Verslag Project Datacommunicatie

Dieter Decaestecker, Sander Demeester, Tom Naessens, Nick Van Haver

21 december 2012

Inhoudsopgave

1 Broncodering

Het grafische bestand dat wij zullen gebruiken voor het illustreren van de broncodering is het bestand 'alberteinstein.bmp'.

1.1 Opstellen Huffmancode

1.1.1 Relatieve frequenties

De matlab code voor deze opgave staat in het bestand `1.1.m` in de bijgeleverde files. Het resultaat hiervan is te zien in tabel 1. De bijbehorende Huffmancode die we met deze frequenties bekomen is terug te vinden in tabel 2 op pagina 4.

1.1.2 Resulterende Huffmanboom

In figuur 1 op pagina 5 staat de boom die we bekomen door de Huffmancode uit tabel 2 te visualiseren. Hier hebben we aan de takken naar de grootste van de twee probabiliteiten (de opwaardse tak) het 0 bit toegekend. De andere takken krijgt dus het bit 1 toegekend. Ter controle hebben we ook de probabiliteiten van de tussenliggende knopen opgenomen in de boom.

1.1.3 Gemiddeld aantal codebits

Het gemiddelde aantal codebits per bronsymbool berekenen we met volgende formule uit de cursus.

$$E[n] = \frac{1}{K} \sum_{i=1}^{N} p_i n_i \tag{1}$$

In deze formule is K het aantal bronsymbolen per macrosymbool, N het aantal verschillende macrosymbolen, p_i de kans om codewoord c_i te bekomen en n_i de lengte van het codewoord c_i. In ons geval hebben we dus dat K = 4, N = 16, p_i zijn de probabiliteiten uit tabel 1 en n_i de lengte van de codewoorden uit tabel 2. Als we deze formule uitwerken, bekomen we een gemiddeld aantal codebits $E[n]$ van afgerond 0.44 bits per bronsymbool. We zullen per pixel dus gemiddeld slechts 0.4 bits nodig hebben.

1.2 Implementatie `create_codebook`()

De implementatie van deze funtie is terug te vinden in het bestand `Source_Coding.m`. Deze functie maakt ook gebruik van het node-object, terug te vinden in `node.m`.

De functie is als volgt geïmplementeerd: Eerst wordt een array opgesteld van node-objecten, met elk zijn letter en frequentie. Daarna sorteren we deze array op frequentie en voegen we de twee laatste nodes samen door

Nummer	Macrosymbool	Abs. Freq.	Rel. Freq.
1	□ □ □ □	12478	0.407777777777778
2	□ □ □ ■	193	0.006307189542484
3	□ ■ □ □	172	0.005620915032680
4	□ ■ □ ■	227	0.007418300653595
5	□ □ ■ □	168	0.005490196078431
6	□ □ ■ ■	235	0.007679738562092
7	□ ■ ■ □	18	0.000588235294118
8	□ ■ ■ ■	147	0.004803921568627
9	■ □ □ □	194	0.006339869281046
10	■ □ □ ■	8	0.000261437908497
11	■ ■ □ □	269	0.008790849673203
12	■ ■ □ ■	157	0.005130718954248
13	■ □ ■ □	175	0.005718954248366
14	■ □ ■ ■	99	0.003235294117647
15	■ ■ ■ □	173	0.005653594771242
16	■ ■ ■ ■	15887	0.519183006535948

Tabel 1: Weergave nummers t.o.v. macrosymbolen samen met hun absolute en relative frequentie

Nummer	Huffmancode
1	10
2	110000
3	110011
4	11110
5	110100
6	11101
7	11011110
8	110110
9	11111
10	11011111
11	11100
12	110101
13	110001
14	1101110
15	110010
16	0

Tabel 2: Weergave nummers van de macrosymbolen met hun corresponderende Huffmancode

deze het linker- en rechterkind te maken van een oudernode. Deze oudernode wordt dan terug achteraan de array toegevoegd. Dit herhalen we tot er nog 1 element in deze array zit; de wortel.

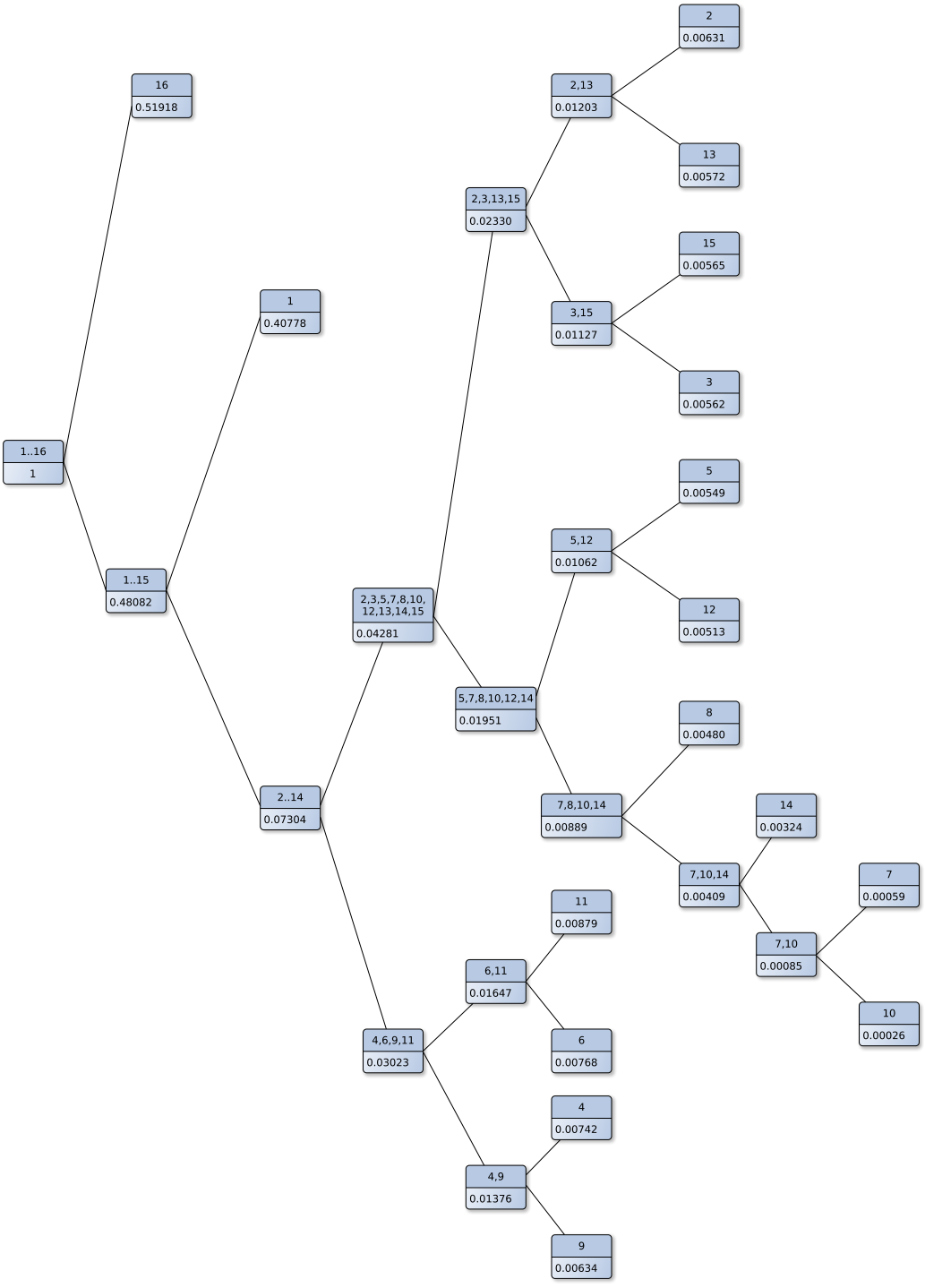
Daarna doorlopen we onze boom vanuit de wortel naar de bladeren en stellen we stapsgewijs de code op. Telkens we in een blad komen voegen we de huidige code toe aan de codewordsarray. Eenmaal we alle bladeren gehad hebben geven we deze array terug als resultaat.
Meer codegerelateerd commentaar is bijgevoegd in de codebestanden.

1.3 Entropie (K = 1..10)

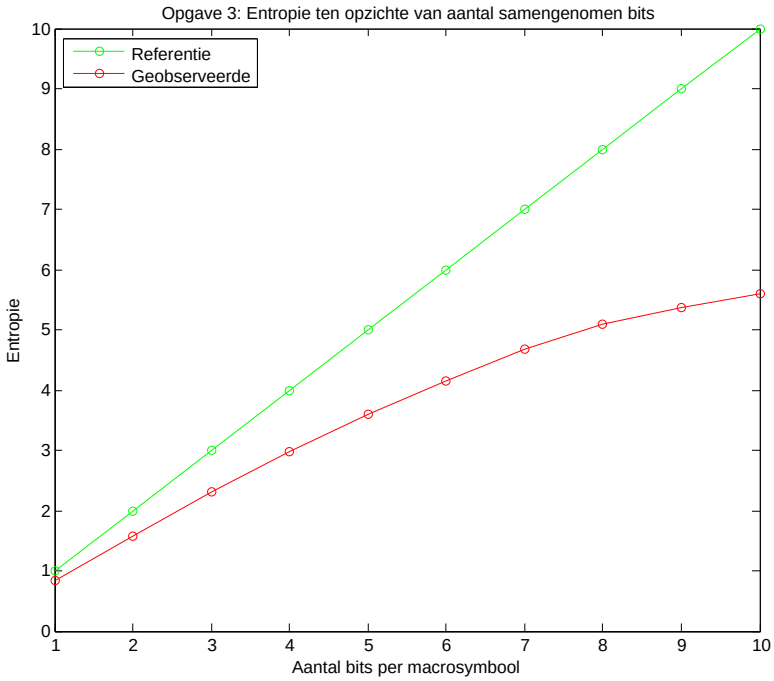
De entropie corresponderend met de frequenties van het bestand `'Xfile4.mat'` bekomen we met het script `1.3.m`. Hierin maken we gebruik van volgende formule uit de cursus:

$$H(X) = -\sum_{x \in A} p(x) log_2(p(x)) \tag{2}$$

Hier is X de toevalsveranderlijke, p(x) is de kans dat $X = x$, dus dat we een bepaald macrosymbool hebben. A is de verzameling van alle mogelijke macrosymbolen. De plot voor deze entropie voor een aantal K samengenomen bronssymbolen is te zien in figuur 2 op pagina 6.



Figuur 1: Huffmanboom voor Huffmancode uit tabel 2



Figuur 2: Entropie t.o.v. aantal samengenomen bits

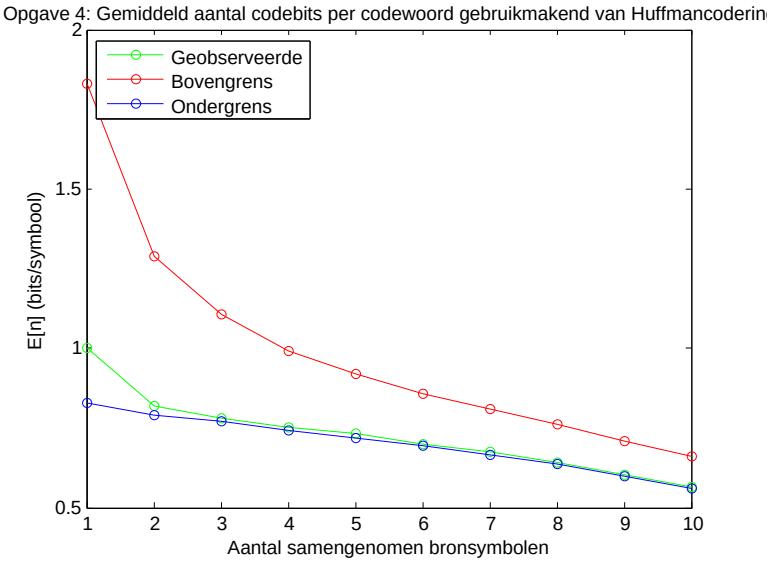
We zien dat de entropie voor de frequenties van de afbeelding eerder logaritmisch stijgt ten opzichte van de referentie die lineair stijgt. Hierdoor zal de entropie voor grotere waarden van K verder afwijken van de referentie. We zien geen duidelijke waarde die beter is dan de andere. We moeten dus een afweging maken van de complexiteit ten opzichte van de entropie. Bij te grote waarden voor K krijgen we immers het probleem dat de berekening te complex wordt.

1.4 Gemiddeld aantal codebits (K = 1..10)

Het gemiddelde aantal codebits per bronsymbool hebben we berekend aan de hand van script `'1.4.m'`. Als bovengrens en ondergrens hebben we respectievelijk de entropie $H(X)$ en $H(X) + \frac{1}{K}$. We zien dat het gemiddeld aantal codebits al bij macrosymbolen van grootte 3 al zeer dicht bij de ondergrens aanligt. Om een bestand optimaal te comprimeren zijn twee zaken belangrijk: enerzijds willen we na het comprimeren een zo klein mogelijk bestand en anderzijds willen we ook dat het en- en decoderen niet té veel tijd in beslag neemt. Het samennemen van meer bits voor de macrosymbolen zal leiden tot een kleiner bestand, maar zal ook leiden tot een langere decodeertijd.
Als we kiezen voor een zo klein mogelijk gecomprimeerd bestand dan kiezen we best voor een aantal samengenomen bits van 10. Zo krijgen we een op-

timaal gecomprimeerd bestand.
Als we echter willen kiezen voor een zo snel mogelijke decompressie kiezen we beter voor een lager aantal bits die toch nog steeds dicht bij de ondergrens aansluit zoals 3.
Uiteindelijk kiezen we voor een compromis tussen beide, dat iets zwaarder zal doorwegen naar een meer optimale compressie aangezien dit hier belangrijker is. We kiezen uiteindelijk om 8 bits samen te nemen per macrosymbool.

De compressiefactor komt overeen met de efficiëntie van de code. Deze efficiëntie kunnen we aflezen uit figuur 3 op pagina 7. Voor de compressie zal het bestand een grootte hebben van 55440 bits. Na de compressie zal dit bestand, met een optimale aantal samengenomen bits (namelijk 10 bits) slechts 31323 bits innemen. We hebben dus een compressiefactor van $\frac{31323}{55440} = 0.564989177\%$.
Met een suboptimaal aantal samengenomen bits (namelijk 8) verkrijgen we een compressiefactor van $\frac{35605}{55440} = 0.64222583\%$.



Figuur 3: Gemiddeld aantal codebits per codewoord bij Huffmancodering

1.5 Bespreking E[n] tegenover ondergrens

Zoals we zien op figuur 3 op pagina 7 zien we dat het gemiddeld aantal bits per bronsymbool al zeer snel in de buurt komt van de ondergrens. We zien dus dat we een hele efficiënte code hebben. Exact de ondergrens bereiken is theoretisch mogelijk; ten eerste moet onze gekozen K 'oneindig' zijn, wat in de praktijk overeenkomt met K gelijk aan de grootte van ht bestand.

Nummer	Huffmancode
1	10
2	111000
3	111001
4	11000
5	111010
6	11001
7	11111110
8	111011
9	11010
10	11111111
11	11011
12	111100
13	111101
14	1111110
15	111110
16	0

Tabel 3: Weergave nummers van de macrosymbolen met hun corresponderende Canonische Huffmancode

Ten tweede moeten de bronsymbolen statistisch onafhankelijk zijn en alle uitkomsten even waarschijnlijk zijn. Voor onze files is dit uiteraard niet correct en is de complexiteit van het algoritme ook ondoenbaar groot voor een te grote K.

1.6 Canonische Huffmancode

Om de Canonische Huffmancode op te stellen maken we gebruik van het algoritme in appendix A van de opgave. We sorteren dus de Huffmancodes eerst op lengte van de code en vervolgens op volgorde van de macrosymbolen. Aan het eerste element kennen we dan even veel nullen toe als de originele code lang is. De volgende code zal dan de vorige code met een verhoogd zijn met achteraan extra nullen tot de lengte van het originele codewoord. In tabel 3 op pagina 8 staat de bekomen Canonische Huffmancode.

1.7 Opstellen canonische vorm

Voor het alfabet A,B,C,D,E,F,G met codes van lengtes 5,3,3,1,4,5,3 bekomen we aan de hand van het algoritme uit appendix A de Canonische Huffmancode weergeven in tabel 4 op pagina 9. We merken dus dat de precieze originele Huffmancode geen belang heeft voor het opstellen van de Canonische Huffmancode, enkel de lengte van de code.

Nummer	Huffmancode
A	11110
B	100
C	101
D	0
E	1110
F	11111
G	110

Tabel 4: Macrosymbolen met hun corresponderende Canonische Huffmancode

1.8 Implementatie `create_canonical_codebook`()

De implementatie van deze funtie is terug te vinden in het bestand `Source_Coding.m`. Deze functie is als volgt geïmplementeerd: Eerst stellen we een matrix op, bestaande uit 2 kolommen: de eerste kolom is het codewoord, de tweede kolom bevat de lengte van de bijhorende huffman codes. We sorteren deze eerst stijgend op de tweede kolom, en daarna op de eerste kolom.

Daarna overlopen we deze matrix rijgewijs waarbij we telkens een teller bijhouden: de huidige code. Deze teller wordt in binaire vorm omgezet waarna er 'padding zeros' worden bijgevoegd zodat de lengte van de code overeen komt met de lengte van het origineel codewoord. Daarna wordt de code opgeslagen in zijn bijhorende plaats in de codewords array. Nu berekenen we de teller opnieuw aan de hand van de nieuwe, gepadde code. Dit herhalen we voor elke rij in de matrix.
Meer codegerelateerd commentaar is bijgevoegd in de codebestanden.

2 Kanaalcodering

2.1 Bespreking Hammingcode

Zoals in de opgave vermeld vertrekken we de cyclische (15,11) Hammingcode met generator veelterm $g(x) = x^4 + x^3 + 1$.

2.1.1 Minimale Hammingafstand

In de cursus vinden we dat als de generatorveelterm $x^4 + x^3 + 1$ primitief is, de minimale Hammingafstand 3 is. We zien dat, als we proberen de veelterm $x^l + 1$ te delen door onze generatorveelterm, de kleinste waarde l waarvoor we kunnen delen 15 is, wat onze bloklengte is. We hebben dus bewezen dat de generatorveelterm een primitieve veelterm is. Hieruit halen we dus dat de minimale Hammingafstand 3 is.

2.1.2 Foutdetecterend en foutcorrigerend vermogen

Het gegarandeerd foutdetecterend vermogen is de minimale Hammingafstand $d - 1$. Hier is dit dus 2. Het gegarandeerd foutcorrigerend vermogen bekomen we door het foutdetecterend vermogen te delen door 2. Hier is dit dus 1.

2.1.3 Checkveelterm

De checkveelterm halen we uit de vergelijking $x^{15} + 1 = g(x) \cdot h(x)$ met $g(x)$ de generatorveelterm en $h(x)$ de checkveelterm. Hieruit halen we dat $h(x) = \frac{x^{15}+1}{x^4+x^3+1}$. Als we dit uitrekenen komen we uit dat $h(x) = x^{11} + x^{10} + x^9 + x^8 + x^6 + x^4 + x^3 + 1$.

2.1.4 Generator- en checkmatrix

De generatormatrix G corresponderend met de generatorveelterm $g(x)$ ziet er als volgt uit:

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Deze generatormatrix zetten we vervolgens om naar zijn systematische vorm G_{sys} aan de hand van rij-operaties:

$$G_{sys} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Syndroom	Coset-leider
0000	000000000000000
1001	100000000000000
1101	010000000000000
1111	001000000000000
1110	000100000000000
0111	000010000000000
1010	000001000000000
0101	000000100000000
1011	000000010000000
1100	000000001000000
0110	000000000100000
0011	000000000010000
1000	000000000001000
0100	000000000000100
0010	000000000000010
0001	000000000000001

Tabel 5: Syndroomtabel aan de hand van H_{sys}

Alternatief konden we dit ook doen met de veeltermnotaties, zoals op pagina 66 in de cursus, wat hetzelfde resultaat oplevert.

Om de checkmatrix op te stellen gebruiken we de checkveelterm $h(x)$. Ook deze zetten we om naar een rijcanonieke vorm:

$$H_{sys} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

2.2 Opstellen en bespreking van syndroomtabel

Om de syndroomtabel op te stellen, transponeren we eerst de checkmatrix H_{sys}. Iedere rij van deze nieuwe matrix is een syndroom. De coset-leiders die corresponderen met deze syndromen bekomen we door te kijken naar de originele checkmatrix H_{sys}. We zetten in de coset-leiders een 1 op de plaatsen van de kolommen in H_{sys} die we samentellen om het syndroom te bekomen. Het resultaat is te zien in tabel 5 op pagina 11.
We veronderstellen nu transmissie over een binair symmetrisch kanaal met bitfoutprobabiliteit p, met de bovenstaande (15,11) code enkel ter foutcorrectie. De kans op een decodeerfout kunnen we dan met volgende formule bepalen:

$$Pr[decodeerfout] = 1 - \sum_{x \in E_{syndr}} p^{w(x)}(1-p)^{n-w(x)}$$

Hierbij is E_{syndr} de verzameling van alle in de syndroomtabel opgenomen waarden van de foutvector. In tabel 5 zien we dat we 1 foutvector hebben met gewicht 0 en 15 met gewicht 1. Als we dit invullen in de bovenstaande formule bekomen we:

$$Pr[decodeerfout] = 1 - ((1-p)^{15} + 15p(1-p)^{14})$$

Rekening houdend met het binomium van Newton is dit:

$$\begin{aligned} Pr[decodeerfout] &= \sum_{i=0}^{15} \binom{n}{i} p^i - ((1-p)^{15} + 15p(1-p)^{14}) \\ &= -14p^{15} + 195p^{14} - 1260p^{13} + 5005p^{12} - 13650p^{11} \\ &\quad +27027p^{10} - 40040p^9 + 45045p^8 - 38610p^7 + 25025p^6 \\ &\quad -12012p^5 + 4095p^4 - 910p^3 + 105p^2 \\ &\approx 105p^2 \text{ (voor } p \ll 1) \end{aligned}$$

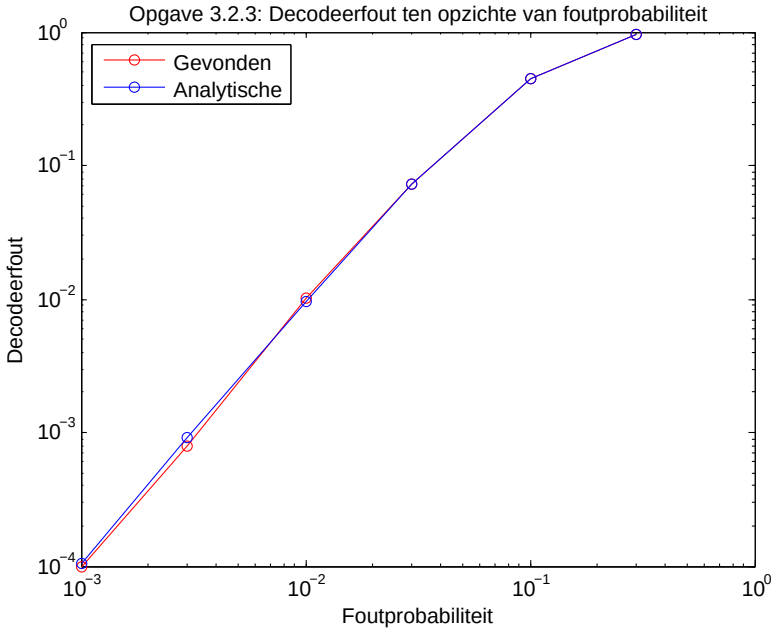
Voor zeer kleine p zal enkel de laagste macht in de sommatie significant zijn. De benadering voor de formule is dus $105p^2$.

2.3 Implementatie `Ham_Encode()` en `Ham_Decode()`

Omdat het encoderen van een informatiewoord een lineaire transformatie is met de systematische code, kunnen we in één keer de hele afbeelding encoderen. We zetten dus eerst de afbeelding om naar een matrix op iedere rij één informatiewoord. Deze vermeningvuldigen we met de codematrix om een matrix van codewoorden te bekomen. Als we deze matrix sequentiëel rij per rij overlopen, bekomen we de geëncodeerde afbeelding.

Om de simulaties uit te voeren, sturen we 25000 informatiewoorden van 11 bits over het binair symmetrische kanaal. We voegen de fouten die het kanaal zou introduceren toe a.d.h.v. een hulpfunctie die de output van de functie `Ham_Encode` verandert (deze flipt elke bit in de bitstring om met kans p), voordat we `Ham_Decode` oproepen. De resultaten hiervan zijn te zien in tabel 6. De resultaten zijn ook nog eens grafisch weergegeven in figuur 4 op pagina 13.

We zien zowel grafisch als in de tabel dat de berekende decodeerfoutkansen zeer dicht bij de praktisch bekomen decodeerfoutkansen liggen.



Figuur 4: Opgave 3.2.3: Decodeerfout ten opzichte van foutprobabiliteit

Kans p	Analytische decodeerfoutkans	Decodeerfoutkans simulatie
0.3	0.964732	0.9641
0.1	0.450957	0.4489
0.03	0.0729725	0.0733
0.01	0.00962977	0.0102
0.003	0.000920759	0.0008
0.001	0.000104094	0.0001

Tabel 6: Tabel met de verwachte en gevonden decodeerfoutkansen voor een informatiewoord voor vraag 3 uit kanaalcodering.

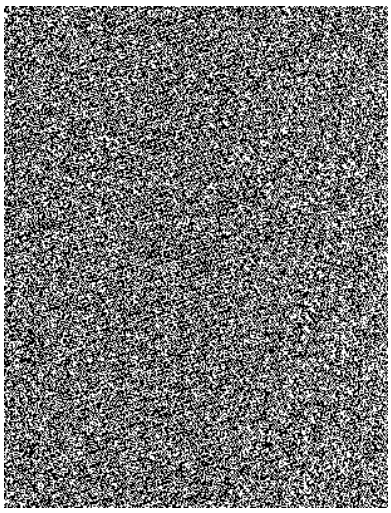
2.4 Productcode `Prod_encode()`

De productcode zal op ieder blok van l codewoorden `Ham_Encode` toepassen. Aan het resultaat hiervan zal een extra rij met voor iedere kolom een pariteitsbit toegevoegd worden.

De minimale hammingafstand van de productcode is het product van de minimale hammingafstanden van de beide foutcorrigerende codes gebruikt om de productcode te construeren. De eerste code heeft een minimale hammingafstand 3, de tweede code die enkel een pariteitsbit zal toevoegen heeft een minimale hammingafstand 2. We bekomen dus dat de minimale hammingafstand van de productcode 6 is.

2.5 Implementatie Decoder en oncorrigeerbare foutpatronen

Het implementeren van de decoder is helaas niet gelukt. Bij het testen krijgen we een afbeelding terug, maar bestaat enkel maar uit ruis. Deze is te zien op figuur 5 op pagina 14. Hoewel we lang op de fout hebben gezocht hebben we deze niet gevonden.



Figuur 5: Foute afbeelding, ge-encodeerd en gedecodeerd met de productcode

We kunnen twee gevallen beschouwen waar oncorrigeerbare foutpatronen voorkomen. Ten eerste is er het geval dat er een pariteitsbit omflipt en op dezelfde rij er een syndroom verschillend van nul is. In dat geval gaan we een bit verbeteren terwijl dit mogelijks niet nodig is.
Ten tweede is er het geval dat er een even aantal (maar meer dan 2) rijen een fout staat in dezelfde kolom waarvan de pariteitsbits nog juist zijn. In dat geval zal de pariteitscheck kloppen en zullen deze fouten niet verbeterd worden.

2.6 Bepaling decodeerfout bij productcode adhv simulaties

De kans dat er in 8 codewoorden minstens 1 decodeerfout optreedt is gegeven door volgende formule:

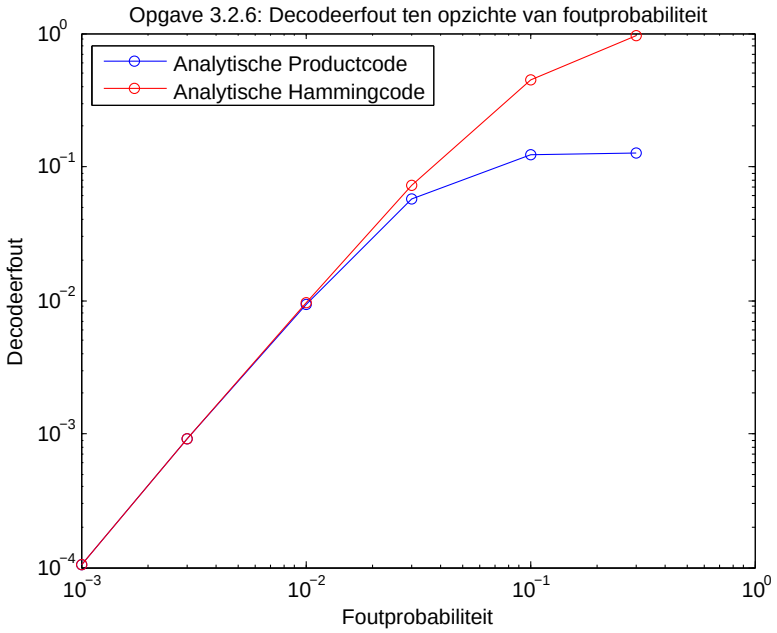
$$1 - Pr[\text{kans op fout in de hammingcode}]^8 = 1 - ((1-p)^{15} + 15p(1-p)^{14})^8$$

De bekomen resultaten zijn te zien in tabel 7 op pagina 15. Aangezien de functie `prod_decode` niet werkt hebben we de decodeerfoutkansen niet kunnen simuleren.
De analytische foutkansens hebben we nog eens visueel voorgesteld in figuur 6 op pagina 15. Hier hebben we de waarden uit de tabel voor de productcode

Kans p	Analytische decodeerfoutkans	Decodeerfoutkans simulatie
0.3	≈ 1	/
0.1	0.99174	/
0.03	0.45468	/
0.01	0.074491	/
0.003	0.0073424	/
0.001	0.00083245	/

Tabel 7: Tabel met de verwachte en gevonden decodeerfoutkansen voor een blok van 8 informatiewoorden voor vraag 6 uit kanaalcodering.

nog eens door 8 gedeeld om de foutkans per informatiewoord te krijgen in plaats van per blok. We zien duidelijk dat de productcode beter presteert dan de Hammingcode.



Figuur 6: Opgave 3.2.6: Analytische decodeerfout Hammingcode ten opzichte van analytische decodeerfout productcode

2.7 Verschillen tussen verstuurde afbeeldingen

De verkregen afbeeldingen zijn te zien in figuur 7 op pagina 7. De derde afbeelding hebben we niet kunnen opstellen aangezien het decoderen met de productcode niet werkt.

Gecodeerde transmissie Bij de niet gecodeerde transmissie zien we duidelijk dat er redelijk wat fouten voorkomen en dat deze fouten gelijkmatig verspreid zijn over de gehele afbeelding. Dit is te wijten aan de foutprobabiliteit van het kanaal.

Gecodeerde transmissie met Hammingcode Door het toepassen van de Hammingcode zal het aantal fouten beperkt worden. We zien ook dat ze gelokaliseerd zijn.

Gecodeerde transmissie met Productcode Door de grotere Hammingafstand van de productcode zullen er hier nog minder fouten optreden. We verwachten dus dat we een bijna foutloze afbeelding krijgen.

(a) Niet gecodeerde transmissie

(b) Gecodeerde transmissie met hammingcode

Figuur 7: Verkregen afbeeldingen

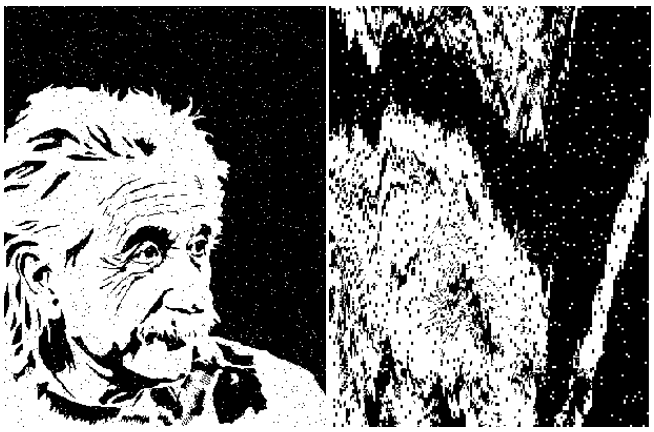
3 Volledig systeem

3.1 Ontvangen afbeeldingen

De verkregen afbeeldingen zijn te zien in figuur 8 op pagina 8. De code gebruikt om deze op te stellen is terug te vinden in het bestand `3.m`. Ook hier hebben we de laatste afbeelding niet kunnen opstellen.

3.2 Bespreking verschillen

Transmissie zonder codering en compressie We zien dat er een aantal fouten zijn opgetreden en dat deze fouten gelijkmatig verspreid zijn over de hele afbeelding. Dit is te wijten aan de foutprobabiliteit van het kanaal.



(a) Transmissie zonder codering en compressie

(b) Gecomprimeerde transmissie

Figuur 8: Verkregen afbeeldingen

Gecomprimeerde transmissie In deze afbeelding kunnen we nog steeds vanuit de verte de vorm van Einstein herkennen. Wat opvalt is dat de hele afbeelding er 'verschoven' uit ziet. Dit is als volgt te verklaren: Stel dat we een alfabet hebben van 0, 1 met als bijhorende codes 0 en 10. Als we als input nu de string '1100' krijgen, 4 bits dus, encoderen we dit als '101000'. Stel nu dat we deze ge-encodeerde string versturen over een kanaal en we krijgen aan de andere kant van dit kanaaal de string '101010' terug. Bij het decoderen krijgen we nu niet '1100' maar '111'. Dit zijn geen 4 bits zoals oorspronkelijk, maar 3.
Dit is exact hetzelfde wat gebeurt bij de afbeelding, maar dan op grotere schaal. Dit uit zich in de figuur als verschuiving.

Gecomprimeerde en gecodeerde transmissie met de productcode We verwachten hier een gelijkaardig verschuivingseffect te zien zoals in de vorige afbeelding maar veel minder extreem. Dit komt omdat de productcode de meeste fouten zal opvangen en corrigeren, maar nog niet alle.