TULOKOODEISTA JA NIIDEN DEKOODAUKSESTA

Iiris Mattila

Pro gradu -tutkielma
Toukokuu 2023

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

Iiris Mattila: Tulokoodeista ja niiden dekoodauksesta
Pro gradu -tutkielma, 40 s.
Matematiikka
Toukokuu 2023

Tulokoodit ovat matriisimuotoisia koodeja. Tulokoodin sana muodostetaan järjestämällä kahden lähtökoodin sanat matriisimuotoon niin, että vaakarivit ovat toisen lähtökoodin sanoja ja pystyrivit toisen lähtökoodin sanoja. Tulokoodien dekoodauksessa hyödynnetään lähtökoodien dekoodausalgoritmeja.

Tutkielmassa aloitetaan tulokoodien dekoodauksen tutkiminen dekoodaamalla tulokoodin sana ensin pystyriveittäin pystyrivikoodin dekooderilla ja sitten vaakariveittäin vaakarivikoodin dekooderilla. Havaitaan, että virhevektorin kuvio vaikuttaa siihen, saadaanko virhe dekoodattua. Eräillä kuvioilla pystytään dekoodaamaan yli puoleen minimietäisyydestä, mutta tiettyjä virhekuvioita ei saada tällä dekooderilla dekoodattua, vaikka niiden paino alittaisikin koodin virheenkorjauskyvyn.

Tutkielmassa esitellään kaksi kehittyneempää tulokoodien dekoodausalgoritmia, joilla saadaan korjattua kaikki minimietäisyyden määrittämät virheet ja pyyhkiymät. Näistä toista voi käyttää silloin, kun lähtökoodeista toinen on majoriteettilogiikalla dekoodattava. Toinen algoritmi sopii mille tahansa lähtökoodeille.

Tulokoodien yksinkertaisesta rakenteesta johtuen ne sopivat pehmeän päätöksen SISO-dekoodaukseen. SISO-dekoodausta käytettäessä tulokoodit ovat kilpailukykyisiä ja kiinnostavia joidenkin käytännön sovellusten kannalta.

Asiasanat: virheitä korjaavat koodit, tulokoodit, virheiden ja pyyhkiymien dekoodaus.

Sisältö

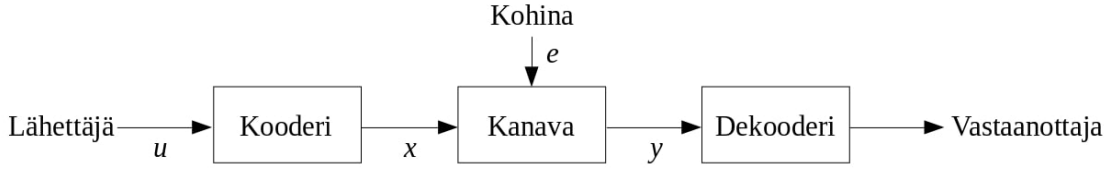
1 Johdanto

Koodausteoria tutkii, miten informaatio koodataan niin, että vastaanottaja saa korjattua kanavassa syntyvät virheet ja siten palautettua alkuperäisen viestin. Virheitä korjaavia koodeja käytetään nykyään kaikissa kommunikaatiosysteemeissä ja digitaalisissa tallennusvälineissä. Koodausteoriaa on tutkittu 40-luvulta alkaen. Shannon osoitti vuonna 1948, että mielivaltaisen pieni virhetodennäköisyys on mahdollista saavuttaa, kunhan koodin informaatiosuhde ei ylitä kanavan kapasiteettia [21]. Seuraavina vuosina Golay ja Hamming kehittivät ensimmäisiä virheitä korjaavia koodeja [8, 9].

Hammingin ja Golayn konstruoimat koodit ovat alan perusteita, jotka sopivat erinomaisesti ideoiden kuvailemiseen, ja tämän tutkielman esimerkeissä käytetäänkin Hammingin koodeja. Toimiakseen Shannonin lause kuitenkin edellyttää pitkiä koodeja, jotta virheiden jakauma keskiarvoistuu. Golayn koodin lohkon pituus on 24 ja esimerkeissä käytetyn Hammingin koodin 7, joten sovelluksiin nämä koodit ovat liian lyhyitä. Esimerkiksi digitelevisiostandardissa lohkon pituus on 16 200 tai 64 800.

Tässä tutkielmassa kanavakoodausta lähestytään yksinkertaisella mallilla (kuva 1). Mallissa on viestin lähettäjä, jolla on informaatiosana $u = (u_0, u_1, ..., u_{k-1})$. Informaatiosanaan lisätään redundanssia koodaamalla se koodisanaksi $x = (x_0, x_1, ..., x_{n-1})$, joka lähetetään kanavaan. Kanavassa kohina aiheuttaa virheitä, jolloin kanavasta luetaan sana $y = x + e$, jossa e on virhevektori $(e_0, e_1, ..., e_{n-1})$. Kanava voi olla systeemistä riippuen esimerkiksi avaruus tai ilmakehä tai sitten tallennusväline, kuten cd tai kovalevy. Kohinalla tarkoitetaan välityksessä tai tallennuksessa tapahtuvia häiriöitä, jotka aiheuttavat viestiin virheitä. Häiriöitä voivat olla esimerkiksi piikki sähkövirrassa tai viallisuus tallennusmediassa, kuten naarmu levyssä.

Kanavasta luettu sana y syötetään dekoodausalgoritmille, joka palauttaa estimaatin informaatiosanasta hyödyntäen sanaan koodauksessa lisättyä redundanssia. Ideaalitilanteessa estimaatti on alkuperäinen sana u. Jos virheitä on kuitenkin syntynyt liikaa, sana saatetaan dekoodataan väärin eli tapahtuu dekoodausvirhe (*decoding error*), tai dekoodaus epäonnistuu eli sanaa ei pystytä dekoodaamaan lainkaan (*decoding failure*).



Kuva 1: Kommunikaatiosysteemin yksinkertainen malli.

Yksi koodausteorian päätutkimuskohteista on kehittää koodeja, joilla on toisaalta suuri informaatiosuhde ja toisaalta suuri minimietäisyys. Suuri informaatiosuhde

kertoo hyvästä tehokkuudessa viestin välityksessä. Mitä suurempi minimietäisyys koodilla on, sitä enemmän se kykenee korjaamaan virheitä. Nämä tavoitteet ovat ristiriidassa keskenään [15].

Käytännön toteutuksen kannalta koodin hyvät matemaattiset ominaisuudet eivät ole riittävä vaatimus. Jotta koodia voidaan hyödyntää sovelluksissa, sillä on oltava riittävän yksinkertainen dekoodausalgoritmi, joka on helposti implementoitavissa. Kommunikaatiosysteemi ei saa olla liian kallis tai kuluttaa liikaa energiaa. Siksi voimakkaimpien koodien käyttämisen suurimpana esteenä onkin niiden dekoodausalgoritmien monimutkaisuus. Tämä tulee erityisesti esiin langattomissa järjestelmissä [1]. Käytettävän koodin valinnassa onkin tehtävä kompromisseja riippuen käyttökohteen vaatimuksista. Huomioon on otettava esimerkiksi tarvittava tarkkuus sekä käytössä oleva kanava ja laskentateho.

Tässä tutkielmassa perehdytään tulokoodeihin. Tulokoodi on matriisimuotoinen koodi, jossa matriisin pysty- ja vaakarivit ovat kahden pienemmän koodin sanoja. Näin saadaan muodostettua lähtökoodeja huomattavasti pitempi ja voimakkaampi koodi. Tulokoodit voidaan dekoodata dekoodaamalla pystyrivit pystyrivikoodin dekooderilla ja vastaavasti vaakarivit vaakarivikoodin dekooderilla. Tätä voidaan jatkaa useamman iteraation ajan. Valitsemalla lähtökoodeiksi koodit, joilla on yksinkertaiset dekoodausalgoritmit, saadaan tulokoodista voimakas koodi, jonka dekoodaus on kuitenkin suhteellisen yksinkertaista.

Tulokoodit esitteli ensimmäisenä Elias vuonna 1954 [5]. Tulokoodien teoriaa laajennettiin ja iteratiivista pehmeän päätöksen dekoodausta (*soft-decision decoding*) tutkittiin 70- ja 80-luvuilla. Tulokoodeista tuli kuitenkin suositumpia vasta 90-luvun lopulla, kun niiden dekoodaukseen alettiin soveltaa iteratiivista turbokoodausta. Käyttökelpoinen pehmen päätöksen SISO-dekoodausalgoritmi (*soft-input soft-output decoding*) tulokoodeille kehitettiin vuonna 1998 [18]. 2010-luvulla SISO-dekoodattuja tulokoodeja on käytetty muun muassa suurinopeuksisissa valokuitujärjestelmissä sekä useissa kommunikaatiostandardeissa (esimerkiksi IEEE 802.20 Mobile Broadband Wireless Access (MBWA)) [17, 16].

Tutkielman alussa käsitellään oleelliset peruskäsitteet, minkä jälkeen esitellään tulokoodin generaattorimatriisiin tarvittava Kroneckerin tulo sekä työssä käytettävät kanavamallit. Kolmannessa luvussa annetaan tulokoodin määritelmä, perehdytään tulokoodien rakenteeseen sekä parametreihin ja määritellään tulokoodin generaattorimatriisi.

Neljännessä ja viidennessä luvussa käsitellään dekoodausta. Neljännessä luvussa tarkastellaan dekoodausta yleisesti aloittaen ensin dekoodauksesta BEC-kanavassa, jossa vastaanotetussa sanassa voi virheiden lisäksi esiintyä pyyhkiymiä. Seuraavaksi esitellään kaksi dekoodausalgoritmia, erityisesti Reedin–Mullerin-koodeille käytetty majoriteettilogiikka sekä Forneyn yleistetyn etäisyyden algoritmi, joka on yleinen pehmeän päätöksen dekoodausalgoritmi.

Viidennessä luvussa siirrytään itse tulokoodien dekoodaukseen, ensin ryöppyvirheiden ja pyyhkiymien kautta. Seuraavaksi esitellään kaksi tulokoodien dekoodausalgoritmia, joissa hyödynnetään edellisen luvun dekoodausalgoritmeja. Luvussa kuusi esitellään eräitä kiinnostavia jatkotutkimuskohteita. Lisäksi esitellään lyhyesti tulokoodien SISO-dekoodaus, joka on oleellinen tulokoodien käytännön sovelluksissa.

2 Taustatietoja

2.1 Koodausteorian peruskäsitteitä

Määritelmä 2.1.1. Olkoon Q äärellinen q:n alkion joukko, $q \geq 2$. Joukon $Q^n = Q \times Q \times ... \times Q^n$ epätyhjää osajoukkoa C kutsutaan n:n pituiseksi q-ariseksi *koodiksi*. Koodin C alkioita sanotaan *koodisanoiksi*. Tässä työssä käsitellään pääasiassa binäärisiä koodeja, jolloin $Q = \mathbb{Z}_2 = \{0, 1\}$.

Koodisanojen $c = (c_1, c_2, ..., c_n)$ ja $c' = (c'_1, c'_2, ..., c'_n)$ *Hammingin etäisyys* $d(c, c')$ on niiden koordinaattien lukumäärä, joissa sanat eroavat toisistaan, eli

$$d(c, c') = |\{i : c_i \neq c'_i\}|.$$

Sanan c *Hammingin paino* $w(c)$ on sanan nollasta eroavien koordinaattien lukumäärä, eli olettaen että $0 \in Q$,

$$w(c) = d(c, 0),$$

jossa 0 on nollasana $(0, 0, ..., 0)$. Hammingin etäisyyttä ja Hammingin painoa kutsutaan myös vain etäisyydeksi ja painoksi.

Koodin C *minimietäisyys* d on pienin kahden koodin C sanan välinen Hammingin etäisyys, eli

$$d = min\{d(c, c') \mid c, c' \in C, c \neq c'\}.$$

Koodin virheenkorjauskyky on käytön kannalta oleellisimpia ominaisuuksia. Virheenkorjauskyky liittyy suoraan koodin minimietäisyyteen. Koodi C, jonka minimietäisyys on d, pystyy korjaamaan e satunnaisvirhettä, jos

$$d \geq 2e + 1.$$

Koodin virheenkorjauskyky antaa teoreettisen rajan korjattavien virheiden määrälle. Jos kanavassa syntyy satunnaisvirheitä $(d-1)/2$ tai vähemmän, on virheet niiden kuviosta riippumatta aina mahdollista korjata. Käytännössä myös koodin muut ominaisuudet, käytetty dekooderi ja syntyneiden virheiden kuvio vaikuttavat siihen, saadaanko syntyneet virheet korjattua. Hyvän dekoodausalgoritmin on korjattava kaikki virheet rajaan $(d - 1)/2$ asti, mutta on mahdollista korjata myös enemmän virheitä, jos virhevektorin kuvio on sopivanlainen.

Määritelmä 2.1.2. Koodi $C \subseteq \mathbb{F}_q^n$ on *lineaarinen*, jos kaikki koodisanojen summat ovat myös koodisanoja, eli kaikille $c = (c_1, c_2, ..., c_n) \in C$, $c' = (c'_1, c'_2, ..., c'_n) \in C$ myös $c + c' = (c_1 + c'_1, c_2 + c'_2, ..., c_n + c'_n) \in C$.

Nollasana kuuluu aina lineaariseen koodiin ja pelkän nollasanan sisältävää koodia kutsutaan triviaaliksi lineaariseksi koodiksi. Lineaarisen koodin minimietäisyys on pienin nollasta eroavan koodisanan paino, eli jos koodi C on lineaarinen, koodin minimietäisyys $d = min\{w(c) \mid c \in C, c \neq 0\}$. Lineaarista koodia sanotaan (n, k, d)-koodiksi, missä n on sanan pituus, k koodin dimensio ja d koodin minimietäisyys.

Lineaarinen koodi $C \subseteq \mathbb{F}_q^n$ on vektoriavaruuden $\mathbb{F}_q^n$ lineaarinen aliavaruus. Koodille C saadaan siis k:n koodisanan kanta, missä k on aliavaruuden C aste. Matriisia, jossa on vaakariveinä kannan vektorit, sanotaan koodin C *generaattorimatriisiksi*.

Koodin C sana saadaan muodostettua kertomalla k:n pituinen informaatiosana koodin C generaattorimatriisilla G. Saadussa sanassa on n bittiä, joista k ovat valitut informaatiobitit ja loput $n-k$ tarkistusbittejä. Suhdetta informaatiobittien ja lähetettyjen bittien välillä kutsutaan koodin *informaatiosuhteeksi* $R = k/n$.

Määritelmä 2.1.3. Olkoon C lineaarinen koodi. Koodia, jonka sanat ovat ortogonaaliset koodin C kanssa, kutsutaan koodin C *duaalikoodiksi*, joka merkitään $C^{\perp}$. Siis

$$C^{\perp} = \{x \in \mathbb{F}_q^n \mid x \cdot c = 0\},$$

missä $x \cdot c$ on sisätulo. Duaalikoodin generaattorimatriisia H sanotaan koodin C *pariteetintarkistusmatriisiksi.*

Lineaarisen koodin generaattorimatriisi G voidaan ilmaista muodossa $G = (I_k \mid P)$, jossa I_k on $k \times k$-identiteettimatriisi. Tätä kutsutaan generaattorimatriisin *standardimuodoksi.* Jos koodin C generaattorimatriisi G on muotoa $(I_k \mid P)$, saadaan pariteetintarkistusmatriisi kaavalla $H = (-P \mid I_{n-k})$.

Esimerkki 2.1.4. Tässä työssä käytetään useassa esimerkissä Hammingin (7,4)-koodia, jolla on generaattorimatriisi $G_{7,4}$ ja pariteetintarkistusmatriisi $H_{7,4}$, sekä laajennettua Hammingin (8,4)-koodia, jolla on generaattorimatriisi $G_{8,4}$ ja pariteetintarkistusmatriisi $H_{8,4}$

$$G_{7,4} = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right) \qquad H_{7,4} = \left(\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right)$$

$$G_{8,4} = \left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{array}\right) \qquad H_{8,4} = \left(\begin{array}{cccc|cccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Määritelmä 2.1.5. Lineaarinen koodi C on *syklinen*, jos myös kaikki koodisanojen sykliset siirrot kuuluvat koodiin eli jokaiselle koodisanalle $c = (c_0, c_1, ..., c_{n-1}) \in C$ on voimassa $(c_{n-1}, c_0, c_1, ..., c_{n-2}) \in C$.

2.2 Kroneckerin tulo

Kroneckerin tulo on eräänlainen matriisitulo. Kroneckerin tuloa hyödynnetään luvussa 3.3 määritettäessä tulokoodille generaattorimatriisi.

Määritelmä 2.2.1. Olkoon $A = (a_{ij})$ $m_1 \times n_1$-matriisi ja $B = (b_{ij})$ $m_2 \times n_2$-matriisi. Näiden Kroneckerin tulo $A \otimes B$ on matriisi, jolle [22]

$$A \otimes B = \begin{pmatrix} a_{00}B & a_{01}B & \dots & a_{0(n_1-1)}B \\ a_{01}B & a_{11}B & \dots & a_{1(n_1-1)}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{(m_1-1)0}B & a_{(m_1-1)1}B & \dots & a_{(m_1-1)(n_1-1)}B \end{pmatrix}.$$

Esimerkki 2.2.2. Lasketaan Kroneckerin tulo matriiseille A ja B, kun

$$A = \begin{pmatrix} 2 & 4 & 3 \\ 3 & 1 & 0 \end{pmatrix} \text{ ja } B = \begin{pmatrix} 1 & 5 \\ 0 & 4 \end{pmatrix}.$$

$$\text{Nyt } A \otimes B = \begin{pmatrix} 2 & 10 & 4 & 20 & 3 & 15 \\ 0 & 8 & 0 & 16 & 0 & 12 \\ 3 & 15 & 1 & 5 & 0 & 0 \\ 0 & 12 & 0 & 4 & 0 & 0 \end{pmatrix}.$$

2.3 Yksinkertaisista kanavamalleista

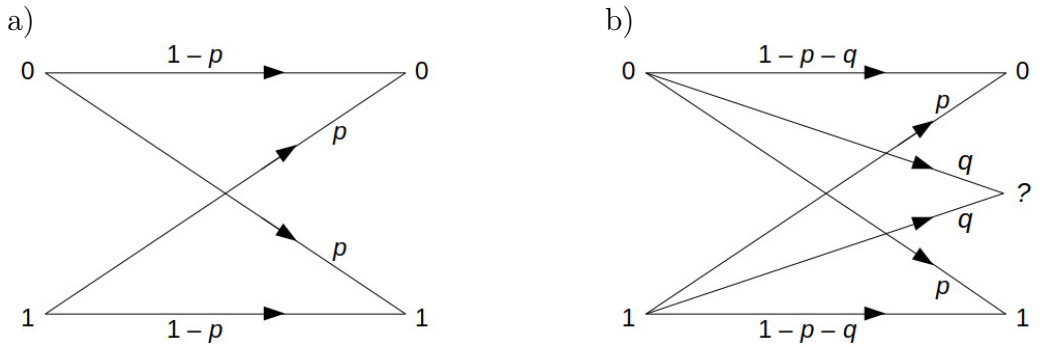
Koodin ja dekoodausalgoritmin valintaan vaikuttaa, millaisen kanavan läpi viesti lähetetään. Tässä työssä käytetään yksinkertaista kanavamallia, *diskreettiä muistitonta kanavaa* (DMC = *discrete memoryless channel*). DMC-kanavassa on M-arinen syöttöaakkosto ja Q-arinen tulosteaakkosto. Muistittomuus tarkoittaa, että tietyllä hetkellä vastaanotettu signaali riippuu ainoastaan kyseisestä lähetetystä signaalista, eikä aiemmin lähetetyillä ole siihen vaikutusta [12].

Diskreetti muistiton kanava voidaan kuvata siirtotodennäköisyyksillä

$$P(j|i), \quad 0 \leq i \leq M-1, \quad 0 \leq j \leq Q-1,$$

missä $P(j|i)$ tarkoittaa todennäköisyyttä vastaanottaa symboli j, kun kanavaan on lähetetty symboli i.

DMC-kanavassa jokaisella bitillä on sama todennäköisyys olla virheellinen, eli muistittomat kanavat ovat satunnaisvirhekanavia. Esimerkkejä kommunikaatiojärjestelmistä, joissa kanavan kohina on satunnaista, ovat useimmat satelliittikanavat ja kommunikointi avaruusluotainten kanssa (*deep-space channel*) [12].



Kuva 2: a) Binäärinen symmetrinen kanava (BSC) ja b) pyyhkiymäkanava (BEC).

Yksinkertaisin DMC-kanava on *binäärinen symmetrinen kanava* (BSC = *binary symmetric channel*), jossa $M = Q = 2$ ja virhetodennäköisyys kumpaankin suuntaan on sama (kuva 2a). Virheen todennäköisyys on p, jolloin BSC-kanavan siirtotodennäköisyydet ovat $P(1|1) = P(0|0) = 1 - p$ ja $P(1|0) = P(0|1) = p$. BSC-kanavassa $Q = 2$ eli tulostus on binäärinen ja siis dekooderille annetaan binäärinen syöte. Tätä kutsutaan kovapäätösdekoodaukseksi [12].

Viesti kulkee kanavassa aaltomuodossa, eikä vastaanotettu signaali välttämättä ole selvästi 0 tai 1. Kovapäätösdekoodauksessa demodulaattorin on kuitenkin jokaisen vastaanotetun signaalin kohdalla tehtävä valinta ja tulkittava signaali siksi symboliksi, jota se enemmän muistuttaa. Epäselvän signaalin kohdalla voi tässä tulla tulkintavirhe. Tämän voi välttää käyttämällä pyyhkiymiä.

Pyyhkiymäkanava eli BEC-kanava (*binary erasure channel*) on DMC-kanava, jossa $M = 2$ ja $Q = 3$ (kuva 2b). Tulosteaakkosto on $\{0, 1, ?\}$, jossa symboli ? tarkoittaa pyyhkiymää. Koordinaatti merkitään pyyhkiymäksi, kun signaali on epäselvä [12]. Pyyhkiymän suurin hyöty tulee siitä, että merkitsemällä bitti pyyhkiymäksi pysyy epäselvyyden paikka tiedossa, kun taas kovapäätösdekoodauksessa virheenkorjausalgoritmi pitää kaikkia sanan bittejä yhtä luotettavina, mikä voi liian usean virheen sattuessa johtaa dekoodausvirheeseen.

Varsinaisesta pehmeän päätöksen dekoodauksesta (*soft-decision decoding*) puhutaan, kun $Q > 3$. Tällöin dekooderin implemointi on kovapäätösdekoodaukseen verrattuna monimutkaisempaa, mutta dekoodausvirheet ja dekoodauksen epäonnistumiset saadaan vähenemään [12].

Todellisuudessa pehmeä ja kovapäätösdekoodaus eivät eroa toisistaan kanavan osalta, vaan valinta kovapäätösdekoodauksen ja pehmeän päätöksen dekoodauksen välillä tehdään vastaanottajapäässä. Viestisignaali luetaan kanavasta ja kvantisoidaan äärellisen moneen vaihtoehtoon sen mukaan, kuinka paljon tallennustilaa järjestelmässä on yhtä bittiä kohden varattu. Jos vaihtoehtoja on kaksi $(0 \mid 1)$ päädytään kovapäätösdekoodaukseen. Kolmella vaihtoehdolla $(0 \mid ? \mid 1)$ on kyseessä BEC, ja sitä useammalla pehmeän päätöksen dekoodaus (esimerkiksi kuudella vaihtoehdolla bitti kvantisoidaan joukkoon $\{0 \mid 0.2 \mid 0.4 \mid 0.6 \mid 0.8 \mid 1.0\}$).

Mitä pehmeämpää dekoodausta tehdään, sitä työläämpää sen toteutus on. Pehmeässä dekoodauksessa saadaan kuitenkin biteille luotettavuusdataa, jonka ansiosta päästään parempiin virheenkorjaustuloksiin. Esimerkiksi bitti, joka on kanavasta luettaessa kvantisoitu arvoon 0.8, on ollut ykkönen todennäköisemmin, kuin bitti, joka on luettu arvoksi 0.6.

Tarkastellaan yksinkertaisena esimerkkinä kolmen bitin pariteetintarkastuskoodia. Lähetetään kanavaan sana 110. Kanavan kohinan seurauksena kanavasta luetaan sana $(0.8, 0.8, 0.6)$. Näistä kolmas bitti on epäluotettavin, joten päätellään se virheelliseksi, ja dekoodataan tämä koodisanaksi 110. Vastaavalla kohinalla olisi kovapäätösdekoodauksessa kaikki bitit kvantisoitu ykkösiksi, kanavasta luettu 111 ja luotettavuustieto menetetty.

Seuraavasta lauseesta saadaan minimietäisyyden suhde korjattaviin virheisiin ja pyyhkiymiin eli koodin virheenkorjauskyky BEC-kanavassa.

Lause 2.3.1. *Koodi C, joka on (n, k, d)-koodi, saa korjattua e virhettä ja ε pyyhkiymää, jos virheille ja pyyhkiymille on voimassa*

$$d \geq 2e + \varepsilon + 1.$$

Todistus. Oletetaan, että kanavaan on lähetetty koodin C sana x ja kanavasta on vastaanotettu sana y, jossa on lauseen mukaiset ε pyyhkiymää ja e virhettä. Pois-

tetaan kaikista koodin C sanoista ne ε bittiä, joissa sanassa y on pyyhkiymä. Saadaan lyhennetty koodi C', jonka pituus on $n - \varepsilon$. Tämän koodin minimietäisyys on vähintään $d - \varepsilon$, ja

$$d - \varepsilon \geq 2e + 1.$$

Koodi C' pystyy siis korjaamaan lyhennetyssä sanassa y' olevat e virhettä. Saadaan sana r, jossa on ε pyyhkiymää, mutta muut bitit vastaavat lähetettyä sanaa x. Koska $d \geq \varepsilon + 1$, sana x on ainoa koodin C koodisana, jossa on sanaa r vastaavat bitit. [12]. □

Lauseesta 2.3.1 huomataan, että pyyhkiymiä saadaan korjattua kaksinkertainen määrä virheisiin nähden. Tämä parannus johtuu siitä, että pyyhkiymien kohdalla dekooderilla on dekoodauksen pohjana enemmän tietoa. Virheiden kohdalla ainoa tieto on, että vastaanotettu sana ei ole koodisana. Virheitä arvoidaan tapahtuneen pienin mahdollinen määrä, jolla vastaanotettuun sanaan olisi voinut koodisanasta päätyä. Pyyhkiymien kohdalla dekooderilla sen sijaan on tiedossa sekä pyyhkiymien lukumäärä että sijainti.

3 Tulokoodien rakenne ja perusominaisuudet

3.1 Johdattelevia esimerkkejä tulokoodista

Esimerkki 3.1.1. Lähestytään tulokoodeja esimerkin kautta. Olkoon C_1 binäärinen kolmen bitin pariteetintarkastuskoodi ja C_2 binäärinen neljän bitin pariteetintarkastuskoodi. Tarkastellaan koodia C, jossa sanoina on 3×4-matriisit, joissa vaakarivit ovat koodin C_2 sanoja ja pystyrivit koodin C_1 sanoja.

Tarkastellaan sanaa

$$c = \left(\begin{array}{ccc|c} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ \hline c_{20} & c_{21} & c_{22} & c_{23} \end{array}\right).$$

Nyt bitit $c_{00}, ..., c_{02}, c_{10}, ..., c_{12}$ ovat informaatiobittejä ja ne voidaan valita vapaasti. Vaakarivit ovat koodin C_2 sanoja, joten bitit c_{03} ja c_{13} määräytyvät yksikäsitteisesti. Samoin bitit c_{20}, c_{21} ja c_{22} määräytyvät yksikäsitteisesti, koska pystyrivit ovat koodin C_2 sanoja.

Ainoa mahdollinen ongelma on, määräytyykö nurkkabitti c_{23} yksikäsitteisesti niin, että alin vaakarivi on koodin C_1 sana ja että oikeanpuoleisin pystyrivi on koodin C_2 sana. Valitaan nurkkabitti c_{23} koodin C_1 mukaan eli niin, että $(c_{03}, c_{13}, c_{23}) \in C_1$. Nyt alin rivi $(c_{20}, c_{21}, c_{22}, c_{23})$ on saatu laskemalla yhteen ylemmät rivit $(c_{00}, c_{01}, c_{02}, c_{03})$ ja $(c_{10}, c_{11}, c_{12}, c_{13})$. Koska C_2 on lineaarinen, $(c_{20}, c_{21}, c_{22}, c_{23}) \in C_2$.

Muodostetaan koodisana informaatiobiteillä $((1,0,1),(0,0,1))$. Pariteetintarkistusbitit määräytyvät seuraavasti

$$\left(\begin{array}{ccc|c} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 \end{array}\right).$$

Tarkastellaan koodin C pienipainoisinta nollasta eroavaa koodisanaa. Koodin C_2 minimipaino on kaksi, joten jos yksi jonkin vaakarivin biteistä on ykkönen, on vaakarivillä oltava toinenkin ykkönen. Samoin koodin C_1 minimipaino on kaksi, joten ykkösiä vastaavilla pystyriveillä on kummallakin oltava myös toiset ykköset. Nollasta eroavassa sanassa on siis vähintään neljä ykköstä eli tulokoodin minimipaino on neljä. Esimerkiksi sana, jossa on ykkönen informaatiobitissä c_{02}, on

$$\left(\begin{array}{ccc|c} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \end{array}\right).$$

Koodin minimipaino on neljä ja informaatiobittejä on kuusi, joten saatu koodi on $(12, 6, 4)$-koodi. Koodi on yhden virheen korjaava.

Esimerkki 3.1.2. Olkoon C_1 binäärinen neljän bitin pariteetintarkastuskoodi C_2 seitsemän bitin Hammingin koodi. Nyt C_1 on (4,3,2)-koodi ja C_2 on (7,4,3)-koodi. Muodostetaan koodi C, jossa pystyrivit ovat koodin C_1 sanoja ja vaakarivit koodin C_2 sanoja. Koodin C koodisanat ovat nyt muotoa

$$c = \left(\begin{array}{cccc|ccc} c_{00} & c_{01} & c_{02} & c_{03} & c_{04} & c_{05} & c_{06} \\ c_{10} & c_{11} & c_{12} & c_{13} & c_{14} & c_{15} & c_{16} \\ c_{20} & c_{21} & c_{22} & c_{23} & c_{24} & c_{25} & c_{26} \\ \hline c_{30} & c_{31} & c_{32} & c_{33} & c_{34} & c_{35} & c_{36} \end{array}\right).$$

Vasemman ylänurkan 12 bittiä ovat informaatiobittejä ja voidaan valita vapaasti. Oikean ylänurkan yhdeksän bittiä määräytyvät Hammingin koodin tarkistusbitteinä kunkin vaakarivin mukaisesti. Vasemman alanurkan neljä bittiä taas muodostuvat pariteetintarkastukoodin tarkistusbitteinä. Nyt kysymyksenä on sama kuin ensimmäisessä esimerkissä: määräytyvätkö kolme oikean alanurkan bittiä (c_{34}, c_{35} ja c_{36}) niin, että alin vaakarivi on Hammingin koodin sana ja pystyrivit pariteetintarkistuskoodin sanoja?

Täydennetään alin rivi loppuun noudattamalla pystyrivien pariteetintarkastuskoodia. Nyt koko alin rivi on saatu laskemalla yhteen ensimmäinen, toinen ja kolmas rivi, eli kolme Hammingin koodin sanaa, joten se on myös Hammingin koodin koodisana. Nurkkabitit noudattavat siis sekä pysty- että vaakarivien kaavaa.

Tarkastellaan pienipainoisinta nollasta eroavaa koodin C sanaa. Valitaan kaksi kolmesta ensimmäisestä vaakarivistä nollasanoiksi. Kolmannen vaakarivin täytyy nyt erota nollasta. Hammingin koodin pienin nollasta eroavan sanan paino on kolme, joten kolmannella vaakarivillä on oltava kolme ykköstä. Pystyrivien pariteetintarkastuskoodi tuo alimmalle vaakariville toiset kolme ykköstä. Tarkastelemamme koodin minimietäisyys on siis kuusi. Koodi C on siis $(28, 12, 6)$-koodi ja pystyy korjaamaan 2 virhettä.

Pohditaan seuraavaksi, miten koodi löytää ja korjaa syntyvät virheet. Noudatetaan seuraavaa algoritmia:

1. Dekoodataan vaakarivit Hammingin $(7, 4)$-koodin dekoodausalgoritmilla. Välitetään pariteetintarkastuskoodille tieto virheellisistä riveistä sekä alustavasti dekoodattu matriisi.

2. Pariteetintarkistuskoodi tarkistaa pystyrivit ja tarvittaessa korjaa ykköskohdassa epäilyksenalaisiksi merkittyjen rivien bittejä.

Oletetaan, että kanavassa on syntynyt kaksi virhettä. Virheet voivat olla joko eri riveillä tai samalla rivillä. Jos virheet ovat matriisin eri riveillä, eli Hammingin koodin eri sanoissa, Hammingin koodin algoritmi korjaa ne kohdassa 1 ja pariteetintarkistuskoodi kelpuuttaa pystyrivit sellaisenaan kohdassa 2.

Käydään esimerkin kanssa läpi tilanne, jossa virheet ovat samalla rivillä. Oletetaan, että kanavaan on lähetetty nollasana ja kanavan kohina on aiheuttanut kaksi virhettä. Kanavasta luetaan sana

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

1. Hammingin koodi toteaa ensimmäisen, toisen ja neljännen vaakarivin virheettömiksi, mutta kolmannen vaakarivin virheelliseksi. Hammingin $(7, 4)$-koodi ei kykene erottamaan, onko tapahtunut yksi vai kaksi virhettä, vaan korjaa kaksivirheisen sanan väärin yhden bitin virheenä. Tässä sana dekoodataan kääntämällä bitti c_{23} ykköseksi. Kolmas rivi merkitään epäluotettavaksi.

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) \begin{array}{c} ok \\ ok \\ ? \\ ok \end{array}$$

2. Pariteetintarkastuskoodi käy läpi pystyrivit ja toteaa ensimmäisen, toisen, kuudennen ja seitsemännen pystyrivin virheettömiksi sekä pystyrivit 3–5 virheellisiksi.

   Pariteetintarkistuskoodin minimietäisyys on kaksi eli se ei itsellään korjaa yhtäkään virhettä. Tässä dekoodattava sana 0010 on yhtä lähellä koodisanoja 0000 ja 0011. Vaakarividekooderin antamien luotettavuustietojen perusteella osataan kuitenkin korjata bitit c_{22}, c_{23} ja c_{24} ja saadaan

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

   Kaikki vaakarivit ovat nyt Hammingin koodin sanoja ja kanavasta vastaanotettu koodisana on saatu korjattua.

3.2 Tulokoodin parametrit

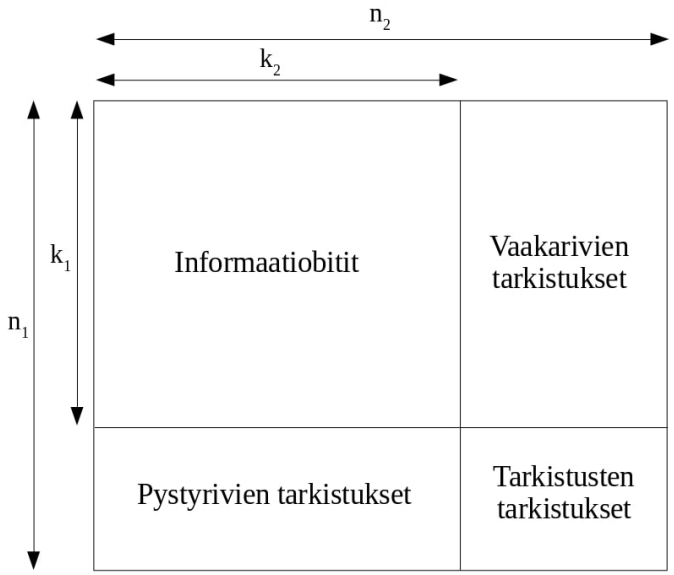
Kahdesta lineaarisesta virheitä korjaavasta koodista voidaan muodostaa isompi koodi koodaamalla informaatiobitit kahdesti. Tehdään tämä luvun 3.1 esimerkkien tapaan järjestämällä bitit matriisimuotoon ja koodaamalla ne ensin yhteen suuntaan ensimmäisellä lähtökoodilla ja sitten toiseen suuntaan toisella. Näin saadaan alkuperäisiä koodeja vahvempi virheenkorjauskyky.

Käytetään lähtökoodeina koodia C_1, joka on (n_1, k_1, d_1)-koodi ja koodia C_2, joka on (n_2, k_2, d_2)-koodi. Muodostetaan luvun 3.1 esimerkkien tavoin suurempi koodi C, jonka koodisana on $n_1 \times n_2$-matriisi. Koodisanassa on k_1k_2 informaatiobittiä, $k_1(n_2 - k_2)$ koodin C_2 tarkistusbittiä, $k_2(n_1 - k_1)$ koodin C_1 tarkistusbittiä ja $(n_1 - k_1)(n_2 - k_2)$ tarkistusten tarkistusbittiä. Koodia C kutsutaan tulokoodiksi.

Tulokoodin koodisanan rakenne on esitetty kuvassa 3. Tässä tutkielmassa käsitellään tulokoodeja selvyyden vuoksi muodossa, jossa tarkastusbitit ovat aina lähtökoodeissa sanan lopussa ja siten tulokoodissa oikealla ja alareunassa.

Luvun 3.1 esimerkeissä havaittiin, että oikean alanurkan bitit eli tarkistusten tarkistukset noudattivat sekä vaaka- että pystyrivikoodin kaavaa. Lineaarialgebrasta saadaan, että tämä pätee yleisestikin. Valitaan informaatiobitit ja täydennetään vaakarivien tarkistukset koodin C_2 mukaan. Ensimmäiset k_1 vaakariviä ovat nyt

koodin C_2 sanoja. Täydennetään sitten jäljellä olevat alemmat $n_1 - k_1$ vaakariviä, eli sekä pystyrivien tarkistukset että tarkistusten tarkistukset, koodin C_1 mukaan. Kaikki pystyrivit ovat siis koodin C_1 sanoja. Lineaarisen koodin tarkistusbitit saadaan koodisanan informaatiobittien lineaarikombinaationa, joten $n_1 - k_1$ alinta vaakariviä on saatu k_1 ylemmän vaakarivin lineaarikombinaationa. Koska ylemmät k_1 vaakariviä ovat koodin C_2 sanoja, ovat myös alimmat vaakarivit koodin C_2 sanoja.



Kuva 3: Tulokoodin koodisanan rakenne.

Määritellään seuraavaksi tulokoodi muodollisesti.

Määritelmä 3.2.1. Olkoon C_1 lineaarinen (n_1, k_1, d_1)-koodi ja C_2 lineaarinen (n_2, k_2, d_2)-koodi yli kunnan $\mathbb{F}_2$. Olkoon C $n_1 n_2$-pituinen koodi, jonka koodisanat ovat $n_1 \times n_2$-matriiseja. C on koodien C_1 ja C_2 *tulokoodi*, joss C:n koodisanat ovat kaikki ne koodisanat, joiden matriisiesityksen kaikki pystyrivit ovat koodin C_1 sanoja ja kaikki vaakarivit ovat koodin C_2 sanoja [13]. Käytetään tulokoodista merkintää

$$C = C_1 \otimes C_2.$$

Kahden tulokoodin koodisanan summassa vaakarivit ovat jokainen kahden koodin C_2 sanan summia, eli koodin C_2 sanoja, ja vastaavasti pystyrivit koodin C_1 sanojen summina koodin C_1 sanoja. Tulokoodin sanojen summa on siis sekin tulokoodin sana eli lineaaristen koodien tulokoodi on lineaarinen.

Lause 3.2.2. *Olkoon C_1 (n_1, k_1, d_1)-koodi ja C_2 (n_2, k_2, d_2)-koodi. Tulokoodi $C_1 \otimes C_2$ on $(n_1 n_2, k_1 k_2, d_1 d_2)$-koodi.*

Todistus. Tulokoodin määritelmästä seuraa, että koodisanan pituus on $n_1 n_2$ ja koodin dimensio $k_1 k_2$. Olkoon c koodin $C_1 \otimes C_2$ nollasta eroava sana. Ainakin yhdellä matriisin c pystyrivillä on nollasta eroava alkio. Koska pystyrivi on koodin C_1 sana, sen paino on vähintään d_1, eli sillä on vähintään d_1 nollasta eroavaa alkiota. Nyt vähintään d_1 vaakarivillä on nollasta eroava alkio. Kukin niistä on koodin C_2 sana, joten niillä on vähintään paino d_2. sanan c paino on siis vähintään $d_1 d_2$ [13]. □

3.3 Tulokoodin generaattorimatriisi

Tutkitaan seuraavaksi tulokoodin generaattorimatriisia. Generaattorimatriisin muodostamisessa käytetään luvussa 2.2 määriteltyä Kroneckerin tuloa ja lähtökoodien generaattorimatriiseja. Tarkastellaan tätä esimerkin kautta.

Esimerkki 3.3.1. Palataan esimerkin 3.1.2 koodeihin. Vaakarivikoodi C_1 on $(4, 3, 2)$-pariteetintarkistuskoodi ja pystyrivikoodi C_2 on Hammingin (7,4)-koodi. Näiden tulokoodin $C = C_1 \otimes C_2$ parametrit ovat $(n_1n_2, k_1k_2, d_1d_2) = (28, 12, 6)$. Koodeilla C_1 ja C_2 on generaattorimatriisit G_1 ja G_2, joille

$$G_1 = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right) \text{ ja } G_2 = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right).$$

Näiden kroneckerin tulo on 12×28-matriisi

$$G_1 \otimes G_2 = \begin{pmatrix} G_2 & 0 & 0 & G_2 \\ 0 & G_2 & 0 & G_2 \\ 0 & 0 & G_2 & G_2 \end{pmatrix}.$$

Esitetään matriisin $G_1 \otimes G_2$ ensimmäinen rivi 4×7-matriisimuodossa, jolloin saadaan koodin $C_1 \otimes C_2$ sana

$$\left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right).$$

Vastaavasti esimerkiksi toisella rivillä on koodin $C_1 \otimes C_2$ sana, jossa koordinaatissa $(0, 1)$ on ykkönen ja muut informaatiobitit ovat nollia, eli tulokoodin sana

$$\left(\begin{array}{cccc|ccc} 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{array}\right)$$

Rivillä 5 taas on sana, jossa koordinaatissa (1,0) on ykkönen ja muut informaatiobitit ovat nollia, eli tulokoodin sana

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right).$$

Matriisin $G_1 \otimes G_2$ riveillä ovat kaikki 12 koodin $C_1 \otimes C_2$ sanaa, joissa informaatiobiteissä on tasan yksi ykkönen, eli kaikki tulokoodin sanat, joiden paino on kuusi. Matriisin $G_1 \otimes G_2$ vaakarivien lineaarikombinaatioina saadaan siis kaikki koodin $C_1 \otimes C_2$ sanat, joten se on koodin generaattorimatriisi.

Vastaavasti yleisessä tapauksessa lähtökoodien generaattorimatriisien Kroneckerin tulona saadaan tulokoodin generaattorimatriisi. Esimerkissä lähtökoodeille käytettiin redusoidussa diagonaalimuodossa olevia generaattorimatriiseja, mutta Kroneckerin tulon ominaisuuksista johtuen lähtökoodien generaattorimatriisien valinnalla ei ole väliä [22].

Tulokoodin $C = C_1 \otimes C_2$ koodisanan voi generoida joko kertomalla k_1k_2-pituisen informaatiovektorin koodin C generaattorimatriisilla tai käyttämällä lähtökoodien generaattorimatriiseja seuraavan lauseen mukaisesti.

Lause 3.3.2. *Olkoon informaatiosana u $k_1 \times k_2$-matriisi ja G_1 ja G_2 koodien C_1 ja C_2 generaattorimatriisit. Tulokoodin C sana x saadaan kaavalla*

$$x = G_1^\top u G_2.$$

Todistus. Kertomalla informaatiomatriisi u vasemmalta koodin C_1 generaattorimatriisin transpoosilla saadaan $n_1 \times k_2$-matriisi, jossa pystyriveinä on koodin C_1 sanoja. Kun tämä kerrotaan oikealta koodin C_2 generaattorimatriisilla, saadaan $n_1 \times n_2$-matriisi, jossa äskeisen matriisin vaakarivit on täydennetty koodin C_2 sanoiksi [1]. □

Lauseen 3.3.2 avulla voidaan operoida tulokoodin suuren $k_1k_2 \times n_1n_2$-generaattorimatriisin sijaan lähtökoodien huomattavasti pienemmillä $k_1 \times n_1$- ja $k_2 \times n_2$-generaattorimatriiseilla. Annetaan tästä esimerkki.

Esimerkki 3.3.3. Olkoon $C = C_1 \otimes C_2$ kuten esimerkissä 3.3.1. Olkoon informaatiosana u 3×4-matriisi

$$u = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Lauseen 3.3.2 mukaisesti saadaan

$$\begin{aligned} x = G_1^\top u G_2 &= \left(\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 1 & 1 & 1 \end{array}\right) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right) \\ &= \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \end{array}\right) \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right) \\ &= \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right). \end{aligned}$$

Sama tulokoodin sana saadaan käyttämällä informaatiosanasta vektorimuotoa $v = (1, 0, 0, ..., 0)$ ja kertomalla tämä generaattorimatriisilla $G_1 \otimes G_2$, joka on 12×28-matriisi. Tulokoodin sana saadaan 28-pituisena vektorina

$$x' = v(G_1 \otimes G_2) = (1, 0, 0, ..., 0) \begin{pmatrix} G_2 & 0 & 0 & G_2 \\ 0 & G_2 & 0 & G_2 \\ 0 & 0 & G_2 & G_2 \end{pmatrix}.$$

4 Dekoodauksesta yleisesti

4.1 Dekoodaus BEC-kanavassa

Esitellään seuraavaksi menetelmä, jolla saadaan korjattua binäärisen koodin virheet ja pyyhkiymät, kun käytetylle koodille on tiedossa kovapäätösdekooderi, joka korjaa virheet rajaan $e \leq (d-1)/2$ asti [20].

Algoritmissa tehdään vastaanotetulle sanalle kaksi komplementtista estimaattia, toisessa pyyhkiymät korvataan kaikki nollilla ja toisessa ykkösillä. Näistä muodostetuista sanoista toisessa on oltava korvatuista biteistä yli puolet oikein. Tämä on riittävä määrä, jotta dekoodausalgoritmi korjaa virheet oikein, jos kanavassa on tapahtunut enintään e virhettä ja ε pyyhkiymää ja $2e + \varepsilon < d$. Estimaatit yritetään kumpikin dekoodata ja jos kummankin dekoodaus tuottaa tulosteena koodisanan, valitaan niistä se jonka dekoodaamisessa tarvittiin vähemmän korjauksia [20].

Algoritmi 4.1.1. *Olkoon C binäärinen (n, k, d)-koodi, jolle on tiedossa dekoodausalgoritmi, joka korjaa kaikki virheet, kun niiden lukumäärä on $\leq (d+1)/2$. Kanavasta on vastaanotettu sana y, jossa osa koordinaateista on pyyhkiymiä.*

1. *Korvataan vastaanotetussa sanassa olevat pyyhkiymät nollilla, jolloin saadaan binäärinen sana r_0. Dekoodataan r_0 koodin C dekoodausalgoritmilla ja saadaan koodin C sana c_0.*

2. *Korvataan vastaanotetussa sanassa olevat pyyhkiymät ykkösillä, jolloin saadaan binäärinen sana r_1. Dekoodataan r_1 koodin C dekoodausalgoritmilla ja saadaan koodin C sana c_1.*

3. *Valitaan dekoodaustulosten väliltä seuraavasti:*

   (i) *Toinen sanoista r_0 ja r_1 palauttaa dekoodauksessa koodin C koodisanan, mutta toisen kohdalla dekoodaus epäonnistuu. Palautetaan onnistuneen dekoodauksen tulos c_i.*

   (ii) *Kummankin sanan r_0 ja r_1 dekoodaus onnistuu ja ne palauttavat saman koodin C sanan $c_0 = c_1$. Palautetaan sana $c_0 = c_1$.*

   (iii) *Kummankin sanan r_0 ja r_1 dekoodaus onnistuu, mutta ne palauttavat koodin C eri sanat c_0 ja c_1, $c_0 \neq c_1$. Palautetaan se koodisana, jonka dekoodauksessa korjattiin vähemmän virheitä.*

   (iv) *Dekoodaus epäonnistuu, jos kummankin sanan r_0 ja r_1 dekoodaus onnistuu yhtä monella korjauksella, mutta ne palauttavat koodin C eri sanat c_0 ja c_1, $c_0 \neq c_1$, tai kummankin sanan dekoodaus epäonnistuu.*

Olkoon x lähetetty sana ja vastaanotetussa sanassa r on ε pyyhkiymää ja e virhettä. Algoritmissa luodut sanat r_0 ja r_1 ovat etäisyyksillä $e+\varepsilon_0$ ja $e+\varepsilon_1$ sanasta x, missä $\varepsilon_0 + \varepsilon_1 = \varepsilon$.

Jos nyt oletetaan, että $2e + \varepsilon + 1 \leq d$, saadaan

$$e + \frac{\varepsilon}{2} \leq \frac{d-1}{2}.$$

Koska koodi C korjaa virheet määrään $(d-1)/2$ asti ja $\varepsilon = \varepsilon_0 + \varepsilon_1$, on ainakin toisen sanoista r_0 ja r_1 etäisyys lähetetystä sanasta x riittävän pieni, jotta koodin C algoritmi dekoodaa sen oikein. Ainakin toinen sanoista c_0 ja c_1 on siis oikea lähetetty sana. [20]

Tarkastellaan tilannetta, jossa molempien sanojen r_0 ja r_1 dekoodauksessa on korjattu yhtä monta virhettä ja saatu koodisanat c_0 ja c_1. Tässä tilanteessa algoritmi hyväksyy dekoodaustuloksen, jos $c_0 = c_1$, mutta dekoodaus epäonnistuu, jos $c_0 \neq c_1$.

Oletetaan, että virheille ja pyyhkiymille pätee $2e + \varepsilon + 1 \leq d$. Nyt siis toinen $c_i = x$. Oletetaan, että $c_0 = x$. Nyt $d(r_0, c_0) = d(r_0, x) = e + \varepsilon_0$. Koska kummankin sanan dekoodauksessa korjattiin yhtä monta virhettä on myös $d(r_1, c_1) = e + \varepsilon_0$. Saadaan

$$d(x, c_1) \leq d(x, r_1) + d(r_1, c_1) = e + \varepsilon_1 + e + \varepsilon_0 = 2e + \varepsilon < d.$$

Siis $c_1 = x$.

Tilanteessa, jossa r_0 ja r_1 saadaan dekoodattua koodisanoiksi c_0 ja c_1 yhtä monella korjauksella ja $c_0 \neq c_1$, on virheitä ja pyyhkiymiä tapahtunut yli rajan $2e + \varepsilon + 1 \leq d$ eikä dekoodaus onnistu luotettavasti. Tässä tilanteessa ei siis kannata hyväksyä kumpaakaan dekoodaustulosta c_0 tai c_1.

Esimerkki 4.1.2. Olkoon C Hammingin $(7, 4)$-koodi. Koodin C minimietäisyys on 3, eli se pystyy korjaamaan yhden virheen tai kaksi pyyhkiymää.

Oletetaan, että kanavasta on vastaanotettu sana 11?00?1. Korvataan pyyhkiymät ja saadaan sanat $r_0 = 1100001$ ja $r_1 = 1110011$. Dekoodataan nämä koodin dekoodausalgoritmilla ja saadaan $c_0 = 1100011$ ja $c_1 = 1100011$. Molemmat antoivat saman koodisanan, joten tämä hyväksytään dekoodaustulokseksi.

Kanavasta on vastaanotettu sana 1011?1?. Nyt $r_0 = 1011010$ ja $r_1 = 1011111$. Saadaan $c_0 = r_0 = 1011010$ ja $c_1 = 1111111$. Kumpikin c_0 ja c_1 ovat koodisanoja, mutta c_0 vaati vähemmän korjauksia, joten saadaan, että dekoodattu sana on $c_0 = 1011010$.

Kolmantena kanavasta on vastaanotettu sana 0110?01. Nyt $r_0 = 0110001$ ja $r_1 = 0110101$. Dekoodauksesta saadaan $c_0 = 0111001$ ja $c_1 = 0100101$. Kumpikin on koodin C sana ja yhtä kaukana vastaanotetusta sanasta. Kanavassa on tapahtunut sekä virhe että pyyhkiymä, eikä koodin minimietäisyys riitä korjaamaan tätä.

Esimerkki 4.1.3. Olkoon C laajennettu Hammingin (8,4)-koodi, jonka minimietäisyys on 4. Kanavasta on vastaanotettu sana 0011110?. Nyt $r_0 = 00111100$ ja $r_1 = 00111101$. Sanan r_0 dekoodaus ei onnistu. Dekoodaamalla r_1 saadaan 00111001. Laajennetulla Hammingin koodilla saatiin siis korjattua yksi virhe ja yksi pyyhkiymä.

Symmetrisessä binäärisessä kanavassa laajennettu Hammingin koodi korjaa tavallisen Hammingin koodin tavoin vain yhden virheen. Sen sijaan pyyhkiymäkanavassa saadaan virheen lisäksi korjattua pyyhkiymä. Ilman virheitä saadaan pyyhkiymiä korjattua kolme ($d = 4 > 0 + 3 = 3$).

4.2 Majoriteettilogiikka

Luvussa 5 käsitellään tulokoodien dekoodausta. Tätä varten esitellään seuraavaksi kaksi dekoodausalgoritmia lähtökoodien dekoodaukseen. Tässä luvussa käsitellään

majoriteettilogiikkaa, jota hyödynnetään luvussa 5.4 käsiteltävässä tulokoodin dekoodausalgoritmissa, jossa toinen koodeista on majoriteettilogiikalla dekoodattava.

Majoriteettilogiikkaa käytti ensimmäisenä Reed vuonna 1954 Reedin–Mullerin koodeille [19]. Massey yleisti menetelmän vuonna 1963 [14]. Majoriteettilogiikalla dekoodatessa hyödynnetään pariteetintarkistusmatriisista saatavia pariteetintarkistusyhtälöitä. Tarkoituksena on löytää jokaiselle bitille riittävän monta ortogonaalista yhtälöä. Tutkittava bitti ratkaistaan yhtälöistä ja niistä enemmistö antaa bitille oikean arvon. Majoriteettilogiikka korjaa kaikki virheet rajaan $e \leq (d-1)/2$ asti, jos koodi on ortogonalisoituva, eli ortogonaalisia pariteetintarkistusyhtälöitä voi löytää rittävän määrän [15].

Majoriteettilogiikan etuja ovat yksinkertainen implementointi ja algoritmin kyky korjata myös useita sellaisia virhevektoreita, joiden paino ylittää luvatun arvon $\lfloor J/2 \rfloor$ [15].

Määritelmä 4.2.1. Pariteetintarkistusyhtälöiden $S_1, S_2, ..., S_J$ joukkoa kutsutaan *ortogonaaliseksi* koordinaatin i suhteen, jos x_i esiintyy näistä jokaisessa, mutta mikään muu x_j ei esiinny useammassa kuin yhdessä joukon yhtälössä [15].

Lause 4.2.2. *Jos kaikille koodin C koordinaateille löydetään J ortogonaalista pariteetintarkistusyhtälöä, koodi C korjaa $\lfloor J/2 \rfloor$ virhettä.*

Todistus. Oletetaan, että kanavassa on syntynyt enintään $\lfloor J/2 \rfloor$ virhettä. olkoon x lähetetty sana, vastaanotettu $y = x + e$, jossa e on virhevektori.

Jos $e_i = 0$, virheet vaikuttavat enintään $\lfloor J/2 \rfloor$:een koordinaatin i suhteen ortogonaalisista pariteetintarkistusyhtälöistä. Yhtälöistä ainakin $\lceil J/2 \rceil$ antaa arvon nolla. Jos tilanne on tasan, otetaan $x_i = y_i$ yhtenä arviona, eli arvioidaan $e_i = 0$.

Jos $e_i = 1$, loput virheet vaikuttavat alle $\lfloor J/2 \rfloor$ yhtälöön. Enemmistö yhtälöistä antaa arvon yksi.

Enemmistö pariteetintarkistusyhtälöistä antaa siis aina oikean e_i:n arvon [15]. □

Selvästi majoriteettilogiikasta on todellista hyötyä vain koodeilla, joilla J on riittävän suuri. Jotta saataisiin dekoodattua kaikki virheet, joiden lukumäärä $e \leq (d-1)/2$, täytyy J:n arvon olla $d-1$. Tästä tulee seuraava määritelmä.

Määritelmä 4.2.3. Olkoon koodin C minimietäisyys d. Koodi C on *(yhdessä vaiheessa) ortogonalisoituva* (engl. *completely orthogonalizable in one step*), joss on mahdollista muodostaa jokaiselle bitille $J = d-1$ pariteetintarkastusyhtälöä, jotka ovat ortogonaaliset kyseisen bitin suhteen [12].

Suurin osa majoriteettilogiikalla dekoodattavista koodeista on syklisiä, sillä sykliselle koodeille riittää löytää pariteetintarkistusyhtälöt yhdelle bitille. Jos koodi C on syklinen ja yhdelle koordinaatille on tiedossa J ortogonaalista pariteetintarkistusyhtälöä, saadaan muille koordinaateille J ortogonaalista pariteetintarkistusyhtälöä alkuperäisen yhtälöjoukon syklisillä siirroilla [15]. Pariteetintarkistusyhtälöihin osallistuu kuitenkin niin suuri joukko bittejä, että voi olla vaikeaa löytää riittävän monta ortogonaalista pariteetintarkistusyhtälöä ensimmäisellekään bitille.

Annetaan seuraavaksi algoritmi yksivaiheiselle majoriteettilogiikalle ja havainnollistetaan sitä esimerkin kautta.

Algoritmi 4.2.4. *Olkoon C (yhdessä vaiheessa) ortogonalisoituva (n, k, d)-koodi. Etsitään jokaiselle koodisanan c bitille c_i $d-1$ bitin c_i suhteen ortogonaalista pariteetintarkastusyhtälöä $S_1^i, S_2^i, ..., S_{d-1}^i$.*

Kanavasta on vastaanotettu sana y. Kukin bitti y_i dekoodataan siksi arvoksi, jonka enemmistö pariteetintarkistusyhtälöistä bitille antaa, tai jos vaihtoehdot menevät tasan, nollaksi. Bitin y_i arvo valitaan siis enemmistöstä joukossa $\{0, S_1^i(y), ..., S_{d-1}^i(y)\}$ [15].

Esimerkki 4.2.5. Tarkastellaan $(7, 3, 4)$-Simplex-koodia, jolla on pariteetintarkastusmatriisi

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

Jotta koodi olisi yhdessä vaiheessa ortogonalisoituva, täytyisi löytää $d-1 = 3$ ortogonaalista pariteetintarkistusyhtälöä kullekin bitille. Bitille x_0 saadaan matriisin ensimmäiseltä riviltä, ensimmäiseltä, toiselta ja kolmannelta riviltä, sekä ensimmäiseltä, toiselta ja neljänneltä riviltä yhtälöt

$$\begin{aligned} x_0 &= x_1 + x_3, \\ x_0 &= x_4 + x_5 \quad \text{ja} \\ x_0 &= x_2 + x_6. \end{aligned}$$

Koska Simplex-koodi on syklinen, saadaan tarkistusyhtälöt muille biteille näiden syklisillä siirroilla. Esimerkiksi bitille x_1 saadaan pariteetintarkistusyhtälöt

$$\begin{aligned} x_1 &= x_2 + x_4, \\ x_1 &= x_5 + x_6 \quad \text{ja} \\ x_1 &= x_0 + x_3. \end{aligned}$$

Näistä ensimmäistä yhtälöä vastaa tarkistusmatriisin rivi 2, toista yhtälöä vastaavat rivit 2, 3 sekä 4, ja kolmatta yhtälöä vastaa rivi 1.

Oletetaan, että kanavaan on lähetetty nollasana ja kanavasta on vastaanotettu sana $y = 0100000$. Ensimmäiselle bitille saadaan tarkastusyhtälöistä

$$\begin{aligned} y_0 &= 1 + 0 = 1, \\ y_0 &= 0 + 0 = 0, \text{ ja} \\ y_0 &= 0 + 0 = 0. \end{aligned}$$

Näistä enemmistö antaa arvon nolla, joten dekoodataan $y_0 = 0$. Koordinaatille y_1 saadaan

$$\begin{aligned} y_1 &= 0 + 0 = 0, \\ y_1 &= 0 + 0 = 0, \text{ ja} \\ y_1 &= 0 + 0 = 0, \end{aligned}$$

joten dekoodataan $y_1 = 0$. Virhe vaikuttaa kunkin koordinaatin $y_2, ..., y_6$ pariteetintarkastusyhtälöistä vain yhteen aivan kuten koordinaatin y_0 kohdalla. Koko sana dekoodattuna on 0000000.

Joillekin koodeille, jotka eivät ole yksivaiheisesti ortogonalisoituvia, majoriteettilogiikkaa voi käyttää monivaiheisesti. Idea on sama, mutta arvio annetaan ensin koordinaattien summille yksittäisten koordinaattien sijaan.

Määritelmä 4.2.6. Pariteetintarkistusyhtälöiden $S_1, S_2, ..., S_J$ joukko on *ortogonaalinen koordinaattien* $a, b, ..., c$ *suhteen*, jos summa $x_a + x_b + ... + x_c$ esiintyy jokaisessa yhtälössä S_i, mutta mikään muu x_j ei esiinny useammassa kuin yhdessä yhtälössä [15].

Ortogonaalista joukkoa käytetään nyt määrittämään arvo ensin summille $x_a + x_b + ... + x_c$. Näiden tietojen avulla saadaan seuraavalla kierroksella arvioitua arvoja summille, joissa on pienempi määrä koordinaatteja. Näin edeten saadaan lopulta arvot yksittäisille koordinaateille. Dekooderia, jossa käydään L kierrosta majoriteettilogiikkaa, sanotaan L-vaiheiseksi dekoodausalgoritmiksi. L-vaiheista majoriteettilogiikkaa käytetään esimerkiksi Reedin-Mullerin koodien dekoodauksessa [15].

4.3 Forneyn yleistetyn minimietäisyyden algoritmi

Esitellään seuraavaksi luvussa 5.5 käsiteltävää yleisempää tulokoodien dekoodausalgoritmia varten tarvittava Forneyn yleistetyn minimietäisyyden algoritmi (*Forney's generalised minimum distance (GMD) decoding algorithm*) [7].

Forneyn algoritmi on pehmeän päätöksen dekoodausalgoritmi, jossa sana dekoodataan yksinkertaisella virheet ja pyyhkiymät korjaavalla dekooderilla hyödyntäen kanavasta biteille saatuja luotettavuustietoja. Algoritmissa edetään kierroksittain merkitsemällä pyyhkiymiksi epäluotettavimmiksi merkittyjä koordinaatteja, ja yritetään dekoodata näin saatu pyyhkiymiä sisältävä sana. Dekoodaustulos hyväksytään menetelmässä määritellyn yleistetyn etäisyyden perusteella. Yhdistämällä kanavan luotettavuusarviot ja algebrallinen dekooderi saadaan Forneyn algoritmissa käyttöön sekä propabilistisen että algebrallisen dekoodauksen hyviä puolia [25].

Olkoon C q-arinen koodi, jonka minimietäisyys on d. Lähetetään koodin C sana $x = (x_0, x_1, ..., x_{n-1})$ kanavaan. Kanavasta luetaan q-arinen sana $r = (r_0, r_1, ..., r_{n-1})$ ja luotettavuusvektori $\alpha = (\alpha_0, \alpha_1, ..., \alpha_{n-1})$, jossa $\alpha_i \in [0, 1]$ ja α_i kertoo bitin r_i luotettavuuden. Mitä suurempi luotettavuusvektorin koordinaatin α_i arvo on, sitä todennäköisemmin kanavasta luetun sanan koordinaatti r_i vastaa lähetetyn sanan koordinaattia c_i.

Dekoodauksen tulosta mitataan yleistetyllä etäisyydellä, joka määritellään seuraavaksi. Vektorin $c = (c_0, c_1, ..., c_{n-1})$ ja vektorin r, jolla on luotettavuusvektori α, *yleistetty etäisyys* (*generalized distance*) $d_G(c, r, \alpha)$ on

$$d_G(c, r, \alpha) = \sum_{i|c_i=r_i} (1 - \alpha_i)/2 + \sum_{i|c_i \neq r_i} (1 + \alpha_i)/2.$$

Algoritmin tulosteeksi valitaan ensimmäinen saatava dekoodaustulos $c^{(j)}$, jolla dekoodatun sanan $c^{(j)}$ ja vastaanotetun sanan r yleistetty etäisyys noudattaa kaavaa $d_G(c^{(j)}, r, \alpha) < d/2$. Dekoodaus tapahtuu seuraavan algoritmin mukaan [25].

Algoritmi 4.3.1. *Olkoon C (n, k, d)-koodi, jolla on tiedossa dekooderi, joka korjaa kaikki virheet ja pyyhkiymät, kun $d \geq 2e + \varepsilon + 1$. Kanavasta on vastaanotettu sana r ja sille luotettavuusvektori α.*

1. *Aloitetaan arvolla $j = 1$. Kierroksella j muodostetaan sana $r^{(j)}$ merkitsemällä sanan r biteistä pyyhkiymiksi $2j - 1$ epäluotettavinta koordinaattia, jos d on parillinen, tai $2j - 2$ epäluotettavinta koordinaattia, jos d on pariton. Tässä epäluotettavin tarkoittaa sitä koordinaattia, jota vastaava vektorin α koordinaatti on arvoltaan pienin.*

2. *Dekoodataan sana $r^{(j)}$ koodin C dekooderilla ja saadaan sana $c^{(j)}$.*

3. *(i) Jos kohdassa 2 dekoodaus onnistui ja*
$$d_G(c^{(j)}, r^{(j)}, \alpha) < d/2,$$
palautetaan dekoodaustuloksena sana $c^{(j)}$.

   (ii) Jos $d_G(c^{(j)}, r^{(j)}, \alpha) \geq d/2$ tai sanan $r^{(j)}$ dekoodaus epäonnistuu, tehdään uusi dekoodausyritys j:n arvolla $j + 1$.

   (iii) Jos j ylittää arvon $\lceil d/2 \rceil$, lopetetaan ja todetaan dekoodaus epäonnistuneeksi.

Esimerkki 4.3.2. Olkoon C Hammingin (7,4)-koodi. Olkoon kanavasta luettu sana $r = 0001001$ ja sanan luotettavuusvektori $\alpha = (1, \frac{9}{10}, \frac{9}{10}, \frac{2}{10}, \frac{9}{10}, \frac{8}{10}, \frac{3}{10})$.

$j = 1$: Koodin C minimietäisyys $d = 3$ on pariton, joten korvataan $2j - 2 = 0$ bittiä pyyhkiymillä, eli dekoodataan sana $r^{(1)} = r = 0001001$.
0001001 dekoodataan sanaksi $c^{(1)} = 1001001$. Lasketaan tälle yleistetty etäisyys $d_G(c^{(1)}, r, \alpha)$. Saadaan

$$\begin{aligned} d_G(c^{(1)}, r, \alpha) =& \frac{1}{20} + \frac{1}{20} + \frac{8}{20} + \frac{1}{20} + \frac{2}{20} + \frac{7}{20} + 1 = 2 \\ >& \frac{d}{2}. \end{aligned}$$

Yleistetty etäisyys on liian suuri, joten tehdään toinen kierros.

$j = 2$: Korvataan $2j - 2 = 2$ bittiä pyyhkiymillä, eli dekoodataan sana $r^{(2)} = 000?00?$. Käytetään pyyhkiymien dekoodaamiseen algoritmia 4.1.1.
Dekoodataan sanat $r_0^{(2)} = 0000000$ ja $r_1^{(2)} = 0001001$, jolloin saadaan koodisanat $c_0^{(2)} = 0000000$ ja $c_1^{(2)} = 1001001$. Näistä dekoodaustulokseksi valitaan $c_0^{(2)}$, koska $c_0^{(2)} = r_0^{(2)}$ eli sen dekoodamisessa korjattiin vähemmän virheitä.
Lasketaan sanalle $c_0^{(2)}$ yleistetty etäisyys $d_G(c_0^{(2)}, r, \alpha)$. Saadaan

$$\begin{aligned} d_G(c_0^{(2)}, r, \alpha) =& 0 + \frac{1}{20} + \frac{1}{20} + \frac{1}{20} + \frac{2}{20} + \frac{12}{20} + \frac{13}{20} = \frac{3}{2} \\ =& \frac{d}{2}, \end{aligned}$$

joten dekoodaustulos $c_0^{(2)} = 0000000$ hyväksytään.

Esimerkissä nähtiin tyypillinen tilanne, jossa luotettavuustietojen käyttö muutti dekooderin toimintaa. Vastaavassa tilanteessa kovapäätösdekoodauksessa kaksi epäluotettavaa päätöstä johtaisi siihen, että yksi hyvin luotettava bitti dekoodattaisiin väärin. Todennäköisempi selitys vastaanotetulle viestille on kuitenkin, että virheellisesti on tulkittu epäluotettavat kaksi bittiä.

Forneyn algoritmia käytetään myös muodossa, jossa valitaan aina paras tehdyistä dekoodausyrityksistä silloinkin, kun algoritmissa 4.3.1 dekoodaus todettaisiin epäonnistuneeksi. Tällöin tilanteessa, jossa ehto $d_G(c^{(j)}, r^{(j)}, \alpha) < d/2$ ei täyty millään kierroksella j, kun $j < d/2$, käytetään yleistettyä etäisyyttä parhaimman dekoodaustuloksen valitsemiseen.

Dekoodaus tapahtuu kuten algoritmissa 4.3.1. Dekoodaustulos $c^{(j)}$ hyväksytään, jos yleistetylle etäisyydelle on voimassa $d_G(c^{(j)}, r^{(j)}, \alpha) < d/2$, sillä tämä on paras saatava tulos. Dekoodaus eroaa algoritmista 4.3.1 kohdassa 3.(iii). Sen sijaan että dekoodaus todettaisiin epäonnistuneeksi, valitaan nyt dekoodaustuloksista $c^{(j)}$ se, jolla yleistetty etäisyys $d_G(c^{(j)}, r^{(j)}, \alpha)$ on pienin.

Simulaatioiden avulla voi tutkia, onko tästä versiosta sovelluksen kannalta hyötyä, vai johtaako se häiritsevän usein virhetulkintaan.

5 Tulokoodien dekoodaus

5.1 Johdanto dekoodaukseen

Tulokoodien dekoodauksessa hyödynnetään lähtökoodien dekoodausalgoritmeja. Yksinkertaisimmillaan voidaan käyttää seuraavanlaista algoritmia.

Algoritmi 5.1.1. *Olkoon $C = C_1 \otimes C_2$. Dekoodataan kanavasta luettu sana y seuraavasti*

1. *Dekoodataan pystyrivit koodin C_1 dekoodausalgoritmilla. Saadaan alustavasti dekoodattu sana y'.*

2. *Dekoodataan sanan y' vaakarivit koodin C_2 dekoodausalgoritmilla ja saadaan sana y''. Jos y'' on koodisana, hyväksytään se dekoodaustulokseksi. Muussa tapauksessa todetaan dekoodauksen epäonnistuneen.*

Algoritmilla 5.1.1 ei kuitenkaan saada korjattua kaikkia painoltaan alle $(d-1)/2$ olevia virheitä. Luvuissa 5.2 ja 5.3 tarkastellaan, millaisia virheitä tällaisella dekooderilla saadaan korjattua ja millaisten virhekuvioiden kohdalla ajaudutaan ongelmiin.

Luvuissa 5.4 ja 5.5 esitellään kehittyneemmät algoritmit, joissa vaakarivien dekoodauksessa hyödynnetään pystyrividekooderilta saatavaa luotettavuusdataa. Tällöin selvitään myös virhekuvioista, jotka yksinkertaisella dekooderilla tuottivat dekoodausvirheitä tai aiheuttivat dekoodauksen epäonnistumisen. Tällöin saadaan korjattua kaikki virheet ja pyyhkiymät rajaan $2e + \varepsilon < d$ asti.

5.2 Ryöppyvirheet

Tulokoodien rakenteesta johtuen peräkkäisiä virheitä saadaan korjattua suurempi määrä kuin koodin minimietäisyydestä saatava $(d-1)/2$ satunnaisvirhettä. Kanavassa virheitä esiintyykin usein useampia peräkkäin esimerkiksi kanavan pulssimaisen häiriön tai tallennusvälineessä olevan virheen, kuten DVD-levyn naarmun, takia. Tällaisia peräkkäisiä virheitä kutsutaan ryöpyiksi.

Ryöppyjä korjataan lisäämällä koodiin lomittelua (*interleaving*), jossa järjestelemällä peräkkäisten sanojen koordinaatteja uudelleen saadaan peräkkäiset virheet jakautumaan useampaan sanaan. Tulokoodien rakenteeseen sisältyy kuitenkin implisiittistä lomittelua: vaakariveille syntyvä ryöppy tekee vaakarivit lukukelvottomiksi, mutta jakautuu useampaan pystyriviin, ja sana saadaan korjattua. Käsitellään seuraavaksi tätä tulokoodien rakenteeseen sisältyvää ryöppyvirheiden korjausta.

Määritelmä 5.2.1. Vektoria $e \in V_n(q)$ sanotaan *ryöpyksi*, jos se on muotoa

$$(0, 0, ..., 0, e_i, e_{i+1}, ..., e_{i+b-1}, 0, 0, ..., 0),$$

jossa $e_i \neq 0$ ja $e_{i+b-1} \neq 0$. Ryöpyn e pituus on b. Koodia, joka pystyy korjaamaan mielivaltaisen enintään b:n pituisen ryöpyn, sanotaan b-ryöppyvirheet korjaavaksi koodiksi.

Lause 5.2.2. *Olkoon C_1 (n_1, k_1, d_1)-koodi ja C_2 (n_2, k_2, d_2)-koodi. Jos C_1 on l_1-ryöppyvirheet korjaava ja C_2 on l_2-ryöppyvirheet korjaava, tulokoodi $C_1 \otimes C_2$ on $\max\{n_1 l_2, n_2 l_1\}$-ryöppyvirheet korjaava.*

Todistus. Oletetaan, että koodin $C_1 \otimes C_2$ sana lähetetään kanavaan vaakariveittäin. Kun kanavasta vastaanotettu sana kootaan taas matriisimuotoon, kanavassa syntynyt enintään $n_1 l_2$:n pituinen ryöppy osuu enintään l_2 peräkkäiseen vaakariviin. Ryöppy siis vaikuttaa kussakin pystyrivissä enintään l_2:n pituisena ryöppyvirheenä. Koska C_2 on l_2-ryöppyvirheet korjaava, sen korjausalgoritmi pystyy korjaamaan pystyrivit. Koodi $C_1 \otimes C_2$ on siis $n_1 l_2$-ryöppyvirheet korjaava.

Vastaavasti lähettämällä sana kanavaan pystyriveittäin saadaan korjattua $n_2 l_1$:n pituiset ryöpyt. Koodi $C_1 \otimes C_2$ on siis $\max\{n_1 l_2, n_2 l1\}$-ryöppyvirheet korjaava [12]. □

Koodin C_1 ryöppyvirheiden korjauskyky l_1 on vähintään yhtä suuri kuin satunnaisvirheiden korjauskyky $(d_1 - 1)/2$, ja samoin koodilla C_2 pätee $l_2 \geq (d_2 - 1)/2$. Tulokoodin ryöppyvirheiden korjauskyky $\max\{n_1 l_2, n_2 l_1\}$ on siis selvästi suurempi kuin satunnaisvirheitä korjattava määrä $(d_1 d_2 - 1)/2$. Tarkastellaan tätä esimerkin kautta.

Esimerkki 5.2.3. Olkoon C kahden Hammingin $(7, 4)$-koodin tulokoodi. Hammingin $(7, 4)$-koodin minimietäisyys on 3, joten tulokoodin minimietäisyys on 9. Koodi pystyy siis korjaamaan neljä satunnaisvirhettä. Hammingin koodi korjaa ryöppyvirheitä vastaavan määrän kuin satunnaisvirheitä, eli yhden virheen. Tulokoodilla saadaan kuitenkin lauseen 5.2.2 mukaan korjattua peräkkäisiä virheitä $7 \times 1 = 7$ pituisiin ryöppyihin asti.

Käytetään dekoodaukseen algoritmia 5.1.1. Koska algoritmissa dekoodaus aloitetaan pystyriveistä, kannattaa koodisana lähettää kanavaan vaakariveittäin. Näin kanavassa mahdollisesti syntyvä ryöppyvirhe jakautuu mahdollisimman moneen pystyriviin ja sen dekoodaus helpottuu.

Oletetaan, että kanavaan lähetetään nollasana ja siihen syntyy seitsemän bitin mittainen ryöppyvirhe alkaen koordinaatista $(2, 6)$:

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Ryöppy vaikuttaa jokaiseen pystyriviin enintään yhdessä koordinaatissa, joten Hammingin koodin dekoodausalgoritmi korjaa pystyrivit oikein ja ryöppy saadaan korjattua.

Esimerkki 5.2.4. Lause 5.2.2 lupaa kahden Hammingin (7,4)-koodin tulokoodin korjaavan seitsemän mittaiset ryöpyt. Koodi korjaa kuitenkin myös kaikki kahdeksan mittaiset ryöppyvirheet. Tarkastellaan kahdeksan mittaista ryöppyä, joka alkaa samasta koordinaatista kuin edellisessä esimerkissä. Ryöppyvirhe on muotoa

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Pystyrivien dekoodaus korjaa näistä kaikki paitsi viimeisen pystyrivin, jossa tapahtuu dekoodausvirhe. Saadaan alustavasti korjattu sana

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Koska virheitä on vain yhdellä pystyrivillä, vaakarividekooderi korjaa tämän oikein ja saadaan nollasana.

Esimerkki 5.2.5. Toisen iteraation lisääminen algoritmiin 5.1.1 auttaa erityistapauksissa korjaamaan enemmän virheitä. Tarkastellaan yhdeksän pituista ryöppyä alkaen koordinaatista $(1, 6)$:

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Pystyrivien dekoodaus tuottaa matriisin

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

ja vaakarivien dekoodaus matriisin

$$\left(\begin{array}{cccc|ccc} 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 1&0&0&1&0&0&1\\ 0&0&0&0&0&0&0\\ \hline 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{array}\right).$$

Jos nyt dekoodataan pystyrivit vielä kerran, saadaan matriisi dekoodattua nollamatriisiksi.

5.3 Pyyhkiymien hyödyntäminen tulokoodin dekoodauksessa

Algoritmi 5.1.1 toimii vain sellaisilla virhekuvioilla, joilla pystyrivien dekoodaus tuottaa matriisin, johon jäävien virheiden kuvio on vaakarivikoodin dekoodattavissa [12]. Jos pystyrivikoodilla C_1 on dekooderi, joka korjaa $\leq (d_1 - 1)/2$ virhettä ja vaakarivikoodilla C_2 on dekooderi, joka korjaa $\leq (d_2 - 1)/2$ virhettä, tällainen dekoodaus korjaa vain noin $d_1 d_2/4$ virhettä. On helppo löytää virhe, joka ei ole dekoodattavissa, ja jonka paino on $(\lfloor (d_1 - 1)/2 \rfloor + 1)(\lfloor (d_2 - 1)/2 \rfloor + 1)$ [6]. Seuraavana on esimerkki virhekuviosta, jota ei algoritmilla 5.1.1 saada korjattua.

Esimerkki 5.3.1. Tarkastellaan taas esimerkin 5.2.3 koodia. Koodi C on kahden Hammingin $(7, 4)$-koodin tulokoodi ja sen minimietäisyys d on 9. Oletetaan, että kanavaan lähetetään nollasana ja kanavassa tapahtuu virheet koordinaateissa $(3, 3)$, $(3, 5)$, $(7, 3)$ ja $(7, 5)$ eli kanavasta luetaan matriisi

$$\left(\begin{array}{cccc|ccc} 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0\\ \hline 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0 \end{array}\right).$$

Nyt virheitä on neljä ja $d \geq 2 \times 4 + 1$ eli koodin C pitäisi pystyä korjaamaan ne. Algoritmi 5.1.1 ei kuitenkaan dekoodaa sanaa oikein, koska virhekuvio muodostaa kaksi virhettä sekä pysty- että vaakariveille, eikä Hammingin $(7, 4)$-koodin algoritmi kykene korjaamaan niitä oikein. Neljännellä ja kuudennella pystyrivillä dekooderi olettaa virheen tapahtuneen ensimmäisessä bitissä. Saadaan matriisi

$$\left(\begin{array}{cccc|ccc} 0&0&0&1&0&1&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0\\ \hline 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0 \end{array}\right).$$

Vaakarivien dekoodauksessa on sama kahden virheen ongelma, ja matriisi dekoodataan virheellisesti sanaksi

$$\left(\begin{array}{cccc|ccc} 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array}\right).$$

Esimerkki 5.3.2. Jos edellisen esimerkin tilanteessa käytössä olisi ollut BEC-kanava, ja virheelliset bitit c_{33}, c_{35}, c_{53} ja c_{55} olisi ykkösten sijaan tulkittu pyyhkiymiksi, olisi pystyrividekooderi saanut ne korjattua.

Hyödynnetään tulokoodien hyvää ryöppyvirheiden sietokykyä ja tietoa, että Hammingin koodi pystyy korjaamaan pyyhkiymiä kaksi. Merkitään virheelliset rivit kokonaan pyyhkiymiksi, jolloin saadaan matriisi

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & ? & 0 & ? & 0 \\ 0 & 0 & 0 & ? & 0 & ? & 0 \\ 0 & 0 & 0 & ? & 0 & ? & 0 \\ ? & ? & ? & ? & ? & ? & ? \\ \hline 0 & 0 & 0 & ? & 0 & ? & 0 \\ 0 & 0 & 0 & ? & 0 & ? & 0 \\ ? & ? & ? & ? & ? & ? & ? \end{array}\right).$$

Dekoodataan tämä pystyriveittäin. Esimerkissä 4.1.2 nähtiin, että Hammingin (7,4)-koodi korjaa kaksi pyyhkiymää, joten dekooderi selviää kaikista pystyriveistä lukuunottamatta niitä, joilla kaikki bitit ovat pyyhkiymiä. Saadaan

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & ? & 0 & ? & 0 \\ 0 & 0 & 0 & ? & 0 & ? & 0 \\ 0 & 0 & 0 & ? & 0 & ? & 0 \\ 0 & 0 & 0 & ? & 0 & ? & 0 \\ \hline 0 & 0 & 0 & ? & 0 & ? & 0 \\ 0 & 0 & 0 & ? & 0 & ? & 0 \\ 0 & 0 & 0 & ? & 0 & ? & 0 \end{array}\right).$$

Vaakariveillä on nyt kullakin kaksi pyyhkiymää, joten vaakarividekooderi selviää niistä ja koko sana saadaan dekoodattua:

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Tässä lähestymisessä ongelmana on tietää, milloin yrittää suoraan dekoodata ja milloin merkitä virheelliset rivit pyyhkiymiksi. Hamming (7,4) on täydellinen koodi, joten kaikki ei-koodisanat ovat yhden bitin päässä jostain koodisanasta. Koodi ei siis tunnista, onko tapahtunut yhden vai kahden bitin virhe ja kahden bitin virhe dekoodataan virheellisesti yhden bitin virheenä.

Jos taas ensimmäisellä kierroksella merkittään kaikki virheelliset rivit pyyhkiymiksi, syntyy tilanteita, joissa pyyhkiymiä on liikaa korjattavaksi, mutta itse virheet olisivat olleet korjattavissa. Tällainen tilanne on, jos virheet ovat kolmella tai neljällä eri vaaka- ja pystyrivillä, kuten esimerkiksi virhekuviolla

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Jos nyt yritetään muuttaa virheelliset rivit pyyhkiymiksi, päädytään tilanteeseen, jossa on kolme pyyhkiymää sekä pysty- että vaakariveillä, mikä on liikaa Hammingin (7,4)-koodille. Tämä virhekuvio olisi sen sijaan algoritmilla 5.1.1 suoraan korjattavissa: kaikki pystyrivit viimeistä lukuunottamatta menevät nolliksi. Viimeisellä pystyrivillä tehdään dekoodausvirhe, mutta se korjaantuu vaakarivien dekoodauksessa.

Esimerkki 5.3.3. Toisin kuin Hammingin $(7, 4)$-koodi, laajennettu Hammingin (8,4)-koodi tunnistaa, onko kanavassa tapahtunut virheitä pariton vai parillinen määrä. Jos siis valitaan lähtökoodeiksi C_1 ja C_2 Hammingin (8,4)-koodit, voidaan tätä ominaisuutta hyödyntää dekoodauksessa. Tulokoodin $C = C_1 \otimes C_2$ minimietäisyys on 16, joten sen tulisi korjata 7 virhettä.

Dekoodataan tulokoodin sana seuraavasti: jos pystyrivi ei ole koodisana ja pystyrivikoodin dekooderi huomaa virheitä parittoman määrän, pystyrivi merkitään pyyhkiymiksi. Jos virheitä on pariton määrä, yritetään pystyrivi korjata.

Nyt dekoodaus onnistuu esimerkiksi virhekuviolla

$$\left(\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Pystyrivien dekoodauksen jälkeen saadaan

$$\left(\begin{array}{cccc|cccc} 0 & ? & 0 & ? & ? & 0 & 0 & 0 \\ 0 & ? & 0 & ? & ? & 0 & 0 & 0 \\ 0 & ? & 0 & ? & ? & 0 & 0 & 0 \\ 0 & ? & 0 & ? & ? & 0 & 0 & 0 \\ \hline 0 & ? & 0 & ? & ? & 0 & 0 & 0 \\ 0 & ? & 0 & ? & ? & 0 & 0 & 0 \\ 0 & ? & 0 & ? & ? & 0 & 0 & 0 \\ 0 & ? & 0 & ? & ? & 0 & 0 & 0 \end{array}\right).$$

Hammingin (8,4)-koodi korjaa kolme pyyhkiymää, saadaan kaikki pyyhkiymät vaakarivien dekoodauksessa korjattua.

Kaikkien seitsemänpainoisten virhekuvioiden dekoodaus ei kuitenkaan onnistu tällä menetelmällä. Tarkastellaan virhekuviota

$$\left(\begin{array}{cccc|cccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Vektori 11010000 on yhden päässä koodisanasta 11011000, joten pystyrivien dekoodauksen jälkeen saadaan

$$\left(\begin{array}{cccc|cccc} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Vektoria 01100000 ei saada vaakarividekooderilla dekoodattua. Jos taas vaakarivit nyt merkittäisiin pyyhkiymiksi, olisi pystyriveillä kullakin neljä pyyhkiymää, mikä on pystyrividekooderille liian paljon.

5.4 Wainbergin majoriteettilogiikka-algoritmi

Esitellään Wainbergin tulokoodien dekoodausalgoritmi [24], joka korjaa kaikki virheet ja pyyhkiymät rajaan $d \geq 2e + \varepsilon + 1$ asti [23]. Ainakin toisen lähtökoodeista on oltava majoriteettilogiikalla dekoodattava.

Algoritmissa oletetaan, että vaakarivikoodi C_2 on yhdessä vaiheessa ortogonalisoituva, jolloin jokaiselle koodin C_2 informaatiobitille on olemassa $d_2 - 1$ ortogonaalista pariteetintarkistusyhtälöä. Bitin vastaanotettu arvo mukaan luettuna bitin

arvolle saadaan siis d_2 lineaarisesti riippumatonta estimaattia. Menetelmä mukautuu myös L-vaiheisesti ortogonalisoituvalle koodille.

Pystyrivikoodin C_1 ei tarvitse olla majoriteettilogiikalla dekoodattava, vaan riittää, että sillä on dekoodausalgoritmi, joka korjaa e virhettä ja ε pyyhkiymää, kun $d_1 \geq 2e + \varepsilon + 1$ [24].

Algoritmi 5.4.1. *Olkoon $C = C_1 \otimes C_2$ tulokoodi, jossa C_1 on (n_1, k_1, d_1)-koodi ja C_2 on (n_2, k_2, d_2)-koodi. Olkoon C_2 yhdessä vaiheessa ortogonalisoituva koodi. Kanavasta on vastaanotettu sana y.*

1. *Dekoodataan matriisin y pystyrivit käyttäen koodin C_1 dekoodausalgoritmia. Jokaiselle pystyriville lasketaan paino $(-2e_i - \varepsilon_i)$, jossa e_i on arvioitu virheiden määrä ja ε_i pyyhkiymien määrä pystyrivillä i, $0 \leq i \leq n_2$.*

   Saadaan alustavasti dekoodattu sana r sekä painovektori $\alpha = (\alpha_0, \alpha_1, ..., \alpha_{n_2-1})$, jossa $\alpha_i = -2e_i - \varepsilon_i$.

2. *Dekoodataan alustavasti dekoodatun sanan r vaakarivit käyttämällä majoriteettilogiikan antamia estimaatteja. Estimaateista valitaan luotettavin pystyrividekoodauksessa lasketun luotettavuusvektorin avulla.*

   Jokaisella vaakarivin bitillä on d_2 riippumatonta estimaattia. Näille määrätään painot seuraavasti: asetetaan paino d_1, johon lisätään estimaatissa esiintyvien bittien vektorissa α esiintyvät painot. Jos summasta tulee negatiivinen, määrätään paino nollaksi. Nyt informaatiobitille valitaan sen estimaatin antama arvo, jonka paino on suurin.

Esimerkki 5.4.2. Olkoon $C = C_1 \otimes C_2$, jossa C_1 on Hammingin $(7, 4)$-koodi ja C_2 on esimerkissä 4.2.5 käytetty Simplex $(7, 3, 4)$-koodi. Koodin C minimietäisyys on 12.

Oletetaan, että kanavaan on lähetetty nollasana ja kanavasta on vastaanotettu sana

$$y = \left(\begin{array}{ccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & ? & 1 & 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

1. Dekoodataan sanan y pystyrivit koodin C_1 dekooderilla. Saadaan alustavasti dekoodattu sana

$$\left(\begin{array}{ccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & ? & 1 & 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right).$$

Dekoodauksessa ensimmäisessä pystyrivissä arvoitiin olleen yksi virhe, joten saadaan $\alpha_0 = -2$. Toisella pystyrivillä on kolme pyyhkiymää, joten saadaan $\alpha_1 = -3$. Kolmannella pystyrivillä saadaan ensimmäisen tavoin $\alpha_2 = -2$, ja lopuilla pystyriveillä ei dekoodauksessa tarvittu korjauksia, joten $\alpha_3 = \alpha_4 = \alpha_5 = \alpha_6 = 0$. Painovektori α on siis

$$\alpha = (-2, -3, -2, 0, 0, 0, 0).$$

2. Dekoodataan vaakarivit.

   Rivi 1: Ensimmäisellä rivillä kaikkien koordinaattien arvo on 0, joten se selvästi dekoodataan nollasanaksi.

   Rivi 2: $r = (101000)$, $\alpha = (-2, -3, -2, 0, 0, 0, 0)$ Käytetään esimerkissä 4.2.5 saatuja pariteetintarkistusyhtälöitä ja bitille r_0 saadaan

   $$\begin{aligned} &r_0 = 1, && \text{paino } 4 - 2 = 2 \\ &r_0 = r_1 + r_3 = 0 + 0 = 0, && \text{paino } 4 - 3 + 0 = 1 \\ &r_0 = r_4 + r_5 = 0 + 0 = 0, && \text{paino } 4 + 0 + 0 = 4 \\ &r_0 = r_2 + r_6 = 1 + 0 = 1, && \text{paino } 4 - 2 + 0 = 2 \end{aligned}$$

   Näistä suurin paino on kolmannella estimaatilla, josta saadaan $r_0 = 0$. Bitille r_1 saadaan estimaatit

   $$\begin{aligned} &r_1 = 0, && \text{paino } 4 - 3 = 1 \\ &r_1 = r_2 + r_4 = 1 + 0 = 1, && \text{paino } 4 - 2 + 0 = 2 \\ &r_1 = r_5 + r_6 = 0 + 0 = 0, && \text{paino } 4 + 0 + 0 = 4 \\ &r_1 = r_0 + r_3 = 1 + 0 = 1, && \text{paino } 4 - 2 + 0 = 2 \end{aligned}$$

   Näistä suurin paino on taas kolmannella estimaatilla, josta saadaan $r_1 = 0$. Jatketaan rivin dekoodausta vastaavasti, ja saadaan 0000000.

   Rivi 3: Kuten rivi 1.

   Rivi 4: Pyyhkiytyneen bitin r_1 korkeapainoisin estimaatti on $r_1 = r_5 + r_6 = 0$, joten se dekoodataan nollaksi. Saadaan rivi 0000000.

   Rivi 5: Bitin r_0 laadukkain estimaatti on edelleen $r_4 + x_5$, josta saadaan $r_0 = 0$. Samoin bitin r_1 laadukkain estimaatti on $r_5 + r_6 = 0$. Bitille r_2 valitaan estimaatti $r_3 + r_5 = 0$.

   Rivi 6: Kuten rivi 4.

Rivi 7: Kuten rivi 2.

Lopullinen dekoodattu sana on

$$\left(\begin{array}{ccc|cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

5.5 Wainbergin yleinen dekoodausalgoritmi

Seuraavaksi käsitellään algoritmia, jolla voi dekoodata minkä tahansa tulokoodin $C = C_1 \otimes C_2$. Ainoat kriteerit koodeille C_1 ja C_2 ovat, että niille on käytettävissä dekooderit jotka korjaavat kaikki virheet ja pyyhkiymät rajaan $2e + \varepsilon + 1 \leq d_i$ asti [24]. Algoritmilla saadaan korjattua kaikki tulokoodin virheet ja pyyhkiymät, kunhan niiden lukumäärä noudattaa rajaa $2e + \varepsilon + 1 \leq d$ [23].

Algoritmi 5.5.1. *Olkoon $C = C_1 \otimes C_2$ tulokoodi, jossa C_1 on (n_1, k_1, d_1)-koodi ja C_2 on (n_2, k_2, d_2)-koodi. Kanavasta on vastaanotettu sana y.*

1. *Dekoodataan sanan y pystyrivit käyttäen koodin C_1 dekooderia, jolloin saadaan alustavasti dekoodattu sana r. Kullekin pystyriville i määritetään luotettavuuspaino (engl. reliability weight) α_i, jolle*

$$\alpha_i = \begin{cases} \dfrac{d_1 - 2e_i - \varepsilon_i}{d_1}, & \text{kun } d_1 - 2e_i - \varepsilon_i > 0 \\ 0, & \text{kun } d_1 - 2e_i - \varepsilon_i \leq 0 \\ & \text{tai pystyrivin dekoodaus epäonnistuu,} \end{cases}$$

   missä e_i on arvio i:nnessä pystyrivissä olleiden virheiden määrästä ja ε_i on pystyrivissä olleiden pyyhkiymien määrä.

2. *Dekoodataan sanan r vaakarivit käyttäen Forneyn dekoodausalgoritmia (algoritmi 4.3.1) käyttäen i:nnen bitin luotettavuuspainona painoa α_i, $0 \leq i < n_2$.*

Havainnollistetaan algoritmia esimerkillä.

Esimerkki 5.5.2. Olkoon C kahden Hammingin $(7, 4)$-koodin tulokoodi. Hammingin koodin minimietäisyys on 3, joten koodin C minimietisyys on 9. Oletetaan, että kanavaan on lähetetty nollasana ja kanavasta on vastaanotettu sana

$$y = \left(\begin{array}{ccc|cccc} 0 & 0 & ? & 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

1. Dekoodataan pystyrivit. Ensimmäinen pystyrivi on koodisana. Toisella ja kolmannella pystyrivillä on tapahtunut sekä virhe että pyyhkiymä, joten käy kuten esimerkissä 4.1.2, eikä niitä saa korjattua. Neljäs, viides ja seitsemäs rivi ovat koodisanoja. Kuudes rivi dekoodataan nollasanaksi.

   Lasketaan näille luotettavuuspainot. Pystyriveillä 1, 4, 5 ja 7 ei tarvinnut korjata yhtään virhettä, joten $\alpha_0 = \alpha_3 = \alpha_4 = \alpha_6 = 1$. Toisella ja kolmannella rivillä dekoodaus epäonnistui, joten $\alpha_1 = \alpha_2 = 0$. Kuudes rivi dekoodattiin koodisanaksi korjaamalla yksi koordinaatti, joten $\alpha_5 = \frac{1}{3}$.

   Saadaan alustavasti dekoodattu sana r ja painovektori α

$$r = \left(\begin{array}{cccc|ccc} 0 & 0 & ? & 0 & 0 & 0 & 0 \\ 0 & ? & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) \qquad \text{ja} \qquad \alpha = (1, 0, 0, 1, 1, \tfrac{1}{3}, 1).$$

2. Dekoodataan vaakarivit r_i, $0 \leq i \leq 6$, Forneyn algoritmilla käyttäen luotettavuusvektorina painovektoria α. Koodin C_2 minimietäisyys $d = 3$ on pariton, joten korvataan $2j - 2$ bittiä pyyhkiymillä.

   $j = 1$:

   Korvataan $2j - 2 = 0$ bittiä pyyhkiymillä.

   Rivi 1: $r_0^{(1)} = 00?0000$, $c_0^{(1)} = 0000000$:

$$\begin{aligned} d_G(c_0^{(1)}, r_0^{(1)}, \alpha) &= \sum_{i|c_i = r_i} (1 - \alpha_i)/2 + \sum_{i|c_i \neq r_i} (1 + \alpha_i)/2 \\ &= \sum_{i \neq 2} (1 - \alpha_i)/2 + \sum_{i=2} (1 + \alpha_i)/2 \\ &= 0 + \tfrac{1}{2} + 0 + 0 + \tfrac{1}{3} + 0 + \tfrac{1}{2} \\ &= \tfrac{4}{3} < \tfrac{d}{2} = \tfrac{3}{2}, \end{aligned}$$

   joten hyväksytään $c_0^{(1)}$.

   Rivi 2: Kuten rivi 1.

   Rivi 3: $r_2^{(1)} = 0000000$, $c_2^{(1)} = 0000000$:

$$\begin{aligned} d_G(c_2^{(1)}, r_2^{(1)}, \alpha) &= \sum_{i} (1 - \alpha_i)/2 \\ &= \tfrac{4}{3} < \tfrac{d}{2}, \end{aligned}$$

   joten hyväksytään $c_2^{(1)}$.

Rivi 4: $r_3^{(1)} = 0110000$, $c_3^{(1)} = 1110000$:

$$\begin{aligned} d_G(c_3^{(1)}, r_3^{(1)}, \alpha) &= \sum_{i \neq 0} (1 - \alpha_i)/2 + \sum_{i=0} (1 + \alpha_i)/2 \\ &= \tfrac{1}{2} + \tfrac{1}{2} + 0 + 0 + \tfrac{1}{3} + 0 + 1 \\ &= \tfrac{7}{3}. \end{aligned}$$

Tämä on suurempi kuin $d/2$, joten neljännelle riville tarvitaan uusi iteraatio.

Rivi 5: Kuten rivi 3.

Rivi 6: Kuten rivi 3.

Rivi 7: Kuten rivi 3.

$j = 2$:

Korvataan $2j - 2 = 2$ bittiä pyyhkiymillä. Vektorin α koordinaateista pienimmät arvot ovat α_1 ja α_2, joten pyyhkiymiksi merkitään toinen ja kolmas bitti. Toinen iteraatio tehdään riville neljä, jolla yleistetty etäisyys oli ensimmäisellä iteraatiolla liian suuri.

Rivi 4: $r_3^{(2)} = 0??0000$, $c_3^{(2)} = 0000000$:

$$\begin{aligned} d_G(c_3^{(2)}, r_3^{(2)}, \alpha) &= \sum_{i | c_i = r_i} (1 - \alpha_i)/2 + \sum_{i | c_i \neq r_i} (1 + \alpha_i)/2 \\ &= \sum_{i \notin \{1,2\}} (1 - \alpha_i)/2 + \sum_{i \in \{1,2\}} (1 + \alpha_i)/2 \\ &= \tfrac{4}{3} < \tfrac{d}{2}, \end{aligned}$$

joten $c_3^{(2)}$ hyväksytään.

Saadaan dekoodattu sana

$$\left(\begin{array}{cccc|ccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Neljännellä rivillä nähtiin tilanne, jossa vaakarivikoodin eli Hammingin (7,4)-koodin oma dekooderi ei saanut dekoodattua sanaa oikein, koska virheitä oli liikaa. Forneyn algoritmin ja biteille annettujen luotettavuuspainojen avulla saatiin kuitenkin virheiden paikat selville ja rivi dekoodattua oikein.

6 Lopuksi

6.1 Peittosäteestä

Tulokoodin $C = C_1 \otimes C_2$ minimietäisyys on $d = d_1 d_2$, joten se korjaa kaikki satunnaisvirheet, joiden paino on $\leq (d-1)/2$. Vaikka tulokoodin minimietäisyys on lähtökoodien minimietäisyyksiin verrattuna suuri, ei se kuitenkaan ole erityisen suuri suhteessa tulokoodin pituuteen. Tulokoodien minimietäisyys onkin pieni verrattuna sellaisiin optimaalisiin koodeihin, joilla on sama pituus ja informaatiosuhde. Jos siis korjataan virheitä vain painoon $(d-1)/2$ asti, ei tulokoodi ole vaihtoehtoisiin koodeihin verrattuna erityisen kiinnostava [1]. Tulokoodien rakenne kuitenkin mahdollistaa sen, että tiettyjen virhekuvioiden tapauksessa on mahdollista korjata selvästi enemmänkin virheitä.

Esimerkki 6.1.1. Käsitellään tulokoodia $C_1 \otimes C_2$, jossa sekä C_1 että C_2 ovat Hammingin $(7, 4, 3)$-koodeja. Tulokoodin minimietäisyys $d = 3 \times 3 = 9$, eli koodi korjaa kaikki virhevektorit painoon 4 asti. Kanavasta on vastaanotettu vektori y, jonka paino on 5 ja kaikki viisi ykköstä ovat samalla vaakarivillä. Osoitetaan, että nollasana on vektoria y lähin koodisana.

Todistus: Jos jokin toinen koodisana x on etäisyydellä $d(x, y) < 5$ vektoria y, niin sanan x Hammingin paino on enintään $5 + d(x, y) < 10$. Koska x on nollasana, niin sen on oltava painoa 9. Jokaisessa nollasta eroavassa koodisanassa ykkösiä on ainakin kolmella pystyrivillä, ja näillä jokaisella vähintään kolme ykköstä. Voidaan päätellä, että sanassa x on sanasta y eroavilla kahdella rivillä vähintään 3 ykköstä. Näin ollen sen Hammingin etäisyys vektorista y on vähintään $3 + 3 + 2 = 8$. Tämä on ristiriita.

Esimerkissä 6.1.1 tarkasteltiin tilannetta, jossa virhekuvion paino ylittää rajan $(d-1)/2$, mutta on silti olemassa yksiselitteinen koodisana, joka on lähimpänä vastaanotettua sanaa y. Tulokoodien dekoodaamiseen kannattaakin käyttää dekooderia, jolla pyritään löytämään vastaanotettua vektoria lähin koodisana silloinkin, kun virheiden määrä ylittää minimietäisyyden antaman rajan $(d-1)/2$. Tulokoodien koko virheenkorjauspotentiaalin selvittämiseksi niitä kannattaakin tarkastella minimietäisyyden lisäksi muillakin mittareilla. Määritellään seuraavaksi peittosäde, jonka avulla voidaan määrittää väli, jolle osuvista virheistä osa on korjattavissa.

Määritelmä 6.1.2. Koodin $C \subseteq \mathbb{F}_q^n$ *peittosäde* on pienin sellainen kokonaisluku ρ, että kaikki joukon $\mathbb{F}_q^n$ vektorit ovat enintään etäisyyden ρ päässä jostakin koodin C sanasta, eli

$$\rho = \rho(C) = max\{d(x, C) | x \in \mathbb{F}_q^n\},$$

jossa $d(x, C) = min\{d(x, c) | c \in C\}$.

Peittosäde mittaa siis etäisyyden koodisanojen ja kauimpien ei-koodisanojen välillä. Virheitä korjaavalla koodilla peittosäde on suurin mahdollinen korjattavissa olevan virheen paino [4].

Pitkille koodeille peittosäteen arviointi on vaikeaa [1]. Tulokoodien peittosäteelle on kuitenkin löydetty seuraava alaraja [4]. Olkoon $C = C_1 \otimes C_2$ tulokoodi ja lähtökoodeilla C_1 ja C_2 peittosäteet $\rho(C_1)$ ja $\rho(C_2)$. Tulokoodin peittosäteelle $\rho(C)$ on voimassa

$$\rho(C) \geq \max\{n_2\rho(C_1), n_1\rho(C_1)\}.$$

Tulokoodin peittosäde ρ on selvästi suurempi kuin sen minimietäisyys, joten rajan $(d-1)/2$ ja peittosäteen ρ väliin jää suuri joukko mahdollisesti korjattavissa olevia virheitä. Tulokoodien koko virheenkorjauspotentiaalin saa siis käyttöön ainoastaan dekooderilla, joka korjaa virheitä myös yli painon $(d-1)/2$ [1]. Tällöin osa painoltaan arvon $(d-1)/2$ ylittävistä virheistä saadaan korjattua, mutta kommunikaatiosysteemin täytyy sallia bittivirheitä, koska kaikkia tällaisia virheitä ei pystytä korjaamaan.

6.2 Tulokoodien laajennuksia

Tässä työssä käsitellään tavallisia kaksiulotteisia tulokoodeja. Tulokoodin käsitteen voi laajentaa myös useampaan kuin kahteen ulottuvuuteen käyttämällä useampaa lähtökoodia. Tällöin koodien C_i tulona saadaan $(\prod_i n_i, \prod_i k_i, \prod_i d_i)$-koodi. Käyttämällä lähtökoodeina lyhytsanaisia koodeja, joilla on korkea informaatiosuhde, saadaan koodi, jolla on pitkät koodisanat ja jonka dekoodaus on suhteellisen yksinkertaista [16].

Moniulotteisten tulokoodien kohdalla päädytään tekemään suurempi määrä dekoodausoperaatioita, mutta ne tehdään lyhyemmille lähtökoodeille. Esimerkiksi kaksiulotteisella tulokoodilla tehdään joka iteraatiolla $2n$ lähtökoodin dekoodausoperaatiota ja kolmeulotteisella tulokoodilla vastaavasti $3n^2$. Vertailtaessa kaksi- ja moniulotteista tulokoodia, joilla on sama sanan pituus ja sama informaatiosuhde, kaksiulotteinen koodi yleensä voittaa moniulotteisen koodin virheenkorjauskapasiteetissa. Erot eivät kuitenkaan ole suuria [16].

Tulokoodin voi muodostaa myös ei-binäärisistä lähtökoodeista. Binääristen tulokoodien ongelmana on, että korkean koodisuhteen saavuttamiseksi täytyy käyttää huomattavan pitkiä koodeja. Esimerkiksi binäärisillä BCH-tulokoodeilla, joita on käytetty muun muassa satelliittikommunikaatiosysteemeissä, sanan pituuden täytyy olla yli 60 000 bittiä, jotta saavutetaan koodisuhde 0.9. Tällöin järjestelmän käytännön rajoitukset voivat tulla vastaan. Pitkä lohkon pituus lisää dekoodauksen viivettä ja dekooderin tarvitseman muistin kokoa [26].

Ei-binäärisen tulokoodin lähtökoodeina voi käyttää esimerkiksi Reedin-Solomonin koodeja. Nämä suoriutuvat hyvin vertailussa BCH-tulokoodeja vastaan. Esimerkiksi kahden RS(15,13)-koodin tulokoodilla on sama koodisuhde ja virheenkorjauskyky kuin kahden BCH(64,57)-koodin tulokoodilla. RS(13,15)-tulokoodin sanan pituus on 900 bittiä, kun taas BCH(64,57)-tulokoodin sanan pituus on 4096 bittiä, joten RS-tulokoodi on kevyempi dekoodata [26].

Esimerkki 6.2.1. DVD:issä käytetty virheenkorjauskoodi RSPC (Reed Solomon Product Code) on kahden Reedin-Solomonin koodin tulokoodi. Koodin aakkostona on 256 alkion kunta. Yksi kunnan alkio sisältää 8 bittiä eli yhden tavun

verran informaatiota, ja informaatiosanassa on 32 KB dataa. Informaatiosana on 192×172-matriisi, jonka vaakarivit koodataan $(182, 172, 11)$-Reedin-Solomonin koodilla. Pystyrivit koodataan $(208, 192, 17)$-Reedin-Solomonin koodilla, jolloin saadaan 208×182-koodisana [3]. RSPC korjaa enintään 2200 tavun ryöpyn, mikä vastaa 4.6 millimetriä levyn pinnalla [10].

6.3 SISO-dekoodaus

Pehmeän päätöksen SISO-dekoodaus (*soft-input soft-output decoding*) vaatii kovapäätösdekoodaukseen verrattuna huomattavasti enemmän laskentatehoa ja on siksi yleensä hyvin raskas menetelmä. Joillakin koodeilla vaadittu lisälaskentateho kuitenkin kannattaa, sillä SISO-dekoodauksella saadaan dekoodaustuloksiin huomattava parannus. Tulokoodin rakenne on sellainen, että SISO-dekoodausta on mahdollista käyttää ilman, että dekooderista tulee liian monimutkainen. Tulokoodeille onkin kehitetty hyviä pehmeän päätöksen dekoodausalgoritmeja, joiden myötä tulokoodit ovat nousseet kilpailukykyisiksi sovelluksissa [16].

Tulokoodien iteratiivisessa SISO-dekoodauksessa dekoodataan vuorotellen pysty- ja vaakarivejä käyttäen näille SISO-dekoodereita. Kun tätä jatketaan useiden iteraatioiden ajan, saadaan jäljelle jäävien virheiden määrä hyvin matalaksi. Kullakin kierroksella dekoodauksessa hyödynnetään sekä kanavasta saatua että toisen suunnan dekoodauksessa laskettua pehmeää informaatiota [11].

Tulokoodin rakenne mahdollistaa hyvin myös dekoodauksen parallelisoimisen, jolloin koodisanan dekoodaus nopeutuu. Tulokoodin koodisanassa jokainen pystyrivi on riippumaton muista pystyriveistä ja samoin jokainen vaakarivi on riippumaton muista vaakariveistä. Tätä ominaisuutta voidaan hyödyntää dekoodauksessa ja dekoodata kukin pystyrivi itsenäisesti, jolloin vastaanotetusta matriisista voidaan käsitellä useita rivejä samanaikaisesti dekooderia monistamalla [11]. Tulokoodien dekoodauksen parallelisoiminen siis suurentaa virtapiirin kokoa ja vaatii lisätehoa, mutta ei ole monimutkaista. Dekoodauksen parallelisoiminen on erityisesti kiinnostava järjestelmissä, joissa kulkee suuria datavirtoja [2].

Eräissä tutkimuksissa dekoodausviive (*decoding latency*) on saatu pienenemään jopa puoleen käyttämällä parallelisoitua dekooderia, jonka pysty- ja vaakarividekooderit päivittävät toisiaan heti rivin dekoodauksen valmistuttua [2]. Sovelluksissa dekoodausviiveeseen vaikuttaa kuitenkin dekoodauksen kompleksisuuden lisäksi erityisesti koodin lomittelusyvyys.

6.4 Yhteenveto

Tulokoodien rakenne on erittäin yksinkertainen, mikä tekee niiden analysoinnista ja implementoinnista helppoa [1]. Tulokoodin tärkeimpiä etuja onkin yksinkertainen koodaus ja dekoodaus [16]. Tulokoodien dekoodaus myös parallelisoituu hyvin, mikä tekee niistä sopivia suurnopeuksisiin sovelluksiin [2, 16]. Lisäksi tulokoodi muistuttaa rakenteeltaan koodien ketjuttamista (*concatenated codes*) sekä monikerroksista koodausta (*multilevel coding*), joten tulokoodeille kehitetyt ratkaisut ja sovellukset ovat laajennettavissa näille koodausmenetelmille [1].

Tulokoodien minimietäisyys on pieni verrattuna saman pituisiin optimaalisiin

koodeihin, mutta niiden virheenkorjauspotentiaali on verrattain suuri [1]. Tulokoodeilla virheenkorjauskyky riippuukin erityisesti virhevektorin kuviosta eikä virheiden määrästä [16]. Niinpä käytännön sovellusten kannalta ei liene mielekästä keskittyä koodaamaan tulokoodeja vain puoleen minimietäisyydestä, vaan kannattaa käyttää dekooderia, jolla löydetään vastaanotettua sanaa mahdollisimman hyvin vastaava sana silloinkin, kun virhevektorin paino ylittää rajan $(d-1)/2$ [1]. Tällöin systeemin täytyy kuitenkin sallia bittivirheitä, koska kaikkia rajan $(d-1)/2$ ylittäviä virhevektoreita ei voida korjata.

Tulokoodien rakenne sopii iteratiiviseen dekoodaukseen, jossa pysty- ja vaakarivit dekoodataan vuorotellen lähtökoodien dekoodausalgoritmeja käyttäen. Iteratiivinen SISO-dekoodaus ei muodostu tulokoodeilla liian raskaaksi sovellusten kannalta, ja tulokoodeista tulikin suositumpia, kun niille kehitettiin käyttökelpoinen SISO-dekoodausalgoritmi [18]. Iteratiivisella SISO-dekoodauksella päästään tulokoodeilla lähelle optimaalista dekoodausta [16].

Kvalitatiivisessa testauksessa tulokoodit ovat SISO-dekoodauksella tuottaneet positiivisia tuloksia eräisiin toisiin koodeihin nähden. Esimerkiksi erittäin suurilla koodisuhteilla tulokoodien dekoodaus on kevyempää ja virheenkorjaus parempaa kuin PCCC-koodeilla (*parallel concatenation convolutional codes*). Suurilla koodisuhteilla tulokoodeilla on pienempi dekoodausviive ja parempi virheenkorjaus kuin LDPC-koodeilla (*low density parity check codes*) [16]. Kilpailevia koodeja on kuitenkin laaja kirjo, joten tarvitaan lisätestejä ja enemmän analyysiä, jotta tulokoodien suorituskyvystä suhteessa muihin koodeihin saadaan selvyys [16].

Kirjallisuutta

[1] Al-Askary O. (2003) *Iterative Decoding of Product Codes*. Lisensiaatin tutkielma, Kungliga Tekniska Högskolan, Tukholma.

[2] Argon C., McLaughlin S. W. (2002) *A parallel decoder for low latency decoding of turbo product codes*. 70–42. IEEE Communications Letters, Volume 6, Issue 2.

[3] Coene W., Pozidis H., van Dijk M., Kahlman J., van Woudenberg R., Stek B., (2001) *Channel Coding and Signal Processing for Optical Recording Systems Beyond DVD*. 682–688. IEEE Transactions on Magnetics, Volume 37, No. 2.

[4] Cohen G., Karpovsky M., Mattson H,. Schatz J. (1985) *Covering radius–Survey and recent results*. 328–343. IEEE Transactions on Information Theory, Volume 31, Issue 3.

[5] Elias P. (1954) *Error-free Coding*, 29–37. Transactions of the IRE Professional Group on Information Theory, Volume 4, Issue 4.

[6] Ericson T. (1988) *A simple ananlysis of the blokh-zyablov decoding algorithm*. 43–57. Lecture Notes in Computer Science, No. 307.

[7] Forney G. D. Jr. (1966) *Generalized Minimum Distance Decoding*. 125–131. IEEE Transactions on Information Theory, Volume IT-12.

[8] Golay M. J. E. (1949) *Notes on Digital Coding*. 657. Proceedings of the IRE, Volume 37, Issue 6.

[9] Hamming R. W. (1950) *Error detecting and error correcting codes*. 147–160. The Bell System Technical Journal, Volume 29, Issue 2.

[10] Immink K. A. S. (1996) *The digital versatile disc (DVD): System requirements and channel coding*. 483–489. SMPTE Journal.

[11] Leroux C., Jego C., Adde P., Gupta D., Jezequel M. (2011) *Turbo Product Code Decoder Without Interleaving Resource: From Parallelism Exploration to High Efficiency Architecture*. 17–29. Journal of Signal Processing Systems, Volume 64, Issue 1.

[12] Lin S., Costello D. J. Jr. (2004) *Error Control Coding*, toinen painos. Pearson Education, New Jersey.

[13] van Lint J. H. (1973) *Direct-product codes*. 36–40. Coding Theory. Springer-Verlag, Berliini.

[14] Massey J. L. (1963) *Threshold decoding*. The MIT Press, Cambridge.

[15] MacWilliams F. J., Sloane N. J. A. (1997) *The theory of error correcting codes*. North-Holland Publishing Company, Amsterdam, New York, Oxford.

[16] Mukhtar H., Al-Dweik A., Shami A. (2016) *Turbo Product Codes: Applications, Challenges, and Future Directions*. 3052–3069. IEEE Communications Surveys & Tutorials, Volume 18, No. 4.

[17] Pfister H. D., Emmadi S. K., Narayanan K. (2015) *Symmetric Product Codes*. 282–290. 2015 Information Theory and Applications Workshop (ITA).

[18] Pyndiah R. M. (1998) *Near-Optimum Decoding of ProductCodes: Block Turbo Codes*. 1003–1010. IEEE Transactions on Communications, Volume 46, Issue 8.

[19] Reed I. S. (1954) *A class of multiple-error-correcting codes and the decoding scheme*. 38–49. Transactions of the IRE Professional Group on Information Theory, Volume 4, Issue 4.

[20] Sarwate D. (2016) Viitattu 16.5.2023. https://math.stackexchange.com/questions/2068881/extended-hamming-code

[21] Shannon C. E. (1948) *A mathematical theory of communication*. 379–423. Bell System Technical Journal, Volume 27, Issue 3.

[22] Slepian D. (1960) *Some Further Theory of Group Codes*. 1219–1252. Bell System Technical Journal, Volume 39.

[23] Wainberg S. (1972) *Burst-error and Random-error Correction over q-ary Input, p-ary Output Channels*. Väitöskirja, Polytechnical Institute, Brooklyn, N.Y.

[24] Wainberg S. (1972) *Error-Erasure Decoding of Product Codes*. 821–823. IEEE Transactions on Information Theory.

[25] Weber J. H., Abdel-Ghaffar K. A. S. (2001) *Reduced GMD Decoding of Concatenated Codes*. 901–905. GLOBECOM'01. IEEE Global Telecommunications Conference.

[26] Zhou R., Le Bidan R., Pyndiah R., Goalic A. (2007) *Low-complexity high-rate Reed–Solomon block turbo codes*. 1656–1660. IEEE Transactions on Communications, Volume 55, Issue 9.