OSA D
MUITA MENETELMIÄ

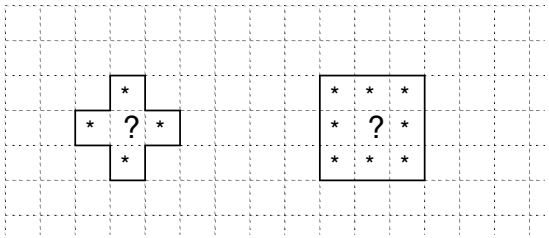
1. SOLUAUTOMAATTISIMULOINTI

1.1. Johdanto

J. von Neumann, joka tutki tietokoneiden teoriaa ja tunnetaan perinteisen *peräkkäislaskenta-arkkitehtuurin,* ns. von Neumann-arkkitehtuurin isänä, huomasi nopeuden muodostuvan peräkkäislaskentaan perustuvan tietojenkäsittelyn pullonkaulaksi laskentatehoa lisättäessä. *Rinnakkaislaskennan* mahdollisuuksia tutkiessaan soluautomaattien idean keksivät jo vuonna 1948 juuri J. von Neumann ja S. Ulam, vaikka tietokoneiden ensimmäiset sukupolvet ovatkin perustuneet nimenomaan nopeaan peräkkäislaskentaan.

J. von Neumann tutki myös mm. biologista lisääntymistä ja pyrki sitä mallintaessaan luomaan "itseään monistavan koneen". Vuosina 1952 – 53 von Neumann saikin suunnitelluksi soluautomaatin, joka pystyi itsensä kopioimiseen ja lisäksi vielä ns. *universaalilaskentaan* eli periaatteessa ratkaisemaan minkä tahansa laskettavissa olevan tehtävän.

Soluautomaatit (cellular automata, yks. automaton) ovat diskreettejä dynaamisia systeemejä, jotka koostuvat joukosta säännölliseksi hilaksi järjestettyjä soluja. Kaikki solut ovat samanlaisia ja ne ovat yhteydessä toisiinsa naapurisuhteidensa perusteella. Solua ja sen naapureita sanotaan *naapurustoksi.*



Kunkin solun mahdolliset tilat ovat *diskreettejä* ja tiloja on tyypillisesti vain muutama. Solun seuraava tila määräytyy naapuruston tilojen perusteella ns. *soluautomaattisäännön* mukaan. Tavallisesti kaikki solut vaihtavat tilaansa samanaikaisesti.

Formaalisti soluautomaatit ovat diskreettejä dynaamisia systeemejä, jotka koostuvat komponenteista (G, Q, V, f, t), missä

- G on *soluavaruus,* yleensä säännöllinen hila, esim. Z tai Z^2.
- Q on diskreetti *tila-avaruus,* esim $\{0, 1\}$.
- V on *naapurusto,* joka on määritelty samoin kaikille soluille.
- f on *soluautomaattisääntö,* jonka avulla lasketaan solun tila ajanhetkellä $t+1$, kun naapuruston tilat ajanhetkellä t tunnetaan.
- t on diskreetti "aika".

Soluautomaattisääntöä sanotaan *totalistiseksi* (totalistic rule), jos se perustuu vain naapuruston solujen, poislukien solu itse, tilojen sopivalla tavalla määriteltyyn "summaan"; *ulkototalistiseksi* (outer totalistic rule), jos edellisten lisäksi myös solun itsensä tilaan; ja muuten *yleiseksi säännöksi* (general rule).
S. Wolfram, joka tutki erityisesti yksiulotteisten soluatuomaattien teoriaa, otti käyttöön tavan koodata soluautomaattisäännöt yhdeksi luvuksi, ks. esim. "Life-peli" kappaleessa D.1.3.

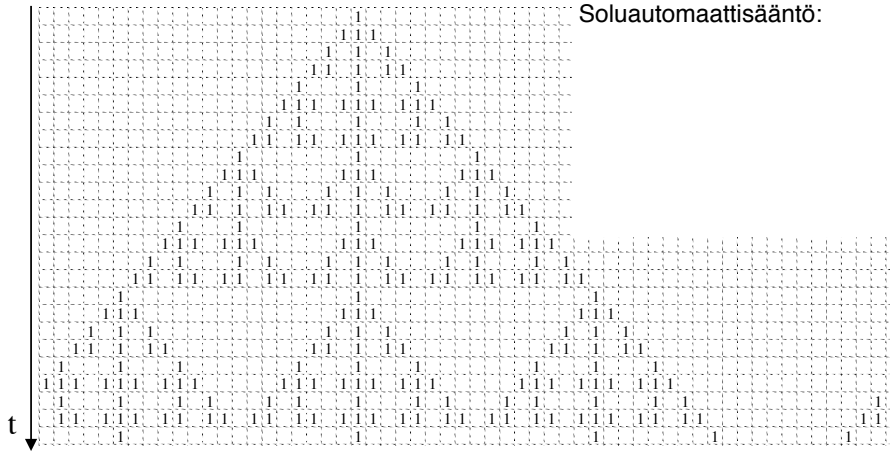
Sovellutuksia:

- hilakaasut
- kiteen kasvu ja kasvuvirheet
- pintareaktiot ja diffuusio
- itsekopiointi
- galaksien evoluutio
- kaaos
- fraktaalit
- hydro-dynamiikka
- "self-organized criticality"
- TIETOKONEET (rinnakkais-laskenta)
- hahmontunnistus
- kasvainten kehittyminen
- immunologia

1.2. Yksiulotteiset soluautomaatit

Yksiulotteinen soluautomaatti on jono soluja. Jos jono on äärellinen se voidaan sulkea myös renkaaksi (eli soveltaa periodisia reunaehtoja). Yksiulotteisia soluautomaatteja on neljä tyyppiä:

1) Homogeeniset soluautomaatit: Evoluutio tuottaa homogeenisen konfiguraation. Täten alkutilan kuviot katoavat ajan kuluessa.
2) Jaksolliset soluautomaatit: Evoluutio tuottaa stabiileja tai periodisia rakenteita. Alkutilan kuviot kasvavat vain äärellisen kokoisiksi.
3) Kaoottiset soluautomaatit: Evoluutio johtaa kaoottiseen tai fraktaaliseen tilaan. Alkutilan kuviot kasvavat rajatta vakionopeudella.
4) Laskennalliset soluautomaatit: Evoluutio tuottaa monimutkaisia paikallisia rakenteita, jotka kykenevät laskennallisiin operaatioihin ja jopa universaalilaskentaan. Alkutilan kuviot voivat kasvaa tai supistua ajan kuluessa.



Kuva 1.2.1. Yksiulotteinen soluautomaatti, jonka koodinumero on 10 (tai 1010_2). Toisena dimensiona kuvassa on aika. Etsi soluautomaattisääntö!

1.3. Kaksiulotteiset soluautomaatit

Kaksiulotteisia soluavaruuksia voi olla useita eri tyyppejä, koottuna kolmion, neliön tai kuusikulmion muotoisista soluista. Tarkastellaan seuraavassa ensin soluautomaattisäännön koodaamista luvuksi ja sitten erästä tunnettua kaksidimensioista soluautomaattia, ns. Life-peliä, esimerkkinä.

Ulkototalistisen kaksidimensioisen soluautomaatin koodinumero on

$$C = \sum_{a=0,k-1} \sum_{n} k^{kn+a} f(a,n) \,, \qquad (1.3.1)$$

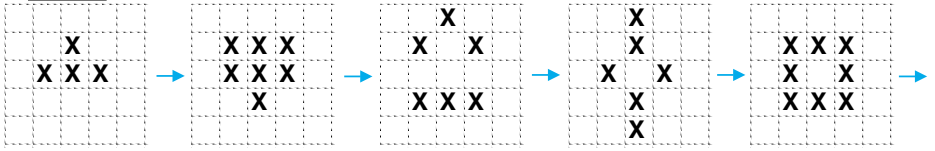
missä k on solun tilojen lukumäärä, a on solun tila, n on naapuruston solujen tilojen summa ja $f(a,n)$ on solun uusi tila.

Life-peli (Game of Life) (M. Gardner, Scientific American **223**, 120 (1970)) on kaksiulotteinen suorakulmaiseen hilaan perustuva ulkototalistinen soluautomaatti. Sen soluilla on kaksi tilaa: elossa (1) tai kuollut (0). Soluautomaattisäännössä sovelletaan Mooren naapurustoa seuraavasti:

GAME OF LIFE:
(M. Gardner 1971)

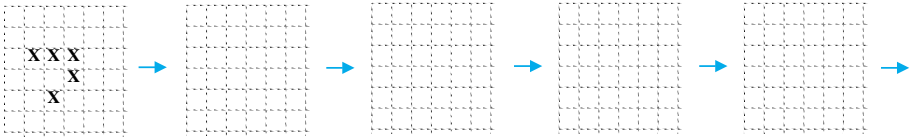
Otus säilyy, jos 2 – 3 naapuria
kuolee, jos 0, 1, 4 – 8 naapuria
syntyy, jos 3 naapuria

Esim.



Siis, populaation otuksen syntymiseen tarvitaan kolme otusta edellisessa sukupolvessa ja otus voi kuolla joko liikakansoitukseen tai yksinäisyyteen.

Esim. Tutki miksi seuraavaa Life-pelin konfiguraatiota sanotaa liitokoneeksi (glider) ?



Life-peli kuvaa jossakin määrin biologisten populaatioiden kasvuilmiöitä, joten "elämän peli" nimitys on varsin onnistunut. Lisäksi Life-peli kykenee universaalilaskentaan eli simuloimaan mitä tahansa muuta tietokonetta. Tämä perustuu siihen, että valitsemalla sopivat alkukonfiguraatiot Life-pelillä voidaan mallintaa kaikkia tietokoneen rakentamiseen tarvittavia piirejä ja niiden kytkentöjä.

Esim. Osoita, että Life-pelin (ulkototalistinen) koodinumero on 224 (kymmenjärjestelmässä).

Hilakaasumallit (lattice gas) ovat soluautomaatteja, joissa tila-avaruuden arvot kuvaavat solussa sijaitsevia hiukkasia. Soluautomaattisäännön tulee tällöin toteuttaa joukko säilymislakeja, esim. hiukkasluvun, energian ja liikemäärän säilyminen. Samalla tavoin voidaan kuvata myös jatkuvan massajakautuman esim. nesteen dynamiikkaa. Kaksiulotteisen hydrodynamiikan kuvaamisessa ongelmana on se, että neliöhila ei ole fysikaalisesti isotrooppinen. Käytettäessä kolmiohilaa tältä ongelmalta vältytään.

Kolmessa ulottuvuudessa tämä sama ongelma on vielä vakavampi, mutta siellä se voidaan kiertää "neljännen ulottuvuuden kautta".

1.4. Soluautomaattien sovellutuksia

Soluautomaatit tietojenkäsittelysysteemeinä. Soluautomaattien evoluution voi tulkita tietojenkäsittelyksi, jossa prosessoidaan alkutilan kofiguraatioon sisältyvää informaatiota. Jos kyseessä on *kääntyvä soluautomaatti* (eli sellainen, jolle on olemassa samat konfiguraatiot käänteisessä järjestyksessä tuottava soluautomaatti), ei informaation määrä käsittelyssä muutu. Useat soluautomaatit ovat kuitenkin ei-kääntyviä, ns. *itseorganisoituvia* (self-organizing). Tällöin satunnaisesta alkutilasta lähdettäessä voi syntyä monimutkaisia järjestyneitä rakenteita.

Massiivisesti rinnakkaistetun superlaskennan teho perustuu useisiin (jopa tuhansiin) prosessoreihin ja niiden välisiin kytkentöihin. Thinking Machines Corp. -yhtiön valmistama Connection Machine esimerkiksi perustuu yhden bitin prosessoreihin (CM-2:ssa 65536 kpl), jotka on fyysisesti kytketty soluautomaatiksi. Tällaisten koneiden suurin ongelma on tavallisesti niiden ohjelmoiminen ja korkean tason ohjelmointikielten kääntäjien tekeminen. Connection Machinelle on saavissa rinnakkaistava FORTRAN-kääntäjä.

	Yksi käskyvuo	Monta käskyvuota
Yksi datavuo	SISD perinteinen arkkitehtuuri	MISD useita käskyjä operoi samaa dataa yhtä aikaa
Monta datavuota	SIMD vektoriprosessorit prosessorihilat	MIMD "useita eri tietokoneita"

Taulukko 1.4.1. Tietokoneiden luokittelu Flynnin taksonomiassa.

Kriittiset ilmiöt. Luonnosta löytyy ilmiöitä, jotka ovat itsesimilaarisia aika- tai paikkaskaalauksen suhteen. Tällaisia ovat esim. vuoristojonot, jokiverkostot ja rantaviivat paikallisesti skaalautuvina, sekä esim. maanjäristykset ja pörssikurssit ajallisesti skaalautuvina. Skaalautuvuus on usein ns. $1/f$ -tyyppiä, jolloin signaalin amplitudin A ja frekvenssin f jakautumat noudattavat lakia

$$A(f) \propto 1/f^{\alpha}, \qquad (1.4.1)$$

missä $\alpha \approx 1$. Tämä tarkoittaa esim. sitä, että pieniä maanjäristyksiä on usein ja suuria harvoin.

Fraktaalien teoria antaa välineet itsesimilaaristen ilmiöiden analysointiin ja ns. kriittiset ilmiöt, esim. fysikaalisen systeemin käyttäytyminen faasimuunnoksen läheisyydessä skaalautuu em. eksponenttilain mukaisesti.

Esim. ferromagneettinen magnetoituma $M(T)$ skaalautuu yhtälön

$$M(T) \propto (T_c - T)^{\beta} \qquad (1.4.2)$$

mukaisesti kriitisen lämpötilan T_c alapuolella.

Neuronimallilla voidaan mallittaa aivojen neuronien aktivoitumista ja laukeamista ja siten signaalin kulkua ja prosessointia aivoissa. DLA eli Diffusion Limited Aggregation tuottaa fraktaaleja soluautomaattihilassa random-walk -tekniikalla, hiekkakekomalli kuvaa havainnollisesti itseorganisoituvaa kriittistä tilaa (maanjäristykset, pörssikurssit) ja metsäpalomalli selittää esim. tarttuvien sairauksien leviämistä.

Esim. Hiekkakekomalli:

- G *soluavaruus:* säännöllinen suorakulmainen hila
- Q diskreetti *tila-avaruus:* $\{0, 1, 2, 3, ...\}$ = hiekanjyvien lkm.
- V *naapurusto:* viereiset solut "pareittain"
- f *soluautomaattisääntö:* kun vierekkäisten solujen arvojen erotus on suurempi kuin jokin määrätty kriittinen arvo x_c, tietty määrä hiekanjyviä putoaa solusta toiseen (alamäkeen).
- t *aika:* diskreetti.

Nyt yhteen soluun (johon muodostuu keon huippu) lisätään yksi tai useampia hiekanjyviä jokaisella aika-askeleella.

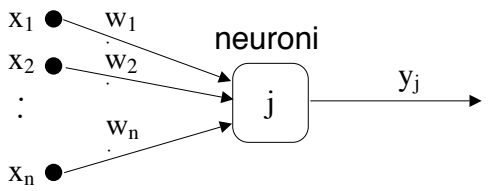
Kaksiulotteisessa hiekkakekomallissa tapahtuvien vyörymien keskimääräinen koko skaalautuu neliöllisesti soluautomaattihilan koon funktiona.

Edellisestä erikoistapauksena on ns. toimistomalli, jossa siirrellään kansioita pöydältä toiselle ja $x_c = 4$.

2. JOHDATUS NEURAALIVERKKOIHIN

2.1. Rakenne

Neuraaliverkot muodostuvat toisiinsa kytketyistä *neuroneista,* jotka kaikki toimivat samalla tavalla. Kytkentöjen voimakkuus on määritetty jokaiselle neuroniparille erikseen. Kytkennän voimakkuutta neuronien i ja j välillä kutsutaan painokertoimeksi, jota useimmiten merkitään w_{ij}. *Painokertoimet* muodostavat matriisin $W = \{w_{ij}\}$.



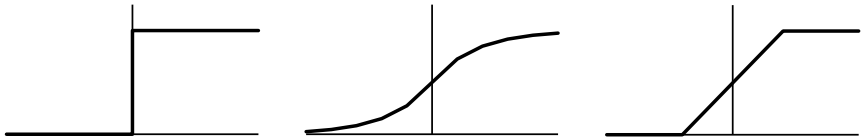
Kuva 2.1.1. Neuronin rakenne.

Neuronin j sisääntulot muodostavat vektorin $X^{(j)} = (x_1^{(j)}, ..., x_n^{(j)})$, missä n on neuronien lukumäärä. Jokainen neuroni laskee yhteen kaikkien muiden neuronien sille lähettämät signaalit vastaavilla painokertoimilla kerrottuna. Neuronin arvo (ulostulo) riippuu sisääntulojen painotetusta summasta etukäteen määritellyn kynnysfunktion g mukaisesti. Siten

$$y_j = g(W X + \theta) = g(\sum_i w_{ij} x_i^{(j)} + \theta), \qquad (2.1.1)$$

missä θ on vakio, ns. *kynnysparametri.*

Kynnys- eli aktivaatiofunktio on kaikille neuroneille yleensä sama ja tavallisimpia muotoja on esitetty kuvassa 2.1.2.



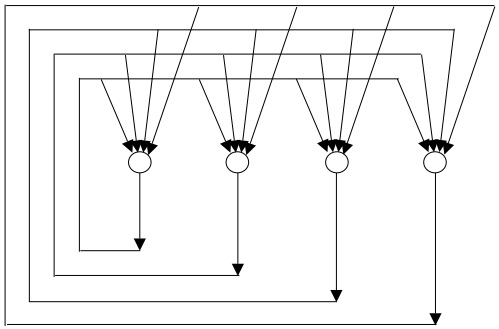
Kuva 2.1.2. Yleisimpiä neuronin kynnysfunktioita: askelfunktio (eli binäärifunktio), sigmoidi $[1+e^{-x}]^{-1}$ ja paloittain lineaarinen funktio.

2.2. Toiminta

Kytkentätavan mukaan verkot määritellään *kokonaan kytketyiksi* tai kerroksittaisiksi *feed-forward -tyyppisiksi.* Kokonaan kytketyissä verkoissa kaikille neuroneille annetaan ensin alkuarvot, minkä jälkeen neuronien arvoja päivitetään säännön

$$x(t+1) = g(W X(t)) \qquad (2.2.1)$$

mukaisesti *synkronisesti* tai *asynkronisesti.* Tässä t tarkoittaa diskreettiä aikaa. Lopputuloksen muodostavat neuronien arvot, kun iterointi on lopetettu.

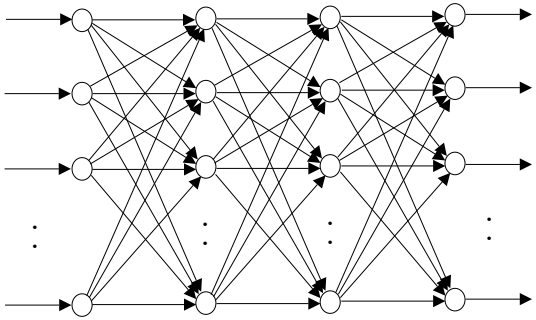


Kuva 2.2.1. Kokonaan kytketty neuraaliverkko.

Feed–forward-tyyppisissä verkoissa neuronit on jaettu tasoihin (kerroksiin), joiden sisällä neuroneja ei ole kytketty toisiinsa. Alkuarvot annetaan vain ensimmäisen tason neuroneille. Neuronien arvot päivitetään säännön

$$x(t+1) = g(W(t)\, X(t)) \qquad (2.2.2)$$

mukaan. Nyt $t = 1, 2, ..., k,$ on neuronitason numero ja k on tasojen lukumäärä. Tulosvektorin muodostavat viimeisen tason neuronien arvot k päivityksen jälkeen.



Kuva 2.2.1. Feed-forward -tyyppinen neuraaliverkko.

Jos alkuarvoiksi annettu vektori X on määritelty joukossa $\mathbb{D}$, ja tulosvektori joukossa $\mathbb{E}$, niin kokonaan kytketty neuraaliverkko kuvaa X:n Y:lle ($f: \mathbb{D} \rightarrow \mathbb{E}$) N iteroinnin jälkeen funktion

$$\begin{aligned} Y &= f(X) = g(W(g(W \ldots g(WX) \ldots))) \\ &= (g \circ W)^N X \end{aligned} \qquad (2.2.3)$$

mukaisesti.

Samoissa joukoissa, $\mathbb{D}$ ja $\mathbb{E}$, k -tasoisen feed-forward -tyyppisen neuraaliverkon suorittama kuvaus on muotoa

$$Y = g(W_k(g(W_{k-1} \ldots g(W_1 X) \ldots))) . \qquad (2.2.4)$$

Kokonaan kytkettyä verkkoa voidaan pitää Feed–forward -tyyppisen verkon yleistyksenä. Kokonaan kytketty verkko muodostaa feed–forward-tyyppisen verkon, jos sen painokertoimien annetaan muuttua ajassa.

2.3. Opetus

Kokonaan kytketyn verkon suorittamassa kuvauksessa on korkeintaan n^2 vapaasti määritettävää parametria ja feed–-forward-tyyppisen verkon tapauksessa on parametrejä yhtä paljon kuin neuronien välisiä kytkentöjä. Näille parametreille tulisi löytää arvot siten, että neuraaliverkon suorittama kuvaus f on mahdollisimman lähellä haluttua kuvausta. Parametrien arvojen määrittämistä kutsutaan verkon *opettamiseksi.*

Opetusalgoritmi tai -sääntö määrää, miten parametrien arvot määritetään. Erilaisia opetussääntöjä on lukuisia. Useimmat niistä ovat kokeilemalla löydettyjä ja hyväksi havaittuja ilman sen tarkempaa analyyttistä perustaa johtuen siitä, että neuraaliverkoilla usein approksimoidaan kuvauksia, jotka ovat matemaattisesti katsoen epätarkasti määriteltyjä. Usein myös määrittely- ja kuvajoukkojen rakenne ja laajuus tunnetaan huonosti. Riippumattomien muuttujien lukumäärä, esimerkiksi, saattaa olla tuntematon.

Opetusalgoritmit jakautuvat kahteen luokkaan, *ohjaamattomiin eli itsejärjestyviin* ja *ohjattuihin algoritmeihin.* Itsejärjestyviä opetusalgoritmeja käytettäessä ensin määritetään säännöt, joiden mukaan verkon parametreja muutetaan alkutilan perusteella. Opetusta varten määrittelyjoukosta valitaan sopivia alkutiloja. Verkon parametrien annetaan muuttua vuorotellen kunkin alkutilan perusteella annettujen sääntöjen mukaan. Itsejärjestyviä opetusalgoritmeja voidaan käyttää vain silloin, kun haluttu verkon suorittama kuvaus voidaan ilmaista määrittelyjoukon laskettavien ominaisuuksien avulla.

Jos neuraaliverkon suorittamaa kuvausta on vaikea yksiselitteisesti yhdistää määrittelyjoukossa havaittaviin säännönmukaisuuksiin, voidaan käyttää ohjattua opetusta. Tällöin opetus perustuu siihen, että tarjolla on riittävästi tavalla tai toisella oikeiksi havaittuja alkutila–lopputila -pareja. Ohjattu opetus voidaan suorittaa kahdella tavalla. Luotettavampi niistä vaatii sen, että voidaan määrittää jokin *verkon toiminnan hyvyyttä kuvaava, laskettava suure.* Kun tällainen suure M on löydetty ja opetusalgoritmi on valittu, suoritetaan opetus tyypillisten esimerkkitapausten avulla.

Ensin annetaan painokertoimille W satunnaisarvot ja valitaan m kappaletta opetuspareja (x_i,y_i) siten, että y_i on haluttu x_i:n kuva. Seuraavaksi lasketaan M:n arvo annettujen painokertoimien W ja opetusparien (x_i,y_i) avulla,

$$M = M(W, (x_1,y_1), (x_1,y_1), ..., (x_m,y_m)). \qquad (2.3.1)$$

M:n arvosta riippuen muutetaan seuraavaksi painokertoimia valitun opetusalgoritmin määräämällä tavalla. Sen jälkeen lasketaan taas M:n arvo jne., kunnes on saavutettu toivottu tarkkuus.

Joskus on vaikea löytää neuraaliverkon toiminnan hyvyyttä kuvaavaa globaalia parametria. Siinä tapauksessa voidaan neuraaliverkkoa opetettaessa muuttaa painokertoimia jokaiselle opetusparille erikseen valitun opetussäännön mukaisesti. Silloin määritellään *virhefunktio* $e = e(W,\{(x_i,y_i)\})$, joka kuvaa saadun tulosvektorin ja halutun tulosvektorin välistä etäisyyttä esim. pienimmän neliösumman mielessä. Virhe e riippuu luonnollisesti painokerroinmatriisista ja se lasketaan vuorotellen jokaiselle opetusparille. Opetusalgoritmi määrää, miten painokertoimia muutetaan virheen e perusteella. Verkon toimintaa kuvaavan parametrin puuttuessa opetuksen onnistuminen voidaan todeta vain jälkikäteen kokeilemalla.

Mikäli verkon suorittama kuvaus tunnetaan ja voidaan lausua

joko kokonaan kytketyn tai feed–forward -tyyppisen neuraaliverkon muodossa, kaavat (2.2.3) ja (2.2.4), ei opetusta tarvita lainkaan, vaan painokertoimet voidaan lukea suoraan kaavasta. Tämä on usein mahdollista yksinkertaisissa tunnistustehtävissä, joiden suorittamiseen riittävät pienet, muutaman neuronin verkot.

Esim. Suunnittele neuraaliverkko, joka tunnistaa 4:llä jaolliset luvut.

2.4. Käyttömahdollisuudet

Neuraaliverkkoja voidaan käyttää kahdenlaisten tehtävien suorittamiseen. Ne voivat suorittaa *tunnistustehtäviä* (vrt. luokittelua) ja *mallinnusta.* Jälkimmäisiä tehtäviä kutsutaan myös yleistystehtäviksi. Raja näiden tehtävätyyppien välillä ei ole yksiselitteinen. Niiden välinen ero piilee lähinnä ongelman esitystavassa ja kuvajoukon laajuudessa.

Esimerkiksi kuvankäsittelyssä neuraaliverkoilla poistetaan kohinaa, korjataan virheitä ja täydennetään kuvaa. Nämä ovat tyypillisiä tunnistustehtäviä. Samantapaisia ongelmia tavataan myös signaalinkäsittelyssä. Mitä tahansa dataa tunnistettaessa verkon toimintaa on yleensä helppo mitata jollakin globaalilla parametrilla. Tämä johtuu siitä, että kuvauksen f kuvajoukko $\mathbb{E}$ on suhteellisen pieni. Tällöin opetus kannattaa tehdä globaalin parametrin M avulla.

Luokittelu vaatii tunnistamista sekin. Erona on vain se, että luokittelussa neuraaliverkon suorittaman kuvauksen kuvajoukko $\mathbb{E}$ ei ole määrittelyjoukon $\mathbb{D}$ osajoukko niinkuin tunnistustehtävissä. Luokittelutehtävissä useimmiten luokittelun perusteena olevat kriteerit tunnetaan yksiselitteisesti. Luokittelutehtäviin soveltuvatkin erityisesti itsejärjestyvät verkot.

Yleistystehtävissä kuvauksen f kuvajoukko $\mathbb{E}$ saattaa olla hyvinkin laaja ja monimutkainen. Verkon toimintaa kuvaavaa parametria on usein vaikea löytää, joten verkon opettaminen on tehtävä virhefunktion e avulla. Yleistystehtävissä tutkitaan tyypillisesti datavektorin muodostavien muuttujien välisiä riippuvuuksia. Opetuksella pyritään muokkaamaan neuraaliverkko siten, että se havaitsee ja tunnistaa datavektorien sisältämiä riippuvuuksia, ennemminkin kuin datavektoreita sinänsä.

Varsinaisen tehtävän suorittamisen lisäksi neuraaliverkot voidaan opettaa siten, että ne kuvaavat odottamattoman alkuarvon (so. datavektori määrittelyjoukon ulkopuolella) etukäteen määritellyn kuvajoukon ulkopuolelle. Yhdistelemällä erilaisia neuraaliverkkoja saadaan toimivia kokonaisuuksia hyvinkin monimutkaisten ja erilaisista osatehtävistä muodostuvien ongelmien ratkaisemiseen.

2.5. Toteutukset

Neuraaliverkot voidaan toteuttaa ohjelmallisesti (esim. MATLAB), jolloin yleiskäyttöinen tietokone ohjelmoidaan simuloimaan neuraaliverkkoa, tai elektronisina piireinä. Piiritoteutukset ovat useimmiten PC-ohjattavia kortteja, jolloin ne ovat helposti käytettävissä erilaisten ongelmien ratkaisemiseen. Verrattaessa simulointeihin neuraaliverkkokortit ovat huomattavasti nopeampia. Mikäli neuraaliverkolta vaaditaan vieläkin suurempaa nopeutta, on myös sen käyttöympäristö suunniteltava alusta alkaen määrättyä sovellutusta silmällä pitäen. Tällaista kokonaisuutta voi jo kutsua neuraaliverkkotietokoneeksi. Toinen syy erityisen neuraaliverkkokokokonaisuuden rakentamiseen voi olla tilanpuute.

Kaupallisia neuraaliverkkojen simulointiohjelmia on saatavana useitakin. Nämä ohjelmat sisältävät yleensä useampia neuraaliverkkorakenteita ja/tai opetusalgoritmeja. Erittäin vaativia tai laajoja ongelmia ne eivät kuitenkaan pysty ratkaisemaan. Kaupallisia piiritoteutuksia neuraaliverkoista on vain hyvin harvoja.

2.6. Sovellutukset

Neuraaliverkkoja käytetään hyvin moniin eri tarkoituksiin mm. data-analyysissä, signaalinkäsittelyssä, ohjaus- ja säätöjärjestelmissä. Vaikka opetusalgoritmien matemaattiset ominaisuudet ovatkin suureelta osin vielä tutkimatta, kokeilemalla löydetyt opetusalgoritmit ovat niin suoraviivaisesti sovellettavissa, että jopa vajavaisesti formuloidut tehtävät voidaan useimmiten ratkaista. Tämä juuri onkin neuraaliverkkojen suurin etu.

Tilastollista analyysia suoritettaessa käytetään hyväksi neuraaliverkkojen yleistysominaisuutta, kuten esim. mittaustulosten analysoinnissa kokeellisessa fysiikassa. Säätöjärjestelmissä neuraaliverkot suorittavat säädettävän järjestelmän mallin invertointia. Robottien ym. ohjauksessa neuraaliverkot mallintavat ulkoista ympäristöä sensorien antamien signaalien perusteella. Neuraaliverkot soveltuvat myös tiedon etsimiseen asiantuntijajärjestelmissä sekä puheen, tekstin ja kuvien tunnistamiseen.

3. GENEETTISET ALGORITMIT

Julkaisussaan *Adaptation in Natural and Artificial Systems* John Holland osoitti 1975 kuinka monimutkaisia rakenteita voidaan koodata bittijonoiksi ja kuinka bittijonojen yksinkertaisilla transformaatioilla voidaan muuttaa (parantaa) em. rakenteiden "sopeutumista ympäristöönsä". John Holland kuvasi myös geneettisen algoritmin (GA), jonka mukaan bittijonojen, tai niitä vastaavien koodaamattomien rakenteiden, evoluutiota voidaan simuloida samaan tapaan kuin esim. eläinpopulaation evoluutiota sen sopeutuessa elinympäristöönsä.

Tällaista evoluutiota voidaan soveltaa optimointiongelman ratkaisemiseen. Eräs tärkeä Hollandin tulos oli se, että jopa suurissa ja monimutkaisissa systeemeissä tällaiset geneettiset algoritmit suppenevat kohti sellaisia "ratkaisuja", jotka ovat globaaleja optimeja tai hyvin lähellä niitä.

3.1. Peruskäsitteet

Geneettiset algoritmit pyrkivät siis matkimaan elävän luonnon evoluutiomekanismeja optimointiongelman ratkaisemiseksi, kun bittijonot ja niihin koodattu informaatio rinnastetaan kromosomeihin (ja geeneihin) seuraavalla tavalla:

- populaation yksilöt (tai oliot) ominaisuuksineen edustavat yritteitä optimointiongelman ratkaisuiksi
- olion ominaisuudet koodataan *kromosomin* muotoiseksi ketjuksi
- yritteen hyvyys *(sopeutuminen)* lasketaan sen ominaisuuksista kustannus- eli sakkofunktioksi, vrt. simulated annealing
- oliot lisääntyvät vanhemmista tyttäriksi siten, että tyttärien kromosomit muodostuvat vanhempien kromosomeista geneetisillä prosesseilla: *crossover, inversio ja mutaatio*
- populaatiosta karsitaan huonoimmat yritteet ja hyvien annetaan lisääntyä, ja mahdollisesti vielä tehokkaammin *(luonnollinen valinta)*

Perusideana on siis *rakennuspalikkahypoteesi* (building block hypothesis): hyvät kromosomit sisältävät osaratkaisuja, joita yhdistelemällä on mahdollista löytää vielä parempia ratkaisuja.

Tällaista menetelmää voidaan käyttää varsin monimutkaisten optimointiongelmien ratkaisemiseen silloin, kun kohtuullisen hyvän ratkaisun löytäminen riittää. Tarkan optimin löytämiseksi perinteiset menetelmät saattavat olla tehokkaampia, jos hyvä alkuarvaus on saatavilla. GA voidaan käyttää tällaisten alkuarvauksien tuottamiseen.

GA, samoin kuin simulated annealing", soveltuvat erityisen hyvin sellaisiin tapauksiin, joissa optimointitehtävän muuttujat ovat epäjatkuvia tai kustannusfunktiolla on paljon paikallisia minimejä. Tällaisia ovat esim. sijoittelu- tai järjestysongelmat, kuten esim. kauppamatkustajan ongelma. Myös funktioiden "optimointi" moniulotteisissa avaruuksissa on sopiva ratkaistavaksi GA-menetelmillä.

Erityisen sopivia ovat GA-menetelmät silloin, kun halutaan löytää useampia erilaisia suhteellisen hyviä ratkaisuja.

3.2. Geneettisten algoritmien rakenne

Optimointiongelman ratkaisemiseen soveltuvalla geneettisellä algoritmilla täytyy olla ainakin seuraavat 5 komponenttia:

1. Probleeman ratkaisun esitettävyys kromosomimuodossa (mutta ei välttämättä bittijonona) P_j ; $j = 1, 2, ..., M$; missä M on "geenien" lukumäärä.
2. Tapa muodostaa yritepopulaatio eli optimointitehtävälle sopivia alkuarvoja $P_{i,j}^0$; $i = 1, 2, ..., N$; missä N on populaation koko.
3. Kustannus- eli sakkofunktio, joka on määritelty kromosomien muodostamassa joukossa (geenien avulla)

   $$f = f(\{P_{i,j}^k\}_{j=1,M}) \qquad (3.2.1)$$

   jokaiselle sukupolvelle k.
4. Geneettiset operaatiot tyttärien risteyttämiseksi vanhemmista, esim. crossover, inversio ja mutaatio, sekä mahdollisesti muita kulloinkin tarkoitukseen sopivia operaatioita.
5. Muut tarvittavat parametrit: populaation sallittu koko, todennäköisyydet erilaisille geneettisille operaatioille, luonnollisen valinnan määrittely: säilymis/tuhoutumisehto ja -algoritmi (uusitaanko koko populaatio samalla kertaa vaiko yksitellen).

Mikäli edellinen sukupolvi poistetaan kokonaisuudessaan uutta muodostettaessa, on vaarana se, että koko evoluutio historian absoluuttisesti parhaimmat yksilöt katoavat. Tämä voidaan estää säilyttämällä aina ainakin pieni joukko parhaimpia yksilöitä tyttärien lisäksi. Tätä kutsutaan *elitismiksi.*

Esim. Kuinka geneettistä algoritmia voitaisiin käyttää lukujärjestyksen laatimiseen koulussa, jossa on useita opettajia ja useita luokkia.

Esim. Usean muuttujan funktion minimointi.

3.3. Kromosomien koodaaminen

Edellinen esimerkki voitaisiin ratkaista myös binäärisiä kromosomeja käyttäen, jos sopiva koodausalgoritmi on käytössä. Binääriluku $\{P_j\}_{j=1,M} = \{P_1, P_2, ..., P_M\}$, missä $P_j = 0$ tai 1, voidaan koodata reaaliluvuksi esim. algoritmilla

$$x = \sum_{j=1}^{M} P_j \, 2^{j-1}, \qquad (3.3.1)$$

joka voidaan vielä skaalata välille $[-r, r]$ seuraavasti

$$x_r = \sum_{j=1}^{M} \left[\frac{2r}{2^M - 1} P_j \, 2^{j-1} \right] - r \, . \qquad (3.3.2)$$

Huomaa, että käänteistä algoritmia ei tarvita!

Järjestyksen kirjoittaminen kromosomiin on helppoa, mutta geneettisten operaatioiden soveltaminen ei tällöin tuota "oikeita" yksilöitä tyttäriksi. Eräs tähän soveltuva algoritmi on ns. *tasainen tasopohjainen risteytys.* Siinä muodostetaan ensin satunnainen binäärinen lukujono, esim.

0 1 1 0 1 1 0 0,

jonka pituus on sama kuin populaation kromosomien pituus. Lukujonoa käyttäen muodostetaan vanhemmista

1: 1 2 3 4 5 6 7 8 ja
2: 8 6 4 2 7 5 3 1

tyttäret

a: – 2 3 – 5 6 – – ja
b: 8 – – 2 – – 3 1,

ja tästä täydentämällä samassa järjestyksessä kuin luvut ovat toisessa vanhemmista

a: 8 2 3 4 5 6 7 1 ja
b: 8 4 5 2 6 7 3 1.

3.4. Geneettisten algoritmien sovellutuksia

Geneettisiä algoritmeja on sovellettu optimointitehtäviin esim. tietoliikenneverkkojen suunnittelussa, puolijohderakenteiden suunnittelussa, peliteoriassa ja mm. kirjoituskoneen näppäimistön suunnittelussa sellaisia kieliä varten, joissä kirjainmerkkien lukumäärä on erityisen suuri.

Esim. Kauppamatkustajan ongelman "koodaus".

4. SUMEA LOGIIKKA JA JÄRJESTELMÄT

Sumea logiikka (fuzzy logic) kehitettiin alunperin perinteisen (täsmällisen) logiikan vastineeksi tapauksiin, joissa käsiteltävä tieto ei ole tarkasti tunnettua. Sumeassa päättelyssä käytetään täsmällisen tiedon ohella myös *epätäsmällistä* ja *epävarmaa* tietoa. Päättely perustuu *jäsenyysfunktioihin* ja *sumeisiin sääntöihin.* Päättelyn tuloksena saadaan jatkuva-arvoinen luku, joka kertoo *päätöksen totuusarvon.*

Sumeiden systeemien teorian perusteet loi L. Zadeh 1960- ja 70-luvulla esittämällä *sumean joukko-opin* (fuzzy set theory) ja siihen liittyvän sumean logiikan. Vuonna 1991 perustettiin Euroopassa kansainvälinen ELITE-järjestö, jonka päämääränä on koordinoida tieteellistä aktiviteettia älykkäiden teknologioiden alueella. Näihin teknologioihin luetaan sumeat järjestelmät, neurolaskenta, geneettiset algoritmit ja asiantuntija-järjestelmät.

Sumean logiikan sovellutukset ovat levinneet jo kulutuselektroniikkaankin. Tuote-esimerkkejä ovat pesukone, pölynimuri, riisikeitin, mikroaaltouuni, kerosiinilämmitin sekä kameroiden automaattitarkennus ja -valotus.

4.1. Epätäsmällisyys ja epävarmuus

Sumeiden systeemien teoriassa käytetään käsitteitä *epätäsmällisyys* (vagueness, imprecision, inexactness) ja *epävarmuus* (uncertainty).

Epätäsmällisiä käsitteitä ovat esim. "suuri" ja "pieni", joiden merkitykset riippuvat asiayhteydestä ja vertailukohdasta. Samoin esim. makuaistimukset ja ihmisten asenteet jäävät usein varsin epätäsmällisten kuvausten varaan. Väittämä: "henkilö on noin 25-vuotias" on epätäsmällinen.

Epävarmuutta on perinteisesti kuvattu todennäköisyyksien avulla tai tapahtuman frekvenssiin perustuen. Väittämä: "henkilö on ehkä 25-vuotias" tai "henkilö on todennäköisyydellä 30% 25-vuotias" on epävarma.

Siten epätäsmällinen väite voi olla varma tai epävarma, ja vastaavasti epävarma väite voi olla täsmällinen tai epätäsmällinen, esim. edeltä: "henkilö on todennäköisyydellä 30% noin 25-vuotias". Yleisesti tarkastellaankin juuri tällaisia tapauksia, joissa on otettava huomioon sekä epätäsmällisyys että epävarmuus.

Koska todennäköisyyteen liitetään yleensä vain täsmälliset käsitteet, tarkastellaan sumeissa järjestelmissä todennäköisyyden sijasta *mahdollisuutta.*

4.2. Sumeat joukot

Sumeiden järjestelmien teorian perustana on *sumea joukko-oppi,* joka on klassisen joukko-opin laajennus. Klassisessa joukko-opissa objektit (tai alkiot) x joko kuuluvat ($x \in A$) tai eivät kuulu ($x \notin A$) annettuun joukkoon A. Joukko A voi olla ääretön tai äärellinen ja se voidaan määritellä monella eri tavalla.

Klassisen joukko-opin perusoperaatiot ovat leikkaus, yhdiste (unioni) ja komplementti.

Sumeassa joukko-opissa objekti voi kuulua joukkoon kokonaan tai vain osittain. Esim. harmaan objektin voidaan ajatella kuuluvan osittain sekä mustien että valkoisten obektien joukkoihin.

Klassisen joukko-opin perusoperaatiot voidaan esittää käyttäen *karakteristista funktiota*

$$c_A(x) = \begin{cases} 1, & \text{kun } x \in A \\ 0, & \text{kun } x \notin A. \end{cases} \qquad (4.2.1)$$

Tällöin saadaan

- komplementti $B = A', \Leftrightarrow c_B(x) = 1 - c_A(x)$ (4.2.2)
- leikkaus $c_{A\cap B}(x) = \min(c_A(x), c_B(x))$ (4.2.3)
- yhdiste $c_{A\cup B}(x) = \max(c_A(x), c_B(x))$ (4.2.4)

kaikille $x \in E$, joista seuraa mm.

$$A \cap A' = \varnothing \qquad (4.2.5)$$

ja

$$A \cup A' = E. \qquad (4.2.6)$$

Sumeassa joukko-opissa voidaan määritellä ns. jäsenyysfunktio, joka karakterisoi perusjoukon E sumeaa osajoukkoa, karakteristisen funktion tavoin. *Jäsenyysfunktion* $\mu(x)$ arvoalue on laajennettu esimerkiksi suljetuksi väliksi $[0, 1]$ siten, että objektin x täysi jäsenyysaste joukossa A saadaan, kun $\mu_A(x) = 1$, ja jos objekti x ei kuulu ollenkaan kyseiseen joukkoon, niin $\mu_A(x) = 0$. Yleisesti, kun x kuuluu joukkoon A osittain, niin

$$0 \le \mu_A(x) \le 1. \qquad (4.2.7)$$

Mitä vähäisempää osittainen joukkoon kuuluminen on sitä lähempänä nollaa vastaavasti on jäsenyysaste μ.

Jäsenyysfunktion tulkinta on siis erilainen kuin todennäköisyysfunktion, vaikka molemmilla on sama arvojoukko $[0, 1]$. Jäsenyysaste tulkitaan "mahdollisuus"käsitteen (possibility) avulla. On perusteltua vaatia, että mahdollisuus on aina vähintään yhtä suuri kuin todennäköisyys.

Jäsenyysfunktio voidaan määritellä jatkuvana, esim.

$$\mu_A(x) = \begin{cases} 1 - (x-6)^2, & \text{kun } 5 \le x \le 7 \\ 0, & \text{muulloin} \end{cases}$$

tai vaikkapa kokonaislukujen joukossa

$$\mu_A(x) = \{0.2/4, 0.7/5, 1/6, 0.5/7, 0.1/8\}.$$

Molemmat yllä jäsenyysfunktiot yllä määrittelvät sumean joukon "noin 6".

Sumeissa järjestelmissä pyritään käyttämään *konvekseja sumeita joukkoja,* mutta joskus päättelyiden tuloksena voidaan saada myös ei-konvekseja joukkoja.

Tavallisin jäsenyysfunktio on kolmiofunktio eli Λ-funktio. Tavallisia ovat myös puolisuunnikkaan muotoinen funktio tai kellokäyrä. Monimutkaiset jäsenyysfunktiot esitetään usein paloittain lineaarisina funktioina.

Sumean joukko-opin säännöt ja operaatiot ovat seuraavat:

- tyhjä joukko $\mu_{\varnothing}(x) \equiv 0$ (4.2.8)
- perusjoukko $\mu_{E}(x) \equiv 1$ (4.2.9)
- identtisyys $A = B \Leftrightarrow \mu_A = \mu_B$ (4.2.10)
- osajoukko $A \quad B \Leftrightarrow \mu_A \leq \mu_B$ (4.2.11)
- komplementti $B = A', \Leftrightarrow \mu_B(x) = 1 - \mu_A(x)$ (4.2.12)
- leikkaus $\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$ (4.2.13)
- yhdiste $\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x))$ (4.2.14)

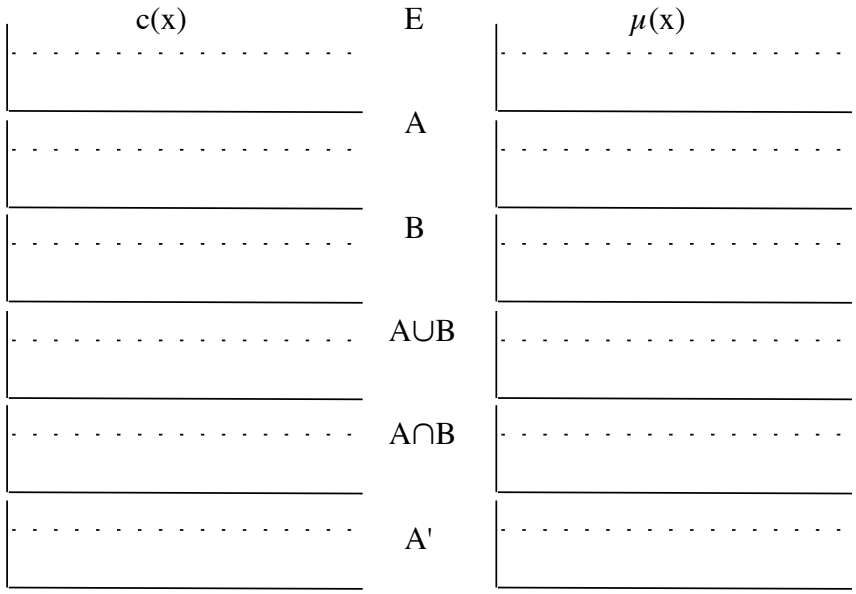
kaikille $x \in E$, joista seuraa, että yleisesti

$$A \cap A' \neq \varnothing \quad (4.2.15)$$

ja

$$A \cup A' \neq E. \quad (4.2.16)$$

Huomaa, että yhtälöt (4.2.12 – 4.2.16) ovat samat kuin yhtälöt (4.2.2 – 4.2.6) ja jäsenyysfunktioilla voidaan operoida samoin kuin karakterisitilla funktioilla sivulla 25.



4.3. Sumea päättely

Klassisen joukko-opin joukoilla voidaan voidaan "laskea" ns. intervallianalyysin laskusäännöillä

$$[a, b] + [c, d] = [a+c, b+d] \quad (4.3.1)$$

$$[a, b] - [c, d] = [a-d, b-c] \quad (4.3.2)$$

$$[a, b] \cdot [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \quad (4.3.3)$$

$$[a, b] / [c, d] = [a, b] \cdot [\,1/d, 1/c], \text{ jos } 0 \notin [c, d]. \quad (4.3.4)$$

Nämä laskusäännöt voidaan yleistää myös sumeille joukoille.

Matemaattisen logiikan ns. *modus ponens* päättely sääntö on seuraava:

Jos A on tosi
ja A ⇒ B
niin B on tosi.

Sumea päättelyssä on perustana joko *yleistetty modus ponens* tai *kompositionaalinen modus ponens.* Edellisestä on esimerkkinä

tomaatti on hyvin punainen
jos tomaatti on punainen niin tomaatti on kypsä
tomaatti on hyvin kypsä,

ja jälkimmäisestä

m on pieni sumea luku
m on hieman suurempi kuin n
n on melko pieni sumea luku.

5. KVANTTITIETOKONEET

5.1. Johdanto

Laskenta on fysikaalinen prosessi, mutta matematiikka, joka määrää laskennan tuloksen on siitä riippumaton abstrakti järjestelmä. Laskenta perustuu algoritmeihin, joilla on rajoituksensa matematiikasta riippumatta. Kvanttimekaniikkaan perustuva kvanttilaskenta antaa uudenlaisia mahdollisuuksia algoritmien laadintaan.

Kvanttilaskennan ja kvanttitietokoneen idean keksi R. Feynman jo 1980-luvulla, mutta varsinainen mielenkiinto kvanttilaskentaan heräsi vasta v. 1994. Tällöin Peter Shor osoitti, että kvanttitietokone pystyy etsimään suurten kokonaislukujen alkutekijät ratkaisevasti nopeammin kuin "klassinen" tietokone. Tätä voitaisiin käyttää ns. julkisen avaimen salausjärjestelmien eli kryptografian murtamiseen.

5.2. Kvanttibitti ja loogiset perusoperaatiot

Kaikki tieto voidaan koodata bitteinä (binäärilukuina) kaksiarvoisten rekisterien jonoihin. Kvanttilaskennassa bitti korvataan *kvanttibitillä* (engl. qubit), jolla voi olla kaksi arvoa $|0\rangle$ tai $|1\rangle$ tai näiden *koherentti superpositio*

$$c_0\,|0\rangle + c_1\,|1\rangle, \tag{5.2.1}$$

missä c_0 ja c_1 ovat kompleksilukuja, joille pätee

$$|c_0|^2 + |c_1|^2 \;=\; 1. \tag{5.2.2}$$

Kvanttibiteille eli qubiteille voidaan määritellä *loogiset perusoperaatiot*

AND, OR, XOR, NAND ja NOT

tavalliseen tapaan.

Klassisten bittien ja qubittien välinen ero on superpositiotilassa olevien qubittien keskinäinen interferenssi loogisissa perusoperaatioissa.

Kvanttilaskennassa käytetään yleensä *laskennallista kantaa*

$$\{\, |a_1\, a_2\, \ldots\, a_L\rangle \,\}, \tag{5.2.3}$$

missä $a_i = 0$ tai 1. Siten kanta virittää 2^L-ulotteisen Hilbertin avaruuden. Esim. jos $L = 2$, niin kanta on

$$\{\, |0\;0\rangle,\; |0\;1\rangle, |1\;0\rangle,\; |1\;1\rangle \,\}.$$

Kahden qubitin kokonaistila (aaltofunktio) on niiden superpositioiden tulo

$$\begin{aligned}&(c_0^1\,|0\,\bullet\rangle + c_1^1\,|1\,\bullet\rangle)\;\;(c_0^2\,|\bullet\,0\rangle + c_1^2\,|\bullet\,1\rangle) \;=\\ &=\; c_0^1\,c_0^2\,|0\;0\rangle + c_0^1\,c_1^2\,|0\;1\rangle + c_1^1\,c_0^2\,|1\;0\rangle + c_1^1\,c_1^2\;|1\;1\rangle.\end{aligned} \tag{5.2.4}$$

Kvanttimekaaniset laskentaoperaatiot ovat unitaarisia muunnoksia, jotka voidaan esittää laskennallisessa kannassa matriiseina. Tarkastellaan esimerkkinä kontrolloitua NOT-porttia C_{12}.

Siten

$$\begin{aligned}&C_{12}\,\{\,(c_0^1\,|0\,\bullet\rangle + c_1^1\,|1\,\bullet\rangle)\;\;(c_0^2\,|\bullet\,0\rangle + c_1^2\,|\bullet\,1\rangle)\,\} \;=\\ &=\; c_0^1\,c_0^2\,|0\;0\rangle + c_0^1\,c_1^2\,|0\;1\rangle + c_1^1\,c_0^2\,|1\;1\rangle + c_1^1\,c_1^2\;|1\;0\rangle.\end{aligned} \tag{5.2.5}$$

Tulos on kahden qubitin *lomittunut* (engl. *entangled*) *tila.* Em. laskennallisessa kannassa

(5.2.6)

5.3. Alkulukutekijöiden etsintä

Shorin faktorointialgoritmin lähtökohtana on, että kokonaisluvun N alkutekijöitä voidaan etsiä tarkastelemalla funktiota

$$F_N(x) \;=\; a^x \bmod N \qquad (5.3.1)$$

eli osamäärän a^x / N jakojäännöstä, kun a on kokonaisluku väliltä $[0, N]$. Kun x on kokonaisluku, niin $F_N(x)$ on jaksollinen, jaksona jokin kokonaisluku r.

Kun jakso r tunnetaan, saadaan luvun N tekijät etsimällä lukujen N ja $a^{r/2}\pm 1$ suurin yhteinen tekijä. Suurimman yhteisen tekijän etsimiseen on olemassa nopeita algoritmeja.

Esim. 5.1. Olkoon $N = 15$ ja $a = 7$. Etsi funktion $F_N(x)$ jakso ja sen avulla luvun 15 alkutekijät.

Kvanttitietokoneen tai siihen perustuvan algoritmin tehokkuus perustuu sen kykyyn löytää jakso r nopeasti. Shorin algoritmissa käytetään kahta L:n qubitin mittaista rekisteriä, kun $2^L \approx N^2$. Aluksi ensimmäinen rekisteri alustetaan siten, että sen jokainen qubitti tulee tilaan

$$1/\sqrt{2}\;\;(\,|0\rangle + |1\rangle\,) \qquad (5.3.2)$$

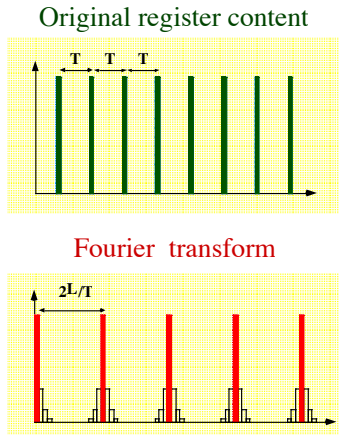
Siten siinä on edustettuna binäärimuodossa jokainen kokonaisluku x väliltä $[0, 2^L{-}1]$ samalla painolla tai todennäköisyydellä. Seuraavaksi lasketaan toiseen rekisteriin funktion $F_N(x)$ arvo. Näin tämä operaatio tehdään tavallaan kaikille kokonaisluvuille x välillä $[0, 2^L{-}1]$ samalla kertaa, massiivisesti rinnakkaistettuna.

Funktion arvon laskemisen seurauksena rekisterit lomittuvat ja koko tietokoneen tilaksi (aaltofunktioksi) tulee

$$\psi \;=\; |0\;\; F_N(0)\rangle \;+\; |1\;\; F_N(1)\rangle \;+\; \ldots \;+\; |2^L{-}1\;\; F_N(2^L{-}1)\rangle. \qquad (5.3.3)$$

Kun nyt mitataan toisen rekisterin tila eli rekisterin sisältämän binääriluvun arvo, romahtaa (engl. collapse) myös ensimmäisen rekisterin tila pienemmäksi superpositioksi. Se on nyt niiden lukujen x superpositio, joille $F_N(x)$ on toisen rekisterin mittauksessa saatu arvo.

Jos nyt mitattaisiin ensimmäisen rekisterin tila, saataisiin tulokseksi jokin näistä jäljelle jääneistä luvuista x. Mittaamisen sijaan tehdäänkin rekisterille diskreetti Fourier-muunnos, joka antaa tulokseksi jakson r. Tällainen Fourier-muunnos on kvanttilaskentaoperaatio, jota ei edellä kuvatulla funktion F laskennan tavalla voi verrata rinnakkaislaskentaan.



Esim. 5.2. Etsi luvun $N = 15$ alkutekijät simuloimalla edellä esitettyä kvanttitietokoneen algoritmia.

5.4. Dekoherenssi

Ideaalinen kvanttitietokone (tai jokainen kvanttibitti) olisi täydellisesti eristetty ympäristöstään, lukuunottamatta laskuoperaatioiden vaatimia vuorovaikutuksia. Vuorovaikutus ympäristön kanssa aiheuttaa yleensä joko bitin siirtymistä tilalta toiselle tai interferenssille välttämättömän tarkan vaiheinformaation eli koherenssin katoamisen.

5.5. Käytännön toteutuksista

Periaatteessa mikä tahansa kaksitilasysteemi sopii kvanttibitiksi. Bitin arvoja 0 ja 1 vastaavat kaksi tilaa voivat olla esimerkiksi elektronin tai ytimen spintilat, fotonin polarisaatiotilat, puolijohteessa olevaan kvanttipisteeseen vangitun elektronin energiatilat tai Cooperin parit suprajohtavissa saarekkeissa.