Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

Coderingstheorie, een blik op verscheidene methoden en het Min-Sum algoritme

(Engelse titel: Codingtheory, a view on several methods and the Min-Sum algorithm)

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

Jorien Knipping

Delft, Nederland
December 2015

BSc verslag TECHNISCHE WISKUNDE

“Coderingstheorie, een blik op verscheidene methoden en het Min-Sum algoritme”

(Engelse titel: “Codingtheory, a view on several methods and the Min-Sum algorithm”)

JORIEN KNIPPING

Technische Universiteit Delft

Begeleider

Dr. J.A.M. de Groot

Overige commissieleden

Dr. ir. M. Keijzer

Dr. D.C. Gijswijt

December, 2015

Delft

SAMENVATTING

Coderen is een manier om fouten op te vangen, die optreden tijdens het versturen van data. In dit bachelor project blijft de coderingstheorie beperkt tot de codering en decodering van bits. Hierdoor wordt er gerekend in de ruimte $\mathbb{F}_2$, dus modulo 2. Daarnaast zal de data in woorden van een zeker aantal bits worden gehakt. De coderingstheorie bestaat uit drie algemene stappen het coderen van een woord, vervolgens het verzenden van dit woord door een kanaal, en daarna het decoderen van het ontvangen woord.

Er zijn talloos veel codes die verschillen in: Het aantal bits in een woord, efficiëntie en het fout-detecterend en fout-verbeterend vermogen. Voorbeelden van expliciete codes zijn de herhalingscode, de Hamming code en de LDPC code. Daarnaast kan een deel van de codes onderverdeeld worden in lineaire en cyclische codes. Bij lineaire en cyclische codes bestaan er directe en goed toepasbare coderings en decoderings methoden. Lineaire codering en decodering berust op twee matrix vermenigvuldigingen. De matrix verantwoordelijk voor de decodering heet de parity check matrix en wordt veelvuldig toegepast in andere decoderings methoden. Cyclische codering en decodering wordt uitgevoerd door middel van polynoom vermenigvuldiging. De Hamming code is een voorbeeld van een lineaire én cyclische code, dus hier kan beide coderingsmethoden op toegepast worden.

Na de codering wordt elk woord verstuurd door een kanaal. In dit kanaal kunnen fouten optreden om verschillende redenen. Meestal wordt deze fout gesimuleerd met behulp van de Gaussiche verdeling. Het is van belang om hoge prestaties te behalen in het verbeteren van deze opgetreden fouten. Daarnaast is het van belang dat de code efficiënt is, zodat er weinig energie nodig is voor het verzenden van de woorden. Hoe meer bits er toegevoegd moeten worden aan de woorden bij de codering hoe meer energie nodig is in de verzending.

Er wordt onderscheid gemaakt in twee verschillende manieren van decoderen, hard en soft decision decodering. Hard decision decodering is vaak makkelijk toe te passen maar gebruikt de informatie van de output uit een kanaal maar deels. Daarom is soft decision decodering ingebracht. Deze manier van decodering kan vaak ingewikkelder zijn in zijn toepassing. maar levert betere prestaties op. De optimale prestatie mogelijk voor een code wordt gegeven door de Shannon limiet. De Shannon limiet geeft aan dat met een optimale coderingsmethode het mogelijk is om een error kans van nul te benaderen met een erg lage energie besteding. Met soft decision decoderen benaderen de Turbo en LDPC codes de Shannon limiet.

In iteratieve decodering kan zowel hard als soft decision decodering toegepast worden. De kracht van iteratief decoderen is dat het decoderings proces kan onderverdeeld worden in subprocessen. Vaak wordt bij iteratief decoderen een graaf gebruikt, de zogenaamde Tanner graaf. Deze graaf wordt meestal ontwikkeld uit de parity check matrix, waardoor er zogenaamde check knopen ontstaan. Door gebruik te maken van deze check knopen kan het decoderen van de graaf onderverdeeld worden in het decoderen van elke check knoop.

Het Min-Sum algoritme is een decoderings algoritme, die iteratief decoderen en de Tanner graaf toepast. De Tanner graaf die gebruikt wordt in dit algoritme moet ontwikkeld zijn uit een parity check matrix. Het algoritme berust op het minimaliseren van kosten functies. Er is een stelling behorende bij dit algoritme die aangeeft dat het algoritme altijd convergeert voor een code met een Tanner graaf zonder cykels. De stelling zegt alleen iets over cykel vrije codes, toch wordt het Min-Sum algoritme veel toegepast in de praktijk op codes met cykels in de Tanner graaf. De LDPC en Turbo codes zijn voorbeelden van grafen met cykels in de Tanner graaf en waar het Min-Sum algoritme op toegepast wordt in de praktijk.

VOORWOORD

Voordat ik aan mijn bachelor Technische Wiskunde begon, interesseerde ik me op de middelbare school al erg voor wiskunde. Vandaar dat mijn profielwerkstuk over wiskunde ging, sterker nog de Hamming code. De Hamming code is een simpele en bekende coderingsmethode, die vroeger veel was toegepast in de telecommunicatie. Toen de tijd daar was om aan mijn bachelor eind project te beginnen, kwam ik al snel op het idee om de coderingstheorie verder te onderzoeken. Het leek me erg interessant om te zien wat ik met mijn nieuw opgedane wiskunde kennis zou kunnen bereiken als ik me nog een keer op het onderwerp coderingstheorie zou storten. Tot op heden werden er bij Technische Wiskunde nog geen vakken gegeven over coderingstheorie, echter Joost de Groot heeft daar dit jaar verandering in gebracht en zal in het derde kwartaal een keuze vak geven, die toegespitst is op coderingstheorie. Zo ontstond ook de match tussen Joost en mij, beide redelijk nieuw in het onderwerp coderingstheorie en Joost met de onderzoekers en wiskunde kennis om mij te helpen in mijn onderzoek. Het leek me een leuke uitdaging om het onbekende gebied van coderingstheorie te betreden, er waren meerdere hobbels op de weg, maar mede door deze hobbels heb ik mezelf verrijkt met veel kennis over coderingstheorie en onderzoeks-valkuilen.

Ik wil Joost de Groot bedanken voor alle tijd en kennis, waarmee hij mij heeft ondersteund. Als ik er niet meer uitkwam was Joost er altijd om me te helpen en samen met mij door talloze artikelen te zoeken naar een oplossingsrichting. Daarnaast wil ik Jos Weber bedanken voor het delen van zijn brede kennis over coderingstheorie met mij. Door de hulp van Jos Weber heb ik alle losse stukjes kennis aan elkaar kunnen verbinden. Ik heb erg veel plezier gehad in het opdoen van zoveel kennis over de coderingstheorie, en ik hoop daarom dat de lezer dit plezier zal delen door deze scriptie te lezen. Veel leesplezier toegewenst.

J. Knipping
Delft, December 2015

INHOUDSOPGAVE

0

INLEIDING

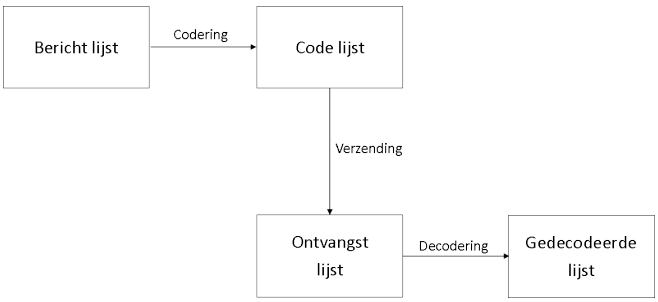
Coderingstheorie komt veel meer voor dan de meeste mensen misschien zouden denken. Een cd-speler zonder coderingstheorie zou bijvoorbeeld heel veel gruis laten horen tijdens het afspelen van cd's. Elk minuscuul krasje zou dan hoorbaar zijn. Daarnaast wordt coderingstheorie ook toegepast op de hardware van computers, in wifi, in telecommunicatie en nog veel meer.

Het onderwerp coderingstheorie is erg breed en het was daardoor lastig om een goede onderzoeksvraag op te stellen. Het onderzoek is uiteindelijk een literatuur studie geworden. In het begin wist ik nog nauwelijks iets over coderingstheorie en ben ik begonnen met de benodigde wiskunde achtergrond te verbeteren door toegepaste algebra boeken te lezen. Toen lag de focus nog zwaar bij cyclische codes. Echter, nadat ik de Tanner graaf tegen kwam in een artikel ontstond het idee om op het onderwerp iteratief decoderen te focussen. Al gauw kwamen er meerdere algoritmes tevoorschijn, waarvan het Min-Sum algoritme mij het meest aansprak. Er is dus niet een afgebakende onderzoeksvraag, maar deze scriptie geeft een introductie geven in verscheidene soorten codes, lichtelijk uitweiden over de prestaties van codes en vervolgens dieper ingaan op het iteratief decoderen waarbij ook een algoritme besproken zal worden.

1

INTRODUCTIE IN CODERINGSTHEORIE

Voordat er dieper op de wiskunde achter decoderen ingegaan wordt, is het van belang om te weten wat decoderen nu precies inhoudt. In dit verslag zal het beperkt blijven tot coderingsprocessen van bit getallen. Hieronder is een schematische weergave van het coderingsproces gegeven. Het proces begint met een lijst woorden, woorden bestaande uit enen en nullen, de bericht lijst. Vervolgens wordt deze lijst woorden gemodificeerd door middel van codering en ontstaat de zogeheten code lijst. Deze lijst zal door een kanaal naar de ontvanger verzonden worden. In het kanaal kunnen door ruis fouten optreden in het woord. De lijst die bij de ontvanger aankomt heet de ontvangst lijst en zal hoogstwaarschijnlijk verschillen van de code lijst door fouten die opgetreden zijn. Vervolgens wordt decodering toegepast om zoveel mogelijk fouten op te vangen en de lijst terug te krijgen naar een leesbare lijst, de gedecodeerde lijst. In de perfecte omstandigheden is de gedecodeerde lijst gelijk aan de bericht lijst.



Voorbeeld 1. *Hier is een voorbeeld gegeven van een codering. In het blauw de bits die toegevoegd zijn en in het rood de bits waar een fout is opgetreden. In de laatste kolom is een heel woord rood weergegeven, dit omdat dat woord fout verbeterd is. Hoe het woord gecodeerd en gedecodeerd is, is nog niet van belang.*

Bericht lijst	*Code lijst*	*Ontvangst lijst*	*Gedecodeerde lijst*
0101	*0101011*	*1101011*	*0101*
0000	*0000000*	*0000000*	*0000*
1100	*1100001*	*1101011*	*0101*
1001	*1001010*	*1001011*	*1001*

1.1. CODERING

Zoals al eerder vermeld, worden de woorden uit de bericht lijst gemodificeerd door middel van codering. Deze modificatie kan op meerdere manieren gebeuren. Er worden bits toegevoegd, dit komt voor bij verscheidene lineaire coderings methoden zoals bijvoorbeeld de Hamming Code. Alle bits worden aangepast, dit gebeurt bijvoorbeeld bij cyclische codering. Ten slotte kan het ook nog voorkomen dat er bits worden toegevoegd én originele bits worden aangepast, dit kan ook voorkomen bij onder andere cyclische codering. Als er bits worden toegevoegd aan het woord heten de toegevoegde bits parity bits.

1.2. KANAAL

Als de code lijst verzonden wordt dan wordt het via een kanaal getransporteerd. Dit is een ruim begrip en kan dus veel betekenen. Een voorbeeld van een kanaal is de kosmos. Als een ruimte sonde een bericht naar de aarde stuurt, zullen er fouten in het bericht op kunnen treden door ruis. Met 'fouten' wordt meestal het omklappen van bitjes bedoeld, dus een 1 wordt een 0 en andersom. Fouten kunnen ook anders geïnterpreteerd worden, maar dat komt pas in hoofdstuk 3 aanbod. Een ander voorbeeld is het lezen van een cd door een cd speler, als er krasjes zitten op de cd zal de ontvanger, in dit geval de cd speler, het fout lezen. Hoe slechter het kanaal, hoe meer fouten er zullen optreden bij de verzending.

1.3. DECODERING

Bij decodering zal getracht worden de fouten die opgetreden zijn in de verzending te verbeteren. In praktijk blijkt dat de simpelere coderings methoden maar 1 fout verbeterend zijn. Dus als er in een woord meerdere fouten optreden kan het zijn dat in het decoderings proces het woord fout verbeterd wordt. Het decoderen bestaat uit drie stappen, de fouten detecteren, verbeteren en vervolgens de woorden weer in normale vorm schrijven. Door deze laatste stap zullen de woorden weer dezelfde lengte krijgen als in de bericht lijst. Als alle fouten goed verbeterd zijn zal de gedecodeerde lijst er exact hetzelfde uitzien als de bericht lijst.

1.4. DE BINAIRE RUIMTE

Zoals al eerder vermeld wordt alleen de binaire codering beschouwd. Er wordt in de ruimte $\mathbb{F}_2$ gewerkt. Dit betekent dat er modulo 2 gewerkt zal worden. Dus $1+1=2 \mod 2 = 0$.

1.5. HAMMING AFSTAND EN FOUT VERBETEREN

Met de kwaliteit van een code wordt het vermogen van deze code om fouten te detecteren en of te verbeteren bedoeld. De kwaliteit van een code wordt beïnvloed door de minimale Hamming afstand d. Met de Hamming afstand wordt het verschil in bits tussen twee woorden bedoeld. De afstand voldoet aan de volgende drie eisen, met x en y worden woorden bedoeld:

- $d(x,y) \geq 0$ en $d(x,y) = 0 \iff x = y$ voor alle $x, y \in W$
- $d(x,y) = d(y,x)$ voor alle $x, y \in W$
- $d(x,y) \leq d(x,z) + d(z,y)$ voor alle $x, y, z \in W$

Met W wordt de ruimte van alle woorden aangeduid, dit wordt later in hoofdstuk 4 verder uitgelegd. Door deze drie eisen is de afstand d een metrische afstand op de ruimte W.

Voorbeeld 2 (Hamming afstand d)**.** *De Hamming afstand tussen* 1101001 *en* 1110101 *is gelijk aan 3. In het rood zijn de bits die verschillen weergegeven.*

Elke code C heeft een minimale afstand $d(C)$. Met de minimale afstand wordt de kleinste afstand van alle afstanden tussen alle woorden uit C bedoeld.

Zoals al eerder vernoemd is er een verband tussen de minimale afstand en het fout detecterend en verbeterend vermogen van een code. Wanneer er een code C is met minimale afstand $d(C)$ dan zal deze code $d(C)-1$ fouten detecteren. Het is lastiger om zo een uitspraak te doen over het fout verbeteren, want hier hangt het fout-verbeterend vermogen ook af van de toegepaste decoderings methode. Hieronder een voorbeeld, waarbij wel iets te zeggen is over het fout verbeterend vermogen van de code C.

Voorbeeld 3 (Nearest neighbour decoding)**.** *Stel nearest neighbour decoding wordt toegepast op een Code C. Deze decodering berust op het kiezen van het woord die de kleinste Hamming afstand heeft tot het 'foute woord'. Dan bestaat de volgende welbekende stelling over het fout-verbeterend vermogen van de code C met minimale afstand $d(C)$:*

- *Als de afstand $d(C)$ oneven is dan is de code C $\frac{1}{2}(d(C)-1)$ fout verbeterend.*
- *Als de afstand $d(C)$ even is dan is de code C $\frac{1}{2}(d(C)-2)$ fout verbeterend.*

Deze stelling geldt omdat foute woorden verbeterd worden naar het dichtstbijzijnde woord.

Dus in het voorbeeld hierboven hangt het fout-verbeterend vermogen geheel van $d(C)$ af.

2

SOORTEN CODES

Er zijn oneindig veel codes in de coderingstheorie. In dit hoofdstuk zal ik een paar bekende en belangrijke binaire codes toelichten.

2.1. HERHALINGSCODE

De meest eenvoudige coderings methode is de herhalingscode. Dit berust op het drie keer herhalen van een te versturen bit. Als bit 1 verstuurd dient te worden, codeert deze methode het woord eerst naar 111, waarna het verstuurd wordt. Een ontvangen bericht wordt al dan niet verbeterd. De verbetering is erg simpel, de bit die het meest voorkomt wordt gezien als de verstuurde bit. Deze manier van verbeteren wordt nearest neighbour decoding genoemd, zie ook voorbeeld 3. Bij nearest neighbour decoding wordt het woord verbeterd naar een woord uit de code lijst met de kleinste Hamming afstand tot het te verbeteren woord. Hieronder een voorbeeld.

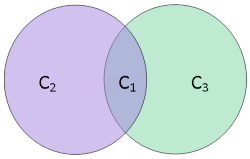
Voorbeeld 4 (Herhalingscode)**.** *Stel het woord* 0 *wordt gecodeerd met de herhalingscode, dan zal het gecodeerde woord gelijk zijn aan* 000*. Stel het woord* 010 *wordt ontvangen, dan zal dit verbeterd worden naar* 000 *en vervolgens gedecodeerd naar* 0*. Hier is de verbetering goed gegaan, maar stel het woord* 011 *wordt uit het kanaal ontvangen. Er zijn twee bits veranderd en dit woord wordt verbeterd naar* 111*, wat na decodering* 1 *wordt. Bij het laatst ontvangen woord zijn de fouten dus niet goed verbeterd.*

Bij deze methode krijgt elke bit twee parity bits, dit is niet erg efficiënt. Het versturen van een bit kost energie bij deze methode wordt de benodigde energie verdriedubbeld. Het aantal verschillende woorden die verstuurd kunnen worden is 2, 0 en 1. Daarnaast is deze code 2 fout-detecterend en 1 fout-verbeterend. Als er 2 fouten optreden in het kanaal dan detecteert deze methode het, echter de code zal niet in staat zijn het woord goed te verbeteren. Als er 1 fout optreedt, zal de code de fout detecteren en goed verbeteren.

2.2. HAMMING CODE

De Hamming code is een fout-verbeterende coderings methode vernoemd naar zijn uitvinder Richard Wesley Hamming. De Hamming code heeft vroeger veel toepassingen gekend in onder andere telecommunicatie. Tegenwoordig zijn er efficiëntere coderingsmethoden die gebruikt worden voor de telecommunicatie. Toch wordt ook de Hamming code vermeld, omdat de Hamming code berust op een simpel intuïtief idee. De decodering bij de Hamming code werkt via het nearest neighbour decoding. Daarnaast zal de Hamming code vaak gebruikt worden in voorbeelden.

De Hamming code voegt parity bits toe aan de originele woorden uit de bericht lijst. De code is te tekenen als een samenstelling van vlakke figuren. De bedoeling is dat de som van elk vlak gelijk is aan nul. In het algemeen geldt voor de Hamming codes hoe meer vlakke figuren hoe efficiënter en ingewikkelder de code wordt. Er bestaan oneindig veel soorten Hamming codes, er zal eerst verder uitgeweid worden over de simpelste Hamming code. Deze code is gelijk aan de herhalingscode eerder genoemd en wordt genoteerd als Hamming(3,1). Hierbij staat 1 voor de lengte van het woord in de berichtlijst en 3 voor de lengte van het woord na de codering, dus de lengte van het code woord. Hieronder is de code illustratief weergegeven:

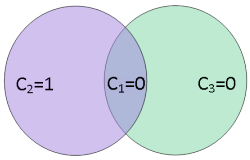


Het woord uit de bericht lijst heeft lengte 1 en wordt hierboven weergegeven als c_1. De parity bits zijn hier c_2 en c_3. c_1 kan alleen de waardes 0 en 1 aannemen want er wordt gewerkt in de ruimte $\mathbb{F}_2$ (de verzameling getallen modulo 2). Het doel is dat de som van beide cirkels gelijk aan 0 is. Dus stel $c_1 = 0$ dan moeten c_2 en c_3 beide de waarde 0 aannemen en als $c_1 = 1$ dan geldt $c_2 = 1$ en $c_3 = 1$. Vandaar dat deze vorm van de Hamming code ook wel de herhalingscode genoemd wordt. Het decoderen van een woord berust op het feit dat elke cirkel een som van nul moet zijn. Als dit niet geldt dan zijn er een of meerdere fouten opgetreden. De bit die in contact is met alleen de cirkels die som 1 hebben moet dan veranderd worden. Deze verbetering is daardoor ook op te vatten als een lineair stelsel. Hieronder is het stelsel voor Hamming(3,1) weergegeven.

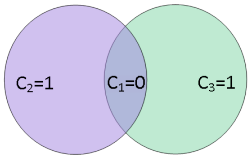
$$\begin{cases} c_1 & +c_2 & & = & 0 \\ c_1 & & +c_3 & = & 0 \end{cases} \tag{2.1}$$

Als er nu een van de sommen geen nul geeft kan met het lineaire stelsel gevonden worden welk bit veranderd moet worden. Later in lineaire codes zal een snellere manier van decoderen voor Hamming codes gegeven worden. Zoals al eerder vermeld bij de herhalingscode, is deze code zeer inefficiënt, 1-foutverbeterend en 2-foutdetecterend. Dit is te zien in het volgende voorbeeld.

Voorbeeld 5 (Hamming(3,1))**.** *Stel het woord* 0 *wordt gecodeerd met de Hamming(3,1) methode, dan zal het gecodeerde woord gelijk zijn aan* 000*. Stel het woord* 010 *wordt ontvangen uit het kanaal. Dan ziet de illustratieve weergave er als volgens uit:*

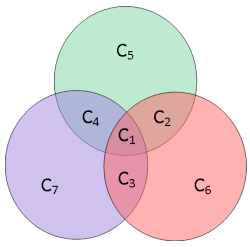


Alleen in de linker cirkel is een fout opgetreden. De bit c2 komt alleen in de linker cirkel voor en zal dus de bit zijn die veranderd moet worden. Dus na verbetering is het woord gelijk aan 000*, wat na decodering* 0 *wordt. Echter als* 011 *ontvangen wordt uit het kanaal zal de illustratieve weergave er anders uitzien:*



Nu is de fout opgetreden in beide cirkels. c1 is de bit die in beide cirkels zit dus zal veranderd moeten worden, dus 111 *is verkregen na verbetering. En gedecodeerd zal er* 1 *uitkomen, wat niet klopt.*

Een andere wat uitgebreidere Hamming code is de Hamming(7,4). Dit is de Hamming code die vaak in voorbeelden gebruikt gaat worden. Deze code werkt met woorden lengte 4 en maakt er code woorden van lengte 7 van. Het aantal woorden die met deze code verstuurd kunnen worden zijn $2^4 = 16$, al significant meer dan de Hamming(3,1). Daarnaast is deze code ook veel efficiënter per 4 bits worden 7 bits verstuurd, dus energieverbruik wordt ongeveer verdubbeld in plaats van verdriedubbeld. Deze coderingsmethode is schematisch weer te geven door middel van 3 cirkels.



De bits c_1 tot en met c_4 vormen de woorden uit de bericht lijst en de bits c_1 tot en met c_7 de woorden uit de codelijst. Dus zijn de bits c_5 tot en met c_7 de parity bits. De minimale afstand tussen de woorden uit deze code is gelijk aan drie. Hieronder een voorbeeld met decoderen via cirkels.

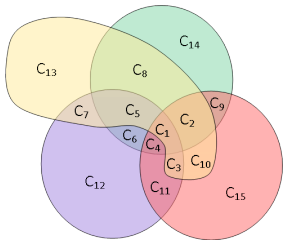
Voorbeeld 6 (Hamming(7,4))**.** *Stel het woord* 0101 *wordt gecodeerd met de Hamming(7,4) methode, dan zal het gecodeerde woord gelijk zijn aan* 0101011.

Ook bij deze code kan de keuze van deze drie parity bits opgevat worden als een lineair stelsel.

$$\begin{cases} c_1 & +c_2 & & +c_4 & +c_5 & & & = & 0 \\ c_1 & +c_2 & +c_3 & & & +c_6 & & = & 0 \\ & & & & & & & & \\ c_1 & & +c_3 & +c_4 & & & +c_7 & = & 0 \end{cases} \tag{2.2}$$

Als er een fout opgetreden is, wordt afhankelijk van welke sommen niet gelijk aan nul zijn 1 bit verbeterd.

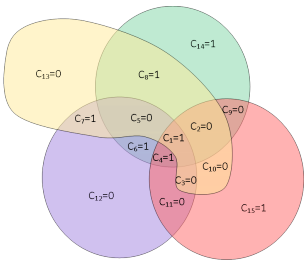
Naarmate er grotere Hamming codes worden beschouwd zal de complexiteit toenemen. Als de Hamming(15,11) behandeld wordt, is dit te zien in de toename van complexiteit in de illustratieve weergave:



Deze methode heeft woorden lengte 11 en de bits c_{12} tot en met c_{15} zijn de parity bits. Dusdanig is het aantal woorden die te versturen zijn door middel van deze codering gelijk aan $2^{11} = 2048$, op deze manier is veel meer informatie te versturen en de efficiëntie van deze code is ook veel groter aangezien er per 11 bits 4 parity bits toegevoegd worden. Hieronder is het bijbehorende lineaire systeem gegeven.

$$\begin{cases} c_1+c_2 & & +c_4 & +c_5+c_6 & & +c_8 & +c_9 & & & & +c_{14} & & = 0 \\ c_1+c_2 & +c_3+c_4 & & & & & +c_9+c_{10} & +c_{11} & & & & +c_{15} & = 0 \\ c_1 & +c_3+c_4 & & +c_5+c_6 & +c_7 & & & +c_{11}+c_{12} & & & & & = 0 \\ c_1+c_2 & +c_3 & & +c_5 & +c_7+c_8 & & +c_{10} & & +c_{12} & +c_{13} & & & = 0 \end{cases} \tag{2.3}$$

Voorbeeld 7 (Hamming(15,11))**.** *Stel het woord* 11010111000 *wordt gecodeerd met de Hamming(15,11) methode, dan zal het gecodeerde woord gelijk zijn aan* 110101110000011.
Stel het woord 100101110000011 *wordt ontvangen uit het kanaal. Dan ziet de illustratieve weergave er als volgens uit:*

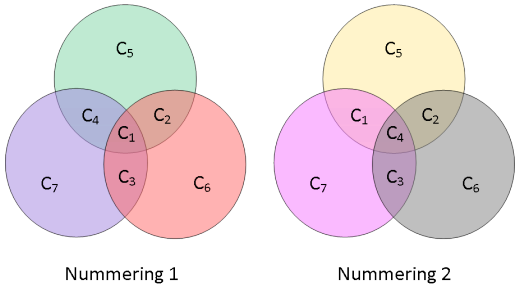


Als van elk vlak de som genomen wordt, blijkt dat de groene, gele en rode vlak som 1 hebben. Hieruit volgt dat c2 aangepast moet worden, deze komt namelijk in die drie vlakken voor en niet in het paarse vlak. Na aanpassing zal het woord dus 110101110000011 *zijn. Wat na decodering* 11010111000 *wordt.*

2.2.1. NUMMERING

Bij de meeste codes is de codering onder andere afhankelijk van de keuze van de nummering van de bits. Alleen bij cyclische codes is er geen afhankelijkheid van de nummering. Er moet dus erg goed gelet worden op de nummering bij het coderen en decoderen, hieronder is een voorbeeld van het verschil in codering gegeven.

Voorbeeld 8 (Twee verschillende nummeringen voor Hamming(7,4))**.** *Hieronder zijn twee verschillende nummeringen weergegeven.*



Stel het woord 0101 *wordt gecodeerd. Volgens nummering 1 zal er dan het woord* 0101011 *uitkomen, terwijl er bij nummering 2 het woord* 0101001 *verkregen wordt. Dit zijn duidelijk twee verschillende woorden. Dus elke nummering heeft een eigen specifieke code lijst.*

2.3. LINEAIRE CODES

Er zijn coderings methodes die lineair zijn. Deze codering en decodering zijn daardoor op te vatten als een matrix vermenigvuldiging. Eerst zullen de woorden uit de bericht lijst vermenigvuldigd worden met een zogenaamde generator matrix, waarna de code lijst verkregen is. Na de verzending door een kanaal zullen er fouten in de ontvangst lijst optreden. Vervolgens zal geprobeerd worden deze fouten te detecteren door middel van een nieuwe matrix vermenigvuldiging. Dit keer met de parity check matrix, deze zal de fouten detecteren en indiceren welke fout opgetreden is. In de volgende paragrafen zal er verder ingegaan worden op deze twee matrixen. Meer informatie over het lineair decoderen is te vinden in "The ABCs of Linear Block Codes"[1].

2.3.1. GENERATOR MATRIX

De matrix waarmee een woord uit de bericht lijst vermenigvuldigd moet worden om een woord uit de code lijst te verkrijgen heet de generator matrix. Eenzelfde generator matrix wordt gebruikt om alle bericht woorden om te zetten in code woorden. Noem k de lengte van een woord uit de bericht lijst en n de lengte van een woord uit de code lijst, de rang van de generator matrix moet gelijk zijn aan k. Meestal heeft de matrix evenveel kolommen als zijn rang, dit omdat de matrix vermenigvuldiging dan minder werk is. Definieer $\underline{a}$ als een woord uit de bericht lijst en $\underline{c}$ als een woord uit de codelijst, de woorden zijn kolom vectoren en de $n \times k$ generator matrix G ziet er als volgt uit:

$$G = \begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} & \cdots & g_{1,k} \\ g_{2,1} & g_{2,2} & g_{2,3} & \cdots & g_{2,k} \\ g_{3,1} & g_{3,2} & g_{3,3} & \cdots & g_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n,1} & g_{n,2} & g_{n,3} & \cdots & g_{n,k} \end{bmatrix}$$

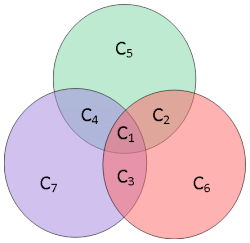
Dus de vermenigvuldiging van G met het woord $\underline{a}$ uit de berichtlijst, zal er zo uitzien:

$$\begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} & \cdots & g_{1,k} \\ g_{2,1} & g_{2,2} & g_{2,3} & \cdots & g_{2,k} \\ g_{3,1} & g_{3,2} & g_{3,3} & \cdots & g_{3,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n,1} & g_{n,2} & g_{n,3} & \cdots & g_{n,k} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix}$$

Hierbij is de generator matrix G afhankelijk van de gebruikte coderings methode. Sterker nog de kolommen van de generator matrix vormen een base van woorden voor de code. Het aantal kolommen van de generator matrix is dus gelijk aan het aantal woorden in de basis van de code. Vaak worden bij het coderen van een woord alleen parity bits toegevoegd. Als een coderings methode berust op het toevoegen van parity bits dan is er gelijk al meer informatie over de generator matrix G. Omdat k de lengte van een woord uit de bericht lijst is en n de lengte van een woord uit de code lijst, zijn er $n-k$ parity bits. Omdat er bits gelijk blijven zal G een samenstelling worden van de $k \times k$ identiteits matrix I_k en een zogeheten $(n-k) \times k$ parity submatrix P, $G = \begin{bmatrix} I_k \\ P \end{bmatrix}$. Nu is alleen nog P afhankelijk van de gebruikte coderings methode. De vermenigvuldiging van G en $\underline{a}$ zal er als volgt uitzien, waarbij de toegevoegde bits worden aangeduid met c_i:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,k} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,k} \\ p_{3,1} & p_{3,2} & p_{3,3} & \cdots & p_{3,k} \\ \vdots & \vdots & \ddots & \vdots & \\ p_{n-k,1} & p_{n-k,2} & p_{n-k,3} & \cdots & p_{n-k,k} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ c_{k+1} \\ \vdots \\ c_n \end{bmatrix} = \underline{c}$$

Voorbeeld 9 (Hamming(7,4))**.** *Om het bovenstaande te illustreren zal hier een voorbeeld gegeven worden door middel van de Hamming(7,4). De parity submatrix is afhankelijk van de keuze van de nummering van de bits. De generator matrix, die hier gemaakt wordt berust op dezelfde nummering als weergegeven in de afbeelding hieronder.*



De som in elke cirkel moet gelijk aan 0 zijn dus $c_5 = c_1 + c_2 + c_4$, $c_6 = c_1 + c_2 + c_3$ *en* $c_7 = c_1 + c_3 + c_4$. *Dit kan vertaald worden in de parity submatrix door de bits in de som gelijk aan 1 te stellen. Hieronder is de generator matrix G behorende bij deze Hamming code te zien:*

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} I_k \\ \left.\vphantom{\begin{matrix}1\\1\\1\end{matrix}}\right\} P \end{matrix}$$

coderen van het woord $[0101]^T$ *uit de bericht lijst gaat nu als volgt:*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Dus uit de vermenigvuldiging met de generator matrix volgt $[0101]^T \rightarrow [0101011]^T$, *waarbij in het blauw de parity bits zijn weergegeven. Dit kan vervolgens gedaan worden voor alle woorden uit de bericht lijst.*

2.3.2. PARITY CHECK MATRIX

Om de woorden uit de ontvangst lijst te decoderen, worden deze vermenigvuldigd met de zogenaamde parity check matrix H. Voor elke $n \times k$ generator matrix G kan er een $n \times (n-k)$ parity check matrix H gemaakt worden. Als H zo gemaakt wordt dat de kolommen van G en de kolommen van H orthogonaal zijn, dan zal er gelden $H^T G = 0$. Omdat alle woorden $\underline{c}$ uit de code lijst een lineaire combinatie van de kolommen van G zijn zal er ook gelden dat $H^T \underline{c} = \underline{0}$. Door deze eigenschap is het volgende te zeggen over een woord $\underline{r}$ uit de ontvangst lijst: Als $H^T \underline{r} = \underline{0}$ dan is er geen detecteerbare fout opgetreden, is dit ongelijk aan $\underline{0}$ dan is er wel een fout gedetecteerd en zal er geprobeerd worden om de fout te verbeteren. De waarde die uit de vermenigvuldiging met de parity check matrix volgt heet het syndroom S van een woord, dus $S = H^T \underline{r}$. Er zal later verder ingegaan worden op het syndroom, eerst zal de matrix H nader gespecificeerd worden.

Als de generator matrix van de vorm is zoals eerder beschreven dus $G = \begin{bmatrix} I_k \\ P \end{bmatrix}$, dan is de parity check matrix ook erg makkelijk te bepalen. Omdat er moet gelden $H^T G = 0$, is H een samenstelling van de $(n-k) \times (n-k)$ identiteits matrix I_{n-k} en de getransponeerde van de parity submatix P^T, dus $H = \begin{bmatrix} P^T \\ I_{n-k} \end{bmatrix}$. De rang van H

moet gelijk zijn aan $n-k$ en zal er dus als volgt uitzien, waarbij de $p_{i,j}$ al bekend zijn vanuit de generator Matrix:

$$H = \begin{bmatrix} p_{1,1} & p_{2,1} & p_{3,1} & \cdots & p_{n-k,1} \\ p_{1,2} & p_{2,2} & p_{3,2} & \cdots & p_{n-k,2} \\ p_{1,3} & p_{2,3} & p_{3,3} & \cdots & p_{n-k,3} \\ \vdots & \vdots & \ddots & \vdots & \\ p_{1,k} & p_{2,k} & p_{3,k} & \cdots & p_{n-k,k} \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Nu zal er wat dieper ingegaan worden op hoe een gedetecteerde fout verbeterd kan worden. Het ontvangen woord wordt genoteerd als $\underline{r}$. De nulruimte van H^T is gelijk aan de code C, $Nul(H^T) = \{\underline{r} | H^T \underline{r} = \underline{0}\}$. Dus als er geen fouten in een woord zijn opgetreden na de verzending, zal de vermenigvuldiging $H^T \underline{r}$ nul opleveren. Daaruit volgt dat als $H^T \underline{r} \neq 0$ dan zijn er een of meer fouten opgetreden. De uitkomst van deze vermenigvuldiging wordt het syndroom genoemd $S = H^T \underline{r}$. Het syndroom is $(n-k) \times 1$ groot en er zijn dus 2^{n-k} mogelijke syndromen. Het aantal mogelijke fouten die kunnen optreden zijn groter dan het aantal syndromen, daarom zal van te voren vastgelegd moeten worden welk syndroom bij welke fout hoort. De kans dat er één fout opgetreden is voor een kleine foutkans is groter dan dat er twee of meer fouten optreden. Dus bij de keuze waar fouten gekoppeld worden aan syndromen zal daar rekening mee gehouden worden. Hieronder is een tabel te zien waarbij syndromen aan fouten e_i toegewezen zijn. De waardes in de tabel zijn volkomen willekeurig.

	Syndroom	Ontvangen woorden				
Code woorden	000...	0000...0	1011...	1101...	0001...	...
e_1	100...	1000...0	0011...	0101...	1001...	...
e_2	010...	0100...0	1111...	1001...	0101...	...
e_3	001...	0010...0	1001...	1111...	0011...	...
...	...	...	...	...	...	...

De fouten die optreden worden nu genoteerd als e_i net zoals in de tabel hierboven. Dan geldt er $\underline{r} = \underline{c} + e_i$, dus:

$$S = H^T \underline{r} = H^T (\underline{c} + e_i) = H^T \underline{c} + H^T e_i = 0 + H^T e_i.$$

Hieruit volgt dat het syndroom ook te schrijven is als de bijbehorende fout vermenigvuldigd met de getransponeerde Parity Check matrix, $S = H^T e_i$.

In het voorbeeld hieronder is een syndroom tabel gegeven, deze is te bepalen door de vermenigvuldiging $H^T e_i$ uit te voeren voor bepaalde e_i. Als de fout eenmaal gevonden is, is het eenvoudig terug te gaan naar het code woord: $\underline{c} = \underline{r} - e_i$. Om nu van dit woord over te gaan op een woord uit de gedecodeerde lijst is wat lastiger. Daarom is de methode van parity bits toevoegen wel zo handig. Als de coderings methode berust op het toevoegen van parity bits dan kunnen bij $\underline{c}$ simpelweg de laatste $n-k$ bits verwijderd worden om $\underline{a}$ te verkrijgen.

Voorbeeld 10 (Hamming(7,4))**.** *Als er voor dezelfde codering als het vorige voorbeeld gekozen wordt, dus Hamming(7,4) (*$k = 4$ *en* $n = 7$*) met ook dezelfde nummering, dan is de parity check matrix gelijk aan:*

$$H = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} 1 \\ 1 \\ 0 \\ 1 \end{matrix}}\right\} P^T \\ \left.\vphantom{\begin{matrix} 1 \\ 0 \\ 0 \end{matrix}}\right\} I_{n-k} \end{matrix}$$

Hieronder is de syndroom tabel gegeven behorende bij deze code:

Error	*Syndroom*
0000000	*000*
1000000	*111*
0100000	*110*
0010000	*011*
0001000	*101*
0000100	*100*
0000010	*010*
0000001	*001*

Stel het woord $\underline{c} = [0101011]^T$ *wordt vermenigvuldigd met* H *dan* $H^T\underline{c} = 0$. *Echter als nu hetzelfde woord met een opgetreden fout, namelijk het rode bitje,* $\underline{r} = [1101011]$ *getest wordt, geeft* $H^T\underline{r} = [111]$. *Dan is er dus een fout gedetecteerd. Het bijbehorende syndroom* $S_j = [111]$, *geeft aan dat de fout* $e_j = [1000000]$ *opgetreden is. Nu is het originele woord dus* $[1101011] - [1000000] = [0101011] = \underline{c}$.

Hoeveel fouten een lineaire code kan detecteren en of verbeteren hangt af van de code. De Hamming code hierboven verbetert precies alle woorden, waar maar één fout opgetreden is en is dus 1 fout-verbeterend.

2.4. CYCLISCHE CODES

Cyclische codes zijn makkelijk toe te passen op hardware en worden daarom toegepast in onder andere computers en cd-spelers. Bij een cd-speler wordt bijvoorbeeld de Reed-Solomon code gebruikt, welke een cyclische code is. Het toepassen in de hardware is redelijk eenvoudig omdat de codering berust op shifts. Deze shifts kunnen op hardware uitgevoerd worden door middel van shift registers. De informatie over cyclische codes is opgehaald uit de boeken [2], [3] en [4].
Cyclische codes zijn een speciale vorm van lineaire codes. In dit hoofdstuk worden de woorden echter als polynomen beschouwd in plaats van vectoren. $\mathbb{F}_2[x]$ is een ring van polynomen met coefficiënten uit $\mathbb{F}_2$. Dit is de verzameling waar in gewerkt gaat worden, want alleen de binaire woorden worden beschouwd.
Voordat er verder in gegaan wordt op de polynomen, worden de woorden nog tijdelijk als vectoren behandeld. Dit zodat de cyclische shift beter te begrijpen is. Bij een cyclische shift verplaatsen alle bits in de vector een of meerdere plaatsen naar beneden en de onderste naar boven.

Voorbeeld 11 (Cyclische shift)**.** *Stel je voert één cyclische shift uit op* $\begin{bmatrix}0\\1\\1\\0\\1\end{bmatrix}$*, dan wordt* $\begin{bmatrix}1\\0\\1\\1\\0\end{bmatrix}$ *verkregen.*

Cyclische codes zullen beschouwd worden als deelverzamelingen van de ringen van de vorm:

$$R_n = \mathbb{F}_2[X]/(X^n - 1)$$

Omdat er in $\mathbb{F}_2$ gewerkt wordt is dit hetzelfde als:

$$R_n = \mathbb{F}_2[X]/(X^n + 1)$$

Dit betekent dat polynomen beschouwd worden met coëfficiënten 0 of 1 en modulo $x_n - 1$ wordt over het hele polynoom genomen. Door dit laatste werkt de shift hetzelfde als bij de vector, echter nu wordt een shift uitgevoerd door te vermenigvuldigen met X^i voor alle $i \in \mathbb{N}$. Na de vermenigvuldiging moet er altijd weer mod $(X^n + 1)$ gewerkt worden. De laatste bit in de vector wordt omhoog geplaatst. Bij een shift in R_n gebeurt praktisch hetzelfde: $X^{n-1} * X = X^n = 1$. Dus die polynoom wordt als het ware naar voren geplaatst. Een polynoom p uit R_n ziet er als volgt uit:

$$p(X) = p_0 + p_1X + p_2X^2 + ... + p_{n-1}X^{n-1}$$

Om een cyclische shift op een polynoom in R_n wat duidelijker te maken, volgt nu een voorbeeld.

Voorbeeld 12 (Cyclische shift bij polynoom)**.** *Beschouw het polynoom* $a(X) \in C$*, C cyclisch.*

$$a(X) = a_0 + a_1X + a_2X^2 + ... + a_{n-1}X^{n-1}$$

Een shift door middel van vermenigvuldiging met X levert het volgende op:

$$a(X)X = a_{n-1} + a_0X + a_1X^2 + ... + a_{n-2}X^{n-1}$$

Alle coëficienten zijn nu 1 plek verplaatst.

Cyclische codes behoren aan de volgende twee eisen te voldoen, zoals gedefinieerd in [4]:

Definitie 1 (Cyclisch code)**.** *Een code C is cyclisch als*

i C een lineaire code is

ii C is invariant onder elke cyclische shift

De laatste eis betekent dat na een cyclische shift het nieuwe woord ook in de code zit. Zoals al eerder vermeld vindt een cyclische shift plaats door het polynoom te vermenigvuldigen met X^i voor alle $i \in \mathbb{N}$. Hieronder nog wat voorbeelden van shiften en lineaire combinaties van shiften.

Voorbeeld 13 (Cyclische polynoom)**.** *Beschouw het polynoom $a(X) \in C$, C cyclisch. Als $a(X)$ vermenigvuldigd wordt met X^n, dan:*

$$a(X)X^n = a_0 + a_1X + a_2X^2 + ... + a_{n-1}X^{n-1} = a(X)$$

Dit is een shift van n plekken, dus de coëficienten eindigen op dezelfde plek als vóór de shift. Ook combinaties van shiften zitten weer in de code C:

$$a(X)(1+X) = (a_0 + a_{n-1}) + (a_1 + a_0)X + (a_2 + a_1)X^2 + ... + (a_{n-1} + a_{n-2})X^{n-1}$$

2.4.1. GENERATOR POLYNOOM

De generator polynoom is een polynoom die de gehele cyclische code voortbrengt.

Stelling 1 (Generator polynoom)**.** *Een cyclische code C kan voortgebracht worden door een polynoom $g(x) \in \mathbb{F}_2[x]$. Dit is de generator polynoom en $\langle g(X)\rangle = \{r(X)g(X) | r(X) \in R_n\}$*

De volgende stelling zal dezelfde en nog een aantal aanvullende eigenschappen geven over deze generator polynoom. Bij de bovenstaande stelling geef ik geen bewijs, die staat namelijk al in het bewijs van de volgende stelling uit [2].

Stelling 2. *Laat $C \in R_n$ een cyclische code zijn ongelijk aan nul. Dan bestaat er een polynoom $g(X) \in C$ met de volgende eigenschappen:*

i $g(X) = g_0 + g_1X + ... + g_{r-1}X^{r-1}$ *is het unieke monische polynoom met minimum graad in C*

ii $C = \langle g(X)\rangle$

iii $g(X) | (X^n - 1)$.

iv *Laat $k = n -$ deg $g(X)$, en laat $g(X) = \sum_{i=0}^{n-k} g_iX^i$, met $g_{n-k} = 1$. Dan is k de dimensie van C en*

$$\{g(X), Xg(X), X^2g(X), ..., X^{k-1}g(X)\}$$

een basis van C.

v *elk element van C is uniek uit te drukken als een product van $g(X)f(X)$, met $f(X) = 0$ of deg$f(X) < k$*

vi *De generator matrix is:*

$$\begin{bmatrix} g_0 & 0 & 0 & \cdots & 0 \\ g_1 & g_0 & 0 & \cdots & 0 \\ g_2 & g_1 & g_0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ g_{n-k} & g_{n-k-1} & g_{n-k-2} & \cdots & g_0 \\ 0 & g_{n-k} & g_{n-k-1} & \cdots & g_1 \\ 0 & 0 & g_{n-k} & \cdots & g_2 \\ 0 & 0 & 0 & \cdots & g_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & g_{n-k} \end{bmatrix}$$

Bewijs. Eerst *i* en *ii* laten zien. Kies $g(X) \in C$ met $g(X)$ polynoom van kleinste graad in C. Deze bestaat want C is niet leeg. Kies nu $c(X) \in C$ willekeurig dan geldt $c(X) = g(X)h(X) + r(X)$ met $r(X) = 0$ of deg $r(X) <$deg $g(X)$. Omdat $c(X) \in C$ en $g(X)h(X) \in C$ geldt $c(X) - g(X)h(X) = r(X) \in C$ en omdat $g(X)$ het minimum polynoom is geldt er $r(X) = 0$. Dus $c(X) = g(X)h(X)$ dus $c(X) \in \langle g(X)\rangle$. Omdat $g(X)$ als polynoom met kleinste graad is gekozen geeft dit *i* en *ii*.
Nu *iii* laten zien, dus dat $g(X) | (X^n - 1)$. Er geldt $(X^n - 1) = g(X)h(X) + r(X)$ met $r(X) = 0$ of deg $r(X) <$deg $g(X)$. $(X^n - 1)$ is gelijk aan het nulwoord in C en $r(X) \in C$ om dezelfde reden als eerder beschreven, wat een tegenspraak is behalve als $r(X) = 0$ anders is $(X^n - 1)$ niet het nulwoord van C. Maar als $r(X) = 0$ dan geldt $g(X) | (X^n - 1)$.

Nu iv, v en vi laten zien met de aanname dat $k = n-$ deg $g(X)$ en $g(X) = \sum_{i=0}^{n-k} g_i X^i$, met $g_{n-k} = 1$. Dus dan geldt deg $g(X) = n-k$ en volgens ii en iii geldt als $c(X) \in C$ met $c(X) = 0$ of deg $c(X) < n$ dan geldt er $c(X) = g(X)h(X)$. Als $c(X) = 0$ dan $h(X) = 0$. Als $c(X) \neq 0$ en dus deg $c(X) < n$ dan deg $h(X) < k$, want de graad van het product van twee polynomen is de som van de twee graden van de polynomen. Dus nu is C te schrijven als:

$$C = \{g(X)f(X) | f(X) = 0 \text{ of deg } f(X) < k\}$$

Dus de dimensie van C is op zijn hoogst k en C wordt opgespannen door

$$\{g(X), Xg(X), \ldots, X^{k-1}g(X)\}.$$

Omdat deze polynomen allemaal een andere graad hebben zijn ze onafhankelijk in $\mathbb{F}_q$. Omdat de graad op zijn hoogst $n-1$ is, zijn ze ook onafhankelijk in R_n. Dus iv en v gelden. En als de basis in n-tuples (vectoren lengte n) geschreven wordt verkrijg je G, dus ook vi geldt. □

Zoals hierboven gegeven is de generator polynoom ook te schrijven als een matrix. Deze is door rij optellingen altijd in de standaard vorm te schrijven, zoals gegeven in de sectie 2.3. Nu kunnen we met deze generator matrix net als bij lineaire codes de woorden uit de bericht lijst coderen.

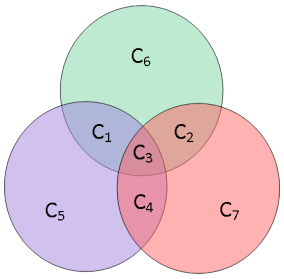
Voorbeeld 14 (Hamming(7,4))**.** *Beschouw de Cyclische code $C(7,4) \subseteq R_7$, waarbij 7 de lengte van een code woord en 4 de lengte van een bericht woord, met generator polynoom $g(X) = 1 + X + X^3$. De generator matrix ziet er dan als volgt uit:*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Deze matrix is door vegen van de kolommen ook te schrijven als:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Dit is de standaard vorm van een generator matrix. Deze cyclische code is equivalent met de Hamming(7,4), namelijk de volgende:



Het is handig om te weten hoe je een generator polynoom vindt bij een gegeven cyclische code C. Zoals in de stelling al aangegeven moet de generator polynoom een deler zijn van $x^n - 1$. Daarvoor moet $x^n - 1$ of $X^n + 1$ gefactoriseerd worden, dit kan erg lastig zijn.

Voorbeeld 15 (R_7)**.** *Het polynoom* X^7+1 *kan als volgt gefactoriseerd worden:*

$$X^7+1=(1+X)(1+X+X^3)(1+X^2+X^3)$$

Dus de volgende cyclische codes kunnen in R_7 *geproduceerd worden:*

Generator polynoom	*Codes*
$g(X)=1+X$	*C(7,6)*
$g(X)=1+X+X^3$	*C(7,4)*
$g(X)=1+X^2+X^3$	*C(7,4)*
$g(X)=(1+X)(1+X+X^3)$	*C(7,3)*
$g(X)=(1+X)(1+X^2+X^3)$	*C(7,3)*
$g(X)=(1+X+X^3)(1+X^2+X^3)$	*C(7,1)*

Stelling 3. *(Uniciteit) Een minimum graad generator polynoom* $g(X)$ *met graad* r*, met* $g(X)\neq 0$*, is uniek voor een gegeven code* $C(n,k)$*.*

Bewijs. Bewijs door tegenspraak. Neem aan dat er twee generator polynomen van minimum graad zijn $g_1(X)$ en $g_2(X)$, beide met graad r. Omdat de cyclische code C lineair is geldt dat de som van deze twee generator polynomen ook weer in C zit. Dus $p(X)=g_1(X)+g_2(X)$ zit weer in C en de coëfficiënt behorende bij X^r is gelijk aan 0, want $g_1(X)$ en $g_2(X)$ zijn beide van graad r dus de coëfficiënt is voor beiden 1 bij X^r en $1+1=0$ mod 2. Dit betekent dat $p(X)$ een lagere graad heeft dan r, echter dit is in tegenspraak met het gegeven dat de generator polynoom de minimum graad heeft. □

Elk codewoord $c(X)\in C$ is een meervoud van de generator polynoom, dus:

$$c(X)=a(X)g(X)=(a_0+a_1X+a_2X^2+\ldots+a_{k-1}X^{k-1})(g_0+c_1X+g_2X^2+\ldots+g_{r-1}X^{r-1})$$

Hierbij zijn $a(X)$ de woorden uit bericht lijst en $c(X)$ de bijbehorende code woorden. En r is de lengte van de generator polynoom. De lengte r van de generator polynoom geeft de vorm van de cyclische code aan en er geldt $r=n-k$, waarbij n de lengte van een code woord en k de lengte van een woord uit de bericht lijst.

2.4.2. CODEREN

Een polynoom coderen kan door deze terug te schrijven naar een vector en te vermenigvuldigen met de hierboven beschreven generator matrix. Daarnaast kan een cyclische polynoom ook gecodeerd worden door het te vermenigvuldigen met een generator polynoom. Vaak zijn codes met parity bits veel betrouwbaarder. Dit gebeurt op een wat andere manier, daarom is hieronder een systematische aanpak gegeven om parity bits aan een woord $a(X)$ uit de bericht lijst mee te geven:

Stap 1

Maak het polynoom $X^{n-k}a(X)$.

Stap 2

Deel vervolgens $X^{n-k}a(X)$ met rest door het generator polynoom $g(X)$:

$$X^{n-k}a(X)=q(X)g(X)+p(X)$$

Waarbij $p(X)$ het rest polynoom is. Dan is dit te herschrijven tot:

$$q(X)g(X)=X^{n-k}a(X)+p(X)$$

Hier geldt dat $q(X)g(X)\in C$, want het is een meervoud van $g(X)$. Nu staat $X^{n-k}a(X)$ voor een shift van het bericht woord en $p(X)$ is dan het parity polynoom (wat te beschouwen is als de polynoom met parity bits).

Stap 3

$c(X)=X^{n-k}a(X)+p(X)=p_0+p_1X+\ldots+p_{n-k-1}X^{n-k-1}+m_0X^{n-k}+m_1X^{n-k+1}+\ldots+m_{k-1}X^{n-1}$ geeft de systematische weergave van code woord $c(X)\in C$ behorende bij het bericht woord $a(X)$. Bij deze codering geldt dat de parity bits in het begin van het woord worden toegevoegd.

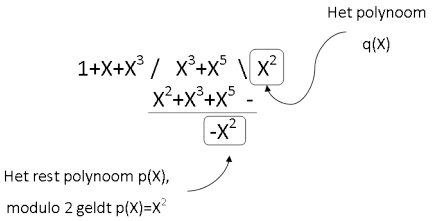
Voorbeeld 16. *Beschouw de cyclische code* $C(7,4)$*, ook wel de Hamming(7,4), met generator polynoom* $g(X) = 1 + X + X^3$*. Bericht woord* $a = (1010)$ *heeft dan de volgende polynoom:*

$$a(X) = 1 + X^2$$

Vervolgens wordt de polynoom als volgt gecodeerd:

$$X^{n-k}a(X) = X^{7-4}(1 + X^2) = X^3 + X^5$$

Deel dit vervolgens door $g(X)$ *door middel van staartdeling:*



Dus er geldt:

$$X^3 + X^5 = X^2(1 + X + X^3) + X^2$$

$p(X) = X^2$*, waaruit volgt:*

$$c(X) = X^{n-k}a(X) + p(X) = X^2 + X^3 + X^5$$

Dus het codewoord behorende bij $a = (1010)$ *is* $c = (0011010)$.

2.4.3. DECODEREN

Decoderen gaat net als bij lineaire codering via syndroom berekening. Hieronder een stappen plan voor de decodering van een ontvangen woord $r(x)$.

Stap 1

Deel het ontvangen woord $r(X) = r_0 + r_1X + ... + r_{n-1}X^{n-1}$ door $g(X)$ met rest:

$$r(X) = q(X)g(X) + S(X)$$

Hierbij heet $S(X)$ het syndroom.

Stap 2

Als het ontvangen woord een code woord is dan is het een veelvoud van $g(X)$, dus geldt er $S(X) = 0$. Als $S(X) \neq 0$, dan is er dus een fout gedetecteerd.

Stap 3

Het ontvangen woord $r(X)$ is te schrijven als

$$r(X) = c(X) + e(X)$$

Hierbij is $c(X)$ het verzonden woord en $e(X)$ de fout die opgetreden is. Dit omschrijven geeft:

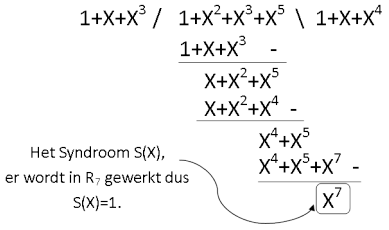
$$e(X) = c(X) + r(X) = c(X) + q(X)g(X) + S(X) = (f(X) + q(X))g(X) + S(X).$$

Dus $S(X) = e(X) \mod g(X)$.

Voorbeeld 17. *Beschouw de cyclische code* $C(7,4)$ *met generator polynoom* $g(X) = 1 + X + X^3$*, dus* $n = 7$ *en* $k = 4$*. De bijbehorende syndromen zijn in de volgende tabel gezet:*

Error	*Syndroom polynoom*	*Syndroom vector*
$e_0 = 1$	1	(100)
$e_1 = X^1$	X	(010)
$e_2 = X^2$	X^2	(001)
$e_3 = X^3$	$1+X$	(110)
$e_4 = X^4$	$X+X^2$	(011)
$e_5 = X^5$	$1+X+X^2$	(111)
$e_6 = X^6$	$1+X^2$	(101)

Stel het woord $a(X) = 1 + X^2$ wordt gecodeerd tot het woord $c(X) = X^2 + X^3 + X^5$. Tijdens de verzending treedt er een fout op en wordt het woord $r(X) = 1 + X^2 + X^3 + X^5$ ontvangen. Nu zal $r(X) \mod g(X)$ uitgerekend worden:



Dus als $S(X) = 1$ dan $e(X) = 1$. Het verbeteren van deze fout geeft $c = X^2 + X^3 + X^5$. Als nu de laatste helft van de polynoom een shift ondergaat van X^{k-n}, dan wordt $a(X) = 1 + X^2$ verkregen, dus de verbetering is goed gegaan.

2.5. LOW DENSITY PARITY CHECK CODES

Een tegenwoordig veel gebruikte coderings methode is de low density parity check code, LDPC. De LDPC is geïntroduceerd in 1963 door Gallager, helaas raakte zijn werk in de vergetelheid. Dit kwam omdat het in die tijd nog erg moeilijk te implementeren was. Hier is pas vanaf 1981 verandering in gekomen door het werk van Tanner [8], waarna deze code veel toepassingen heeft gekregen, zoals onder andere in satelliet transmissie, digitale televisie en nog veel meer. Op dit moment wordt bijna overal of de LDPC of de turbo code toegepast, er zal niet verder ingegaan worden op hoe de turbo code in elkaar steekt. De LDPC code kan gecodeerd en gedecodeerd worden zoals eerder is laten zien in sectie 2.3. Later zal er ook een iteratieve wijze besproken worden waarmee de LDPC gedecodeerd kan worden.
De LDPC is een lineaire code met een bijzondere parity check matrix. Het verschil met de eerder beschreven parity check matrix is dat de matrix geen volle rang hoeft te hebben, in plaats van $n \times (n-k)$ is de afmeting $n \times m$. Hierbij is k de lengte van een woord uit de bericht lijst en n de lengte van een woord uit de code lijst. In de parity check matrix van een LDPC code komen weinig enen voor, vandaar de naam low density. Het aantal enen per rij en kolom staat vast. Elke rij heeft c aantal enen en elke kolom heeft r aantal enen. Als de parity check matrix $n \times m$ is dan geldt het volgende verband: $nc = mr$. Dit verband volgt uit het feit dat het aantal enen in de parity check matrix gelijk is aan nc en aan mr. De LDPC codes worden als volgt genoteerd: LDPC$(n, c, rang(H))$. De volgende beperking geldt op de afmetingen van de parity check matrix:

$$n - k \leqslant m \leqslant 2^{n-k}$$

Er zijn minimaal $n-k$ kolommen nodig, want de kolommen van H $(col(H))$ moeten loodrecht staan op $col(G)$ dus $\dim(col(H)) = \dim(colG)^{\perp}$. De dimensie van het orthogonale complement van G is volgens stelling 6.1 op bladzijde 331 uit "Linear Algebra"[5] gelijk aan $n-k$, dit omdat k de dimensie is van de kolomruimte G. Omdat $\dim(col(H)) = n-k$ moeten er minimaal $n-k$ kolommen in H zitten, anders gaat er informatie verloren. Er zullen hooguit 2^{n-k} kolommen zijn, want $(colG)^{\perp}$ bevat 2^{n-k} vectoren. Meer keuzes zijn er niet. Ondanks dat H bij een LDPC code meer kolommen bevat geldt wel nog steeds $H^T G = 0$. Voor meer informatie over de LDPC code kunnen het artikel "Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results"[6] en de slides van het vak "Error Correcting Codes"van Jos Weber [7] geraadpleegd worden.

Voorbeeld 18 (LDPC(7,3,4))**.** *De rang van H is 4, het aantal rijen is 7 en de density c is gelijk aan 3. Er geldt $nc = 21$ dus moet ook gelden $mr = 21$, 21 is te ontbinden in $3 * 7$ of $21 * 1$. Hieronder is de parity check matrix gegeven voor het geval dat er 7 kolommen zijn en $r = 3$.*

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

3

DECODERING EN PRESTATIE

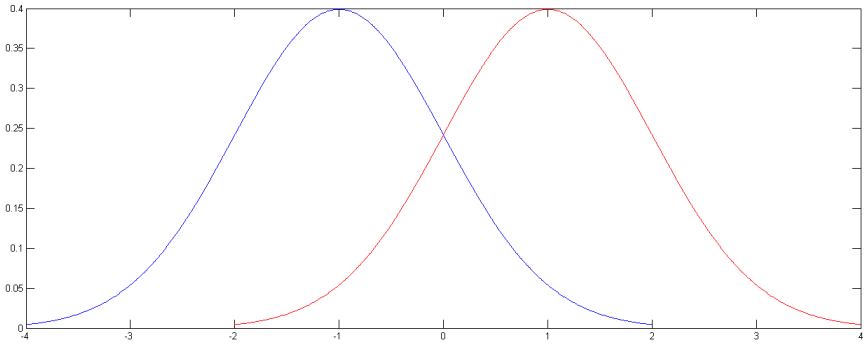
Er zijn twee conceptueel verschillende decoderings methoden, namelijk de zogenaamde hard en soft decision decodering. Tot nu toe is alleen de hard decision decodering besproken, waarbij ervan uitgegaan wordt dat er altijd of een 1 of een 0 ontvangen wordt. Om verder in te gaan op het verschil tussen hard en soft decision decodering, zal eerst de verzending besproken worden. Aanvullende informatie bij dit hoofdstuk kan gevonden worden in [1] en [6].

3.1. GAUSSISCHE RUIS

Meestal worden de fouten die optreden in het kanaal gesimuleerd met Gaussische ruis. Deze Gaussische ruis kan veroorzaakt worden door natuurlijke bronnen, zoals radiatie van de aarde en andere warme objecten, de zon of bijvoorbeeld de hitte in de geleiders van een computer. Gaussische ruis heeft een dichtheidsfunctie gelijk aan de normale verdeling. De Gaussische verdeling ziet er als volgt uit:

$$f(x|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

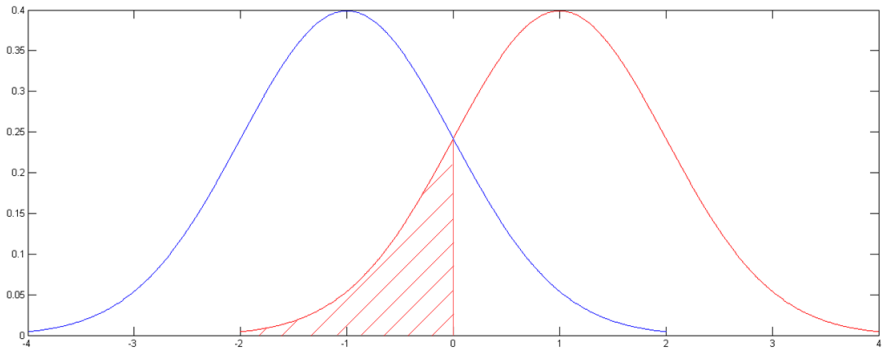
Hierbij is μ de verwachting en σ de standaardafwijking. Vaak als bits verstuurd worden dan geldt er dat 0 een puls met waarde $+1$ is en 1 een puls met waarde -1. Bij de verzending vindt Gaussische ruis plaats, hieronder is in een grafiek weergegeven wat voor effect de Gaussische ruis heeft op beide pulsen. Het effect op de puls $+1$ is in het rood weergegeven en het effect op -1 in het blauw.



In de werkelijkheid worden na verzending niet bits ontvangen maar reële getallen, die dicht bij de oorspronkelijke verzending liggen, als model voor de verdeling wordt Gaussische ruis aangenomen.

3.1.1. HARD DECISION DECODERING

Bij hard decision decoderen worden de ontvangen getallen eerst afgerond en daarna wordt er pas gedecodeerd. Dit afronden gebeurt afhankelijk van het teken. Stel een positief reëel getal wordt ontvangen dan wordt dit getal omgezet naar 0, als er een negatief reëel getal ontvangen wordt, wordt dit afgerond naar 1. Dus er treedt een fout op als bijvoorbeeld de puls +1 door de ruis vervormd wordt naar −0.2. Hieronder zijn weer de Gaussische verdelingen geplot, de waarde van het oppervlakte gearceerd onder de rode grafiek geeft de kans dat een bit van 0 omgeklapt naar 1, dus een opgetreden fout.



Er zal een voorbeeld behandeld worden van hard decision bij de herhalingscode.

Voorbeeld 19 (Herhalingscode)**.** *Stel het woord* 0 *wordt gecodeerd naar* (000) *en vervolgens wordt het verzonden door een kanaal met Gaussische ruis. Het ontvangen woord is nu* $(0.9; -0.1; -0.1)$*, dit wordt eerst afgerond naar* (011)*, vervolgens bij de decodering omgezet naar* (111)*. Daaruit volgt dat na decodering het woord* 1 *is, dus het ontvangen woord fout verbeterd.*

De absolute waarde van een ontvangen puls geeft de zekerheid aan, dus de kans dat een woord goed ontvangen is. Als +0.9 bijvoorbeeld ontvangen wordt is er een grote kans dat dit daadwerkelijk een 0 is geweest. Echter, als +0.1 ontvangen wordt is de kans aanzienlijk kleiner dat er daadwerkelijk een 0 verzonden was. Dus bij +0.1 is de kans groter dat de puls ten onrechte afgerond wordt naar 0. Echter, bij hard decision decodering wordt geen rekening gehouden met de absolute waarde van de puls.

3.1.2. SOFT DECISION DECODERING

Soft decision decoderen houdt rekening met de absolute waarde van de ontvangen puls. De afronding naar de bits 0 of 1 gebeurt pas na de decodering. In het volgende hoofdstuk 4 zal een voorbeeld van iteratieve soft decision decodering gegeven worden. Hieronder is hetzelfde voorbeeld als bij hard decision gegeven. Echter, dit keer zal er een goede verbetering plaatsvinden.

Voorbeeld 20 (Herhalingscode, euclidische afstand)**.** *Stel het woord* 0 *wordt verzonden en* $(0.9; -0.1; -0.1)$ *ontvangen. Bij decodering zal de euclidische afstand beschouwd worden. Het woord met de kleinste afstand wordt gekozen. In de herhalingscode zijn er twee mogelijke woorden, dus de afstanden tot deze woorden moet berekend worden. De euclidische afstand ziet er als volgt uit:*

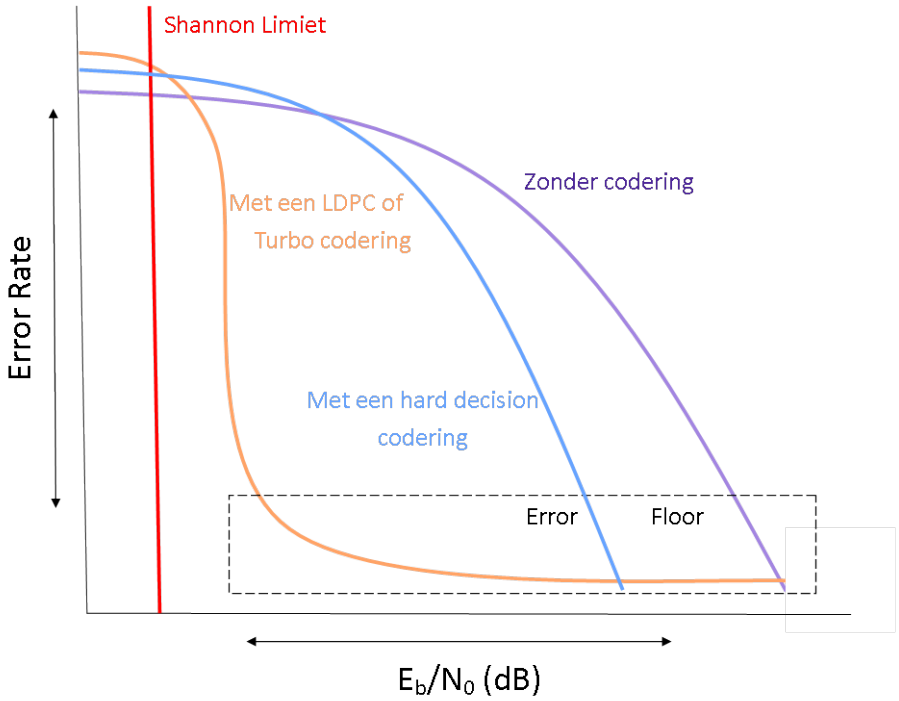
$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Waarbij x *een woord uit de code is en* y *het ontvangen woord. De euclidische afstand van het codewoord* (111) *(dus het bericht woord* 0*) tot het ontvangen woord* $(0.9; -0.1; -0.1)$ *is gelijk aan* 1.63*. De euclidische afstand van het codewoord* $(-1; -1; -1)$ *(dus het bericht woord* 1*) tot het ontvangen woord* $(0.9; -0.1; -0.1)$ *is gelijk aan* 2.25*. Dus het woord* $(1; 1; 1)$ *wordt gekozen, wat na decodering* 0 *wordt. Dus de fout is dit keer wel goed verbeterd.*

Soft decision decodering wordt veelvuldig toegepast in de praktijk, dit omdat de foutverbeterings prestatie beter is dan bij hard decision decodering. Hierboven is ook een voorbeeld gegeven van soft decision decodering die de fout goed verbeterd, terwijl dezelfde fout niet verbeterd werd bij hard decision decodering. Implementatie van soft decision decodering kan echter wel ingewikkelder zijn. In het volgende hoofdstuk volgt er een voorbeeld van een veel toegepaste soft decision decodering.

3.2. SHANNON LIMIET

Om de prestaties in de praktijk van een decoderings methode vast te leggen, wordt de error rate vaak uitgezet tegen de gebruikte energie gedeeld door een waarde N_0 die afhankelijk is van de ruis, $\frac{E_b}{N_0}$ (dB). Met de error rate wordt de kans op een fout woord na de codering en decodering bedoeld. Bij Gaussische ruis wordt N_0 bepaald door de standaardafwijking, $N_0 = 2\sigma^2$. Al eerder is besproken dat bits verzonden worden met pulsen van grootte +1 en −1. Als nu de informatie in hetzelfde kanaal, dus met dezelfde ruis, verzonden wordt met pulsen van +5 en −5, dan is de foutkans veel kleiner omdat de standaardafwijking van de Gaussische ruis hetzelfde blijft. Des te groter de pulsen des te meer energie nodig is, daarom is er een negatieve correlatie tussen error rate en energie. Hieronder is een grafiek te zien van error rate voor een verzending zonder codering, een verzending met een willekeurige hard decision codering en een verzending met een willekeurige turbo of LDPC codering. De Shannon limiet is ook afgebeeld, daar zal later veder op ingegaan worden.



Zoals hierboven te zien, ziet de grafiek van de Turbo of LDPC codering eruit als een waterval. De coderingen met een prestatie grafiek die op een waterval lijkt, hebben een erg goede prestatie en worden daarom veel toegepast in de praktijk. Vanaf een gegeven energie, afhankelijk van de methode, zal de grafiek steil dalen, waarna het uiteindelijk weer uitvlakt. Dit laatste stadium van de grafiek wordt ook wel de error floor genoemd. De prestatie van een coderingsmethode hangt af van wanneer de steile daling en de error floor begint. Hoe eerder de steile daling begint hoe minder energie er nodig is voor verzending bij de gebruikte methode. Hoe lager de error floor begint hoe lager de error rate wordt bij een relatief lage energie.

Shannon is een wiskundige die veel fundamentele wiskundige relaties in de coderingstheorie heeft gelegd. Hij heeft onder andere aangetoond dat een kanaal een capaciteit C heeft. Als een verzending onder deze capaciteit C zit, dan bestaan er codes die willekeurig kleine error kansen hebben. Uit deze vindingen volgde de Shannon limiet. De Shannon limiet geeft aan dat met een optimale coderingsmethode het mogelijk is om een error kans van nul te benaderen met een energie ruis constante verhouding zo laag als: $\frac{E_b}{N_0} \approx -1.6dB$. Shannon heeft aangetoond dat de codes bestaan, maar niet hoe deze gemaakt kunnen worden. In 1993 kwam de eerste code die de Shannon limiet benaderde, de zogenaamde turbo code. Samen met LDPC is turbo code de meest gebruikte code in de praktijk. Ze hebben een vergelijkbare performance ten overstaan van de Shannon limiet. De LDPC code kreeg meer belangstelling vanaf 1996 en heeft een error van 10^{-6} en een $\frac{E_b}{N_0}$ die 0.04 dB verwijderd is van de Shannon limiet. Daarnaast heeft de LDPC code een lagere error floor dan de turbo code, wat de LDPC code aantrekkelijk maakt.

4

ITERATIEF DECODEREN

In 1962 publiceerde Gallager al de eerste ideeën over iteratief decoderen, deze raakten echter in de vergetelheid. Iteratief decoderen kwam weer op in 1981, toen Tanner een artikel over iteratief decoderen door middel van een graaf publiceerde [8]. Deze graaf werd de Tanner graaf genoemd. Iteratief decoderen werkt zo goed, omdat het coderings proces onderverdeeld kan worden in subprocessen. In de context van een Tanner graaf betekent dit dat niet de hele graaf in een keer gedecodeerd moet worden, maar dat de graaf opgedeeld wordt in subgraven die allemaal apart gedecodeerd kunnen worden. Eerst zal een notatie ingevoerd worden, daarna zal de Tanner graaf behandeld worden, en ten slotte wordt het iteratieve decoderings algoritme Min-Sum behandeld.

4.1. STRUCTUUR

Een handige manier om de Tanner graaf en iteratieve algoritmes te noteren is met behulp van de volgende terminologie gebaseerd op de structuur uit artikel [9]. Het systeem dat gebruikt gaat worden is een tripel en wordt genoteerd als (N, W, B).

Er zal wat dieper ingegaan worden op de tripel en zijn ruimtes. W is een configuratie ruimte en kan beschreven worden als een product $W = \prod_{s \in N} A_s$ van toestanden. De set N bestaat uit gehele getallen $\{1, \ldots, n\}$ en A_s is voor elke $s \in N$ een deelverzameling van $\mathbb{F}_q$. Daaruit volgt dat $W \subseteq \mathbb{F}_q^n$, dus W is een ruimte van n-tupels. Verder wordt aangenomen dat N en A_s een eindig aantal elementen bevatten. Elementen van N zijn aanduidingen van posities in een woord en elementen van W zijn woorden. $B \subseteq W$ is een gedrag op W. Alle elementen van B heten geldige configuraties. In de context van codes zitten in B de woorden die voldoen aan de code, dus de code woorden. Om deze notatie wat tastbaarder te maken zal hier nu een klein voorbeeld volgen.

Voorbeeld 21 (Het alfabet)**.** *Stel de code C bestaat uit alle woorden van lengte drie, die beschreven staan in het woordenboek van Van Dale. Dan bestaat de configuratie ruimte W uit alle mogelijke woorden gemaakt door drie letters afkomstig uit het Nederlandse alfabet. Het gedrag B bestaat uit alle woorden uit het Van Dale woordenboek met lengte 3. $N = \{1,2,3\}$ en A_s bestaat uit de letters van het alfabet voor $s \in N$. Stel het woord $x = DOP$ is ontvangen dan geldt er $x \in B$. Echter als het woord $y = DRP$ ontvangen wordt dan geldt er $y \notin B$, maar wel $y \in W$.*

Zoals eerder vermeld zal deze scriptie beperkt blijven tot het coderen en decoderen in de binaire ruimte. Voor de configuratie ruimte geldt dan $W = \mathbb{F}_2^n$, dus W is een ruimte van binaire n-tupels, en $B \subseteq \mathbb{F}_2^n$. Daarnaast is de toestand A_s beperkt tot het aannemen van de waardes 0 of 1. De waardes in N geven nu een aanduiding van welke bit in een woord bedoeld wordt. Hier een voorbeeld van zo een systeem in de binaire ruimte $\mathbb{F}_2^7$.

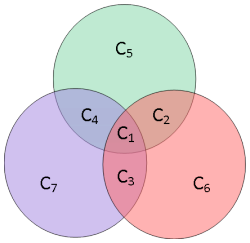
Voorbeeld 22 (Hamming(7,4))**.** *Bij Hamming(7,4) hebben de te verzenden woorden lengte 7 dus $W = \mathbb{F}_2^7$, en $W = \prod_{s \in N} A_s$ met $A_s = \{0,1\}$ voor $s \in N$, $N = \{1,2,\cdots,7\}$. B bestaat hier uit alle woorden die voldoen aan de Hamming code. Bijvoorbeeld het woord* $(0101110) \in W$ *zit in B, maar* $(0110111) \in W$ *zit niet in B, want dit woord voldoet niet aan de Hamming voorwaarden.*

Hier volgen diverse notaties die in het vervolg gebruikt gaan worden:

- De bits van een woord $x \in W$ zullen genoteerd worden met x_s, met $s \in \{1,\ldots,n\}$. Hierbij geeft s de positie van de bedoelde bit weer.
- De restrictie van een woord $x \in W$ op de deelverzameling $R \subseteq N$ wordt genoteerd als x_R. Dus bijvoorbeeld de rij $x_{s_1},\ldots,x_{s_k}$ is gelijk aan x_R met $R = \{s_1,\ldots,s_k\}$.
- Voor een deelverzameling woorden $X \subseteq W$ en een deelverzameling bits $R \subseteq N$ wordt de volgende notatie gebruikt:
$$X_R = \{x_R : x \in X\}$$

Een check structuur Q op (N,W,B) is een collectie deelverzamelingen E van N zodat voor elk woord $x \in W$ geldt: als $x_E \in B_E$ voor alle check verzamelingen $E \in Q$ dan is x geldig, dus $x \in B$. Hierbij wordt B_E het lokale gedrag genoemd. Een element x is lokaal geldig op E als $x_E \in B_E$. Een belangrijke opmerking hierbij is dan: Een woord x is geldig dan en slechts dan als het lokaal geldig is op alle check verzamelingen. Door deze check verzamelingen kan het decoderen onderverdeeld worden in het decoderen van kleinere systemen.

Voorbeeld 23 (Hamm(7,4))**.** *Stel de Hamming ziet er als volgt uit:*

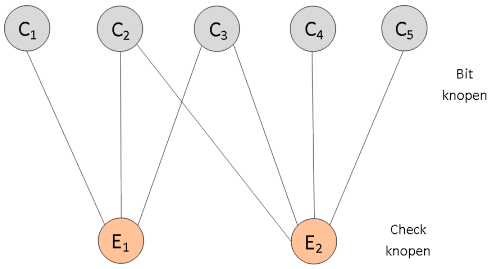


Elke cirkel kan gezien worden als een check verzameling, dus $Q = \{E_1 = \{1,2,4,5\}, E_2 = \{1,2,3,6\}, E_3 = \{1,3,4,7\}\}$. *De voorwaarde van alle* E_i *is dat de som van de bits in* E_i *modulo 2 gelijk aan nul is. Het spreekt voor zich als een woord aan alle check verzamelingen voldoet dat deze in* B *zit. Neem nu bijvoorbeeld* $x = [0110111]$*, dan* $x_{E_2} \notin B_{E_2}$ *want* $0+1+1+1 = 1 \mod 2$. *Dus* $x \notin B$.

4.2. TANNER GRAAF

De Tanner graaf is een breed begrip, het is een gereedschap om codes mee te decoderen. De Tanner graaf kan bij verscheidene codes toegepast worden, waaronder turbo codes, LDPC codes en Hamming codes. De basis voor deze graaf is gelegd door Tanner in "A Recursive Approach to Low Complexity Codes"[8]. In deze paragraaf zal de graaf alleen toegepast worden in het geval van binaire decodering.

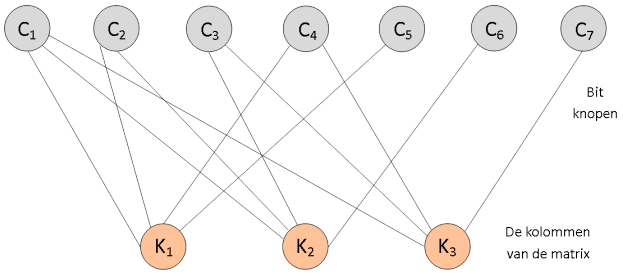
De Tanner graaf is een bipartite graaf. Één van de twee partities bestaat uit knopen die bits voorstellen (dus de posities in de woorden aangeduid door elementen van N), de andere partitie bestaat uit knopen die de checkverzamelingen representeren, dus elk element van Q is een knoop. Door deze check verzamelingen kan het decoderen onderverdeeld worden in het decoderen van subgrafen in plaats van de hele graaf door per check knoop te decoderen. Hieronder een algemene Tanner graaf van een code met woorden lengte 5 en twee check verzamelingen.



Meestal wordt de Tanner graaf ontwikkeld uit een parity check matrix. Elke rij in de parity check matrix H levert een bit knoop, en elke kolom een check knoop. Als er een 1 in de parity check matrix H staat dan zal er een tak tussen de bijbehorende rij en kolom zijn. Doordat de Tannergraaf uit de parity check matrix is gemaakt, geldt er voor alle check knopen dat de som van alle verbonden bit knopen gelijk aan nul moet zijn. Dit laatste kan gebruikt worden bij iteratieve decodering, waar de graaf gedecodeerd wordt door middel van subgrafen.

Voorbeeld 24 (Tanner graaf gemaakt uit een parity check matrix)**.** *Hieronder een parity check matrix met bijbehorende Tanner graaf.*

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



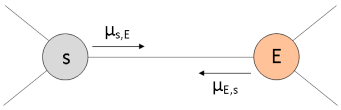
4.3. MIN-SUM ALGORITME

Als een Tanner graaf ontwikkeld wordt uit een parity check matrix is er een iteratieve methode om te decoderen via subgrafen. Hier zal één algoritme behandeld worden dat op een iteratieve wijze de graaf doorloopt, het Min-Sum algoritme. Eerst zal het algoritme algemeen beschreven worden, waarna er een toepassing behandeld wordt.

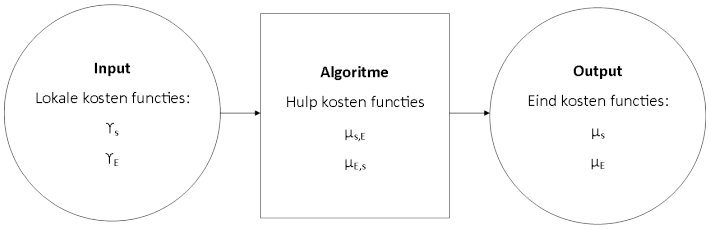
Het Min-Sum algoritme is oorspronkelijk ontworpen door Tanner, genaamd Tanner's algorithm B. Later werd het algoritme verder ontwikkeld en in "Codes and Iterative Decoding on General Graphs"[9] is het uitgebreid beschreven. Het algoritme maakt gebruik van soft decision decodering, dus zal reële getallen als input hebben. Daarnaast moeten de gebruikte Tanner grafen ontwikkeld zijn uit parity check matrices, zodat elke check knoop inhoudt dat de som van alle aanliggende bit knopen aan een check knoop gelijk aan 0 moet zijn.

Het algoritme zal genoteerd worden met het eerder genoemde tripel (N, W, B) met een check structuur Q. Het algoritme heeft als input reële lokale kosten functies en als output eind kosten functies. Het doel van de eind kosten functies is het samenvoegen van alle lokale kosten functies in de bit en check knopen. Elke bit knoop heeft zijn eigen bit lokale kosten functie $\gamma_s : A_s \to \mathbb{R}$ voor $s \in N$. Omdat er in $\mathbb{F}_2$ gewerkt wordt, geldt $A_s = \{0, 1\}$. Hetzelfde geldt voor de check knopen, die hebben een check lokale kosten functie $\gamma_E : W_E \to \mathbb{R}$ voor $E \in Q$. Daarnaast hebben alle bit en check knopen een eigen eind kosten functie: $\mu_s : A_s \to \mathbb{R}$ voor $s \in N$ en $\mu_E : W_E \to \mathbb{R}$ voor $E \in Q$.

Tijdens het algoritme zullen er voor elk paar (s, E) met $s \in E$, dus voor elke tak, twee hulp kosten functies aangemaakt worden. De eerste is een check naar bit kosten functie, $\mu_{E,s} : A_s \to \mathbb{R}$. De tweede is een bit naar check kosten functie, $\mu_{s,E} : A_s \to \mathbb{R}$. Zoals hieronder is te zien kan de functie $\mu_{E,s}(a)$ geïnterpreteerd worden als de bijdrage van E naar s op de bit $x_s = a_s$, voor $a \in W$ en $s \in N$. Zo kan ook de functie $\mu_{s,E}(a)$ geïnterpreteerd worden als de bijdrage van s naar E op $x_s = a_s$, voor $a \in W$ en $s \in N$.



Hieronder is schematisch de werking van de functies weergegeven.



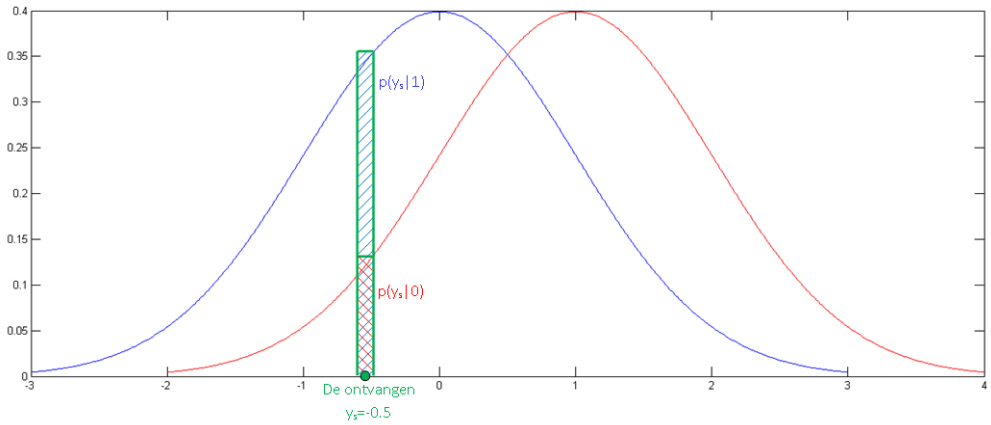
Het doel van dit algoritme is het vinden van een geldige configuratie $x \in B$ (dus het vinden van een woord die voldoet aan de code), zodat de som van de lokale kosten zo klein mogelijk is. De globale kosten functie, met $x \in B$, die dit weergeeft is gelijk aan:

$$G(x) = \sum_{E \in Q} \gamma_E(x_E) + \sum_{s \in N} \gamma_s(x_s)$$

Nu wordt de toepassing van het algoritme in codering verder uitgelicht. Stel het woord y wordt ontvangen uit een geheugenloos kanaal. Dan worden de check lokale kosten functies $\gamma_E(x_E)$ op nul gezet. Daarnaast worden de bit lokale kosten functies gelijk gesteld aan de log-likelihood: $\gamma_s(a_s) = -log p(y_s|a_s)$, dit is het negatieve logaritme van de kans dat y_s ontvangen is, terwijl a_s verzonden was. Omdat er maar twee toestanden zijn is de begin kosten functie voor elke bit knoop ook te schrijven als:

$$\gamma_s(a_s) = \begin{cases} -log p(y_s|0) & \text{als } a_s = 0 \\ -log p(y_s|1) & \text{als } a_s = 1 \end{cases}$$

De kans $p(y_s|a_s)$ is gelijk aan het oppervlak van de rechthoek onder de grafiek van de Gaussische verdeling zoals hieronder in de afbeelding is te zien. Stel $y_s = -0.5$ is ontvangen hieronder is de kans $p(y_s|0)$ in het rood weergegeven en de kans $p(y_s|1)$ in het blauw.



De kans $p(y_s|x_s)$ moet zo groot mogelijk zijn. De rede hiervoor is dat de negatieve log-likelihood van de kans wordt gebruikt. In de lokale kosten functie wordt de lokale kosten functie dus geminimaliseerd door $p(y_s|x_s)$ zo groot mogelijk te maken. Om deze reden moet de globale kosten functie geminimaliseerd worden.

In een standaard geheugenloos kanaal situatie waar de lokale check knoop kosten functies $\gamma_E(x_E)$ gelijk aan nul worden gesteld en de lokale bit knoop kosten functies gelijk zijn aan $\gamma_s(x_s) = -\log p(y_s|x_s)$, geldt dat de globale kosten functie $G(x)$ de log-likelihood $-\log p(y|x)$ wordt. Zoals later duidelijk wordt kan de min-sum algoritme de minimalisatie uitvoeren als de check structuur cykel vrij is. Met een cykel vrije code wordt bedoeld dat er geen cykels mogen voorkomen in de Tanner graaf behorende bij de code. Wanneer het algoritme termineert is afhankelijk van de toepassing. Er kan voor gekozen worden na een strikt aantal iteraties het proces te stoppen of als de hulpfuncties geconvergeerd zijn.

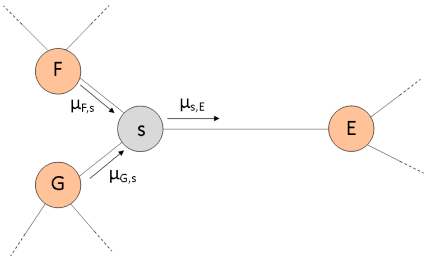
4.3.1. HET ALGORITME

Initialisatie

De lokale kosten functies γ_s en γ_E worden vastgesteld, zoals hierboven beschreven. De hulp kosten functies $\mu_{E,s}$ en $\mu_{s,E}$ worden op nul gezet.

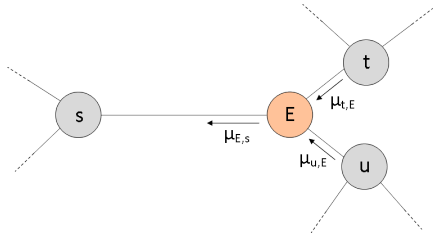
Iteratie

Vervolgens worden om de beurt de hulp kosten functies $\mu_{E,s}$ en $\mu_{s,E}$ een aantal keer geupdate. De hulp kosten functie $\mu_{s,E}(a_s)$, met $a \in W$ en $s \in N$, wordt berekend als de som van de lokale kosten functie γ_s en alle inkomende $\mu_{E',s}$ behalve vanuit E. Hieronder de functie en een afbeelding die het schematisch weergeeft:



$$\mu_{s,E}(a_s) := \gamma_s(a_s) + \sum_{E' \in Q: s \in E', E' \neq E} \mu_{E',s}(a_s)$$

De andere hulp kosten functie $\mu_{E,s}(a_E)$, met $a \in W$ en $s \in N$, wordt verkregen door de geldigheid van a in de omgeven bit knopen s te checken. Voor elke bit knoop worden de lokale kosten functies en alle inkomende $\mu_{s',E}$ in E behalve vanuit s gesommeerd. Het minimum van deze wordt gekozen als de kost $\mu_{E,s}(a_E)$. Hieronder de functie en een schematische weergave:



$$\mu_{E,s}(a_E) := \min_{x_E \in B_E : x_s = a_s} \{\gamma_E(x_E) + \sum_{s' \in E : s' \neq s} \mu_{s',E}(x_{s'})\}$$

Terminatie

De uiteindelijke kosten functies μ_s en μ_E worden dan als volgt berekend. De eind bit kosten functies $\mu_s(a_s)$, met $a \in W$ en $s \in N$, zijn te bepalen als de som van de lokale kosten functie van de bit en alle bijdragen $\mu_{E,s}$ die in s komen:

$$\mu_s(a_s) := \gamma_s(a_s) + \sum_{E' \in Q : s \in E'} \mu_{E',s}(a_s)$$

De eind check kosten functies $\mu_E(a_E)$ worden achterhaald als de som van de lokale kosten functie en alle bijdragen $\mu_{s,E}$ inkomende in E:

$$\mu_E(a_E) := \gamma_E(a_E) + \sum_{s' \in E} \mu_{s',E}(a_{s'})$$

De beste configuratie $x \in B$ is te vinden door de eind bit kosten functies $\mu_s(x_s)$ te minimaliseren naar de waarde $x_s \in A_s$ voor elke $s \in N$.

4.3.2. DE STELLING

Stelling 4 (Min-sum algoritme)**.** *Als de check structuur eindig en cykel vrij is dan convergeert de kosten functie na eindig veel iteraties en voor de uiteindelijke kosten functies geldt dan :*

$$\mu_s(a) = \min_{x \in B : x_s = a} G(x)$$

en

$$\mu_E(a) = \min_{x \in B : x_E = a} G(x)$$

Het bewijs van deze stelling is te vinden in appendix in sectie A.1. De convergeer snelheid en de prestatie van dit algoritme is volledig afhankelijk van de code, waar het algoritme op toegepast wordt. Het is niet het algoritme met de beste prestatie, maar heeft wel een goede prestatie en wordt daarom toegepast in onder andere parallelle hardware. Door alle toepassingen heeft het algoritme veel vormen aangenomen. In de toepassing hieronder is een recente en minder algemene vorm van het algoritme beschreven.

4.4. EEN TOEPASSING

Er zal een toepassing van het algoritme gegeven worden zowel op papier als met Matlab. De toepassing is een vernieuwde versie van het hierboven beschreven algoritme. Het grootste verschil zit erin dat dit algoritme elke begin kosten functies die twee waarden heeft, vervangt door één getal. Daarnaast wordt 0 omgezet naar 1 en 1 naar -1. In het algoritme hierboven beschreven hebben 0 en 1 beide een eigen kans dat is nu ook zo, dus de begin kosten functies moeten gecombineerd worden in een getal, L. Hieronder worden de log likelihood ratio's van de begin kosten functies gecombineerd in L met onder andere de regel van Bayes en logaritme rekenregels:

$$L_s(x_s|y_s) = \log \frac{p(x_s=1|y_s)}{p(x_s=-1|y_s)} = \log \frac{p(x_s=1)}{p(x_s=-1)} + \log \frac{p(y_s|x_s=1)}{p(y_s|x_s=-1)}$$

Er wordt aangenomen dat de kansen $p(x_s=1)$ en $p(x_s=-1)$ gelijk zijn, dus beide 0.5 zijn. Dit betekent dat $\log \frac{p(x_s=1)}{p(x_s=-1)} = 0$, dus:

$$L_s(x_s|y_s) = \log \frac{p(y_s|x_s=1)}{p(y_s|x_s=-1)}$$

Merk op dat:

$$p(x_s=1|y_s) = \frac{e^{L_s(x_s|y_s}}{1+e^{L_s(x_s|y_s)}}$$

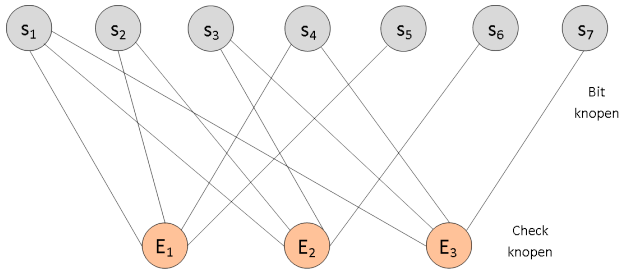
want:

$$L_s(x_s|y_s) = \log \frac{p(x_s=1|y_s)}{1-p(x_s=1|y_s)}$$

$$\Rightarrow e^{L_s(x_s|y_s)} = \frac{p(x_s=1|y_s)}{1-p(x_s=1|y_s)} \Rightarrow p(x_s=1|y_s) = \frac{e^{L_s(x_s|y_s)}}{1+e^{L_s(x_s|y_s)}}$$

De hierboven beschreven log likelihood ratio staat verder beschreven in [10].

Bij hard decision decoderen geeft het teken van L aan of er voor 1 of -1 gekozen moet worden. Er zal nu een voorbeeld volgen van het algoritme dat gebruik maakt van soft decision decoderen op een Hamming(7,4) code. Hieronder is de Tanner graaf te zien gemaakt uit de parity check matrix. Zoals is te zien in de Tanner graaf is deze code niet cykel vrij, toch zal blijken dat het algoritme al snel convergeert voor deze code.



Stel het woord 1010100 wordt na codering verzonden, dus $(-1;1;-1;1;-1;1;1)$. In het kanaal ondervindt het woord Gaussische ruis, vervolgens worden de log likelihood ratio's berekend. Dit gaat als volgt: Stel g is ontvangen uit het kanaal, dan:

$$p(g|x_s=1) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(g-1)^2}{2\sigma^2}}$$

en

$$p(g|x_s=-1) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(g+1)^2}{2\sigma^2}}$$

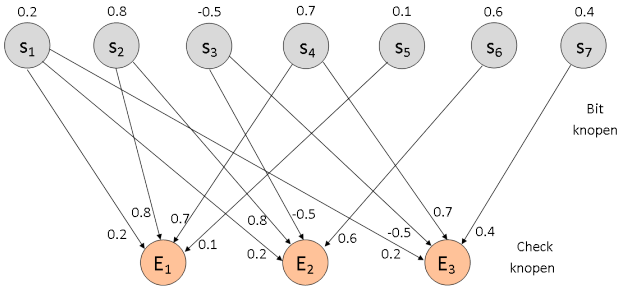
Dus de likelihood ratio is gelijk aan:

$$L_s(x_s|y_s) = \log \frac{\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(g-1)^2}{2\sigma^2}}}{\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(g+1)^2}{2\sigma^2}}} = \log e^{\frac{-(g-1)^2}{2\sigma^2} + \frac{-(g+1)^2}{2\sigma^2}}$$

$$= \frac{-(g-1)^2}{2\sigma^2} + \frac{-(g+1)^2}{2\sigma^2} = \frac{1}{2\sigma^2} * g$$

Zoals al eerder gemeld, geldt $N_0 = 2\sigma^2$, dus $L_s(x_s|y_s) = \frac{4}{N_0} * g$. Na het uitrekenen van de verscheidene log likelihood ratio's geldt $L = (0.2; 0.8; -0.5; 0.7; 0.1; 0.6; 0.4)$. Dit zijn dus de gecombineerde begin kosten functies. Stel het woord wordt na de log likelihood ratio naar hard decision vorm geschreven dan zijn er twee fouten op te merken, (**0**010**0**00).

In de eerste stap van de iteratie worden de waardes van de bit knopen naar de check knopen gestuurd ($\mu_{s,E}$), dit is te zien in de afbeelding en tabel hieronder:



Iteratie 1	$\mu_{s,E}$						
	s_1	s_2	s_3	s_4	s_5	s_6	s_7
L_s	0.2	0.8	−0.5	0.7	0.1	0.6	0.4
E_1	0.2	0.8		0.7	0.1		
E_2	0.2	0.8	−0.5			0.6	
E_3	0.2		−0.5	0.7			0.4

In de volgende stap worden de check verzamelingen naar de bit knopen gestuurd ($\mu_{E,s}$). Hier is het de bedoeling om de geldigheid van de bits te checken. Dit gebeurt door de log likelihood $L(x_{s_1} \oplus x_{s_2} \oplus \cdots \oplus x_{s_k})$ uit te rekenen, met $s_1, s_2, \ldots, s_k \in E$. Omdat in $\mathbb{F}_2$ wordt gewerkt en 1 de identiteit is, geldt:

$$1 \oplus 1 = -1 \oplus -1 = 1$$

$$1 \oplus -1 = 1 \oplus -1 = -1$$

Verder geldt:

$$L(x_1 \oplus x_2|y_1, y_2) = \log \frac{p(x_1 \oplus x_2 = 1|y_1, y_2)}{p(x_1 \oplus x_2 = -1|y_1, y_2)}$$

Nu geldt:

$$p(x_1 \oplus x_2 = 1|y_1, y_2) = p(x_1 = 1|y_1)p(x_2 = 1|y_2) + (1 - p(x_1 = 1|y_1))(1 - p(x_2 = 1|y_2))$$

$$\frac{e^{L(x_1|y_1)}e^{L(x_2|y_2)}}{(1+e^{L(x_1|y_1)})(1+e^{L(x_2|y_2)})} + \frac{1}{(1+e^{L(x_1|y_1)})(1+e^{L(x_2|y_2)})}$$

$$= \frac{1 + e^{L(x_1|y_1)}e^{L(x_2|y_2)}}{(1+e^{L(x_1|y_1)})(1+e^{L(x_2|y_2)})}$$

Op dezelfde manier geldt:

$$p(x_1 \oplus x_2 = -1|y_1, y_2) = \frac{e^{L(x_1|y_1)} + e^{L(x_2|y_2)}}{(1+e^{L(x_1|y_1)})(1+e^{L(x_2|y_2)})}$$

Dus:

$$L(x_1 \oplus x_2|y_1, y_2) = \log \frac{1 + e^{L(x_1|y_1)}e^{L(x_2|y_2)}}{e^{L(x_1|y_1)} + e^{L(x_2|y_2)}}$$

Dus voor $\mu_{E,s} = L(x_{s_1} \oplus x_{s_2} \oplus \cdots \oplus x_{s_k})$ en uit [10] volgt dat:

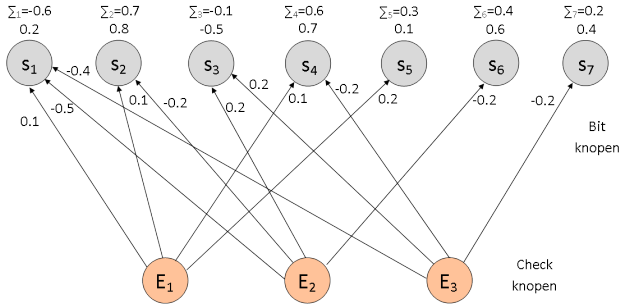
$$L(x_{s_1} \oplus x_{s_2} \oplus \cdots \oplus x_{s_k}) = \log \frac{1 + e^{L(x_{s_1})}e^{L(x_{s_2})} \cdots e^{L(x_{s_k})}}{e^{L(x_{s_1})} + e^{L(x_{s_2})} + \cdots + e^{L(x_{s_k})}}$$

$$\approx \text{sign}(L(x_{s_1})) \cdot \text{sign}(L(x_{s_2})) \cdots \text{sign}(L(x_{s_k})) \cdot \min(|L(x_{s_1})|, |L(x_{s_2})|, \cdots, |L(x_{s_k})|)$$

Hieronder is een klein voorbeeld gegeven voor check knoop E_1 en bit s_1:

$$\mu_{E_1,s_1} = (+1) * (+1) * (+1) * \min 0.8; 0.7; 0.1 = 0.1$$

Na elke volledige iteratie heeft elke bit knoop een waarde μ_s, welk de som is van L_s en de verkregen waardes van de aanliggende check knopen. Hieronder zijn een afbeelding en tabel met de waardes van $\mu_{E,s}$ en μ_s gegeven.



Iteratie 1	$\mu_{E,s}$						
	s_1	s_2	s_3	s_4	s_5	s_6	s_7
L_s	0.2	0.8	−0.5	0.7	0.1	0.6	0.4
E_1	0.1	0.1		0.1	0.2		
E_2	−0.5	−0.2	0.2			−0.2	
E_3	−0.4		0.2	−0.2			−0.2
μ_s	−0.6	0.7	−0.1	0.6	0.3	0.4	0.2

μ_s kan gebruikt worden om de hulpfunctie μ_{s,E_i} te bepalen. Deze hulpfunctie is namelijk de som van alle bijdragen $\mu_{E_j,s}$ behalve vanuit E_j plus de begin kosten functie, die in dit algoritme vervormd is naar L. Dus de hulpfunctie is te schrijven als $\mu_{s,E_j} = \mu_s - \mu_{E_j,s}$. In de volgende iteraties zal deze methode gebruikt worden om de bijdragen $\mu_{s,E}$ te berekenen. Daarnaast kan μ_s gezien worden als de bit eind kosten functie, echter de eind kosten is iets anders. Dit keer is het weer een getal in plaats van een functie, maar het getal wordt wel zoals in het Min-Sum algoritme berekend dus:

$$\mu_s = L_s + \sum_{E' \in Q: s \in E'} \mu_{E',s}$$

Het is de bedoeling $|\mu_s|$ te maximaliseren, want $|\mu_s|$ zegt iets over de zekerheid van het teken van de bit. Dit gezegd hebbende valt het op dat na de eerste iteratie de fout in het eerste bit weg is, maar bits 2,3,4,6,7 zijn lager in zekerheid geworden en bit 5 heeft een slechtere waarde verkregen. Dit laatste omdat de waarde eerst 0.1 was en nu 0.3, dus de bit heeft een grotere afstand gekregen tot een negatief teken.

Bij de volgende iteraties zal de hulpfunctie $\mu_{s,E}$ berekent worden zoals hierboven beschreven. Hieronder is de volgende iteratie te vinden. Iteratie 3 tot en met 6 zijn in de appendix te vinden in sectie A.2.

Iteratie 2	$\mu_{s,E}$						
	s_1	s_2	s_3	s_4	s_5	s_6	s_7
L_s	0.2	0.8	−0.5	0.7	0.1	0.6	0.4
E_1	−0.7	0.6		0.5	0.1		
E_2	−0.1	0.9	−0.3			0.6	
E_3	−0.2		−0.3	0.8			0.4
Iteratie 2	$\mu_{E,s}$						
E_1	0.1	−0.1		−0.1	−0.5		
E_2	−0.3	0.1	−0.1			0.1	
E_3	−0.3		−0.2	0.2			0.2
μ_s	−0.3	0.8	−0.8	0.8	−0.4	0.7	0.6

Na 6 iteraties is de eind kosten functie μ_s bepaald en de volgende vector verkregen: $(-0.7; 0.8; -0.9; 0.9; -0.7; 0.7; 0.7)$. Na afronding wordt (1010100) verkregen, we weten dat dit gelijk is aan het originele woord, de verbetering is

dus goed gegaan. Daarnaast zijn de absolute waarde allemaal 0.7 of hoger, dus er kan met een aanzienlijke zekerheid gezegd worden dat het originele woord (1010100) was. Daarnaast leverde het algoritme dit resultaat al in 6 iteraties, terwijl de code cykels in de Tanner graaf bevatte.

4.4.1. TOEPASSING IN MATLAB

In de appendix in de sectie A.3 staat een Matlab code de bovenstaande toepassing. Het bleek dat het algoritme na iteratie 6 al geconvergeerd was.

In de Matlab code is nog een tweede Hamming(7,4) decodering gedaan. Het originele woord uit de code lijst was gelijk aan (0110101), na verzending kwam $(-0.2;-0.9;-0.8;0.7;0.1;-0.1;-0.8)$ uit het kanaal. Als dit woord eerst afgerond zou worden wordt (1110011) verkregen. Hier zitten 3 fouten in. Na 4 iteraties was het algoritme al geconvergeerd en werd de vector $(0.2;-0.7;-0.3;0.2;-0.2;0.2;-0.4)$ verkregen. Na afronding wordt dit (0110101), er zijn drie fouten goed verbeterd! De absolute waarden zijn niet erg hoog, dus de ontvanger is na de decodering niet zeer zeker of de verbetering goed gegaan is.

Als hard decision decoding was gebruikt was de fout waarschijnlijk niet goed verbeterd. Hieronder is een voorbeeld gegeven, waarbij het woord opgelost wordt met een parity check matrix en bijbehorend syndroom. Ter illustratie van het betere fout-verbeterend vermogen van soft decision decodering, zal er nu hetzelfde voorbeeld als in de toepassing behandeld worden maar met Hard decision decodering, welk in dit geval maar een 1 fout-verbeterende code levert.

Voorbeeld 25 (Hard decision decodering)**.** *Net als hierboven is het woord* (1010100) *gecodeerd met Hamming(7,4) en verzonden door een kanaal met Gaussische ruis. Na de verzending wordt weer* $(0.2;0.8;-0.5;0.7;0.1;0.6;0.4)$ *ontvangen. Vervolgens wordt het afgerond naar* (0010000)*. De parity check matrix en syndroom tabel behorende bij de gebruikte Hamming code zien er als volgt uit:*

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

Error	*Syndroom*
0000000	*000*
1000000	*111*
0100000	*110*
0010000	*011*
0001000	*101*
0000100	*100*
0000010	*010*
0000001	*001*

Vervolgens wordt het afgeronde woord (0010000) *vermenigvuldigd met* H^T*.*

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Het gevonden syndroom is gelijk aan $\begin{bmatrix}0\\1\\1\end{bmatrix}$. *Dus de error is gelijk aan* $\epsilon = \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\end{bmatrix}$.

Het verbeterde code woord wordt nu:

$$\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\end{bmatrix} - \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix}$$

Het woord is dus fout verbeterd naar (0000000). *Dit woord heeft een nog grotere afstand tot het originele woord* (1010100) *dan het ontvangen woord* (0010000). *Het woord is dus fout verbeterd.*

5

Conclusie en Discussie

Er zijn talloos veel verschillende codes. Als er onderscheid plaats vindt door middel van prestatie wordt er al snel naar de Shannon limiet gekeken. De Shannon limiet geeft het best mogelijke prestatie vermogen van een codering weer. Maar de prestaties hangen niet alleen van de gebruikte code af maar ook van de decodering. De decodering die deze goede prestaties bij de LDPC en turbo codes oplevert is iteratief decoderen. Het Min-Sum algoritme, een iteratieve decoderings methode, is alleen bewezen voor cykel vrije codes. Desondanks wordt het Min-Sum algoritme in de praktijk veelvuldig toegepast op codes met cykels in de Tanner graaf. Het blijkt namelijk dat in de praktijk het algoritme vaak wel convergeert voor codes met cykels en als de hulpfuncties niet convergeren blijkt er meestal toch wel een duidelijk minimum voor de eind kosten functies. Daarnaast is gebleken uit [9] hoe groter de kleinste cykel in een Tanner graaf van de code hoe beter het Min-Sum algoritme presteert. Het is een groot voordeel dat het Min-Sum algoritme toegepast kan worden op codes met cykels in de Tanner graaf, want de best presterende codes van dit moment zijn de LDPC en Turbo codes en beide bevatten cykels.

Als in het vervolg een uitbreidend onderzoek plaats vindt op deze scriptie zou het presterend vermogen van het Min-Sum algoritme op codes met cykels onderzocht kunnen worden. De notatie van het Min-Sum algoritme gegeven in hoofdstuk 4 is zeer algemeen en verouderd, hierdoor is de versie die gebruikt is in de toepassing erg verschillend. Als er meer tijd was geweest, was het een interessante aanvulling geweest om verder onderzoek te doen in de verschillende versies van het Min-Sum algoritme. Daarnaast geldt dat het Min-Sum algoritme een gegeneraliseerde versie is van het Viterbi algoritme. Dit is niet genoemd in de scriptie omdat mijn kennis over het Viterbi algoritme erg gering is. Ook meer onderzoek in de richting van het Viterbi algoritme kan interessante resultaten opleveren. Daarnaast kan een uitgebreide studie van de LDPC of Turbo code interessant zijn, omdat deze codes een extreem goed presterend vermogen hebben. Tenslotte kan het interessant zijn om meer onderzoek te verrichten naar de Reed-Solomon code, welk de code is achter cd-spelers.

A

APPENDIX

A.1. BEWIJS MIN-SUM ALGORTIME

Hier volgt het bewijs van de stelling zoals gegeven in [9].

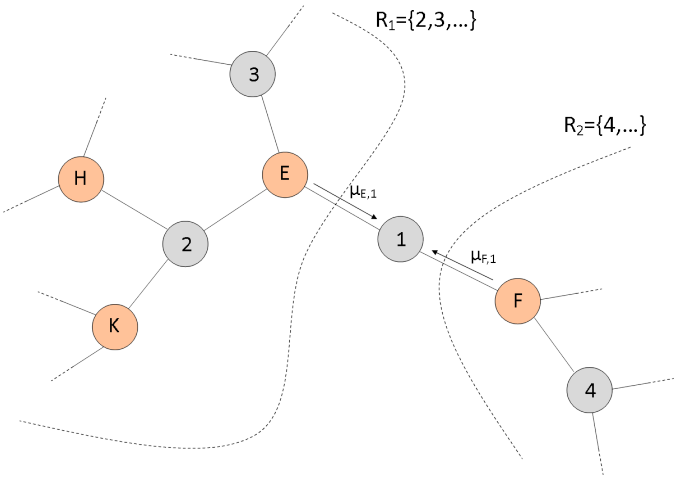
Bewijs. Laat Q een cykel vrije check structuur zijn op de tripel (N, W, B). Voor een willekeurige deelverzameling $R \subseteq N$ wordt de volgende notatie voor de globale kosten functie die correspondeert met R ingevoerd:

$$G_R(x_R) := \sum_{E \in Q : E \subseteq R} \gamma_E(x_E) + \sum_{s \in R} \gamma_s(x_s)$$

In het speciale geval dat $R = N$, geldt er $G_N(x) = G(x)$. Omdat R willekeurig gekozen wordt geldt dit bewijs voor de hele graaf. In de stelling is een van de eisen dat de Tanner graaf geen cykels bevat, dus elke Tanner graaf is een boom. In het bewijs zal er stap voor stap door de boom gelopen worden tot er een blad bereikt is.

DE EERSTE EXPRESSIE

Eerst zal de expressie $\mu_s(a) = \min_{x \in B : x_s = a} G(x)$ aangetoond worden. Dit met een recursieve methode, waarbij stap voor stap teruggewerkt zal worden in de boom, totdat een blad wordt bereikt, waar alleen de lokale kosten functies overblijven. Om de notatie overzichtelijk te houden zal het bewijs toegepast worden op de algemene checkstructuur zoals weergegeven in de afbeelding hieronder. Deze structuur is zo algemeen dat het bewijs werkt op elke structuur doordat alle andere gevallen symmetrisch zullen zijn aan deze. Het verschil met andere structuren zal zitten in het aantal takken waar elke bit knoop en check knoop aan verbonden zijn. Als er meer takken in de graaf zijn, zal R opgesplitst moeten worden in nog meer deelverzamelingen $R_i \in R$ dan in deze algmene checkstructuur.



De eindkosten functie behorende bij bit 1 in de afbeelding hierboven is gelijk aan:

$$\mu_1(a) = \gamma_1(a) + \mu_{E,1}(a) + \mu_{F,1}$$

Om deze uitdrukking verder uit te werken zal de restrictie R opgedeeld worden in $\{1\}$, R_1 en R_2. Waarbij R_1 bestaat uit alle bit knopen s die middels een pad via E met knoop 1 verbonden zijn. Zo is R_2 alle bit knopen s die via F verbonden zijn met bit knoop 1. Nu geldt dat $R = N$ de disjuncte vereniging van R_1, R_2 en $\{1\}$ is.

Stap 1: de eind kosten functies μ_s
Beschouw nu de eind kosten functie $\mu_1(a) = \gamma_1(a) + \mu_{E,1}(a) + \mu_{F,1}(a)$ voor een willekeurige $a \in A_1$ en bit knoop 1, zoals in het figuur hierboven. Laat nu $\nu_1(a) := \min_{x \in B: x_1 = a} G(x)$ zijn, dan wordt er in het vervolg aangetoond dat $\mu_1(a) = \nu_1(a)$. Er zullen ook ν hulpfuncties aangemaakt worden om dit doel te bereiken. Eerst ter herinnering: er geldt $G(x) = \sum_{E \in Q} \gamma_E(x_E) + \sum_{s \in N} \gamma_s(x_s)$, en omdat R opgedeeld is geldt er $\sum_{s \in N} \gamma_s(x_s) = \gamma_1(x_1) + \sum_{s \in R_1} \gamma_s(x_s) + \sum_{s \in R_2} \gamma_s(x_s)$. Verder geldt $\sum_{K \in Q} \gamma_K(x_K) = \gamma_E(x_E) + \gamma_F(x_F) + \sum_{K \in R_1} \gamma_K(x_K) + \sum_{K \in R_2} \gamma_K(x_K)$. Nu zal $\nu_1(a)$ herschreven worden:

$$\nu_1(a) = \min_{x \in B: x_1 = a} G(x) = \min_{x \in B: x_1 = a} \{\gamma_1(a) + G_{R_1}(x_{R_1}) + \gamma_E(x_E) + G_{R_2}(x_{R_2}) + \gamma_F(x_F)\}$$

De γ_E en γ_F zitten apart in de som omdat ze beide afhangen van bit 1 en die zit niet in R_1 of R_2. In de volgende stap wordt het minimum gesplitst in twee minima. In deze minima wordt bit 1 er ook bij getrokken, omdat check knopen E en F afhangen van bit knoop 1.

$$= \gamma_1(a) + \min_{x_{R_1 \cup \{1\}} \in B_{R_1 \cup \{1\}}: x_1 = a} \{G_{R_1}(x_{R_1}) + \gamma_E(x_E)\} + \min_{x_{R_2 \cup \{1\}} \in B_{R_2 \cup \{1\}}: x_1 = a} \{G_{R_2}(x_{R_2}) + \gamma_F(x_F)\}$$

Nu zullen $\nu_{E,1}(a)$ en $\nu_{F,1}(a)$ gedefinieerd worden als:

$$\nu_{E,1}(a) = \min_{x_{R_1 \cup \{1\}} \in B_{R_1 \cup \{1\}}: x_1 = a} \{G_{R_1}(x_{R_1}) + \gamma_E(x_E)\}$$

en

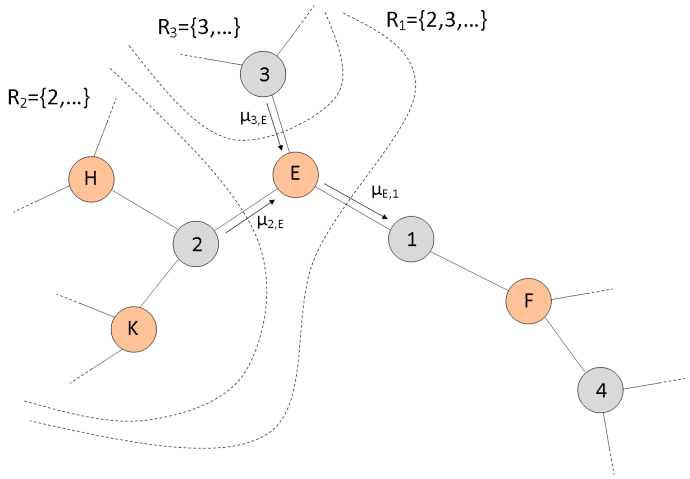
$$\nu_{F,1}(a) = \min_{x_{R_2 \cup \{1\}} \in B_{R_2 \cup \{1\}}: x_1 = a} \{G_{R_2}(x_{R_2}) + \gamma_F(x_F)\}$$

Dus dan:

$$\nu_1(a) = \gamma_1(a) + \nu_{E,1}(a) + \nu_{F,1}(a)$$

Er zal nu aangetoond worden dat $\nu_{E,1} = \mu_{E,1}$ en $\nu_{F,1} = \mu_{F,1}$. Dan is ν_1 gelijk aan de eind kosten functie μ_1. Er zal alleen aangetoond worden dat $\nu_{E,1} = \mu_{E,1}$, de andere gaat namelijk analoog, want het is een symmetrische situatie.

Stap 2: de hulp kosten functie $\mu_{E,s}$
In de afbeelding hieronder is de splitsing van R_1 in R_2, R_3 en $\{E\}$ te zien. Deze splitsing vindt plaats om $\mu_{E,1}(a)$ te berkenen, waarvan we gaan aantonen dat het gelijk is aan $\nu_{E,1}(a)$.



Beschouw voor een willekeurige $a \in A_1$ en bit knoop 1 de functie:

$$\mu_{E,1}(a) = min_{x_E \in B_E: x_1 = a} \{\gamma_E(x_E) + \mu_{2,E}(x_2) + \mu_{3,E}(x_3)\}$$

Het doel is nu te laten zien dat $\nu_{E,1} = \mu_{E,1}$. Eerst kan $\nu_{E,1}$ opgebroken worden in drie onafhankelijke delen: De lokale check knoop kosten functie $\gamma_E(x_E)$ en twee kosten functies geassocieerd met de deelverzamelingen R_2 en R_3. Er geldt:

$$\nu_{E,1}(a) = \min_{x_{R_1\cup\{1\}} \in B_{R_1\cup\{1\}} : x_1 = a} \{\gamma_E(x_E) + G_{R_1}(x_{R_1})\}$$

Merk op:

$$\gamma_E(x_E) + G_{R_1}(x_{R_1}) = \gamma_E(x_E) + \sum_{K\in R_1} \gamma(x_K) + \sum_{s\in R_1} \gamma(x_s)$$

$$= \gamma_E(x_E) + \sum_{K\in R_2\cup R_3} \gamma(x_K) + \sum_{s\in R_2\cup R_3} \gamma(x_s)$$

Daarnaast geldt dat als $K \in R_1$ dan $K \in R_2$ of $K \in R_3$, want er is geen $K \in R_1$ die knopen van R_2 en R_3 bevat. Behalve E, maar die bevat ook knopen buiten R_1. Dus geldt er:

$$= \gamma_E(x_E) + \sum_{K\in R_2} \gamma(x_K) + \sum_{K\in R_3} \gamma(x_K) + \sum_{s\in R_2} \gamma(x_s) + \sum_{s\in R_3} \gamma(x_s)$$

$$= \gamma_E(x_E) + G_{R_2}(x_{R_2}) + G_{R_3}(x_{R_3})$$

Hieruit volgt nu dat:

$$\nu_{E,1}(a) = \min_{x_{R_1\cup\{1\}} \in B_{R_1\cup\{1\}} : x_1 = a} \{\gamma_E(x_E) + G_{R_2}(x_{R_2}) + G_{R_3}(x_{R_3}))\} = (*)$$

Er geldt dat dit gelijk is aan:

$$= \min_{x_E \in B_E : x_1 = a} \gamma_E(x_E) + \min_{x'_{R_2} \in B_{R_2} : x'_2 = x_2} G_{R_2}(x'_{R_2}) + \min_{x'_{R_3} \in B_{R_3} : x'_3 = x_3} G_{R_3}(x'_{R_3}) = (**)$$

Dit zal bewezen worden door eerst te laten zien dat $(**) \leq (*)$. En vervolgens dat $(*) \leq (**)$.

$$(**) \leq (*):$$

Neem aan er is een woord

$$x_{R_1\cup\{1\}} = \min_{x_{R_1\cup\{1\}} \in B_{R_1\cup\{1\}} : x_1 = a} \{\gamma_E(x_E) + G_{R_2}(x_{R_2}) + G_{R_3}(x_{R_3}))\}$$

Nu laten zien dat dat woord ook in $(**)$ zit. $x_{R_1\cup\{1\}}$ is ook te schrijven als $x_1 = a, x_2, x_3, x_{R_2\setminus 2}, x_{R_3\setminus 3}$. Dus $x_1 = a, x_2, x_3 \in B_E$, $x_2, x_{R_2\setminus 2} \in B_{R_2}$ en $x_3, x_{R_3\setminus 3} \in B_{R_3}$. Dus $x_{R_1\cup\{1\}} \in (**)$ dus $(**) \leq (*)$.

$$(*) \leq (*):$$

Stel dat

$$x_1 = a, x_2, x_3 = \min_{x_E \in B_E : x_1 = a} \gamma_E(x_E) + \min_{x'_{R_2} \in B_{R_2} : x'_2 = x_2} G_{R_2}(x'_{R_2}) + \min_{x'_{R_3} \in B_{R_3} : x'_3 = x_3} G_{R_3}(x'_{R_3})$$

Neem vervolgens $x_2, x_{R_2\setminus 2} \in B_{R_2}$ die $G_{R_2}(x_{R_2})$ minimaliseert en $x_3, x_{R_3\setminus 3} \in B_{R_3}$ die $G_{R_3}(x_{R_3})$ minimaliseert. Dus $(*) \leq (**)$.

De eerste twee elementen van de laatste vergelijking zullen nu kortweg genoteerd worden als:

$$\nu_{2,E}(x_2) = \min_{x'_{R_2} \in B_{R_2} : x'_2 = x_2} G_{R_2}(x'_{R_2})$$

en

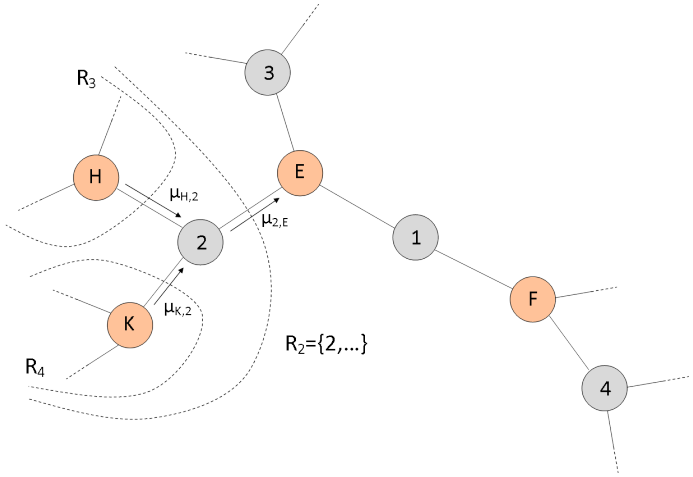
$$\nu_{3,E}(x_3) = \min_{x'_{R_3} \in B_{R_3} : x'_3 = x_3} G_{R_3}(x'_{R_3})$$

Dan is

$$\nu_{E,1}(a) = \min_{x_E \in B_E : x_1 = a} \{\gamma_E(x_E) + \nu_{2,E}(x_2) + \nu_{3,E}(x_3)\}$$

Dus heeft de expressie dezelfde structuur als $\mu_{E,1}$. Om te laten zien dat $\nu_{E,1} = \mu_{E,1}$, moet er nu aangetoond worden dat $\nu_{2,E}(x_2) = \mu_{2,E}(x_2)$ en $\nu_{3,E}(x_3) = \mu_{3,E}(x_3)$. Deze situatie is weer symmetrisch dus het is voldoende om $\nu_{2,E}(x_2) = \mu_{2,E}(x_2)$ aan te tonen.

Stap 3: de hulp kosten functies $\mu_{s,E}$
De onderstaande afbeelding illustreert de berekening van $\mu_{2,E}(a)$.



De functie behorende bij de afbeelding hierboven is:

$$\mu_{2,E}(a) = \gamma_2(a) + \mu_{H,2} + \mu_{K,2}$$

Dus nu gaat er aangetoond worden dat deze gelijk is aan $\nu_{2,E}(a)$. Hiervoor zal ook $\nu_{2,E}(a)$ opgesplitst worden in drie onafhankelijke delen: de lokale bit knoop kosten functie $\gamma_2(a)$ en twee kosten functies geassocieerd met de deelverzamelingen R_3 en R_4.

$$\nu_{2,E}(a) = \min_{x_{R_2} \in B : x_2 = a} G_{R_2}(x_{R_2})$$

$$= \min_{x_{R_2} \in B : x_2 = a} \{\gamma_2(a) + G_{R_3}(x_{R_3}) + \gamma_H(x_H) + G_{R_4}(x_{R_4}) + \gamma_K(x_K)\}$$

$$= \gamma_2(a) + \min_{x_{R_3 \cup \{2\}} \in B_{R_3 \cup \{2\}} : x_2 = a} \{G_{R_3}(x_{R_3}) + \gamma_H(x_H)\} + \min_{x_{R_4 \cup \{2\}} \in B_{R_4 \cup \{2\}} : x_2 = a} \{G_{R_4}(x_{R_4}) + \gamma_K(x_K)\}$$

Dit is op dezelfde manier afgeleid als in stap 1. De laatste twee termen kunnen net als eerder genoteerd worden als:

$$\nu_{H,2}(a) = \min_{x_{R_3 \cup \{2\}} \in B_{R_3 \cup \{2\}} : x_2 = a} \{G_{R_3}(x_{R_3}) + \gamma_H(x_H)\}$$

en

$$\nu_{K,2}(a) = \min_{x_{R_4 \cup \{2\}} \in B_{R_4 \cup \{2\}} : x_2 = a} \{G_{R_4}(x_{R_4}) + \gamma_K(x_K)\}$$

Vervolgens geldt:

$$\nu_{2,E}(a) = \gamma_2(a) + \nu_{H,2}(a) + \nu_{K,2}(a)$$

Nu heeft de expressie $\nu_{2,E}(a)$ dezelfde vorm als $\mu_{2,E}(a)$. De volgende stap is aantonen dat $\nu_{H,2}(a) = \mu_{H,2}(a)$. Dit kan op dezelfde manier als in Stap 2 gedaan worden.

De laatste twee stappen in dit proces kunnen recursief herhaald worden tot er een bit blad bereikt is. Bladeren zijn de bit knopen die maar met één check knoop verbonden zijn. Deze bladeren bestaan omdat er geen cykels in de graaf zitten en de graaf dus een boom is. In zo'n blad zal er gelden dat:

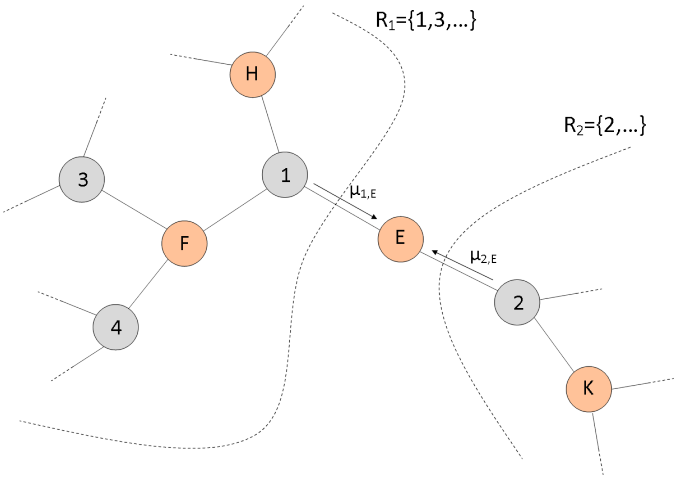
$$\nu_{s,T}(x_S) = \min_{x_s \in B_s} G_{\{s\}}(x_s) = \gamma_s(x_s)$$

Vervolgens kan er weer terug gegaan worden in de boom en dan volgt daaruit:

$$\mu_s(a) = \min_{x \in B : x_s = a} G(x)$$

DE TWEEDE EXPRESSIE

Ten tweede zal de expressie $\mu_E(a) = \min_{x \in B: x_E = a} G(x)$ aangetoond worden. Ook in dit bewijs zal er recursief teruggewerkt worden in de boom, totdat een blad wordt bereikt, waar alleen de lokale kosten functies overblijven. Het bewijs zal veel lijken op het vorige bewijs, alleen de eerste stap is voornamelijk anders. Om ook hier de notatie overzichtelijk te houden zal er weer op de algemene checkstructuur gewerkt worden. Alle andere structuren zullen symmetrisch met deze bewezen kunnen worden en dus is het bewijs op de check structuur in de afbeelding hieronder voldoende.



Stap 1: de eind kosten functies μ_s
Beschouw de eind kosten functie voor een willekeurige $a \in W_E$ en check structuur E, zoals in de afbeelding hierboven.

$$\mu_E(a) = \gamma_E(a) + \mu_{1,E}(a_1) + \mu_{2,E}(a_2)$$

Noteer nu $\nu_E(a) := \min_{x \in B: x_E = a} G(x)$ en herschrijf $\nu_E(a)$ vervolgens, zoals al voorgedaan is in de vorige expressie:

$$\nu_E(a) = \min_{x \in B: x_E = a} G(x) = \min_{x \in B: x_E = a} \{\gamma_E(a) + G_{R_1}(x_{R_1}) + G_{R_2}(x_{R_2})\}$$

$$= \gamma_E(a) + \min_{x_{R_1} \in B: x_E = a} \{G_{R_1}(x_{R_1})\} + \min_{x_{R_2} \in B: x_E = a} \{G_{R_2}(x_{R_2})\}$$

Vervolgens zullen de volgende notaties toegepast worden:

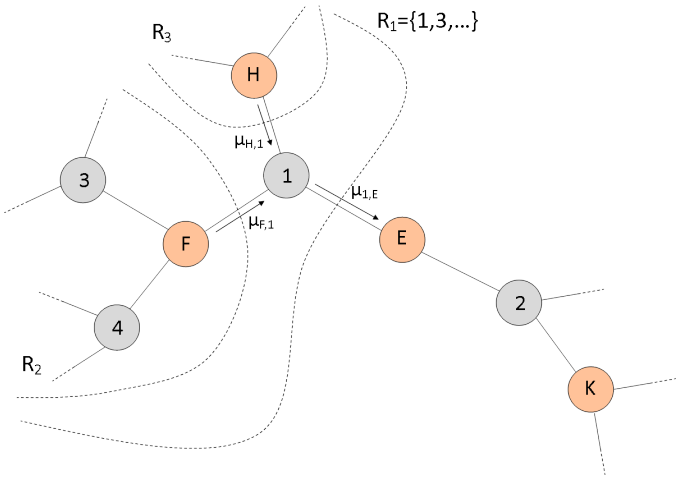
$$\nu_{1,E}(x_1) = \min_{x_{R_1} \in B: x_E = a} \{G_{R_1}(x_1)\}$$

en

$$\nu_{2,E}(x_2) = \min_{x_{R_2} \in B: x_E = a_2} \{G_{R_2}(x_2)\}$$

Dan geldt $\nu_E(a) = \gamma_E(a) + \nu_{1,E}(x_1) + \nu_{2,E}(x_2)$ dus ν_E is van de vorm μ_E dan en slechts dan als $\nu_{1,E} = \mu_{1,E}$ en $\nu_{2,E} = \mu_{2,E}$. Alleen de gelijkheid $\nu_{1,E}(a_1) = \mu_{1,E}(a_1)$ wordt aangetoond, want het andere geval is symmetrisch.

Stap 2: de hulp kosten functies $\mu_{s,E}$
Hieronder is een schematische weergave van de berekening van $\mu_{1,E}$ weergegeven.



De volgende formule hoort bij de afbeelding:

$$\mu_{1,E}(a) = \gamma_1(a) + \mu_{H,1}(a) + \mu_{F,1}(a)$$

Het doel is om te laten zien dat deze formule gelijk is aan $\nu_{1,E}(a)$. Daarom wordt $\nu_{1,E}(a)$ opgesplitst in drie onafhankelijke delen: twee kosten functies behorende bij de deelverzamelingen R_2 en R_3 en een lokale kosten functie.

$$\begin{aligned}\nu_{1,E}(a) &= \min_{x_{R_1} \in B_{R_1} : x_E = a} \{G_{R_1}(x_{R_1})\} \\ &= \min_{x_{R_1} \in B_{R_1} : x_E = a} \{\gamma_1(a) + G_{R_2}(x_{R_2}) + \gamma_F(x_F) + G_{R_3}(x_{R_3}) + \gamma_H(x_H)\} \\ &= \gamma_1(a) + \min_{x_{R_2 \cup \{1\}} \in B_{R_2 \cup \{1\}} : x_E = a} \{G_{R_2}(x_{R_2}) + \gamma_F(x_F)\} + \min_{x_{R_3 \cup \{1\}} \in B_{R_3 \cup \{1\}} : x_E = a} \{G_{R_3}(x_{R_3}) + \gamma_H(x_H)\}\end{aligned}$$

Vervolgens zullen de functies $\nu_{F,1}$ en $\nu_{H,1}$ genoteerd worden als:

$$\nu_{F,1}(a) = \min_{x_{R_2 \cup \{1\}} \in B_{R_2 \cup \{1\}} : x_E = a} \{G_{R_2}(x_{R_2}) + \gamma_F(x_F)\}$$

en

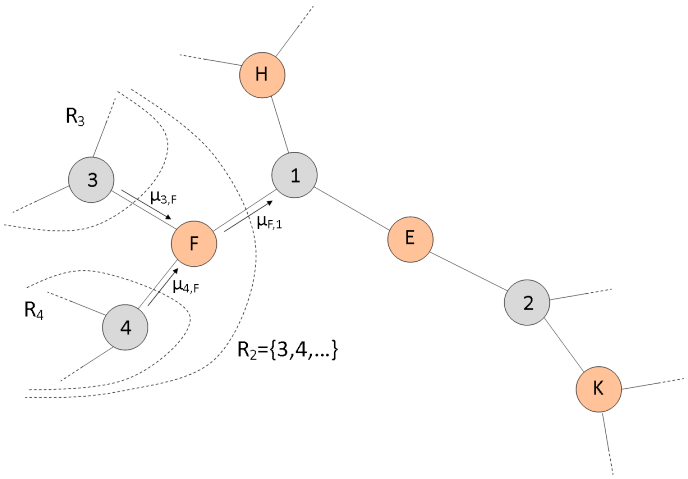
$$\nu_{H,1}(a) = \min_{x_{R_3 \cup \{1\}} \in B_{R_3 \cup \{1\}} : x_E = a} \{G_{R_3}(x_{R_3}) + \gamma_H(x_H)\}$$

dus

$$\nu_{1,E}(a) = \gamma_1(a) + \nu_{F,1}(a) + \nu_{H,1}(a)$$

Dan geldt $\nu_{1,E} = \mu_{1,E}$ dan en slechts dan als $\nu_{F,1} = \mu_{F,1}$ en $\nu_{H,1} = \mu_{H,1}$. Daarom wordt nu de vergelijking $\nu_{F,1} = \mu_{F,1}$ aangetoond, de ander gaat symmetrisch.

Stap 3: de hulp kosten functies $\mu_{E,s}$
In de afbeelding hieronder is een schematische weergave van de volgende berekening gegeven:



Beschouw de gelijkheid

$$\mu_{F,1} = min_{x_F \in B_F : x_E = a}\{\gamma_F(x_F) + \mu_{3,F}(x_3) + \mu_{4,F}(x_4)\}$$

Nu zal $\nu_{F,1}$ opgebroken worden in drie onafhankelijke delen:

$$\nu_{F,1}(a) = \min_{x_{R_2 \cup \{1\}} \in B_{R_2 \cup \{1\}} : x_E = a} \{\gamma_F(x_F) + G_{R_2}(x_{R_2})\}$$

$$= \min_{x_{R_2 \cup \{1\}} \in B_{R_2 \cup \{1\}} : x_E = a} \{\gamma_F(x_F) + G_{R_3}(x_{R_3}) + G_{R_4}(x_{R_4})\}$$

$$= \min_{x_F \in B_F : x_E = a} \{\gamma_F(x_F) + \min_{x'_{R_3} \in B_{R_3} : x'_{R_3} = x_{R_3}} G_{R_3}(x'_{R_3}) + \min_{x'_{R_4} \in B_{R_4} : x'_{R_4} = x_{R_4}} G_{R_4}(x'_{R_4})\}$$

De laatste termen zullen, voor overzichtelijkheid van de notatie, genoteerd worden als:

$$\nu_{3,F}(x_3) = \min_{x'_{R_3} \in B_{R_3} : x'_{R_3} = x_{R_3}} G_{R_3}(x'_{R_3})$$

en

$$\nu_{4,F}(x_4) = \min_{x'_{R_4} \in B_{R_4} : x'_{R_4} = x_{R_4}} G_{R_4}(x'_{R_4})$$

dus

$$\nu_{F,1}(a) = \min_{x_F \in B_F : x_E = a} \{\gamma_F(x_F) + \nu_{3,F}(x_3) + \nu_{4,F}(x_4)\}$$

$\nu_{F,1}$ heeft nu de juiste structuur. Om te laten zien dat $\nu_{F,1} = \mu_{F,1}$, moet aangetoond worden dat $\nu_{3,F} = \mu_{3,F}$ en $\nu_{4,F} = \mu_{4,F}$. Dit kan aangetoond worden op dezelfde manier als bij Stap 2.

De laatste twee stappen in dit proces kunnen recursief herhaald worden tot er een bit blad bereikt is. Bladeren zijn de bit knopen die maar met één check knoop verbonden zijn. Deze bladeren bestaan omdat er geen cykels in de graaf zitten en de graaf dus een boom is. In zo'n blad zal er gelden dat:

$$\nu_{s,T}(x_S) = \min_{x_s \in B_s} G_{\{s\}}(x_s) = \gamma_s(x_s)$$

Vervolgens kan er weer terug gegaan worden in de boom en dan volgt daaruit:

$$\mu_E(a) = \min_{x \in B : x_E = a} G(x)$$

□

A.2. TABELLEN

Hieronder de tabellen drie iteraties van de toepassing van het Min-Sum algoritme.

Iteratie 3	$\mu_{s,E}$						
	s_1	s_2	s_3	s_4	s_5	s_6	s_7
L_i	0.2	0.8	−0.5	0.7	0.1	0.6	0.4
E_1	−0.4	0.9		0.9	0.1		
E_2	0	0.7	−0.7			0.6	
E_3	0		−0.6	0.6			0.4
Iteratie 3	$\mu_{E,s}$						
E_1	0.1	−0.1		−0.1	−0.4		
E_2	−0.6	0	0			0	
E_3	−0.4		0	0			0
$\sum_i$	−0.7	0.7	−0.5	0.6	−0.3	0.6	0.4

Iteratie 4	$\mu_{s,E}$						
	s_1	s_2	s_3	s_4	s_5	s_6	s_7
L_i	0.2	0.8	−0.5	0.7	0.1	0.6	0.4
E_1	−0.8	0.8		0.7	0.1		
E_2	−0.1	0.7	−0.5			0.6	
E_3	−0.3		−0.5	0.6			0.4
Iteratie 4	$\mu_{E,s}$						
E_1	0.1	−0.1		−0.1	−0.7		
E_2	−0.5	0.1	−0.1			0.1	
E_3	−0.4		−0.3	0.3			0.3
$\sum_i$	−0.6	0.8	−0.9	0.9	−0.6	0.7	0.7

Iteratie 5	$\mu_{s,E}$						
	s_1	s_2	s_3	s_4	s_5	s_6	s_7
L_i	0.2	0.8	−0.5	0.7	0.1	0.6	0.4
E_1	−0.7	0.9		1	0.1		
E_2	−0.1	0.7	−0.8			0.6	
E_3	−0.2		−0.6	0.6			0.4
Iteratie 5	$\mu_{E,s}$						
E_1	0.1	−0.1		−0.1	−0.7		
E_2	−0.6	0.1	−0.1			0.1	
E_3	−0.4		−0.2	0.2			0.2
$\sum_i$	−0.7	0.8	−0.8	0.8	−0.6	0.7	0.6

Iteratie 6	$\mu_{s,E}$						
	s_1	s_2	s_3	s_4	s_5	s_6	s_7
L_i	0.2	0.8	−0.5	0.7	0.1	0.6	0.4
E_1	−0.8	0.9		0.9	0.1		
E_2	−0.1	0.7	−0.7			0.6	
E_3	−0.3		−0.6	0.6			0.4
Iteratie 6	$\mu_{E,s}$						
E_1	0.1	−0.1		−0.1	−0.8		
E_2	−0.6	0.1	−0.1			0.1	
E_3	−0.4		−0.3	0.3			0.3
$\sum_i$	−0.7	0.8	−0.9	0.9	−0.7	0.7	0.7

A.3. MATLAB CODE

Hieronder is de Matlab code en zijn functies van de toepassing van het Min-Sum algoritme gegeven. De code is aan te passen naar andere codes dan de Hamming(7,4) door een andere input text file te geven in MakeConnection2.

```
clear all

iterations = 10; %aantal iteraties

L = [-0.2, -0.9, -0.8, 0.7, 0.1, -0.1, -0.8]; %De ontvangen pulsen

%Hieronder wordt bepaald welke connecties er zijn in de Tanner graaf
[ connectionMatrix ] = MakeConnection2();
%Hieronder wordt de eerste iteratie uitgevoerd
[ nodeToCheck ] = FirstIteration(L, connectionMatrix);
[ checkToNode ] = MakeCheckToNode(nodeToCheck, connectionMatrix);

%Na de eerste iteratie wordt de som bepaalt
[ sumMatrix ] = sumCalculator(L, checkToNode);

%Om bij te houden wanneer de som convergeert is er een matrix die de som
%elke iteratie noteert.
ConvergatieMatrix = zeros(size(L,2),iterations);
ConvergatieMatrix(:,1)=sumMatrix;

for i = 1:iterations-1
    %Hier volgen de verdere iteraties
    [ nodeToCheck ] = MakeNodeToCheck( nodeToCheck, checkToNode, sumMatrix, connectionMatrix );
    [ checkToNode ] = MakeCheckToNode(nodeToCheck, connectionMatrix);
    [ sumMatrix ] = sumCalculator(L, checkToNode);
    ConvergatieMatrix(:,i+1)=sumMatrix;
end
```

```
function [ connectionMatrix ] = MakeConnection2()
%Hier worden de connecties van de graaf ingelezen. De input moet een soort
%tabel zijn met een 1 als er een connectie is tussen de desbetreffende bit
%en check knoop en anders een 0.
connectionMatrix = dlmread('connections.txt');
end
```

```
function [ nodeToCheck ] = FirstIteration(L, connectionMatrix)
%Hier wordt de eerste iteratie voor de hulpfunctie van de bijdrage van bit knoop naar check
%knoop uitgevoerd. De connectieMatrix wordt gebruikt om te checken of er
%een tak is tussen twee knopen.
 for i=1:size(connectionMatrix,1)
     for j=1:size(connectionMatrix,2)
         if(connectionMatrix(i,j)==1)
             nodeToCheck(i,j)=L(1,j);
         end
     end
 end


end
```

```
function [ checkToNode ] = MakeCheckToNode(nodeToCheck, connectionMatrix)
%Hier worden de iteraties voor de hulpfunctie van de bijdrage van check
%knoop naar bit knoop uitgevoerd. De connectieMatrix wordt gebruikt om te checken of er
%een tak is tussen twee knopen.
for i=1:3
    for j=1:7
        if(connectionMatrix(i,j)~=0)
            signTemp=1;
            minCheck = zeros(1,1);
            teller = 1;
            for k=1:7
                if(connectionMatrix(i,k)~=0 && k~=j)
                    signTemp = sign(signTemp*nodeToCheck(i,k));
                    minCheck(teller,1)=abs(nodeToCheck(i,k));
                    teller=teller+1;
                end
            end
            checkToNode(i,j)=signTemp*min(minCheck);
        end
    end

end
```

```
function [ sumMatrix ] = sumCalculator(L, checkToNode)
%Hier wordt de som bepaald na elke iteratie.
sumMatrix = L;
for i =1:7
    for j = 1:3
        sumMatrix(1,i) = sumMatrix(1,i) + checkToNode(j,i);
    end
end


end
```

```
function [ nodeToCheck ] = MakeNodeToCheck( nodeToCheck, checkToNode, sumMatrix, connectionMatrix )
%Hier worden de iteraties voor de hulpfunctie van de bijdrage van check
%knoop naar bit knoop uitgevoerd. De connectieMatrix wordt gebruikt om te checken of er
%een tak is tussen twee knopen.
for i=1:7
    for j=1:3
        if(connectionMatrix(j,i)~=0)
            nodeToCheck(j,i)=sumMatrix(1,i)-checkToNode(j,i);
        end
    end
end


end
```

BIBLIOGRAFIE

[1] Bernard Sklar and Fredric J. Harris, *The ABCs of Linear Block Codes*, IEEE Signal Processing Magazine, volume 4, pages 14-35, 2004.

[2] W. Cary Huffman and Vera Pless, *Fundamentals of Error Correcting Codes*, 1st edition, pages 121-207, 2003.

[3] Darel W. Hardy, Fred Richman and Carol L. Walker, *Aplied Algebra, codes, ciphers and discrete algorithms*, 2nd edition, pages 199-239, 2009.

[4] Raymond Hill, *A First Course in Coding Theory*, 1st edition, pages 125-161, 1986.

[5] Fraleigh Beauregard, *Linear Algebra*, 3rd edition, 1995.

[6] Yu Kou, Shu Lin and Marc P.C. Fossorier, *Low-Density Parity-Check Codes Based on Finite Geometries: A Rediscovery and New Results*, IEEE Transactions On Information Theory, volume 47, pages 2711-2736, 1981.

[7] Jos Weber, *Error-Correcting Codes*, et4-030 Lecture Notes, 2013.

[8] R. Michael Tanner, *A recursive Approach to Low Complexity Codes*, IEEE Transactions On Information Theory, volume 27, pages 533-547, 1981.

[9] Niclas Wiberg, Hans-Andrea Loeliger and Ralf Kötter, *Codes and Iterative Decoding on General Graphs*, 1995.

[10] Joachim Hagenauer, Elke Offer and Lutz Papke, *Iterative Decoding of Binary Block and Convolutional Codes*, IEEE Transactions On Information Theory, volume 42, pages 429-445, 1996.