastala ve vzdálenosti 39,4 m. Vždy však byly prokázány opravné schopnosti kódu a reálně můžeme hovořit o prodloužení dosahu v řádu jednotek metrů.

Výraznou limitací a hlavním faktorem pro výběr použitého kódu byla velikost RAM paměti mikrokontroléru. Kvůli tomuto omezení se bohužel nepodařilo implementovat delší kód s lepšími opravnými schopnostmi, než je LDPC (96,48).

Další kódy byly navrhovány a testovány v simulacích s využitím Národní Gridové Infrastruktury MetaCentrum. Průměrný *kódový zisk* LDPC kódů (generovaných vytvořeným softwarem) délky $n = 64$, 128, 256, 512 a 1024 s 50% redundancí pro chybovost 10^{-4} ukazuje graf na obr. 9.1. Porovnání simulace chybovosti nejlepších kódů z množiny několika navržených ukazuje graf na obr. 9.2. Výpočetní náročnost roste s délkou kódů a simulace delších kódů bohužel nebyla v době odevzdání práce dokončena.

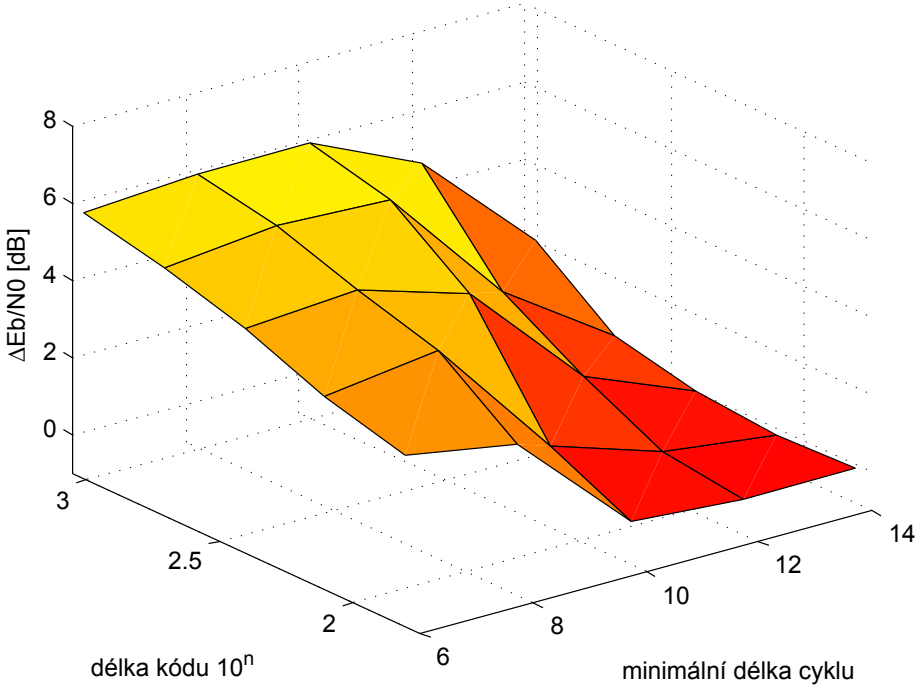
Porovnání simulace binárního a nebinárního kódu v tělese GF(4) je uvedeno na obr. 9.3, přičemž byl vybrán kód LDPC (128,64). Při simulaci nebinárních kódů pracuje vytvořený software řádově mnohem pomaleji (než při simulaci binárních kódů) a existuje zde potenciál pro zlepšení.

Výzvou do budoucna je především implementace rychlého dekodéru v obvodech FPGA. Nabízí se také realizace algoritmu na principu FFT [5] a simulace v jiných typech kanálů, než je AWGN. Zajímavé by jistě bylo podrobné porovnání rychlostí dekódování na různých platformách.

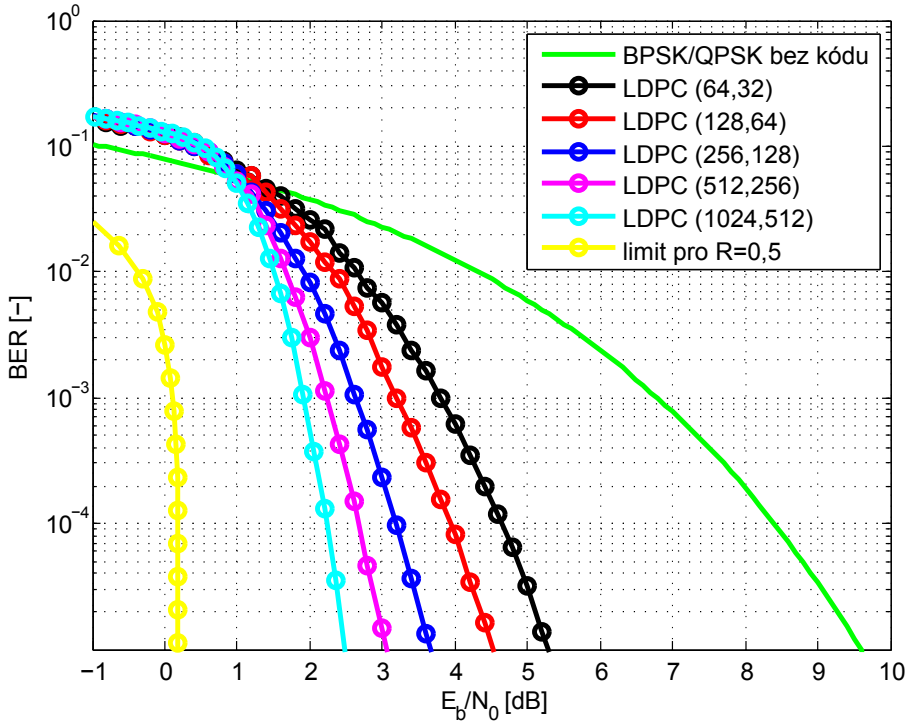
Další věcí, která dosud nebyla v práci diskutována, je simulace pro nízké hodnoty chybovosti. Je možné pozorovat (viz obr. 9.4), že křivka při velmi nízkých hodnotách chybovosti klesá pomaleji. V této souvislosti se hovoří o dvou oblastech - *waterfall* a *error floor*. Tyto simulace jsou však velmi časově náročné.

Tab. 9.1: Doby výpočtu minimální vzdálenosti pro kód použitý při měření

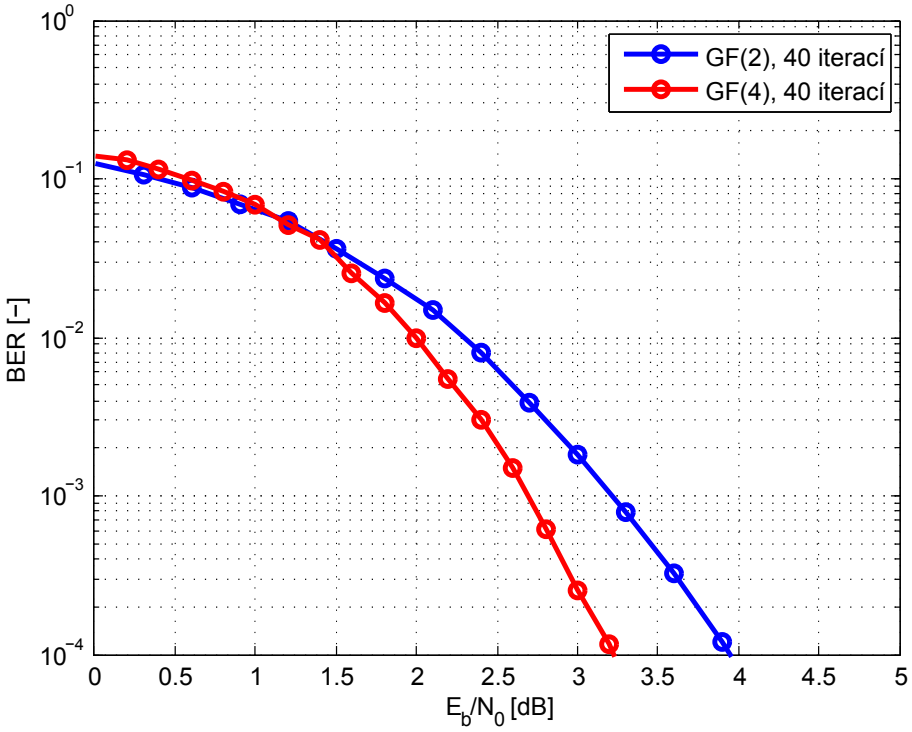
Matlab, funkce gfweight	**54,14 min**
Vlastní řešení, 1 vlákno	**12,26 s**
Vlastní řešení, 2 vlákna	**7,84 s**
Vlastní řešení, 10 vláken	**9,06 s**



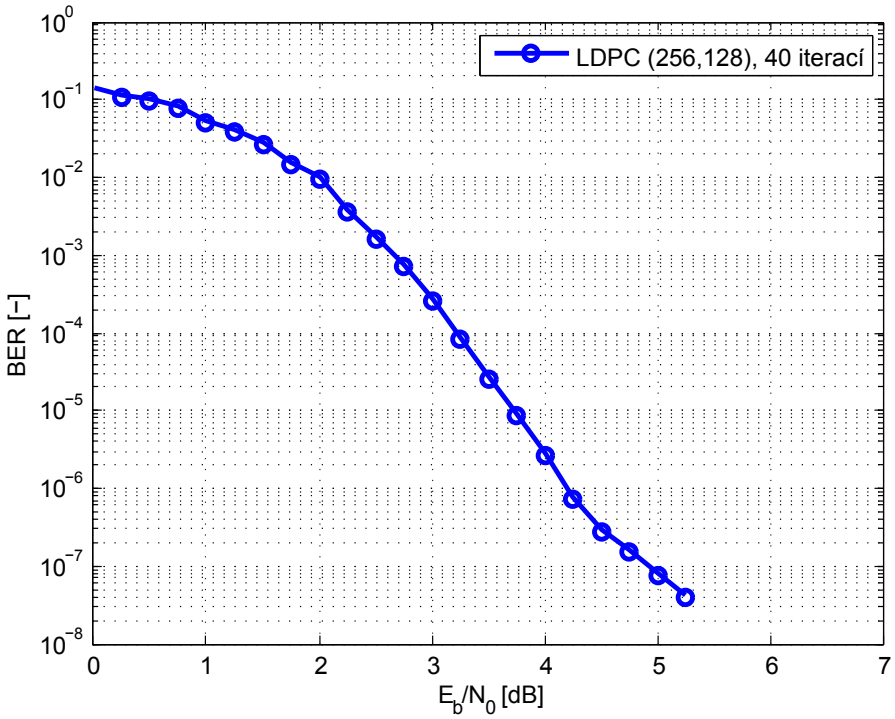
Obr. 9.1: Kódový zisk při $P_b = 10^{-4}$, 40 iterací



Obr. 9.2: Porovnání chybovosti navržených kódů (50 dekódovacích iterací)



Obr. 9.3: Simulace kódu LDPC (128,64) v nebinárním tělese



Obr. 9.4: Simulace pro nízké hodnoty chybovosti

Literatura

[1] ANDREWS, Kenneth, DOLINAR, Sam, DIVSALAR, Dariush., THORPE Jeremy. *Design of Low-Density Parity-Check (LDPC) Codes for Deep-Space Applications.* [online]. [cit. 2012-04-28]. IPN Progress Report 42-159, 2004, 14 s. Dostupné z: `http://tmo.jpl.nasa.gov/progress_report/42-159/159K.pdf`.

[2] BALATSOUKAS-STIMMING, Alex. *Decoding of LDPC codes using the Sum-Product algorithm for the AWGN channel with BPSK modulation.* Technical University of Crete, 2009. [online]. [cit. 2012-04-21]. Dostupné z: `http://www.telecom.tuc.gr/~alex/lectures/lecture5.pdf`.

[3] BEVELACQUA, Peter Joseph. *Antenna Theory.* [online]. [cit. 2012-04-21] Dostupné z: `http://www.antenna-theory.com/`.

[4] BIOLEK, Dalibor. *Datová komunikace : Úvod do teorie informace a kódování.* Vysoké učení technické v Brně, 2002, 78 s. [online]. [cit. 2012-04-21]. Dostupné z: `http://user.unob.cz/biolek/vyukaVUT/skripta/DKO.pdf`.

[5] BYERS, Georey J., TAKAWIRA, Fambirai. *Fourier Transform Decoding of Non-Binary LDPC Codes.* University of KwaZulu-Natal, 2004, 5 s. [online]. [cit. 2012-05-06]. Dostupné z: `http://www.satnac.org.za/proceedings/2004/AccessCoding/No%2078%20-%20Byers.pdf`.

[6] COSTELLO, Daniel J. a G. David FORNEY. *Channel Coding: The Road to Channel Capacity.* Proceedings of the IEEE , 2007, 28 s. DOI: 10.1109/JPROC.2007.895188. [online]. [cit. 2012-04-28]. Dostupné z: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4282117&tag=1`

[7] DAVEY, Matthew C, MACKAY, David. *Low-density parity check codes over GF(q).* Communications Letters, IEEE , vol.2, no.6, 1998, 3 s. doi: 10.1109/4234.681360. [online]. [cit. 2012-04-28]. Dostupné z: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=681360`.

[8] DECHENE, Dan, PEETS, Kevin. *Simulated Performance of Low-Density Parity-Check Codes : A MATLAB Implementation.* Lakehead University, 2006, 46 s. [online]. [cit. 2012-04-21]. Dostupné z: `http://www.dechene.ca/papers/report_4th.pdf`.

[9] DECLERCQ, David. *Status of knowledge on non-binary LDPC decoders.* IEEE SSC SCV Tutorial, 2010. [online]. [cit. 2012-04-21]. Dostupné z: `http://sites.ieee.org/scv-sscs/previous-events/events-2010/october-21-2010-2`.

[10] FAN, Jun, Yung XIAO a Kiseon KIM. Design LDPC Codes without Cycles of Length 4 and 6. 2008, 5 s. DOI: 10.1155/2008/354137. [online]. [cit. 2012-04-28]. Dostupné z: http://www.hindawi.com/journals/jece/2008/354137/.

[11] FITTON, Mike. *Principles of Digital Modulation.* Toshiba Research Europe Limited, 2001, 40 s. [online]. [cit. 2012-04-21]. Dostupné z: http://www.berk.tc/combas/digital_mod.pdf.

[12] GALLAGER, G. Robert. *Low-Density Parity-Check Codes.* IEEE, 1962, 8 s.. [online]. [cit. 2012-04-28]. Dostupné z: http://www.comm.csl.uiuc.edu/~koetter/ece459/LDPC_gallager.pdf.

[13] GALLAGER, G. Robert. *Low-Density Parity-Check Codes.* 1963, 90 s.. [online]. [cit. 2012-04-28]. Dostupné z: http://www.rle.mit.edu/rgallager/documents/ldpc.pdf.

[14] GANEPOLA, Vajira S., CARRASCO, Rolando A., WASSELL, Ian J., LE GOFF, Stéphane Y. *Performance study of non-binary LDPC Codes over GF(q).* Communication Systems, Networks and Digital Signal Processing, 2008., 5 s. doi: 10.1109/CSNDSP.2008.4610743. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4610743.

[15] GEORGIEV, Vjačeslav. *Hammingův kód.* Západočeká univerzita v Plzni, 2005, 3 s. [online]. [cit. 2012-04-21]. Dostupné z: http://webs.zcu.cz/fel/kae/pi/kody/Hamming_code.pdf.

[16] GILL, John. *Algebraic Error Control Codes : Handouts.* Stanford University, 2010. [online]. [cit. 2012-04-21]. Dostupné z: http://www.stanford.edu/class/ee387/handouts.shtml.

[17] GILRA, Deepak. *A Class of Non-Binary LDPC Codes.* Thesis, Texas A&M University, 2003, s. 57. [online]. [cit. 2012-04-28]. Dostupné z: http://repository.tamu.edu/bitstream/handle/1969.1/67/etd-tamu-2003A-2003032814-Gilr-1.pdf.

[18] GURUSWAMI, Venkatesan. *Introduction to Coding Theory : Course notes.* Carnegie Mellon University, 2010. [online]. [cit. 2012-04-21]. Dostupné z: http://www.cs.cmu.edu/~venkatg/teaching/codingtheory/.

[19] HAGIWARA, Manabu, IMAI, Hideki. *Quantum Quasi-Cyclic LDPC Codes.* Chuo University. 2010, 18 s. [online]. [cit. 2012-04-21]. Dostupné z: http://arxiv.org/pdf/quant-ph/0701020.pdf.

[20] HALL, Jonathan I. *Notes on Coding Theory.* Michigan State University, 2012. [online]. [cit. 2012-04-21]. Dostupné z: http://www.mth.msu.edu/~jhall/classes/codenotes/coding-notes.html.

[21] HOWARD, Sheryl. *Energy Efficiency of Error-Correcting Coding in Low-Power Wireless Links.* Northern Arizona University, 2007, s. 23. [online]. [cit. 2012-04-11]. Dostupné z: http://www.cmoset.com/uploads/7.3.pdf.

[22] HU, Xiao-Yu, ELEFTHERIOU, Evangelos, ARNOLD, Dieter M. *Regular and irregular progressive edge-growth tanner graphs.* Information Theory, IEEE Transactions, vol.51, no.1, 2005, 13 s, doi: 10.1109/TIT.2004.839541. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1377521&tag=1.

[23] HU, Xiao-Yu, FOSSORIER, Marc P.C., ELEFTHERIOU, Evangelos. *On the Computation of the Minimum Distance of Low-Density Parity-Check Codes.* Communications, 2004 IEEE International Conference, vol.2, 2004, 5 s. doi: 10.1109/ICC.2004.1312605. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1312605.

[24] CHUNG, Sae-Young, FORNEY, G. David, RICHARDSON, Thomas J., URBANKE, Rüdiger. *On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit.* Communications Letters, IEEE , vol.5, no.2, 2001, 3 s. doi: 10.1109/4234.905935. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=905935.

[25] JARVI, Trent. *RXTX project.* [online]. [cit. 2012-04-13]. Dostupné z: http://rxtx.qbang.org.

[26] JOHNSON Sarah J. *Introducing Low-Density Parity-Check Codes.* The University of Newcastle, Australia, 83 s. [online]. [cit. 2012-04-18]. Dostupné z: http://materias.fi.uba.ar/6624/index_files/outline_archivos/SJohnsonLDPCintro.pdf.

[27] JOHNSON Sarah J. *Low-Density Parity-Check Codes from Combinatorial Designs.* The University of Newcastle, Australia, 2004, 227 s. [online]. [cit. 2012-04-20]. Dostupné z: http://project.sigpromu.org/reports/Report185.pdf.

[28] JORDAN, Ralph. *Channel Coding: Lecture Notes.* University of Ulm, 2003, 107 s. [online]. [cit. 2012-03-29]. Dostupné z: http://tait.e-technik.uni-ulm.de/~schmidt/documents/ccscript.pdf .

[29] KIYANI, Nauman F., WEBER, Jos H. *Analysis of Random Regular LDPC Codes on Rayleigh Fading Channels.* Delft University of Technology, 2006, 8 s. [online]. [cit. 2012-04-20]. Dostupné z: http://www.ewi.tudelft.nl/fileadmin/Faculteit/EWI/Over_de_faculteit/Afdelingen/Telecommunications/Organisation/Sections/WMC/Publications/2006/doc/Kiyani_Proceedings_WIC_SITB2006_Noordwijk.pdf.

[30] KNUDSEN, Joakim Grahl. *Randomised Construction and Dynamic Decoding of LDPC Codes.* University of Bergen, 2005, 116 s. [online]. [cit. 2012-04-21]. Dostupné z: http://www.ii.uib.no/~matthew/Masters/Joakimmaster.pdf.

[31] KSCHISCHANG, Frank R. *Codes Defined on Graphs.* Communications Magazine, IEEE, vol.41, no.8, 2003, 8 s. doi: 10.1109/MCOM.2003.1222727. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1222727.

[32] LEINER, Bernhard M.J. *LDPC Codes – a brief Tutorial.* University of Arizona, 2005, 9 s. [online]. [cit. 2012-04-21]. Dostupné z: http://www2.engr.arizona.edu/~ece506/readings/project-reading/5-ldpc/LDPC_Intro.pdf.

[33] LESLIE, Martin. *Low-Density Parity-Check codes.* University of Arizona, 2009. [online]. [cit. 2012-04-21]. Dostupné z: http://math.arizona.edu/~mleslie/files/ldpc.pdf.

[34] LIN, Shu, COSTELLO, Daniel J. *Error Control Coding : Fundamentals and Applications.* Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1983, 603 s. ISBN 0-13-283796-X.

[35] MACKAY, David J.C. *Encyclopedia of Sparse Graph Codes.* [online]. [cit. 2012-04-21]. Dostupné z: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[36] MACKAY, David J.C., NEAL, Radhord M. *Good Codes Based on Very Sparse Matrices.* [online]. [cit. 2012-04-26]. Dostupné z: http://www.cs.toronto.edu/~mackay/mnc4s.pdf.

[37] MACKAY, David J.C. *Good Error-Correcting Codes Based on Very Sparse Matrices.* Information Theory, IEEE Transactions, vol.45, no.2, 1999, 33 s., doi: 10.1109/18.748992. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=748992.

[38] MACKAY, David J.C. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003, 628 s. [online]. [cit. 2012-04-28]. Dostupné z: http://www.cs.toronto.edu/~mackay/itprnn/book.pdf.

[39] MAHADEVAN, Amitkumar. *On LDPC Codes for ADSL.* University of Maryland, 2002, 298 s. [online]. [cit. 2012-04-21]. Dostupné z: http://userpages.umbc.edu/~amahad1/Publications/ms.pdf.

[40] MEGHDADI, Vahid. *BER calculation.* 2008, s. 9. [online]. [cit. 2012-03-29] Dostupné z: http://perso.ensil.unilim.fr/~meghdadi/notes/ber_awgn.pdf .

[41] MOON, K. Todd. *Error Correction Coding : Mathematical methods and algorithms.* John Wiley & Sons, Inc., 2005, 756 s. ISBN 0-471-64800-0.

[42] PILLAI, Krishna. *Modulation roundup: error rates, noise, and capacity.* 2008. [online]. [cit. 2012-04-15]. Dostupné z: http://www.eetimes.com/design/communications-design/4017668/Modulation-roundup-error-rates-noise-and-capacity.

[43] PRASARTKAEW, Chutima, CHOOMCHUAY, Somsak. *A Design of Parity Check Matrix for Irregular LDPC Codes.* College of Data Storage Technology and Applications, 2009, s.4. [online]. [cit. 2012-04-28]. Dostupné z: http://www.kmitl.ac.th/~kchsomsa/somsak/papers/iscit_2k9.pdf.

[44] QIAN, BeiBei, YAO, Dongping, XIONG Lei. *A precise Approach for Computing Channel Soft Information.* IEEE International Symposium, 2009, 4 s. doi: 10.1109/MAPE.2009.5355835. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5355835.

[45] RICHARDSON, Thomas J., SHOKROLLAHI, M. Amin, URBANKE, Rüdiger L. *Design of capacity-approaching irregular low-density parity-check codes.* Information Theory, IEEE Transactions, vol.47, no.2, 2001, 19 s, doi: 10.1109/18.910578. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=910578.

[46] RICHARDSON, Thomas J. *Error Floors of LDPC Codes.* Flarion Technologies, 2004, 10 s. [online]. [cit. 2012-04-21]. Dostupné z: http://ldpccodes.com/papers/ErrorFloors.pdf.

[47] ROWEIS, Sam. *Equivalent Codes & Systematic Forms.* 2005, 4 s. [online]. [cit. 2012-04-21]. Dostupné z: http://www.cs.nyu.edu/~roweis/csc310-2005/notes/lec16x.pdf.

[48] RYAN, William E. *An Introduction to LDPC Codes.* The University of Arizona, 2003, 23 s. [online]. [cit. 2012-04-21]. Dostupné z: http://tuk88.free.fr/LDPC/ldpcchap.pdf.

[49] SANKAR, Krishna. *Bit Error Rate (BER) for BPSK modulation.* 2007. [online]. [cit. 2012-04-21]. Dostupné z: http://www.dsplog.com/2008/06/18/bounds-on-communication-shannon-capacity/.

[50] SANKAR, Krishna. *Bounds on Communication based on Shannon's capacity.* 2008. [online]. [cit. 2012-04-21]. Dostupné z: http://www.dsplog.com/2007/08/05/bit-error-probability-for-bpsk-modulation/.

[51] SIEGEL, Paul H. *An Introduction to Low-Density Parity-Check Codes.* University of California, 2007. [online]. [cit. 2012-04-21]. Dostupné z: http://cmrr-star.ucsd.edu/psiegel/pubs/07/ldpc_tutorial.ppt.

[52] SHINDE, Rajendra B. *Analysis of Belief Propagation Decoder of LDPC Codes.* 2007, 6 s. [online]. [cit. 2012-04-21]. Dostupné z: http://www.stanford.edu/~rbs/Projects/IS_report.pdf.

[53] SHOKROLLAHI, Amin. *LDPC Codes: An Introduction.* Digital Fountain, Inc., 2003, 34 s. [online]. [cit. 2012-04-17]. Dostupné z: http://www.telecom.tuc.gr/~alex/papers/amin.pdf.

[54] SPAGNOL, Christian. *Aspects of LDPC codes for hardware implementation.* National University of Ireland, 2009, 300 s. [online]. [cit. 2012-04-28]. Dostupné z: http://rennes.ucc.ie/~christians/pdfs/Post_viva_Spagnol_th.pdf.

[55] SPAGNOL, Christian, POPOVICI, Emanuel, MARNANE, William. *New Algorithm For LDPC Decoding Over GF(q).* University College Cork, 2005, 6 s. [online]. [cit. 2012-04-21]. Dostupné z: http://rennes.ucc.ie/~christians/pdfs/Christian_Spagnol_ISSC05.pdf.

[56] SUN, Jian. *An Introduction to Low Density Parity Check (LDPC) Codes.* West Virginia University, 2003, 22 s. [online]. [cit. 2012-04-17]. Dostupné z: http://materias.fi.uba.ar/6624/index_files/outline_archivos/slideldpc.pdf.

[57] TANNER, R. Michael. *A Recursive Approach to Low Complexity Codes.* Information Theory, IEEE Transactions, vol.27, no.5, 1981, 15 s. doi: 10.1109/TIT.1981.1056404. [online]. [cit. 2012-04-28]. Dostupné z: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1056404.

[58] TERVO, Richard. *Addition and Multiplication Tables in Galois Fields GF(2^m).* University of New Brunswick, 2011. [online]. [cit. 2012-04-21]. Dostupné z: `http://www.ee.unb.ca/cgi-bin/tervo/galois3.pl`.

[59] THANGARAJ, Andrew. *Error Control Coding : Recorded Lectures.* Indian Institute of Technology, 2011. [online]. [cit. 2012-04-21]. Dostupné z: `http://www.ee.iitm.ac.in/~andrew/videolectures/EE512/`.

[60] TRACHTENBERG, Ari. *Error-Correcting Codes on Graphs: Lexicodes, Trellises and Factor Graphs.* 2001. [online]. [cit. 2012-04-21]. Dostupné z: `http://ipsit.bu.edu/phdthesis_html/phdthesis_html.html`.

[61] TRAROE Nana, KANT, Shashi, JENSEN, Tobias Lindstrøm. *Message Passing Algortihm and Linear Programming Decoding for LDPC and Linear Block Codes.* Aalborg University, 2007, 108 s. [online]. [cit. 2012-14-18]. Dostupné z: `http://kom.aau.dk/~tlj/mpa_lp_decoding_ldpc.pdf`.

[62] TRIFONOV, Peter. *Design of Structured Irregular LDPC Codes.* IEEE Region 8 International Conference on, Computational Technologies in Electrical and Electronics Engineering, 2008, 5 s. [online]. [cit. 2012-24-24]. Dostupné z: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4602580&tag=1`.

[63] VAZIRANI, Umesh V., DASGUPTA, Sanjoy, PAPADIMITRIOU, Christos H. *Algorithms : Paths in graphs.* University of California at Berkeley, 2006, 24 s. [online]. [cit. 2012-04-21]. Dostupné z: `http://www.cs.berkeley.edu/~vazirani/algorithms/chap4.pdf`.

[64] VLČEK, Karel. *Komprese a kódová zabezpečení v multimediálních komunikacích.* BEN - Technická literatura, Praha, 2004, 258 s. ISBN 80-7300-134-9.

[65] ZAHEER, Shaikh Faisal. *LDPC Code Simulation.* 2005. [online]. [cit. 2012-04-24]. Dostupné z: `http://www.mathworks.com/matlabcentral/fileexchange/8977-ldpc-code-simulation`.

[66] ZHANG, Zhengya. *Study of Permutation Matrices Based LDPC Code Construction.* 2005, 13 s. [online]. [cit. 2012-04-21]. Dostupné z: `http://www.eecs.berkeley.edu/~ananth/229BSpr05/Reports/ZhengyaZhang.pdf`.

[67] ZHONG, Hao, XU, Wei, XIE, Ningde, ZHANG, Tong. *Area-Efficient Min-Sum Decoder Design for High-Rate Quasi-Cyclic Low-Density Parity-Check Codes in Magnetic Recording.* IEEE Transactions vol.43, no.12, 2007, 6 s. doi: 10.1109/TMAG.2007.906890. [online]. [cit. 2012-04-28]. Dostupné z: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4380288&tag=1`.

[68] ZHONG, Zhou, LI, Yunzhou, CHEN, Xiang, HU, Hanying, WANG, Jing. *Modified min-sum decoding algorithm for LDPC codes based on classified correction.* Communications and Networking in China, 2008, 5 s. doi: 10.1109/CHINACOM.2008.4685176. [online]. [cit. 2012-04-28]. Dostupné z: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4685176`.

[69] ŽEMLIČKA, Jan. *Praktická linární algebra a geometrie: Cvičení, Lineární kódy.* MFF UK v Praze, 2005. [online]. [cit. 2012-04-21]. Dostupné z: `http://www.karlin.mff.cuni.cz/~zemlicka/cvic5-6/pla_1.htm`.

[70] Eclipse Foundation. *SWT: The Standard Widget Toolkit.* [online]. [cit. 2012-04-13]. Dostupné z: `http://www.eclipse.org/swt/`.

[71] Maxim. *MAX220–MAX249 Family datasheet.* 2010. [online]. [cit. 2012-04-13]. Dostupné z: `http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf`.

[72] STMicroelectronics. *LExxAB, LExxC : Very low drop voltage regulators with inhibit.* 2008, 39 s. [online]. [cit. 2012-04-30]. Dostupné z: `http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00000545.pdf`.

[73] Texas Instruments. *CC2520 Datasheet.* 2007, 133 s. [online]. [cit. 2012-04-15]. Dostupné z: `http://www.ti.com/lit/ds/symlink/cc2520.pdf`.

[74] Texas Instruments. *CC2520 Software Examples : User's Guide.* 2009 28 s. [online]. [cit. 2012-04-15]. Dostupné z: `http://www.ti.com/lit/ug/swru137b/swru137b.pdf`.

[75] Texas Instruments. *CC2520 Development Kit : User's Guide.* 2007, 45 s. [online]. [cit. 2012-04-15]. Dostupné z: `http://www.ti.com/lit/ug/swru138/swru138.pdf`.

[76] Texas Instruments. *MSP430F261x, MSP430F241x : Mixed Signal Microcontroller.* 2011, 102 s. [online]. [cit. 2012-04-15]. Dostupné z `http://www.ti.com/lit/ds/symlink/msp430f2618.pdf`.

[77] Texas Instruments. *MSP430F261x, MSP430F241x : Device Erratasheet.* 2011, 24 s. [online]. [cit. 2012-04-15]. Dostupné z `http://www.ti.com/lit/er/slaz033j/slaz033j.pdf`.

[78] Texas Instruments. *MSP430x2xx Family : User's Guide.* 2012, 658 s. [online]. [cit. 2012-04-15]. Dostupné z `http://www.ti.com/lit/ug/slau144i/slau144i.pdf`.

10 Přílohy

10.1 Kompletní ukázka průběhu dekódování

Jedná se příklad dekódování chybného slova a ukázku předávaných zpráv v průběhu dekódovacího procesu.

10.1.1 Kontrolní matice kódu, kódový a chybový vektor

Zadáno:

$$\mathbb{H} = \begin{bmatrix}
0&1&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&1&0&0&0\\
0&0&0&1&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&1&0\\
0&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0\\
0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0
\end{bmatrix},$$

$$\mathbf{c} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$

$$\mathbf{e} = [\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ ],$$

$$\mathbf{c}' = \mathbf{c} \oplus \mathbf{e},$$

$$\mathbf{c}' = [\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ].$$

10.1.2 Předávané zprávy pro SP algoritmus

Veškeré hodnoty se vztahují k symbolu 1 ($q_{ij}^{(1)}$, $r_{ij}^{(1)}$).

it.	$q_{6,0}$	$q_{10,0}$	$q_{0,1}$	$q_{7,1}$	$q_{3,2}$	$q_{9,2}$	$q_{5,3}$	$q_{7,3}$	$q_{5,4}$	$q_{12,4}$
1	0,1	0,1	0,9	0,9	0,9	0,9	0,1	0,1	0,9	0,9
2	0,035	0,035	0,965	0,744	0,744	0,744	0,035	0,035	0,500	0,965
3	0,076	0,012	0,992	0,9	0,959	0,823	0,040	0,1	0,906	0,980
it.	$r_{6,0}$	$r_{10,0}$	$r_{0,1}$	$r_{7,1}$	$r_{3,2}$	$r_{9,2}$	$r_{5,3}$	$r_{7,3}$	$r_{5,4}$	$r_{12,4}$
1	0,244	0,244	0,244	0,756	0,244	0,244	0,244	0,244	0,756	0,100
2	0,099	0,426	0,5	0,930	0,341	0,722	0,5	0,275	0,847	0,516
3	0,288	0,151	0,828	0,779	0,042	0,439	0,106	0,221	0,946	0,215

it.	$q_{5,5}$	$q_{6,5}$	$q_{7,6}$	$q_{10,6}$	$q_{0,7}$	$q_{9,7}$	$q_{12,7}$	$q_{4,8}$	$q_{9,8}$
1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,9	0,9
2	0,965	0,965	0,035	0,035	0,756	0,756	0,516	0,744	0,965
3	0,988	0,9	0,076	0,040	0,884	0,756	0,215	0,935	0,366
it.	$r_{5,5}$	$r_{6,5}$	$r_{7,6}$	$r_{10,6}$	$r_{0,7}$	$r_{9,7}$	$r_{12,7}$	$r_{4,8}$	$r_{9,8}$
1	0,756	0,756	0,244	0,244	0,756	0,756	0,9	0,756	0,244
2	0,5	0,901	0,275	0,426	0,5	0,711	0,965	0,060	0,616
3	0,871	0,725	0,236	0,130	0,919	0,423	0,980	0,031	0,646

it.	$q_{2,9}$	$q_{4,9}$	$q_{8,9}$	$q_{13,9}$	$q_{4,10}$	$q_{6,10}$	$q_{0,11}$	$q_{8,11}$	$q_{1,12}$	$q_{3,12}$
1	0,9	0,9	0,9	0,9	0,9	0,9	0,1	0,1	0,9	0,9
2	0,405	0,024	0,405	0,256	0,965	0,965	0,500	0,256	0,744	0,988
3	0,754	0,005	0,034	0,032	0,988	0,769	0,070	0,038	0,867	0,999
it.	$r_{2,9}$	$r_{4,9}$	$r_{8,9}$	$r_{13,9}$	$r_{4,10}$	$r_{6,10}$	$r_{0,11}$	$r_{8,11}$	$r_{1,12}$	$r_{3,12}$
1	0,100	0,756	0,100	0,180	0,756	0,756	0,756	0,9	0,9	0,244
2	0,004	0,725	0,256	0,273	0,270	0,901	0,262	0,405	0,989	0,421
3	0,004	0,912	0,038	0,259	0,082	0,835	0,125	0,034	0,898	0,078

it.	$q_{2,13}$	$q_{5,13}$	$q_{14,13}$	$q_{10,14}$	$q_{13,14}$	$q_{0,15}$	$q_{11,15}$	$q_{14,15}$
1	0,1	0,1	0,1	0,9	0,9	0,1	0,1	0,1
2	0,004	0,100	0,244	0,664	0,965	0,001	0,037	0,037
3	0,004	0,003	0,070	0,936	0,957	0,004	0,035	0,012
it.	$r_{2,13}$	$r_{5,13}$	$r_{14,13}$	$r_{10,14}$	$r_{13,14}$	$r_{0,15}$	$r_{11,15}$	$r_{14,15}$
1	0,9	0,244	0,100	0,756	0,180	0,756	0,100	0,100
2	0,405	0,5	0,037	0,711	0,619	0,5	0,100	0,244
3	0,754	0,136	0,012	0,890	0,747	0,175	0,016	0,070

it.	$q_{1,16}$	$q_{4,16}$	$q_{7,16}$	$q_{3,17}$	$q_{13,17}$	$q_{3,18}$	$q_{6,18}$	$q_{11,18}$	$q_{9,19}$	$q_{10,19}$
1	0,9	0,9	0,9	0,9	0,9	0,1	0,1	0,1	0,9	0,9
2	0,989	0,996	0,996	0,664	0,744	0,004	0,037	0,100	0,965	0,744
3	0,898	0,985	0,912	0,960	0,763	4E-4	0,006	0,016	0,942	0,935
it.	$r_{1,16}$	$r_{4,16}$	$r_{7,16}$	$r_{3,17}$	$r_{13,17}$	$r_{3,18}$	$r_{6,18}$	$r_{11,18}$	$r_{9,19}$	$r_{10,19}$
1	0,9	0,756	0,756	0,244	0,180	0,756	0,244	0,100	0,244	0,756
2	0,744	0,284	0,711	0,264	0,727	0,578	0,097	0,037	0,616	0,642
3	0,867	0,079	0,771	0,043	0,928	0,921	0,318	0,035	0,456	0,891

10.1.3 Předávané zprávy pro min-sum algoritmus

Veškeré hodnoty se opět vztahují k symbolu 1.

it.	$L(q_{6,0})$	$L(q_{10,0})$	$L(q_{0,1})$	$L(q_{7,1})$	$L(q_{3,2})$	$L(q_{9,2})$	$L(q_{5,3})$	$L(q_{7,3})$
1	2,197	2,197	-2,197	-2,197	-2,197	-2,197	2,197	2,197
2	4,394	4,394	-4,394	0,000	0,000	0,000	4,394	4,394
3	2,197	4,394	-6,592	-2,197	-4,394	-2,197	2,197	2,197
4	4,394	4,394	-4,394	-4,394	-0,000	0,000	4,394	6,592
it.	$L(r_{6,0})$	$L(r_{10,0})$	$L(r_{0,1})$	$L(r_{7,1})$	$L(r_{3,2})$	$L(r_{9,2})$	$L(r_{5,3})$	$L(r_{7,3})$
1	2,197	2,197	2,197	-2,197	2,197	2,197	2,197	2,197
2	2,197	0,000	-0,000	-4,394	-0,000	-2,197	-0,000	-0,000
3	2,197	2,197	-2,197	-2,197	2,197	2,197	4,394	2,197
4	-0,000	0,000	-0,000	-2,197	4,394	0,000	-0,000	2,197

it.	$L(q_{5,4})$	$L(q_{12,4})$	$L(q_{5,5})$	$L(q_{6,5})$	$L(q_{7,6})$	$L(q_{10,6})$	$L(q_{0,7})$	$L(q_{9,7})$
1	-2,197	-2,197	-2,197	-2,197	2,197	2,197	2,197	2,197
2	0,000	-4,394	-4,394	-4,394	4,394	4,394	-2,197	-2,197
3	-4,394	-4,394	-4,394	-2,197	2,197	2,197	-2,197	-2,197
4	0,000	-4,394	-4,394	-4,394	4,394	4,394	-0,000	-4,394
it.	$L(r_{5,4})$	$L(r_{12,4})$	$L(r_{5,5})$	$L(r_{6,5})$	$L(r_{7,6})$	$L(r_{10,6})$	$L(r_{0,7})$	$L(r_{9,7})$
1	-2,197	2,197	-2,197	-2,197	2,197	2,197	-2,197	-2,197
2	-2,197	-2,197	0,000	-2,197	-0,000	0,000	-0,000	0,000
3	-2,197	2,197	-2,197	-2,197	2,197	2,197	-2,197	2,197
4	-4,394	2,197	0,000	0,000	2,197	0,000	-4,394	-0,000

it.	$L(q_{12,7})$	$L(q_{4,8})$	$L(q_{9,8})$	$L(q_{2,9})$	$L(q_{4,9})$	$L(q_{8,9})$	$L(q_{13,9})$
1	2,197	-2,197	-2,197	-2,197	-2,197	-2,197	-2,197
2	-2,197	0,000	-4,394	0,000	4,394	0,000	0,000
3	2,197	-2,197	2,197	-2,197	4,394	4,394	4,394
4	2,197	-4,394	0,000	2,197	8,789	2,197	4,394
it.	$L(r_{12,7})$	$L(r_{4,8})$	$L(r_{9,8})$	$L(r_{2,9})$	$L(r_{4,9})$	$L(r_{8,9})$	$L(r_{13,9})$
1	-2,197	-2,197	2,197	2,197	-2,197	2,197	2,197
2	-4,394	4,394	0,000	6,592	0,000	0,000	-0,000
3	-4,394	2,197	-2,197	4,394	-2,197	4,394	2,197
4	-4,394	4,394	0,000	8,789	-4,394	4,394	-2,197

it.	$L(q_{4,10})$	$L(q_{6,10)}$	$L(q_{0,11})$	$L(q_{8,11})$	$L(q_{1,12})$	$L(q_{3,12})$	$L(q_{2,13})$	$L(q_{5,13})$
1	-2,197	-2,197	2,197	2,197	-2,197	-2,197	2,197	2,197
2	-4,394	-4,394	0,000	0,000	0,000	-4,394	6,592	2,197
3	-4,394	-2,197	2,197	4,394	-2,197	-8,789	4,394	4,394
4	-4,394	0,000	6,592	4,394	0,000	-4,394	8,789	4,394
it.	$L(r_{4,10})$	$L(r_{6,10})$	$L(r_{0,11})$	$L(r_{8,11})$	$L(r_{1,12})$	$L(r_{3,12})$	$L(r_{2,13})$	$L(r_{5,13})$
1	-2,197	-2,197	-2,197	-2,197	-2,197	2,197	-2,197	2,197
2	-0,000	-2,197	2,197	0,000	-6,592	0,000	0,000	-0,000
3	2,197	-2,197	2,197	4,394	-2,197	2,197	-2,197	2,197
4	4,394	-4,394	0,000	2,197	-2,197	0,000	2,197	-0,000

it.	$L(q_{14,13})$	$L(q_{10,14})$	$L(q_{13,14})$	$L(q_{0,15})$	$L(q_{11,15})$	$L(q_{14,15})$	$L(q_{1,16})$	$L(q_{4,16})$
1	2,197	-2,197	-2,197	2,197	2,197	2,197	-2,197	-2,197
2	2,197	0,000	-4,394	6,592	2,197	2,197	-6,592	-6,592
3	2,197	-2,197	-2,197	6,592	4,394	4,394	-2,197	-2,197
4	2,197	-4,394	-4,394	10,986	6,592	10,986	-2,197	-6,592
it.	$L(r_{14,13})$	$L(r_{10,14})$	$L(r_{13,14})$	$L(r_{0,15})$	$L(r_{11,15})$	$L(r_{14,15})$	$L(r_{1,16})$	$L(r_{4,16})$
1	2,197	-2,197	2,197	-2,197	2,197	2,197	-2,197	-2,197
2	2,197	0,000	0,000	0,000	2,197	2,197	0,000	-0,000
3	4,394	-2,197	-2,197	2,197	6,592	2,197	-2,197	2,197
4	10,986	-0,000	2,197	0,000	2,197	2,197	0,000	4,394

it.	$L(q_{7,16})$	$L(q_{3,17})$	$L(q_{13,17})$	$L(q_{3,18})$	$L(q_{6,18})$	$L(q_{11,18})$	$L(q_{9,19})$	$L(q_{10,19})$
1	-2,197	-2,197	-2,197	2,197	2,197	2,197	-2,197	-2,197
2	-6,592	0,000	0,000	6,592	2,197	2,197	-4,394	0,000
3	-2,197	-2,197	-2,197	8,789	4,394	6,592	-2,197	-2,197
4	-2,197	-4,394	2,197	8,789	4,394	2,197	-4,394	-0,000
it.	$L(r_{7,16})$	$L(r_{3,17})$	$L(r_{13,17})$	$L(r_{3,18})$	$L(r_{6,18})$	$L(r_{11,18})$	$L(r_{9,19})$	$L(r_{10,19})$
1	-2,197	2,197	2,197	-2,197	2,197	2,197	2,197	-2,197
2	0,000	-0,000	-0,000	-0,000	4,394	2,197	0,000	0,000
3	-2,197	4,394	-2,197	-2,197	2,197	4,394	2,197	-2,197
4	-4,394	0,000	-4,394	-0,000	-0,000	6,592	-0,000	-4,394

10.1.4 Odhady u SP algoritmu

Hodnoty pravděpodobností pro realizaci odhadu:

j	0	1	**2**	3	4	5	6	**7**	**8**	**9**
$p^{(1)}$	0,1	0,9	**0,9**	0,1	0,9	0,9	0,1	**0,1**	**0,9**	**0,9**
$Q_j^{(1)}\|_{\text{it.1}}$	0,01	0,90	0,48	0,01	0,76	0,99	0,01	0,91	**0,90**	0,07
$Q_j^{(1)}\|_{\text{it.2}}$	0,01	0,99	**0,92**	0,04	0,98	0,99	0,03	0,88	0,48	0,01
$Q_j^{(1)}\|_{\text{it.3}}$	0,01	0,99	0,24	4E-3	0,98	0,99	5E-3	0,98	0,35	5E-3
j	10	11	12	13	14	15	16	17	18	19
$p^{(1)}$	0,9	0,1	0,9	0,1	0,9	0,1	0,9	0,9	0,1	0,9
$Q_j^{(1)}\|_{\text{it.1}}$	0,99	**0,76**	0,96	0,04	0,86	4E-3	0,999	**0,39**	0,01	0,90
$Q_j^{(1)}\|_{\text{it.2}}$	0,97	0,03	0,998	3E-3	0,97	4E-3	0,96	0,90	6E-4	0,96
$Q_j^{(1)}\|_{\text{it.3}}$	0,80	6E-4	0,87	7E-4	0,995	3E-5	0,945	0,84	0,02	0,98

Odhady:

$$\hat{\mathbf{c}}|_{\text{it.0}} = [\,0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\,],$$
$$\hat{\mathbf{c}}|_{\text{it.1}} = [\,0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\,],$$
$$\hat{\mathbf{c}}|_{\text{it.2}} = [\,0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\,],$$
$$\hat{\mathbf{c}}|_{\text{it.3}} = [\,0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\,].$$

10.1.5 Odhady u min-sum algoritmu

Hodnoty pravděpodobností v logaritmickém zobrazení pro realizaci odhadu:

j	0	1	**2**	3	4	5	6	**7**	**8**	**9**
$L(p)$	2,20	-2,20	**-2,20**	2,20	-2,20	-2,20	2,20	**2,20**	**-2,20**	**-2,20**
$L(Q_j)\|_{\text{it.1}}$	6,59	-2,20	2,20	6,59	-2,20	-6,59	6,59	-4,39	**-2,20**	2,20
$L(Q_j)\|_{\text{it.2}}$	4,39	-6,59	**-4,39**	2,20	-6,59	-4,39	2,20	-2,20	2,20	4,39
$L(Q_j)\|_{\text{it.3}}$	6,59	-6,59	2,20	8,79	-2,20	-6,59	6,59	-2,20	**-2,20**	6,59
$L(Q_j)\|_{\text{it.4}}$	2,20	-4,39	2,20	4,39	-4,39	-2,20	4,39	-6,59	2,20	4,39
j	10	11	12	13	14	15	16	17	18	19
$L(p)$	-2,20	2,20	-2,20	2,20	-2,20	2,20	-2,20	-2,20	2,20	-2,20
$L(Q_j)\|_{\text{it.1}}$	-6,59	**-2,20**	-2,20	4,39	-2,20	4,39	-8,79	**2,20**	4,39	-2,20
$L(Q_j)\|_{\text{it.2}}$	-4,39	4,39	-8,79	4,39	-2,20	6,59	-2,20	-2,20	8,79	-2,20
$L(Q_j)\|_{\text{it.3}}$	-2,20	8,79	-2,20	6,59	-6,59	13,18	-4,39	**0,00**	6,59	-2,20
$L(Q_j)\|_{\text{it.4}}$	-2,20	4,39	-4,39	15,38	-8E-16	6,59	-2,20	-6,59	8,79	-6,59

Zde je vhodné ještě dodat, že dekódování u *min-sum* algoritmu se podařilo pouze díky zaokrouhlovacím chybám (viz bit č. 14). Zaokrouhlovací chyby se mohou lišit v závislosti na implementaci.

Odhady při použití min-sum algoritmu:

$$\hat{\mathbf{c}}|_{\text{it.0}} = [\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.1}} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.2}} = [\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.3}} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ ],$$
$$\hat{\mathbf{c}}|_{\text{it.4}} = [\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ ].$$

10.2 Kód použitý při měření spolehlivost přenosu

Uvedeno se složenými závorkami jako inicializované pole.

10.2.1 LDPC (96,48)

$\mathbb{H} \Leftrightarrow \mathcal{E} = \{\{i,j\}, i,j : \mathbb{H}_{ij} = 1\} =$ { {0,6}, {0,7}, {0,56}, {0,61}, {0,79}, {0,93}, {1,36}, {1,38}, {1,44}, {1,69}, {1,91}, {1,94}, {2,1}, {2,2}, {2,14}, {2,38}, {2,57}, {2,83}, {3,51}, {3,53}, {3,63}, {3,68}, {3,82}, {4,27}, {4,29}, {4,37}, {4,58}, {4,88}, {4,94}, {5,13}, {5,18}, {5,31}, {5,34}, {5,58}, {6,19}, {6,26}, {6,72}, {7,4}, {7,26}, {7,40}, {7,47}, {7,53}, {7,91}, {8,5}, {8,42}, {8,63}, {8,75}, {8,79}, {8,80}, {9,3}, {9,15}, {9,21}, {9,51}, {9,55}, {9,86}, {10,20}, {10,24}, {10,65}, {10,77}, {11,12}, {11,20}, {11,21}, {11,49}, {11,56}, {11,70}, {12,10}, {12,24}, {12,40}, {12,54}, {12,59}, {12,66}, {13,5}, {13,30}, {13,52}, {13,62}, {13,73}, {13,89}, {14,13}, {14,17}, {14,35}, {14,51}, {14,67}, {14,89}, {15,42}, {15,54}, {15,67}, {15,74}, {16,14}, {16,56}, {16,68}, {16,71}, {17,23}, {17,42}, {17,49}, {17,60}, {17,90}, {18,0}, {18,7}, {18,23}, {18,34}, {18,48}, {18,77}, {19,31}, {19,46}, {19,86}, {19,91}, {20,0}, {20,12}, {20,40}, {20,75}, {20,78}, {20,87}, {21,4}, {21,15}, {21,22}, {21,23}, {22,4}, {22,16}, {22,64}, {22,65}, {22,79}, {23,19}, {23,33}, {23,39}, {23,43}, {23,76}, {23,85}, {24,2}, {24,41}, {24,48}, {24,74}, {25,19}, {25,22}, {25,31}, {25,68}, {25,73}, {25,78}, {26,1}, {26,18}, {26,81}, {26,93}, {27,2}, {27,27}, {27,33}, {27,53}, {28,45}, {28,57}, {28,90}, {28,92}, {29,8}, {29,34}, {29,70}, {29,82}, {30,33}, {30,65}, {30,84}, {30,86}, {30,90}, {30,95}, {31,20}, {31,45}, {31,67}, {31,81}, {32,25}, {32,28}, {32,60}, {32,64}, {33,3}, {33,6}, {33,54}, {33,62}, {33,69}, {33,72}, {34,25}, {34,47}, {34,48}, {34,76}, {35,9}, {35,29}, {35,41}, {35,70}, {35,72}, {35,92}, {36,10}, {36,15}, {36,18}, {36,25}, {36,52}, {36,92}, {37,32}, {37,78}, {37,88}, {37,90}, {38,30}, {38,44}, {38,71}, {38,76}, {38,81}, {38,87}, {39,5}, {39,21}, {39,36}, {39,48}, {40,14}, {40,17}, {40,28}, {40,29}, {40,40}, {40,95}, {41,8}, {41,47}, {41,62}, {41,83}, {41,84}, {42,8}, {42,45}, {42,55}, {42,64}, {42,66}, {42,85}, {42,94}, {43,28}, {43,39}, {43,50}, {43,74}, {43,82}, {43,93}, {44,11}, {44,26}, {44,50}, {44,57}, {44,77}, {44,89}, {45,11}, {45,16}, {45,32}, {45,41}, {45,46}, {45,59}, {45,71}, {46,9}, {46,24}, {46,35}, {46,36}, {46,43}, {46,60}, {46,61}, {47,22}, {47,37}, {47,80}, {47,81}, {47,84} }

10.3 Některé další navržené kódy

10.3.1 LDPC (64,32)

$\mathbb{H} \Leftrightarrow \mathcal{E} = \{\{i,j\}, i,j : \mathbb{H}_{ij} = 1\} =$ { {0,59}, {0,44}, {0,31}, {0,18}, {0,14}, {0,4}, {1,30}, {1,25}, {1,24}, {1,14}, {1,13}, {1,11}, {2,55}, {2,48}, {2,47}, {2,19}, {2,16}, {2,15}, {3,57}, {3,33}, {3,32}, {3,17}, {3,16}, {3,10}, {4,47}, {4,45}, {4,44}, {4,37}, {4,9}, {4,0}, {5,60}, {5,53}, {5,49}, {5,38}, {5,37}, {5,7}, {6,49}, {6,31}, {6,29}, {6,27}, {6,26}, {6,9}, {7,60}, {7,56}, {7,52}, {7,44}, {7,42}, {7,17}, {8,56}, {8,47}, {8,31}, {8,30}, {8,7}, {8,1}, {9,55}, {9,49}, {9,34}, {9,25}, {9,6}, {9,5}, {10,56}, {10,43}, {10,40}, {10,38}, {10,13}, {10,5}, {11,53}, {11,39}, {11,32}, {11,24}, {11,22}, {11,15}, {12,58}, {12,54}, {12,48}, {12,38}, {12,11}, {12,1}, {13,50}, {13,36}, {13,34}, {13,32}, {13,13}, {13,0}, {14,61}, {14,48}, {14,44}, {14,29}, {14,13}, {14,12}, {15,61}, {15,57}, {15,41}, {15,5}, {15,2}, {15,1}, {16,46}, {16,41}, {16,15}, {16,14}, {16,9}, {16,7}, {17,57}, {17,35}, {17,34}, {17,28}, {17,26}, {17,14}, {18,62}, {18,37}, {18,20}, {18,18}, {18,16}, {18,12}, {19,43}, {19,32}, {19,27}, {19,21}, {19,11}, {19,4}, {20,62}, {20,45}, {20,42}, {20,39}, {20,35}, {20,11}, {21,52}, {21,46}, {21,43}, {21,30}, {21,26}, {21,12}, {22,55}, {22,52}, {22,45}, {22,38}, {22,23}, {22,4}, {23,63}, {23,58}, {23,25}, {23,17}, {23,15}, {23,3}, {24,60}, {24,46}, {24,22}, {24,8}, {24,1}, {24,0}, {25,62}, {25,61}, {25,54}, {25,50}, {25,30}, {25,27}, {26,40}, {26,28}, {26,27}, {26,24}, {26,20}, {26,3}, {27,53}, {27,45}, {27,31}, {27,21}, {27,16}, {27,3}, {28,56}, {28,51}, {28,10}, {28,9}, {28,8}, {28,6}, {29,54}, {29,33}, {29,29}, {29,23}, {29,22}, {29,21}, {30,59}, {30,54}, {30,28}, {30,19}, {30,8}, {30,2}, {31,59}, {31,50}, {31,43}, {31,22}, {31,17}, {31,6} }

10.3.2 LDPC (128,64)

$\mathbb{H} \Leftrightarrow \mathcal{E} = \{\{i,j\}, i,j : \mathbb{H}_{ij} = 1\} =$ { {0,127}, {0,105}, {0,88}, {0,83}, {0,74}, {0,60}, {1,114}, {1,111}, {1,107}, {1,101}, {1,29}, {1,14}, {2,124}, {2,103}, {2,95}, {2,52}, {2,32}, {2,4}, {3,113}, {3,108}, {3,55}, {3,46}, {3,34}, {3,0}, {4,98}, {4,80}, {4,63}, {4,49}, {4,44}, {4,37}, {5,109}, {5,99}, {5,97}, {5,65}, {5,61}, {5,23}, {6,121}, {6,112}, {6,110}, {6,48}, {6,27}, {6,11}, {7,121}, {7,98}, {7,87}, {7,40}, {7,35}, {7,28}, {8,102}, {8,72}, {8,68}, {8,54}, {8,30}, {8,24}, {9,91}, {9,88}, {9,58}, {9,53}, {9,43}, {9,29}, {10,125}, {10,90}, {10,83}, {10,65}, {10,64}, {10,53}, {11,123}, {11,102}, {11,96}, {11,83}, {11,67}, {11,37}, {12,111}, {12,100}, {12,95}, {12,94}, {12,87}, {12,83}, {13,119}, {13,94}, {13,63}, {13,33}, {13,11}, {13,8}, {14,126}, {14,117}, {14,105}, {14,28}, {14,25}, {14,18}, {15,120}, {15,118}, {15,114}, {15,110}, {15,67}, {15,4}, {16,115}, {16,69}, {16,66}, {16,57}, {16,45}, {16,44}, {17,71}, {17,48}, {17,43}, {17,42}, {17,24}, {17,1}, {18,116}, {18,111}, {18,41}, {18,30}, {18,19}, {18,16}, {19,113}, {19,101}, {19,86}, {19,77}, {19,31}, {19,10}, {20,107}, {20,89},

{20,75}, {20,50}, {20,38}, {20,15}, {21,121}, {21,117}, {21,104}, {21,91}, {21,81}, {21,34}, {22,86}, {22,73}, {22,42}, {22,39}, {22,23}, {22,22}, {23,125}, {23,92}, {23,86}, {23,51}, {23,32}, {23,9}, {24,122}, {24,115}, {24,74}, {24,54}, {24,52}, {24,21}, {25,108}, {25,97}, {25,72}, {25,63}, {25,20}, {25,2}, {26,64}, {26,49}, {26,42}, {26,40}, {26,25}, {26,13}, {27,123}, {27,81}, {27,64}, {27,27}, {27,19}, {27,10}, {28,106}, {28,87}, {28,82}, {28,61}, {28,46}, {28,36}, {29,107}, {29,91}, {29,56}, {29,45}, {29,42}, {29,0}, {30,84}, {30,79}, {30,45}, {30,35}, {30,12}, {30,2}, {31,124}, {31,115}, {31,109}, {31,106}, {31,26}, {31,24}, {32,85}, {32,75}, {32,61}, {32,33}, {32,31}, {32,13}, {33,119}, {33,102}, {33,101}, {33,76}, {33,66}, {33,5}, {34,126}, {34,115}, {34,90}, {34,71}, {34,20}, {34,14}, {35,112}, {35,104}, {35,103}, {35,72}, {35,29}, {35,13}, {36,123}, {36,93}, {36,76}, {36,59}, {36,52}, {36,0}, {37,98}, {37,70}, {37,65}, {37,55}, {37,5}, {37,4}, {38,116}, {38,79}, {38,67}, {38,61}, {38,21}, {38,18}, {39,96}, {39,55}, {39,40}, {39,12}, {39,11}, {39,7}, {40,118}, {40,99}, {40,92}, {40,44}, {40,7}, {40,6}, {41,127}, {41,113}, {41,89}, {41,69}, {41,62}, {41,30}, {42,94}, {42,92}, {42,85}, {42,68}, {42,35}, {42,14}, {43,108}, {43,95}, {43,78}, {43,66}, {43,56}, {43,25}, {44,93}, {44,73}, {44,57}, {44,34}, {44,32}, {44,15}, {45,97}, {45,46}, {45,44}, {45,29}, {45,22}, {45,18}, {46,112}, {46,109}, {46,105}, {46,80}, {46,76}, {46,15}, {47,123}, {47,100}, {47,84}, {47,71}, {47,70}, {47,51}, {48,125}, {48,72}, {48,69}, {48,59}, {48,47}, {48,28}, {49,112}, {49,62}, {49,58}, {49,54}, {49,25}, {49,6}, {50,88}, {50,82}, {50,56}, {50,47}, {50,31}, {50,7}, {51,106}, {51,91}, {51,73}, {51,70}, {51,69}, {51,33}, {52,79}, {52,62}, {52,48}, {52,32}, {52,22}, {52,17}, {53,124}, {53,90}, {53,58}, {53,41}, {53,39}, {53,8}, {54,126}, {54,118}, {54,93}, {54,87}, {54,50}, {54,43}, {55,120}, {55,109}, {55,81}, {55,62}, {55,56}, {55,3}, {56,121}, {56,77}, {56,26}, {56,20}, {56,16}, {56,6}, {57,117}, {57,77}, {57,65}, {57,38}, {57,17}, {57,1}, {58,126}, {58,102}, {58,89}, {58,82}, {58,27}, {58,23}, {59,116}, {59,68}, {59,57}, {59,53}, {59,17}, {59,11}, {60,60}, {60,59}, {60,40}, {60,36}, {60,14}, {60,3}, {61,122}, {61,81}, {61,50}, {61,49}, {61,39}, {61,12}, {62,96}, {62,93}, {62,85}, {62,78}, {62,30}, {62,22}, {63,80}, {63,78}, {63,51}, {63,36}, {63,17}, {63,13} }

10.3.3 LDPC (256,128)

$\mathbb{H} \Leftrightarrow \mathcal{E} = \{\,\{i,j\}, i,j : \mathbb{H}_{ij} = 1\,\} =$ { {0,255}, {0,204}, {0,197}, {0,191}, {0,116}, {0,16}, {1,225}, {1,196}, {1,127}, {1,68}, {1,43}, {1,2}, {2,200}, {2,182}, {2,149}, {2,72}, {2,41}, {2,39}, {3,235}, {3,211}, {3,173}, {3,152}, {3,107}, {3,15}, {4,225}, {4,174}, {4,161}, {4,90}, {4,48}, {4,22}, {5,200}, {5,192}, {5,190}, {5,188}, {5,27}, {5,23}, {6,250}, {6,241}, {6,240}, {6,157}, {6,92}, {6,77}, {7,230}, {7,149}, {7,142}, {7,40}, {7,35}, {7,21}, {8,229}, {8,227}, {8,218}, {8,188}, {8,113}, {8,46}, {9,255}, {9,220}, {9,139}, {9,132}, {9,119}, {9,49}, {10,187}, {10,169}, {10,122}, {10,51},

{10,33}, {10,29}, {11,180}, {11,146}, {11,109}, {11,52}, {11,38}, {11,28}, {12,201}, {12,151}, {12,108}, {12,36}, {12,22}, {12,0}, {13,206}, {13,176}, {13,129}, {13,85}, {13,47}, {13,27}, {14,150}, {14,120}, {14,95}, {14,89}, {14,40}, {14,13}, {15,201}, {15,163}, {15,105}, {15,83}, {15,30}, {15,5}, {16,234}, {16,207}, {16,159}, {16,128}, {16,120}, {16,70}, {17,236}, {17,227}, {17,209}, {17,186}, {17,75}, {17,53}, {18,211}, {18,185}, {18,168}, {18,96}, {18,61}, {18,34}, {19,252}, {19,194}, {19,192}, {19,184}, {19,158}, {19,6}, {20,220}, {20,207}, {20,190}, {20,108}, {20,71}, {20,67}, {21,160}, {21,58}, {21,44}, {21,41}, {21,29}, {21,13}, {22,222}, {22,160}, {22,155}, {22,121}, {22,66}, {22,23}, {23,249}, {23,222}, {23,184}, {23,151}, {23,138}, {23,1}, {24,186}, {24,185}, {24,180}, {24,156}, {24,114}, {24,1}, {25,191}, {25,80}, {25,45}, {25,19}, {25,3}, {25,1}, {26,252}, {26,231}, {26,203}, {26,167}, {26,81}, {26,29}, {27,203}, {27,195}, {27,143}, {27,123}, {27,91}, {27,22}, {28,221}, {28,204}, {28,172}, {28,135}, {28,117}, {28,36}, {29,157}, {29,139}, {29,127}, {29,118}, {29,83}, {29,25}, {30,215}, {30,161}, {30,158}, {30,89}, {30,85}, {30,45}, {31,215}, {31,172}, {31,113}, {31,103}, {31,57}, {31,21}, {32,253}, {32,157}, {32,148}, {32,131}, {32,40}, {32,20}, {33,195}, {33,155}, {33,100}, {33,96}, {33,78}, {33,56}, {34,246}, {34,243}, {34,226}, {34,147}, {34,129}, {34,93}, {35,218}, {35,210}, {35,199}, {35,91}, {35,74}, {35,24}, {36,160}, {36,142}, {36,115}, {36,102}, {36,55}, {36,36}, {37,232}, {37,196}, {37,107}, {37,69}, {37,54}, {37,44}, {38,165}, {38,128}, {38,82}, {38,75}, {38,22}, {38,7}, {39,224}, {39,173}, {39,158}, {39,100}, {39,42}, {39,25}, {40,183}, {40,173}, {40,68}, {40,63}, {40,32}, {40,26}, {41,219}, {41,205}, {41,136}, {41,113}, {41,54}, {41,10}, {42,234}, {42,200}, {42,134}, {42,119}, {42,99}, {42,15}, {43,243}, {43,175}, {43,104}, {43,88}, {43,76}, {43,19}, {44,230}, {44,222}, {44,221}, {44,164}, {44,124}, {44,11}, {45,220}, {45,166}, {45,111}, {45,62}, {45,46}, {45,13}, {46,176}, {46,133}, {46,86}, {46,61}, {46,59}, {46,43}, {47,228}, {47,224}, {47,217}, {47,185}, {47,142}, {47,90}, {48,238}, {48,233}, {48,225}, {48,103}, {48,88}, {48,58}, {49,216}, {49,211}, {49,141}, {49,106}, {49,82}, {49,16}, {50,150}, {50,136}, {50,135}, {50,134}, {50,126}, {50,9}, {51,234}, {51,223}, {51,172}, {51,138}, {51,44}, {51,8}, {52,241}, {52,215}, {52,177}, {52,145}, {52,30}, {52,12}, {53,214}, {53,165}, {53,122}, {53,80}, {53,28}, {53,2}, {54,207}, {54,179}, {54,176}, {54,79}, {54,54}, {54,32}, {55,245}, {55,243}, {55,87}, {55,30}, {55,28}, {55,11}, {56,249}, {56,229}, {56,198}, {56,193}, {56,110}, {56,85}, {57,247}, {57,224}, {57,214}, {57,145}, {57,95}, {57,47}, {58,188}, {58,187}, {58,146}, {58,98}, {58,45}, {58,34}, {59,235}, {59,197}, {59,177}, {59,167}, {59,27}, {59,20}, {60,254}, {60,242}, {60,237}, {60,145}, {60,24}, {60,16}, {61,197}, {61,193}, {61,159}, {61,125}, {61,102}, {61,52}, {62,162}, {62,159}, {62,141}, {62,101}, {62,88}, {62,6}, {63,216}, {63,172}, {63,121}, {63,80}, {63,67}, {63,25}, {64,231}, {64,228}, {64,179}, {64,116}, {64,104}, {64,14}, {65,254}, {65,233}, {65,205}, {65,73}, {65,50}, {65,42}, {66,225}, {66,221}, {66,192}, {66,93}, {66,92}, {66,62}, {67,193}, {67,140}, {67,112}, {67,74}, {67,49}, {67,44}, {68,208}, {68,203}, {68,112}, {68,89}, {68,76}, {68,71}, {69,215},

{69,180}, {69,179}, {69,62}, {69,39}, {69,17}, {70,236}, {70,223}, {70,189}, {70,181}, {70,162}, {70,152}, {71,252}, {71,236}, {71,226}, {71,132}, {71,65}, {71,57}, {72,204}, {72,188}, {72,181}, {72,127}, {72,79}, {72,4}, {73,169}, {73,131}, {73,128}, {73,73}, {73,38}, {73,31}, {74,239}, {74,144}, {74,140}, {74,123}, {74,9}, {74,4}, {75,171}, {75,159}, {75,144}, {75,139}, {75,56}, {75,29}, {76,212}, {76,170}, {76,165}, {76,154}, {76,88}, {76,23}, {77,235}, {77,212}, {77,123}, {77,87}, {77,84}, {77,62}, {78,236}, {78,224}, {78,137}, {78,125}, {78,11}, {78,3}, {79,254}, {79,246}, {79,231}, {79,229}, {79,201}, {79,150}, {80,208}, {80,142}, {80,141}, {80,138}, {80,59}, {80,53}, {81,161}, {81,116}, {81,94}, {81,74}, {81,60}, {81,5}, {82,208}, {82,173}, {82,147}, {82,124}, {82,116}, {82,41}, {83,237}, {83,223}, {83,156}, {83,144}, {83,108}, {83,98}, {84,219}, {84,143}, {84,121}, {84,101}, {84,34}, {84,12}, {85,232}, {85,146}, {85,130}, {85,126}, {85,66}, {85,6}, {86,238}, {86,109}, {86,71}, {86,61}, {86,60}, {86,7}, {87,249}, {87,212}, {87,119}, {87,105}, {87,33}, {87,32}, {88,242}, {88,194}, {88,147}, {88,131}, {88,121}, {88,46}, {89,232}, {89,178}, {89,164}, {89,122}, {89,96}, {89,46}, {90,250}, {90,237}, {90,160}, {90,109}, {90,68}, {90,3}, {91,238}, {91,187}, {91,124}, {91,97}, {91,84}, {91,70}, {92,218}, {92,163}, {92,71}, {92,48}, {92,35}, {92,9}, {93,138}, {93,123}, {93,65}, {93,60}, {93,31}, {93,2}, {94,248}, {94,170}, {94,129}, {94,117}, {94,99}, {94,56}, {95,253}, {95,189}, {95,185}, {95,99}, {95,94}, {95,58}, {96,237}, {96,226}, {96,193}, {96,178}, {96,135}, {96,90}, {97,202}, {97,190}, {97,168}, {97,162}, {97,87}, {97,40}, {98,213}, {98,181}, {98,154}, {98,148}, {98,93}, {98,37}, {99,223}, {99,195}, {99,157}, {99,104}, {99,102}, {99,33}, {100,245}, {100,240}, {100,203}, {100,198}, {100,126}, {100,8}, {101,251}, {101,248}, {101,152}, {101,151}, {101,130}, {101,64}, {102,253}, {102,196}, {102,182}, {102,85}, {102,84}, {102,78}, {103,250}, {103,247}, {103,219}, {103,117}, {103,87}, {103,14}, {104,191}, {104,134}, {104,81}, {104,63}, {104,37}, {104,0}, {105,202}, {105,201}, {105,158}, {105,133}, {105,31}, {105,18}, {106,200}, {106,137}, {106,114}, {106,77}, {106,26}, {106,7}, {107,227}, {107,213}, {107,177}, {107,69}, {107,51}, {107,18}, {108,240}, {108,217}, {108,129}, {108,105}, {108,52}, {108,4}, {109,217}, {109,171}, {109,97}, {109,67}, {109,18}, {109,10}, {110,242}, {110,213}, {110,117}, {110,49}, {110,17}, {110,7}, {111,254}, {111,122}, {111,115}, {111,112}, {111,86}, {111,57}, {112,209}, {112,202}, {112,199}, {112,136}, {112,80}, {112,78}, {113,251}, {113,250}, {113,210}, {113,70}, {113,65}, {113,37}, {114,196}, {114,180}, {114,132}, {114,95}, {114,23}, {114,0}, {115,197}, {115,189}, {115,166}, {115,76}, {115,68}, {115,10}, {116,239}, {116,175}, {116,131}, {116,125}, {116,8}, {116,0}, {117,176}, {117,149}, {117,130}, {117,111}, {117,50}, {117,19}, {118,175}, {118,174}, {118,167}, {118,78}, {118,59}, {118,49}, {119,226}, {119,195}, {119,166}, {119,163}, {119,141}, {119,73}, {120,238}, {120,206}, {120,153}, {120,148}, {120,140}, {120,15}, {121,198}, {121,174}, {121,147}, {121,109}, {121,83}, {121,69}, {122,248}, {122,233}, {122,216}, {122,164}, {122,133}, {122,20}, {123,244}, {123,137}, {123,118}, {123,84}, {123,81}, {123,66}, {124,244}, {124,163},

{124,115}, {124,110}, {124,75}, {124,14}, {125,219}, {125,167}, {125,153}, {125,120}, {125,5}, {125,1}, {126,245}, {126,178}, {126,114}, {126,37}, {126,35}, {126,25}, {127,182}, {127,152}, {127,143}, {127,139}, {127,103}, {127,28} }

10.3.4 LDPC (512,256)

$\mathbb{H} \Leftrightarrow \mathcal{E} = \{ \{i,j\}, i,j : \mathbb{H}_{ij} = 1 \} =$ { {0,422}, {0,195}, {0,149}, {0,123}, {0,100}, {0,47}, {1,508}, {1,469}, {1,433}, {1,386}, {1,335}, {1,305}, {2,455}, {2,410}, {2,388}, {2,332}, {2,276}, {2,199}, {3,493}, {3,485}, {3,372}, {3,312}, {3,249}, {3,161}, {4,500}, {4,487}, {4,344}, {4,202}, {4,147}, {4,52}, {5,450}, {5,303}, {5,241}, {5,235}, {5,213}, {5,9}, {6,509}, {6,405}, {6,340}, {6,227}, {6,131}, {6,119}, {7,414}, {7,320}, {7,256}, {7,218}, {7,187}, {7,186}, {8,498}, {8,473}, {8,456}, {8,288}, {8,278}, {8,264}, {9,410}, {9,359}, {9,261}, {9,183}, {9,157}, {9,58}, {10,400}, {10,244}, {10,242}, {10,202}, {10,73}, {10,47}, {11,405}, {11,385}, {11,298}, {11,229}, {11,207}, {11,186}, {12,492}, {12,483}, {12,422}, {12,399}, {12,319}, {12,300}, {13,457}, {13,412}, {13,374}, {13,363}, {13,327}, {13,149}, {14,414}, {14,383}, {14,277}, {14,258}, {14,203}, {14,41}, {15,377}, {15,354}, {15,353}, {15,239}, {15,165}, {15,25}, {16,351}, {16,291}, {16,237}, {16,126}, {16,88}, {16,42}, {17,432}, {17,342}, {17,189}, {17,188}, {17,140}, {17,86}, {18,355}, {18,272}, {18,232}, {18,35}, {18,23}, {18,18}, {19,404}, {19,329}, {19,198}, {19,112}, {19,57}, {19,50}, {19,13}, {20,335}, {20,265}, {20,143}, {20,110}, {20,12}, {20,5}, {21,343}, {21,317}, {21,145}, {21,115}, {21,83}, {21,71}, {22,451}, {22,386}, {22,336}, {22,209}, {22,160}, {22,101}, {23,425}, {23,392}, {23,273}, {23,239}, {23,36}, {23,17}, {24,499}, {24,415}, {24,308}, {24,233}, {24,201}, {24,40}, {25,432}, {25,365}, {25,297}, {25,163}, {25,113}, {25,55}, {26,388}, {26,293}, {26,282}, {26,179}, {26,131}, {26,97}, {27,408}, {27,350}, {27,330}, {27,323}, {27,186}, {27,60}, {28,493}, {28,364}, {28,309}, {28,296}, {28,181}, {28,1}, {29,316}, {29,252}, {29,165}, {29,89}, {29,18}, {29,15}, {30,502}, {30,415}, {30,335}, {30,190}, {30,178}, {30,118}, {31,346}, {31,298}, {31,196}, {31,124}, {31,95}, {31,64}, {32,378}, {32,349}, {32,217}, {32,201}, {32,159}, {32,70}, {32,53}, {33,468}, {33,213}, {33,203}, {33,94}, {33,65}, {33,36}, {34,497}, {34,485}, {34,428}, {34,415}, {34,355}, {34,325}, {35,438}, {35,195}, {35,168}, {35,79}, {35,3}, {35,2}, {36,501}, {36,471}, {36,315}, {36,225}, {36,156}, {36,90}, {37,499}, {37,359}, {37,324}, {37,322}, {37,320}, {37,104}, {38,492}, {38,448}, {38,352}, {38,227}, {38,194}, {38,61}, {39,358}, {39,356}, {39,326}, {39,260}, {39,135}, {39,53}, {40,407}, {40,402}, {40,283}, {40,255}, {40,193}, {40,192}, {41,463}, {41,441}, {41,415}, {41,333}, {41,92}, {41,31}, {42,429}, {42,370}, {42,216}, {42,163}, {42,137}, {42,122}, {43,401}, {43,398}, {43,233}, {43,114}, {43,65}, {43,0}, {44,507}, {44,402}, {44,375}, {44,289}, {44,175}, {44,34}, {45,482}, {45,433}, {45,363}, {45,301}, {45,249}, {45,84}, {46,412}, {46,378}, {46,302}, {46,268}, {46,254}, {46,68}, {47,471}, {47,227}, {47,192}, {47,142}, {47,94}, {47,23}, {48,344}, {48,330}, {48,273}, {48,103}, {48,87}, {48,75}, {49,231}, {49,206}, {49,156}, {49,144}, {49,93}, {49,70}, {50,484}, {50,449}, {50,409}, {50,380}, {50,112}, {50,92}, {51,295}, {51,292}, {51,282}, {51,247}, {51,242}, {51,96}, {52,449}, {52,394}, {52,285}, {52,271}, {52,161}, {52,150}, {53,354},

{53,320}, {53,272}, {53,268}, {53,265}, {53,195}, {54,477}, {54,455}, {54,389}, {54,355}, {54,266}, {54,244}, {55,379}, {55,339}, {55,228}, {55,132}, {55,106}, {55,20}, {56,468}, {56,310}, {56,170}, {56,166}, {56,84}, {56,68}, {57,419}, {57,417}, {57,370}, {57,362}, {57,321}, {57,136}, {58,488}, {58,445}, {58,379}, {58,264}, {58,203}, {58,167}, {59,461}, {59,433}, {59,326}, {59,151}, {59,123}, {59,99}, {60,356}, {60,337}, {60,127}, {60,104}, {60,68}, {60,59}, {61,499}, {61,487}, {61,382}, {61,338}, {61,174}, {61,152}, {61,90}, {62,397}, {62,369}, {62,348}, {62,310}, {62,136}, {62,109}, {63,353}, {63,168}, {63,154}, {63,67}, {63,38}, {63,33}, {64,489}, {64,299}, {64,246}, {64,224}, {64,182}, {64,63}, {65,445}, {65,371}, {65,315}, {65,210}, {65,101}, {65,71}, {66,511}, {66,496}, {66,298}, {66,230}, {66,176}, {66,138}, {67,509}, {67,407}, {67,278}, {67,268}, {67,154}, {67,45}, {68,485}, {68,442}, {68,426}, {68,236}, {68,87}, {68,51}, {69,455}, {69,381}, {69,212}, {69,140}, {69,102}, {69,24}, {70,484}, {70,350}, {70,325}, {70,221}, {70,219}, {70,183}, {71,476}, {71,453}, {71,438}, {71,362}, {71,261}, {71,201}, {72,447}, {72,353}, {72,286}, {72,214}, {72,159}, {72,49}, {73,460}, {73,416}, {73,361}, {73,272}, {73,231}, {73,173}, {74,406}, {74,339}, {74,186}, {74,184}, {74,111}, {74,61}, {75,486}, {75,473}, {75,262}, {75,193}, {75,84}, {75,81}, {76,502}, {76,431}, {76,392}, {76,263}, {76,209}, {76,63}, {77,436}, {77,419}, {77,313}, {77,260}, {77,254}, {77,187}, {78,433}, {78,411}, {78,331}, {78,284}, {78,39}, {78,18}, {79,481}, {79,315}, {79,219}, {79,138}, {79,116}, {79,30}, {80,312}, {80,290}, {80,205}, {80,158}, {80,114}, {80,5}, {81,409}, {81,405}, {81,269}, {81,265}, {81,234}, {81,182}, {82,488}, {82,372}, {82,113}, {82,88}, {82,78}, {82,2}, {83,435}, {83,412}, {83,294}, {83,257}, {83,234}, {83,57}, {84,489}, {84,429}, {84,339}, {84,307}, {84,301}, {84,66}, {85,495}, {85,469}, {85,328}, {85,54}, {85,7}, {85,6}, {86,350}, {86,334}, {86,306}, {86,302}, {86,275}, {86,164}, {87,437}, {87,303}, {87,232}, {87,127}, {87,80}, {87,27}, {88,457}, {88,454}, {88,324}, {88,255}, {88,77}, {88,16}, {89,460}, {89,403}, {89,382}, {89,313}, {89,304}, {89,32}, {90,431}, {90,410}, {90,292}, {90,173}, {90,37}, {90,14}, {91,435}, {91,386}, {91,344}, {91,289}, {91,129}, {91,88}, {92,424}, {92,366}, {92,300}, {92,275}, {92,176}, {92,106}, {93,384}, {93,308}, {93,258}, {93,243}, {93,226}, {93,211}, {94,506}, {94,377}, {94,331}, {94,279}, {94,126}, {94,92}, {95,496}, {95,441}, {95,360}, {95,309}, {95,268}, {95,212}, {96,457}, {96,384}, {96,147}, {96,113}, {96,21}, {96,17}, {97,483}, {97,437}, {97,340}, {97,314}, {97,238}, {97,4}, {98,472}, {98,391}, {98,244}, {98,222}, {98,214}, {98,34}, {99,409}, {99,381}, {99,364}, {99,307}, {99,139}, {99,125}, {100,488}, {100,458}, {100,290}, {100,287}, {100,262}, {100,35}, {101,401}, {101,286}, {101,164}, {101,151}, {101,141}, {101,37}, {102,420}, {102,407}, {102,376}, {102,93}, {102,48}, {102,47}, {103,498}, {103,492}, {103,475}, {103,144}, {103,52}, {103,25}, {104,322}, {104,284}, {104,269}, {104,230}, {104,109}, {104,46}, {105,380}, {105,370}, {105,354}, {105,119}, {105,44}, {105,32}, {106,490}, {106,220}, {106,212}, {106,104}, {106,33}, {106,11}, {107,501}, {107,463}, {107,342}, {107,336}, {107,332}, {107,77}, {108,462}, {108,144}, {108,120}, {108,111}, {108,83}, {108,29}, {109,478}, {109,391}, {109,290}, {109,185}, {109,147}, {109,111}, {110,374}, {110,373}, {110,340}, {110,183}, {110,171}, {110,155}, {111,473}, {111,461}, {111,258}, {111,134}, {111,108}, {111,31}, {112,459}, {112,423}, {112,395}, {112,88}, {112,8}, {112,6}, {113,457},

{113,348}, {113,329}, {113,95}, {113,40}, {113,4}, {114,427}, {114,187}, {114,156}, {114,110}, {114,74}, {114,43}, {115,446}, {115,295}, {115,225}, {115,198}, {115,103}, {115,80}, {116,388}, {116,373}, {116,284}, {116,41}, {116,22}, {116,20}, {117,465}, {117,411}, {117,356}, {117,219}, {117,143}, {117,102}, {118,494}, {118,443}, {118,422}, {118,131}, {118,121}, {118,91}, {119,455}, {119,217}, {119,209}, {119,62}, {119,61}, {119,4}, {120,469}, {120,431}, {120,398}, {120,191}, {120,166}, {120,44}, {121,352}, {121,330}, {121,285}, {121,208}, {121,157}, {121,115}, {122,450}, {122,396}, {122,352}, {122,346}, {122,267}, {122,256}, {123,474}, {123,444}, {123,248}, {123,194}, {123,162}, {123,100}, {124,510}, {124,418}, {124,368}, {124,303}, {124,245}, {124,228}, {125,452}, {125,403}, {125,358}, {125,248}, {125,241}, {125,234}, {126,467}, {126,338}, {126,281}, {126,210}, {126,85}, {126,79}, {127,436}, {127,276}, {127,72}, {127,70}, {127,45}, {127,8}, {128,494}, {128,430}, {128,407}, {128,346}, {128,129}, {128,1}, {129,385}, {129,294}, {129,79}, {129,69}, {129,22}, {129,12}, {130,493}, {130,334}, {130,263}, {130,226}, {130,86}, {130,27}, {131,423}, {131,422}, {131,349}, {131,137}, {131,63}, {131,0}, {132,313}, {132,266}, {132,226}, {132,179}, {132,155}, {132,7}, {133,295}, {133,261}, {133,254}, {133,139}, {133,33}, {133,15}, {134,281}, {134,192}, {134,172}, {134,105}, {134,102}, {134,13}, {135,465}, {135,240}, {135,236}, {135,218}, {135,213}, {135,134}, {136,423}, {136,366}, {136,318}, {136,257}, {136,67}, {136,9}, {137,462}, {137,376}, {137,322}, {137,321}, {137,160}, {137,82}, {138,435}, {138,215}, {138,180}, {138,162}, {138,141}, {138,62}, {139,474}, {139,454}, {139,333}, {139,184}, {139,166}, {139,139}, {140,409}, {140,406}, {140,390}, {140,276}, {140,223}, {140,100}, {141,231}, {141,228}, {141,223}, {141,205}, {141,105}, {141,67}, {142,428}, {142,282}, {142,256}, {142,238}, {142,158}, {142,112}, {143,480}, {143,477}, {143,197}, {143,194}, {143,122}, {143,64}, {144,439}, {144,418}, {144,341}, {144,174}, {144,170}, {144,99}, {145,470}, {145,374}, {145,271}, {145,197}, {145,89}, {145,19}, {146,424}, {146,397}, {146,367}, {146,304}, {146,253}, {146,118}, {147,343}, {147,314}, {147,298}, {147,286}, {147,190}, {147,133}, {148,508}, {148,287}, {148,221}, {148,217}, {148,57}, {148,25}, {149,475}, {149,446}, {149,185}, {149,69}, {149,63}, {149,42}, {150,425}, {150,399}, {150,362}, {150,299}, {150,179}, {150,162}, {151,444}, {151,408}, {151,237}, {151,72}, {151,43}, {151,38}, {152,451}, {152,397}, {152,223}, {152,75}, {152,65}, {152,22}, {153,504}, {153,503}, {153,479}, {153,408}, {153,301}, {153,89}, {154,461}, {154,394}, {154,344}, {154,227}, {154,140}, {154,38}, {155,444}, {155,431}, {155,363}, {155,317}, {155,51}, {155,46}, {156,474}, {156,459}, {156,402}, {156,400}, {156,378}, {156,240}, {157,484}, {157,417}, {157,349}, {157,288}, {157,270}, {157,5}, {158,427}, {158,305}, {158,155}, {158,128}, {158,60}, {158,8}, {159,395}, {159,384}, {159,309}, {159,234}, {159,171}, {159,103}, {160,500}, {160,316}, {160,248}, {160,176}, {160,55}, {160,13}, {161,510}, {161,396}, {161,395}, {161,202}, {161,117}, {161,44}, {162,502}, {162,448}, {162,418}, {162,321}, {162,279}, {162,172}, {163,310}, {163,280}, {163,274}, {163,182}, {163,178}, {163,78}, {164,493}, {164,483}, {164,481}, {164,473}, {164,289}, {164,173}, {165,394}, {165,391}, {165,215}, {165,169}, {165,60}, {165,26}, {166,436}, {166,306}, {166,251}, {166,246}, {166,189}, {166,2}, {167,396}, {167,302}, {167,297}, {167,262}, {167,120}, {167,10}, {168,501}, {168,430}, {168,213},

{168,112}, {168,109}, {168,56}, {169,351}, {169,313}, {169,245}, {169,192}, {169,146}, {169,71}, {170,495}, {170,471}, {170,452}, {170,239}, {170,121}, {170,66}, {171,483}, {171,376}, {171,294}, {171,253}, {171,36}, {171,6}, {172,434}, {172,371}, {172,309}, {172,275}, {172,222}, {172,206}, {173,361}, {173,279}, {173,214}, {173,148}, {173,21}, {173,8}, {174,468}, {174,330}, {174,146}, {174,125}, {174,93}, {174,28}, {175,367}, {175,331}, {175,207}, {175,167}, {175,73}, {175,58}, {176,224}, {176,167}, {176,122}, {176,107}, {176,80}, {176,54}, {177,482}, {177,395}, {177,389}, {177,190}, {177,189}, {177,56}, {178,509}, {178,401}, {178,357}, {178,160}, {178,116}, {178,15}, {179,460}, {179,318}, {179,184}, {179,161}, {179,130}, {179,102}, {180,439}, {180,287}, {180,224}, {180,208}, {180,86}, {180,51}, {181,505}, {181,436}, {181,393}, {181,297}, {181,271}, {181,76}, {182,382}, {182,349}, {182,229}, {182,193}, {182,150}, {182,54}, {183,464}, {183,440}, {183,406}, {183,274}, {183,247}, {183,142}, {184,470}, {184,429}, {184,250}, {184,96}, {184,30}, {184,9}, {185,479}, {185,389}, {185,360}, {185,334}, {185,142}, {185,59}, {186,443}, {186,438}, {186,334}, {186,147}, {186,134}, {186,82}, {187,385}, {187,322}, {187,180}, {187,173}, {187,113}, {187,94}, {188,479}, {188,442}, {188,283}, {188,117}, {188,107}, {188,62}, {189,345}, {189,266}, {189,216}, {189,148}, {189,85}, {189,46}, {190,503}, {190,294}, {190,251}, {190,230}, {190,158}, {190,108}, {191,467}, {191,417}, {191,360}, {191,39}, {191,36}, {191,14}, {192,484}, {192,375}, {192,341}, {192,235}, {192,210}, {192,97}, {193,480}, {193,404}, {193,250}, {193,200}, {193,28}, {193,22}, {194,403}, {194,247}, {194,154}, {194,134}, {194,115}, {194,39}, {195,456}, {195,413}, {195,126}, {195,124}, {195,66}, {195,44}, {196,490}, {196,458}, {196,293}, {196,174}, {196,61}, {196,37}, {197,476}, {197,448}, {197,347}, {197,175}, {197,169}, {197,109}, {198,420}, {198,171}, {198,141}, {198,136}, {198,98}, {198,35}, {199,507}, {199,496}, {199,440}, {199,208}, {199,200}, {199,74}, {199,21}, {200,476}, {200,472}, {200,427}, {200,297}, {200,130}, {200,14}, {201,499}, {201,307}, {201,306}, {201,169}, {201,145}, {201,55}, {202,459}, {202,456}, {202,383}, {202,281}, {202,259}, {202,232}, {203,500}, {203,311}, {203,291}, {203,282}, {203,222}, {203,16}, {204,393}, {204,288}, {204,220}, {204,199}, {204,75}, {204,34}, {205,432}, {205,377}, {205,357}, {205,280}, {205,218}, {205,197}, {206,489}, {206,420}, {206,267}, {206,233}, {206,198}, {206,77}, {207,453}, {207,371}, {207,348}, {207,323}, {207,239}, {207,31}, {208,389}, {208,258}, {208,177}, {208,157}, {208,91}, {208,90}, {209,414}, {209,300}, {209,270}, {209,249}, {209,247}, {209,101}, {210,487}, {210,318}, {210,314}, {210,252}, {210,243}, {210,153}, {211,498}, {211,388}, {211,361}, {211,139}, {211,123}, {211,56}, {212,472}, {212,466}, {212,413}, {212,357}, {212,319}, {212,105}, {213,489}, {213,481}, {213,391}, {213,59}, {213,7}, {213,3}, {214,454}, {214,296}, {214,159}, {214,29}, {214,23}, {214,20}, {215,442}, {215,376}, {215,365}, {215,260}, {215,52}, {215,41}, {216,508}, {216,206}, {216,124}, {216,33}, {216,26}, {216,17}, {217,491}, {217,240}, {217,205}, {217,150}, {217,125}, {217,25}, {218,505}, {218,430}, {218,292}, {218,170}, {218,153}, {218,137}, {219,441}, {219,421}, {219,205}, {219,180}, {219,76}, {219,27}, {220,494}, {220,421}, {220,327}, {220,43}, {220,35}, {220,26}, {221,426}, {221,269}, {221,96}, {221,48}, {221,42}, {221,10}, {222,368}, {222,220}, {222,194}, {222,181}, {222,108}, {222,18}, {223,491}, {223,392}, {223,390},

{223,316}, {223,267}, {223,68}, {224,462}, {224,379}, {224,308}, {224,283}, {224,241}, {224,161}, {225,421}, {225,406}, {225,241}, {225,225}, {225,204}, {225,49}, {226,505}, {226,425}, {226,264}, {226,219}, {226,169}, {226,118}, {227,470}, {227,312}, {227,277}, {227,273}, {227,188}, {227,29}, {228,347}, {228,332}, {228,259}, {228,151}, {228,78}, {228,32}, {229,275}, {229,204}, {229,200}, {229,199}, {229,137}, {229,116}, {230,419}, {230,327}, {230,235}, {230,178}, {230,83}, {230,24}, {231,387}, {231,324}, {231,215}, {231,177}, {231,132}, {231,87}, {232,494}, {232,216}, {232,198}, {232,132}, {232,118}, {232,81}, {233,485}, {233,381}, {233,357}, {233,337}, {233,167}, {233,16}, {234,443}, {234,252}, {234,245}, {234,196}, {234,188}, {234,141}, {235,511}, {235,356}, {235,296}, {235,283}, {235,237}, {235,119}, {236,486}, {236,299}, {236,296}, {236,128}, {236,71}, {236,40}, {237,478}, {237,326}, {237,319}, {237,145}, {237,122}, {237,98}, {238,402}, {238,106}, {238,69}, {238,40}, {238,19}, {238,11}, {239,495}, {239,432}, {239,343}, {239,276}, {239,171}, {239,11}, {240,504}, {240,416}, {240,369}, {240,341}, {240,300}, {240,185}, {241,462}, {241,393}, {241,367}, {241,168}, {241,155}, {241,50}, {242,491}, {242,451}, {242,411}, {242,246}, {242,196}, {242,156}, {243,466}, {243,447}, {243,387}, {243,242}, {243,181}, {243,143}, {244,510}, {244,377}, {244,340}, {244,215}, {244,135}, {244,0}, {245,380}, {245,311}, {245,310}, {245,217}, {245,115}, {245,30}, {246,410}, {246,257}, {246,197}, {246,128}, {246,117}, {246,5}, {247,504}, {247,464}, {247,434}, {247,172}, {247,135}, {247,120}, {248,475}, {248,394}, {248,368}, {248,345}, {248,336}, {248,274}, {249,351}, {249,320}, {249,221}, {249,204}, {249,130}, {249,95}, {250,507}, {250,491}, {250,359}, {250,328}, {250,172}, {250,149}, {251,372}, {251,369}, {251,255}, {251,250}, {251,191}, {251,133}, {252,304}, {252,211}, {252,143}, {252,111}, {252,51}, {252,45}, {253,498}, {253,387}, {253,375}, {253,267}, {253,253}, {253,72}, {254,497}, {254,447}, {254,445}, {254,246}, {254,184}, {254,57}, {255,506}, {255,468}, {255,107}, {255,106}, {255,90}, {255,15} }

Evidenční list

Souhlasím s tím, aby má diplomová práce byla půjčována v Univerzitní knihovně ZČU v Plzni.

Datum: Podpis:

Uživatel stvrzuje svým čitelným podpisem, že tuto diplomovou práci použil ke studijním účelům a prohlašuje, že ji uvede mezi použitými prameny.

Jméno a příjmení	Fakulta / Katedra	Datum	Podpis