Luku 5

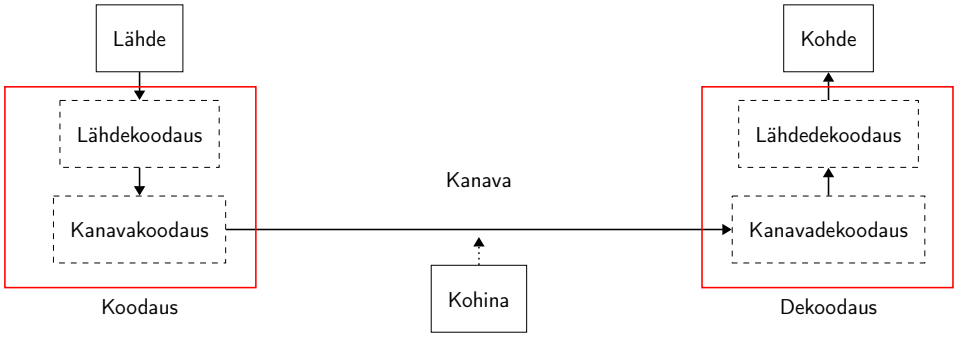
Koodausteoriaa kohinalla

Edellisessä kappaleessa tutkittiin kohihattomalla kanavalla kommunikointia. Tässä kappaleessa lisätään tilanteeseen mahdollisen kohinan vaikutus. Kohinattomassa tilanteessa pyrimme tiivistämään lähetettävät viestit mahdollisimman lyhyiksi nopeuttaaksemme viestintää, eli teimme kompressiota, eli poistimme redundanssia. Kohinaisessa tilanteessa tavoite on edelleen nopea viestintä, mutta haluamme myös varautua mahdollisiin lähetyksen aikana syntyviin virheisiin, ja käytännössä tämä tarkoittaa redundanssia lisäämistä.

Kohinaisessa kommunikaatiossa tilanne siis näyttää siltä, että yhtä aikaa poistamme ja lisäämme redundanssia. Ideana onkin, että ensin poistamme viestistä 'suunnittelemattoman' redundanssin ja tämän jälkeen lisäämme siihen tarkoitushakuista, virheensietokykyä optimoivaa redundanssia.

5.1 Perusosa

Yleisasetelma on sama kuin viime luvussa. Tilanteen yksinkertaistamiseksi rajoitumme tilanteeseen missä kanava on *binäärikanava*, eli siellä kulkee ykkösiä ja nollia. Edelleen jaamme suorittamamme koodauksen kahteen osaan: kompressointiin eli *lähdekoodaukseen* sekä virheensietokyvyn lisäämiseen eli *kanavakoodaukseen*. Viime osion havainnakuva muokkautuu siis seuraavan näköiseksi.



Havainnekuva kohinaisesta kanavasta.

Oletamme edelleen, että lähdekoodaus on tehty optimaalisesti, eli että lähteestä tulevaa dataa ei voi enää edelleen kompressoida, ja että lähdekoodauksen output on binäärisymboleita. Erityisesti lähdekoodaus on meille black box, ja mallinnamme sitä TN-jakaumalla $((0,1),(p_0,p_1))$. Huomataan heti, että jos tämä jakauma ei ole tasajakauma, on sen entropia alle 1, jolloin viime luvun tekniikoilla voisimme edelleen kompressoida viestejä paremmin. Täten voimme siis optimaalisuusoletuksen nojalla olettaa, että $p_0 = 0.5 = p_1$.

Kertauksen vuoksi, oletamme tässä kappaleessa siis että:

- Kanavamme on binäärikanava.
- Koodaus on jaettu lähde- ja kanavakoodauksiin.
- Lähdekoodaus on optimaalinen binäärikoodi, ja mallinnamme sitä tasajakaumalla.

Emme puutu tällä kurssilla siihen, että miten rajoittavia nämä oletukset ovat. Huomautamme tosin, että todistukset näyttävät hyvin samoilta vaikka kanava olisi ei-binäärinen.

5.1.1 Kohinainen kanava

Määritellään seuraavaksi kanava. Kanava on siis jokin kommunikaatioväylä, kuten puhelinlinja, parikaapeli, radioyhteys tai vastaava, johon voi jollain todennäköisyydellä tulla virheitä. Määrittelemme kanavan parina TN-

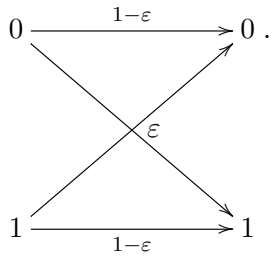
jakaumia, jotka kuvastavat miten todennäköisesti vastaanottopäässä nähdään 0 tai 1 kun on lähetetty 0 tai 1.

Määritelmä 5.1.1. *Kanava* on pari TN-jakaumia (Y_0, Y_1), missä

$$Y_0 = ((0,1),(p_0,p_1)) \quad \text{ja} \quad Y_1 = ((0,1),(q_0,q_1)).$$

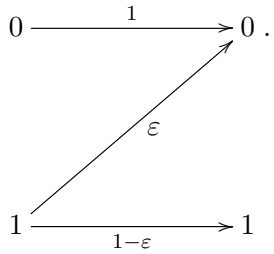
Esimerkki 25. Tärkein esimerkki meille on *symmetrinen ε-kohinainen kanava* $(Y_0^\varepsilon, Y_1^\varepsilon)$, missä $p_0 = 1-\varepsilon$, $p_1 = \varepsilon$, $q_0 = \varepsilon$ ja $q_0 = 1-\varepsilon$.

Sen sijaan että kirjoittaisimme kaikki parin todennäköisyydet aina näkyviin, esitämme kanavia usein seuraavassa muodossa:



Tässä kanavassa siis kummallakin syötteellä on sama todennäköisyys välittyä oikein vastaanottajalle.

Esimerkki 26. Toinen tärkeä esimerkki on n.k. ε-kohinainen Z-kanava:



Tässä kanavassa viesti 0 välittyy siis aina oikein, mutta viesti 1 voi välittyä väärin.

Tulemme 'syöttämään' kanavalle kanavakoodauksestamme syntyviä bittijonoja. Kanavan 'näkökulmasta' kanavakoodauksemme output on TN-jakauma. Kanava ja sille syötteenä toimiva TN-jakauma muodostavat yhteisjakauman seuraavasti.

Määritelmä 5.1.2. Olkoon (Y_0, Y_1) kanava ja $X := ((0,1),(t_0,t_1))$ TN-jakauma. Näiden jakaumista muodostetaan yhteisjakauma (X,Y) asettamalla

$$P(X=0,Y=0) = t_0p_0, \quad P(X=0,Y=1) = t_0p_1,$$
$$P(X=1,Y=0) = t_1q_0, \quad P(X=1,Y=1) = t_1q_1.$$

Tätä yhteisjakaumaa kutsutaan *kanavajakaumaksi*.

Huomaa, että kanavajakaumassa $P(X|Y = 0) = Y_0$, $P(X|Y = 1) = Y_1$. Merkitsemme välillä röyhkeästi sekä kanavaa (Y_0, Y_1) että sen kanssa muodostetun kanavajakauman marginaalitodennäköisyyttä samalla merkillä Y.

Kohinaisella kanavalla kiinnostavaa on se, miten hyvin voimme päätellä lähetetyn bitin kun tiedämme vastaanotetun bitin. Erityisesti meitä siis kiinnostaa miten vahvasti kanavajakauman marginaalijakaumat ovat kytköksissä toisiinsa. Tätä varten meillä on jo olemassa loistava mittari, nimittäin yhteisinformaatio. Muistetaan, että yhteisinformaatio voidaan laskea kaavalla

$$I(X;Y) = H(X) - H(X|Y).$$

Tässä kohtaa kurssia on hyvä pitää mielessä tulkinta, jossa $H(X)$ kuvaa ylipäänsä epätietoisuuttamme lähetetystä bitistä, ja ehdollinen entropia $H(X|Y)$ sitä, että kuinka epätietoisia olemme keskimäärin bitistä lähetetystä bitistä kun olemme nähneet saapuneen bitin. Voitaisiin siis kirjoittaa

Yhteisinformaatio = Keskim. epävarmuus alussa − Keskim. epävarmuus lopussa.

Yhteisinformaatio siis luontevasti kuvaa paljon voimme yleensä toivoa saavamme tietoa viestin vastaanotettuamme. Tämän motivoimana voidaan määritellä kanavan kapasiteetti.

Määritelmä 5.1.3. Kanavan Y *kapasiteetti* on

$$\mathrm{Cap}(Y) = \max_X I(X;Y),$$

missä maksimi otetaan kaikkien kanavan Y kanssa muodostettujen kanavajakaumien yli.

Shannonin kanavakoodauslause, josta myöhemmin puhumme, kertoo että kanavan kapasiteetti on nimensä veroinen, eli kapasiteetti antaa jossain mielessä tarkan rajan sille, miten nopeasti kanavalla voi edes teoreettisesti viestiä.

Demoissa lasketaan symmetrisen ε-kohinaisen binäärikanavan kapasiteetti.

5.1.2 Kanavakoodaukset

Viime kappaleessa määrittelimme TN-jakaumaan (Ω_X, P) liittyvän koodauksen funktioksi $c\colon \Omega_X \to \mathcal{M}_{0,1}$. Tämän jälkeen suunnittelimme blokkikoodeja muokkaamalla alkuperäisestä jakaumasta X jakauman X^N, ja määrittelemällä tälle jonkin koodauksen. Olisimme voineet vaihtoehtoisesti määritellä, että koodauksen määrittelyjoukko saa olla isompi kuin $\Omega_X = \mathcal{M}^1_{\Omega_X}$, erityisesti koodauksen voisi määritellä funktioksi $c\colon \mathcal{M}^{N*}_{\{0,1\}} \to \mathcal{M}_{\{0,1\}}$, jollain

$N \geq 1$, missä $\mathcal{M}_{\{0,1\}}^{N*}$ on kaikkien tasan N merkkiä pitkien binäärisanojen joukko. Teemme näin jatkossa.

Erityisesti jos koodaus on muotoa $c\colon \mathcal{M}_{\{0,1\}}^{K*} \to \mathcal{M}_{\{0,1\}}^{N}$ jollain $N, K \geq 1$, niin kutsumme tällaista koodausta (N,K)-*(blokki)koodaukseksi.* (Keskitymme pelkästään blokkikoodeihin, eli erityiesti emme mieti vaihtelevan pituisia koodeja kanavakoodauksessa. Yksi syy on se, että etuliitevapaan koodit ovat vielä ihan eri tavalla alttiita häiriöille, ja koodisanojen rajojen häilyminen voisi aiheuttaa isoja virheitä.) Blokkikoodauksen *lähetysnopeus* on luku K/N, eli esimerkiksi jos kanava pystyy välittämään yhden merkin sekunnissa, niin käytettäessä blokkikoodia jonka nopeus on R voidaan viestiä keskimäärin R kanavakoodauksen lähteen merkkiä sekunnissa.

Koodaukseen liittyy aina myös koodauksen purku. Kohinattomassa tilanteessa koodin purku oli suoraviivaista, mutta kohinaisessa tilanteessa meidän pitää varautua virheisiin. Mikäli c on (N,K)-koodaus niin sen purku (eli dekoodaus) on funktio $d\colon \mathcal{M}_{\{0,1\}}^{N*} \to \mathcal{M}_{\{0,1\}}^{K}$ jolle pätee $d(c(x)) = x$. Huomaa, että käytännössä aina $N > K$, joten koodin purku antaa saman tuloksen useammalle eri syötteelle. Vertaa halutessasi tilannetta lineaarikuvauksiin $\mathbb{R}^K \to \mathbb{R}^N$ yms.

Seuraava määritelmä on hieman tekninen, mutta sen taustalla on yksinkertaisia ajatuksia. Erityisesti, **symmetrisen kohinaisen kanavan tilanteessa esittelemämme eri virhekäsitteet ovat kaikki samoja, ja erityisesti "todennäköisyys virheelle viestissä" on juuri se, mitä sen arvaisitkin olevan.** Jos se ahdistaa, katso joko myöhempiä esimerkkejä jossa virheitä lasketaan, tai perehdy MacKayn kirjan lukuun 9.

Määritelmä 5.1.4. Olkoon Y kanava ja c (N,K)-koodaus sekä d sen jokin purku. Jos T on kanavalle lähtevä merkkijono, niin merkitsemme $\Xi(T)$ merkkijonoa, joka otetaan kanavan toisessa päässä vastaan – huomaa että c ja d ovat deterministisiä funktioita, mutta kanavalla on stokastinen luonne, joten Ξ on nyt välttämättä satunnaismuuttuja. Mutta minkä suhteen?

Koodauksemme c on itseasiassa satunnaismuuttuja joka on määritelty TN-jakaumassa $((0,1),(0.5,0.5))^K$ (muista, että oletimme että kanavakoodauksen input on luonteeltaan kahden alkion tasajakauma optimaalisen kompression johdosta). Täten kanavakoodauksen output muodostaa uuden TN-jakauman kun sen ajatellaan tuottavan bittejä yksi kerrallaan. Tämä TN-jakauma puolestaan muodostaa kanavan Y kanssa kanavajakauman, jonka avulla voimme viimein laskea todennäköisyyden sille, että annettu viesti välittyy kanavalla oikein. Emme kirjoita näitä kuitenkaan formaalisti auki, ainakaan perusosassa, vaan merkitsemme merkillä Ξ satunnaismuuttujaa joka kuvaa sitä mitä kanava tekee syötteilleen (eli jos kanavaan työntää merkkijonon T niin ulos tulee $\Xi(T)$ ja merkitsemme $P(d(\Xi(c(S))) \neq S)$ todennäköisyyttä sille, että annetuilla (deterministisillä) koodauksella ja purulla saamme takaisin väärän viestin kun olemme koodanneet, lähettäneet kanavalle ja dekoodanneet.

Tästä edelleen määritellään:

- *Keskimääräinen blokkivirhe* p_B on todennäköisyys sille että lähetetty merkkijono purkautuu väärin.

$$p_B = \sum_{S \in \mathcal{M}_{0,1}^K} P(S) \cdot P(d(\Xi(c(S))) \neq S)$$

- *Suurin mahdollinen blokkivirhe* p_{BM}

$$\max_{S \in \mathcal{M}_{0,1}^K} P(d(\Xi(c(S))) \neq S)$$

- *Bittivirhe* p_b on keskimääräinen todennäköisyys sille, että alkuperäisen viestin yksittäinen bitti välittyy määränpäähän väärin.

- Purku on *optimaalinen* mikäli se minimoi keskimääräisen blokkivirheen koodauksen kaikkien purkujen joukossa. (Huomaa, että tämä voi riippua kanavasta.)

Symmetrisellä kohinaisella kanavalla virheen todennäköisyys ei riipu syötteestä, joten kaikissa tutkimissamme esimerkeissä kaikilla syötteillä on sama virhetodennäköisyys ja oletustemme nojalla kaikilla syötteillä sama todennäköisyys, jolloin $p_B = p_{BM} = p_b$.

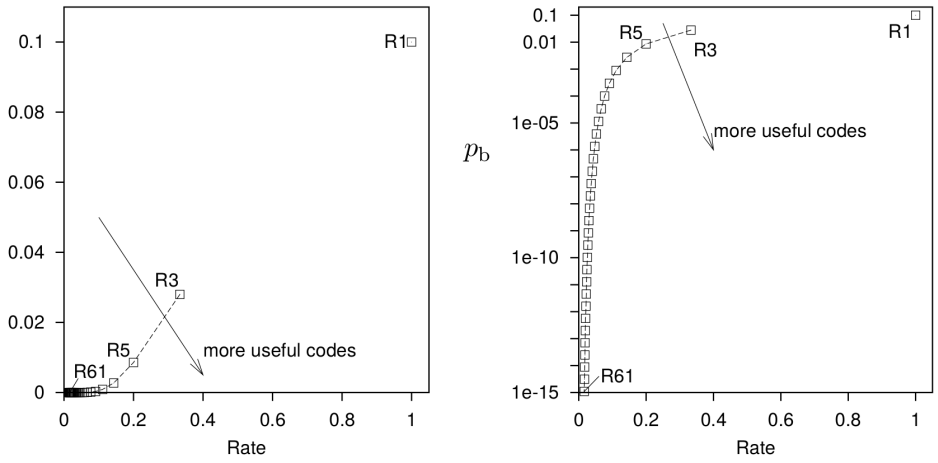
Otetaan nyt muutama esimerkki koodauksista.

Esimerkki 27. Koodi $R_3 \colon \mathcal{M}_{\{0,1\}}^{1*} \to \mathcal{M}_{\{0,1\}}^{3*}$ määritellään asettamalla $0 \mapsto 000, 1 \mapsto 111$. Tämä koodi siis triplaa kaikki merkit. Koodi puretaan vastaanottopäässä katsomalla kutakin kolmen merkin blokkia, ja katsomalla kumpaa merkkiä on enemmän, eli koodin purku vie sanat $111, 101, 110$ ja 011 merkille 1 ja loput sanat merkille 0. Tämä on $(3, 1)$-koodaus.

Oletetaan, että lähetämme tällä koodauksella viestin 1010 symmetrisellä $\varepsilon = 0.1$-kohinaisella kanavalla. Mikä on todennäköisyys, että viesti tulee virheettä perille? Esimerkin tilanteessa kukin merkki lähetetään kolminkertaisena, ja purku onnistuu mikäli enintään yksi merkki vaihtuu, joten todennäköisyys että saamme virheen on

$$P(\text{kaksi virhettä}) + P(\text{kolme virhettä}) = 3 0.1^2 0.9^1 + 0.1^3 \approx 0.03.$$

Verrattuna identtiseen koodaukseen missä kukin symboli vaan lähetetään itselleen (eli $(1, 1)$-koodaus) ja virheen todennäköisyys on $\varepsilon = 0.1$, saamme nyt pienennettyä virheen todennäköisyyden alle kolmannekseen alkuperäisestä. Huonona puolena on se, että joudumme lähettämään kolme kertaa enemmän merkkejä kuin ennen, eli *lähetysnopeutemme tippuu kolmannekseen alkuperäisestä.*



Kuva 5.1: Muutamien koodien virheensietokyvystä ja lähetysnopeudesta. Kummassakin x-akselina on lähetysnopeus ja y-akselina blokin keskimääräinen virhe, vasemmassa lineaarisella asteikolla ja oikeassa logaritmisella. Kuva lähteestä [Mac03, s. 8].

Vastaavasti voitaisiin määritellä $(5,1)$-koodi R_5, jossa $0 \mapsto 00000, 1 \mapsto 11111$ jolloin blokin keskimääräinen virhe on todennäköisyys sille että sanomaan osuu vähintään kolme virhettä;

$$\binom{5}{3}0.1^3 0.9^2 + \binom{5}{4}0.1^3 0.9^2 + 0.1^5 \approx 0.0087.$$

Virheen todennäköisyys siis laskee entisestään, mutta myös lähetysnopeus pienenee huomattavasti.

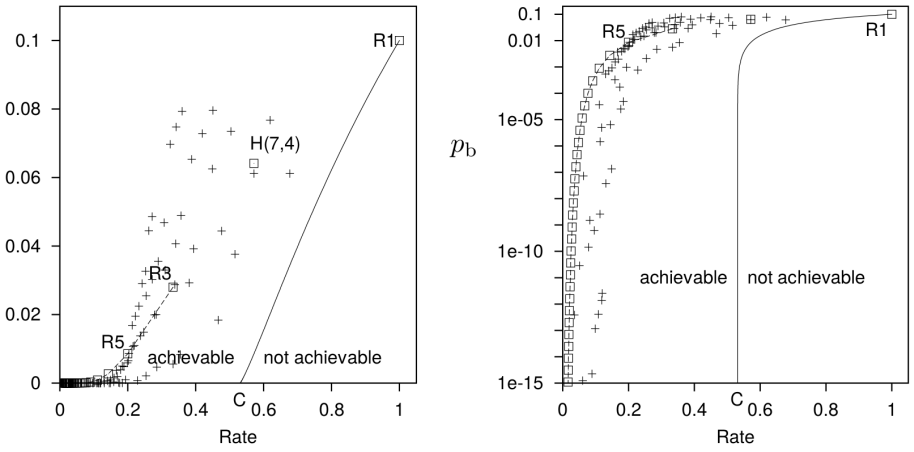
Edelleen voitaisiin määritellä $(k,1)$-koodit R_k, missä k on mikä tahansa pariton luonnollinen luku. Oheisessa MacKayn kirjasta napatussa kuvassa (Kuva 5.1) on plotattu näiden koodien antamaa yksittäisen lähetetyn merkin virhettä suhteessa niiden lähetysnopeuteen.

Yllä käytetyt R_k-koodit ovat luontevan oloisia, joten tässä kohtaa voisi arvata, että ne kuvaavat yleistä tilannetta: parempi virheensietokyky vaatii aina vaan suurempaa lähetysnopeuden uhraamista. Näin ei kuitenkaan ole! Shannonin kanavakoodauslause sanoo seuraavaa.

Lause 5.1.5. *Olkoon Y kanava ja C sen kapasiteetti. Tällöin Kaikilla $\varepsilon > 0$, $R < C$ ja tarpeeksi isoilla*[1] *N on olemassa (N,K)-blokkikoodi jonka lähetysnopeus on vähintään R ja blokin suurin mahdollinen virhe alle ε.*

Todistus. Torstaina. □

[1] 'Tarpeeksi iso' riippu kanavasta sekä parametreista ε ja R.



Kuva 5.2: Muutamien koodien virheensietokyvystä ja lähetysnopeudesta. Kummassakin x-akselina on lähetysnopeus ja y-akselina blokin keskimääräinen virhe, vasemmassa lineaarisella asteikolla ja oikeassa logaritmisella. Kuvassa näkyy myös Shannonin kanavakoodauslauseen antama raja. Kuva lähteestä [Mac03, s. 15].

Tämä lause siis sanoo, että meidän ei tarvitse uhrata viestintänopeutta alle tietyn kanavasta riippuvan vakion saadaksemme virhetodennäköisyydet todella pieniksi. Pienenä uhrauksena joudumme tosin käyttämään aina vaan pidempiä ja pidempiä viestiblokkeja, joka voi tuottaa ongelmia mikäli meillä olisi tarve viestiä silloin tällöin lyhyitä viestejä eikä jatkuvaa datavirtaa. Tähän liittyvän kommentaarin jätämme koodausteorian ammattilaisille. Shannonin lausetta on havainnoitu kuvassa 5.2.

Shannonin lauseen todistus jätetään syventävään osioon, mutta tutustutaan seuraavaksi Hammingin koodiin jota tutkimalla huomaamme, että miksi R_k-koodit eivät pääse lähellekään optimaalista tilannetta.

5.1.3 Hammingin koodi

Miksi R_k-koodit eivät ole optimaalisia? Tarkastellaan koodia tutkimalla merkkijonon $S = 1010$ lähetystä. R_3 triplaa kunkin yksittäisen merkin, eli kanavalle lähtee merkkijono $T = 111000111000$. Huomataan, että koodi osaa korjata minkä tahansa yksittäisen virheen, eli mikäli $T*$ on merkkijono joka saadaan jonosta T yhtä merkkiä muuttamalla, niin $d_3(T*) = S$, missä d on koodauksen R_3 purku.

Huomataan, että todennäköisyys saada tasan yksi virhe viestin T lähetyksessä on $12 * 0.1 * 0.9^{11} \approx 0.38$. Koodaus osaa kuitenkin korjata myös muita virheitä, esimerkiksi virheen jossa lähetyksessä syntyy neljä virhettä,

mutta kukin sijoittuu eri 'triplablokkiin' lähetetyssä viestissä:

$$S = \texttt{1010} \xrightarrow{R_3} T = \texttt{111000111000}$$
$$\texttt{--}\star\star\texttt{---}\star\texttt{---}\star$$
$$S = \texttt{1010} \xleftarrow{d} T' = \texttt{110100101001}$$

Todennäköisyys saada lähetyksen aikana tällainen virhe on $0.1^4 \cdot 0.9^8 \cdot 1 \cdot \frac{3}{4} \cdot \frac{2}{4} \cdot \frac{1}{4} \approx 0.000004$. Tämä on huiman pieni, kun vertaa sitä todennäköisyydeen saada kaksi virhettä samassa blokissa, eli $0.1^2 \cdot 0.9^{10} \cdot 1 \cdot \frac{2}{11} \approx 0.0006$.

Toisin sanoen, koodaus R_3 osaa korjata erilaisia virheitä, mutta ongelma on siinä että sitä ei ole rakennettu korjaamaan parhaiten todennäköisimpiä virheitä. Tilanne on hieman samanlainen kuin Huffmanin koodista keskustellessa; lähdekoodauksessa kannattaa antaa lyhyitä koodeja kaikista todennäköisimmille merkeillä. Vastaavasti kanavakoodauksessa kannattaa luoda koodeja jotka korjaavat parhaiten todennäköisimpiä virheitä.

Lähdekoodaus:	**Kanavakoodaus:**
Kaikkea ei voi kompressoida.	Virhettä ei saa ikinä nollaksi.
Keskimääräinen parannus riittää.	Keskimääräinen parannus riittää.
Annetaan siis yleisemmille asioille parempi kompressio.	*Annetaan siis yleisemmille virheille parempi korjaus.*

Hammingin koodin ideana onkin juuri tämä; se osaa korjata tosi hyvin yksittäisiä virheitä eikä käytä "energiaa" harvinaisempien virheiden korjaamiseen.

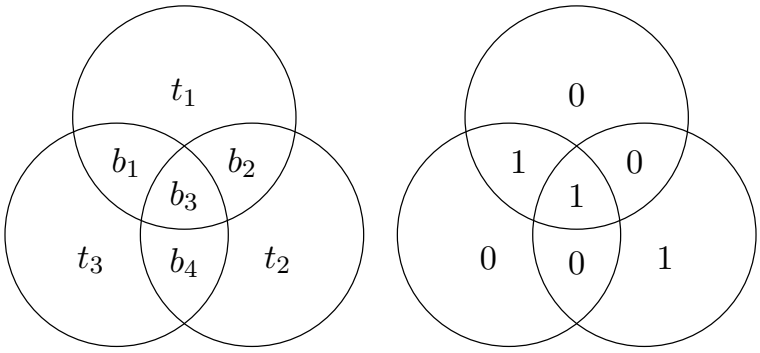
Hammingin koodaus

Hammingin $(7,4)$-koodi on $(7,4)$-koodaus, joka määritellään seuraavasti. Olkoon $S = b_1b_2b_3b_4 \in \mathcal{M}_{0,1}^{4*}$ merkkijono, joka halutaan kooodata. Haluamme määritellä seuraavaksi nk. *parillisuusbitit* $t_1t_2t_3$; lopullinen koodi tulee olemaan $b_1b_2b_3b_4t_1t_2t_3$. Piirretään kolme ympyrää kuten kuvassa 5.3, ja **valitaan parillisuusbitit siten, että kussakin ympyrässä on parillinen määrä ykkösiä.**

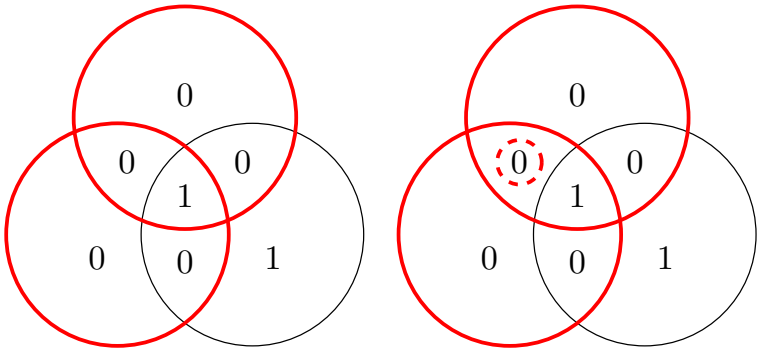
Esimerkillä 1010 lähetetään siis viesti 1010010.

Hammingin purku

Hammingin koodia purettaessa, jos matkalla ei ole tapahtunut yhtään virhettä, löytyy alkuperäinen viesti ensimmäisestä neljästä merkistä. Viestiä vastaanotettaessa pitää kuitenkin tarkistaa, onko virheitä mahdollisesti tullut. Hammingin koodissa tämä tehdään katsomalla, että onko kaikissa ympyröissä edelleen parillinen määrä ykkösiä. Jos on, niin asetetaan $d(b_1b_2b_3b_4t_1t_2t_3) = b_1b_2b_3b_4$.



Kuva 5.3: Hammingin koodin määrittely



Kuva 5.4: Hammingin koodin purkua virheellisellä viestillä $T* = 0010010$.

Mikäli ei ole, niin katsotaan missä ympyröissä on pariton määrä ykkösiä. Tällöin on olemassa yksikäsitteinen bitti jota muuttamalla kaikkien ympyröiden ykkösten määrä palaa parilliseksi. Katsotaan tilannetta kahden esimerkin kautta. Tutkitaan tilanteita, joissa aiemmin koodattuun viestiin 1010010 on tullut virhe joko ensimmäiseen bittiin $S* = 0010010$ tai viimeiseen bittiin $T* = 1010011$. Purku näkyy kuvissa 5.4 ja 5.5.

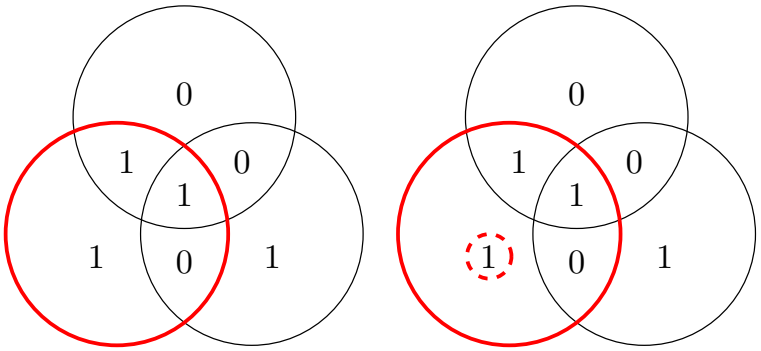
Demoissa tutkitaan mitä käy useamman virheen tilanteessa.

Hammingin koodi matriiseilla

Mikäli varustetaan joukko $\{0, 1\}$ laskutoimituksella

$$0+0=0 \quad 0+1=1$$
$$1+0=1 \quad 1+1=0$$

(eli XOR tai mod 2 -aritmetiikka, as you wish) niin huomataan että bittien $\{a, b, c, d\}$ joukossa on parillinen määrä ykkösiä jos ja vain jos $a+b+c+d = 0$. Erityisesti jos merkataan $t = a+b+c+d$, niin joukossa $\{a, b, c, d, t\}$ on aina parillinen määrä ykkösiä.



Kuva 5.5: Hammingin koodin purkua virheellisellä viestillä $T* = 1010011$.

Täten Hammingin koodi voidaan määritellä asettamalla $t_1 = b_1 + b_2 + b_3$ jne. Käyttämällä lineaarialgebrasta tuttua notaatiota, voidaan Hammingin koodaus siis ilmaista matriisin muodossa:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \mapsto \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_1 + b_2 + b_3 \\ b_2 + b_3 + b_4 \\ b_1 + b_3 + b_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ t_1 \\ t_2 \\ t_3 \end{bmatrix},$$

jossa yhteenlaskut siis tulkitaan aiemmin mainitussa muodossa.

Kirjallisuutta

[Jay03] E. T. Jaynes. *Probability theory*. Cambridge University Press, Cambridge, 2003. The logic of science, Edited and with a foreword by G. Larry Bretthorst.

[Mac03] David J. C. MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, New York, 2003.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Tech. J.*, 27:379–423, 623–656, 1948.

[Siv06] D. S. Sivia. *Data analysis*. Oxford Science Publications. Oxford University Press, Oxford, second edition, 2006. A Bayesian tutorial, With J. Skilling.