Neuroverkkojen tuottaminen geneettisen algoritmin avulla

Harri Salakoski

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Pro gradu –tutkielma
Toukokuu 2002

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Harri Salakoski: Neuroverkkojen tuottaminen geneettisen algoritmin avulla
Pro gradu -tutkielma, 50 sivua, 1 liitesivu
Toukokuu 2002

Tutkimus käsittelee neuroverkkojen tuottamista geneettisen algoritmin avulla. Geneettisen algoritmin käyttämästä genomista kasvatetaan neuroverkko erityistä kasvumallia käyttäen. Kasvatettua verkkoa arvioidaan testidatan ja verkon rakenteen perusteella.
Työssä hahmotellaan myös geneettisen algoritmin kehittämistä biologisten mallien avulla. Ensin määritellään rakenteellinen informaatio ja pohditaan sen merkitystä evolutiivisen algoritmin kehittämiseksi.
Työn empiirisessä osiossa esitellään Dave-kasvumalli. Kasvumallin avulla luodaan boolean-verkko, joka ratkaisee pariteettiongelman.

Avainsanat ja -sanonnat: geneettinen algoritmi, neuroverkko, L-systeemi, biologinen realismi.

1. Johdanto

Tässä tutkimuksessa käsitellään tavoitteiden mukaisen neuroverkon luontia geneettisen algoritmin avulla. Geneettisen algoritmin apuna käytetään kasvualgoritmia, joka kasvattaa genomista neuroverkon. Prosessi esitetään tekoelämän teoreettisessa viitekehyksessä.

Työn tavoitteena on rakentaa ohjelma, joka saa syötteenään joukon tulo-lähtö-pareja ja antaa tulosteena ongelmaan soveltuvan neuroverkon rakenteen. Ongelma oletetaan yksinkertaisuuden vuoksi hyvin määritellyksi funktioksi, joka on muotoa $y^p = f(x^k)$, jossa x^k ja y^p ovat bittivektoreita pituudeltaan k ja p. Prosessin tavoitteena on siis löytää neuroverkko, joka mahdollisimman hyvin approksimoi funktion f.

Tutkimuksessa käsitellään myös evolutiivisen algoritmin ja kasvualgoritmin yleistä kehittämistä, mutta enimmäkseen käsitellään niiden soveltamista neuroverkon kasvattamiseen. Kasvualgoritmia käytetään, koska sen avulla neuroverkko saa toivottuja ominaisuuksia, kuten modulaarisuutta [Gruau]. Kasvualgoritmi saa evolutiivisen algoritmin käyttämän genomin parametrina, josta kasvualgoritmi kasvattaa neuroverkon. Taulukkoon 1 on koottu tutkimuksen kolmea keskeistä osa aluetta sivuavia käsitteitä.

Taulukko 1. Tutkimusalan käsitteistöä

Kattava käsite	Biologinen metafora	Tutkimuksen keskeinen ala	Teknistä jaottelua
Etsintäalgoritmi	Evoluutio	Evolutiiviset algoritmit	Geneettiset algoritmit Geneettinen ohjelmointi
Kielioppi	Solujen erilaistuminen, eliön kasvu	Kasvualgoritmit	Graafi-L-järjestelmät Merkkijono-L-järjestelmät Solukoodaus
Approksimaattori	Biologinen neuroverkko	Neuroverkot	Syklittömät / rekursiiviset verkot

Evolutiivinen algoritmi on nimitys algoritmille, joka jollain tavoin muistuttaa evoluutiota. Evolutiivisista algoritmeista suurimman ja tunnetuimman ryhmän muodostavat geneettiset algoritmit. Evolutiivisen algoritmin käyttämää tietorakennetta kutsutaan genomiksi. Koska muunnos merkkijonosta verkoksi ei onnistu muuten luontevasti, käytetään kasvualgoritmia tuottamaan neuroverkon rakenne. Kasvualgoritmin käytön vaikutuksia käsitellään kohdassa 4.1.

Neuroverkkoja on useita muunnelmia, joita voidaan jaotella ominaisuuksien, käytetyn opetusalgoritmin tai verkon rakenteen mukaan. Taulukon 1 jaottelu syklittömän ja rekursiivisen verkon välillä on vain esimerkki. Neuroverkosta käytetään termiä ‘verkko’, kun mielenkiinto kohdistuu neuroverkon rakenteeseen. Neuroverkolla voidaan tarkoittaa muutakin kuin approksimaattoria, mutta tässä työssä ollaan kiinnostuneita vain neuroverkon kyvystä toimia tavoitefunktion approksimaattorina.

Biologisessa realismissa joudumme tyytymään metaforiin ja karkeisiin yleistyksiin, jotka toisaalta ovatkin tavoitteena. Biologiset metaforat tarjoavat termejä prosessin käsittelyyn ja tuovat esiin geneettisen algoritmin kehitysmahdollisuuksia. Biologisten ilmiöiden monimutkaisuutta ja kompleksisuutta ei siis kielletä, vaan ilmiöt tarjovat perustan matemaattiselle mallille [vertaa Lokki et al., 89]. Matemaattiset merkinnät antavat tutkimukselle konkretiaa ja auttavat algoritmin esittelyssä, mutta prosessin hyvyys voidaan osoittaa vain käytännön testien avulla [Koza, 1992].

Tämän työn aihepiiriä ovat aikaisemmin tutkineet esimerkiksi Gruau [1994], Boers ja Kuiper [1992] sekä Kitano [1990]. Frédéric Gruau on kehittänyt neuroverkkojen etsintään prosessin, jossa etsintäalgoritmina toimii geneettinen ohjelmointi ja neuroverkon kasvualgoritmina ns. *solukoodaus.* Solukoodaus ei ole biologisesti orientoitunut malli, vaikkakin Gruaun mallissa neuroverkko kasvaa solmujen (tai solujen) jakaantuessa.

Boersin ja Kuiperin [1992] kasvualgoritmi esitellään kohdassa 5.1. Mallissa geneettisen algoritmin tuottamasta merkkijonosta tulkitaan L-järjestelmille säännöt. Sääntöjä iteroitaessa kasvaa neuroverkon määräävä merkkijono. Alan pioneeri, Hiroaki Kitano [1990], on myös kehittänyt kasvualgoritmin neuroverkkojen kasvattamiseen. Kitano osoitti ensimmäisenä kasvumallin edut neuroverkkojen tuottajana.

Edellä mainitut tutkimukset ovat keskittyneet kasvualgoritmin kehittämiseen ja jättäneet evolutiivisen algoritmin kehittämisen sivuosaan, vaikka molemmat ovat prosessin toiminnan kannalta tärkeitä. Evolutiivista algoritmia, kasvualgoritmia ja genomin rakennetta tulisi kehittää yhdessä, mutta tämä on jätetty vähälle huomiolle. Tutkimusten tulosten riippumatonta arviointia ja malleja vertailevaa tutkimusta ei toistaiseksi ole saatavilla.

Ongelmaa sivuavat useat tutkimusalueet, esimerkiksi tekoelämä (artificial life), koneoppiminen (machine learning), laskennan teoria (theory of computation),

evolutionäärinen biologia (evolutionary biology), graafiteoria (graph theory), neuroverkot (neural networks) ja tilastotiede (statistical science) ovat avuksi prosessia kehitettäessä.

Prosessiin liittyy useita mielenkiintoisia kysymyksiä ja ongelmia, joita erillisinä on tutkittu paljon. Tässä tutkimuksessa ei pystytä ottamaan kattavasti huomioon erityisalojen tutkimusta. Tärkeimpinä lähteinä on käytetty kirjoja biologian [Niemi et al.; Lokki et al.], geneettisten algoritmien [Mitchell, 1996], geneettisen ohjelmoinnin [Koza, 1992] ja neuroverkkojen [Hecht-Nielsen, 1989, Zeidenberg, 1991] aloilta. Työn innoittajina ovat toimineet mielenkiintoiset kirjoitukset niin geeneistä kuin evoluutiosta yleensäkin [Dawkins; Kauffman; Maynard and Szathmáry].

Tekoelämän teoreettinen viitekehys on valittu parhaiten soveltuvana tässä tutkimuksessa käsitellyn prosessin kuvaamiseksi. Toisaalta tekoelämän tutkimus ei ole kiinnostunut automaattisesta neuroverkkojen generoinnista. Erinäisiä malleja ja merkintöjä joudutaan soveltamaan, joten työn yhtenäisyys kärsii. Vaikka prosessiin liittyviä kehitysehdotuksia riittää, empiirisessä osiossa (7.2) pyritään vain rakentamaan toimiva kokonaisuus, eikä kehitysehdotuksia testata empiirisesti.

2. Evolutiivinen laskenta

Etsintäalgoritmin tai yleisesti *optimoinnin* tavoitteena on löytää ratkaisuehdoitusten joukosta paras mahdollinen ratkaisu annettuun ongelmaan. Ratkaisua etsitään tiettyjen ratkaisuehdoituksia luokittelevien kriteerien perusteella, jotka on määriteltävä ongelmakohtaisesti [Viitanen, 1997, 4]. Ratkaisuehdoituksia luokitellaan ns. *hyvyysfunktion* avulla. Ratkaisuehdotusten joukko on osajoukko *hakuavaruudesta*. Koska hakuavaruus on suuri, usein ääretön, tarvitaan kehittyneitä menetelmiä hakuavaruuden läpikäymiseksi. Evolutiivinen algoritmi liikkuu hakuavaruudessa *geneettisten operaatioiden* avulla.

Evolutiivinen algoritmi on evoluutioon perustuva satunnaistettu etsintäalgoritmi. Se ylläpitää ratkaisuehdotusten joukkoa $P^t = \left\{x_1^t, \ldots, x_n^t\right\}$, missä t on *sukupolvi* ja n populaation koko. Ensimmäistä sukupolvea P^0 kutsutaan *alkupopulaatioksi*. Populaation koko on yleensä vakio (esim. n = 200), mutta voi muuttua prosessin edetessä. Uusi sukupolvi (iteraatio t + 1) saadaan sukupolven t ratkaisuehdotuksista geneettisten operaatioiden avulla. Näitä ovat esimerkiksi *mutaatio* ja *risteytyminen*. Geneettiset operaatiot muuntelevat ratkaisuehdotuksia; tämän tarkempaa määritelmää ei geneettisille operaatioille kannata antaa. Populaatio muuttuu sukupolvesta toiseen, kun hyvät ratkaisuehdotukset vaikuttavat huonoja todennäköisemmin seuraavan sukupolven sisältöön. [Michalewicz, 1]

Evolutiiviset algoritmit ovat osa *heikkojen* etsintäalgoritmien perhettä. “Heikko” on yleisnimi laajasti sovellettaville algoritmeille. Heikon etsintäalgoritmin etu on, ettei se tarvitse ongelmasta etukäteisinformaatiota. Evolutiivisen algoritmin lisäksi heikkoja etsintäalgoritmeja ovat esimerkiksi simuloitu jäähdytys (simulated annealing), tabuetsintä (tabu search) ja vuorikiipeily (hill-climbing). Useimmat etsintäalgoritmit [Aho, Mäkinen ja Poranen]

- luovat joukon ratkaisuehdotuksia,
- evaluoivat ehdotuksia hyvyysfunktiota käyttäen,
- päättävät mitkä ehdotukset säilytetään ja
- luovat uusia ehdotuksia muuntelemalla tai yhdistelemällä valittuja ehdotuksia.

John Holland [Holland, 1975] esitteli geneettisen algoritmin teoksessaan “Adaptation in Natural and Artificial Systems”. Geneettinen algoritmi on viime vuosina ollut ahkeran tutkimuksen kohteena ja siitä on useita muunnelmia.

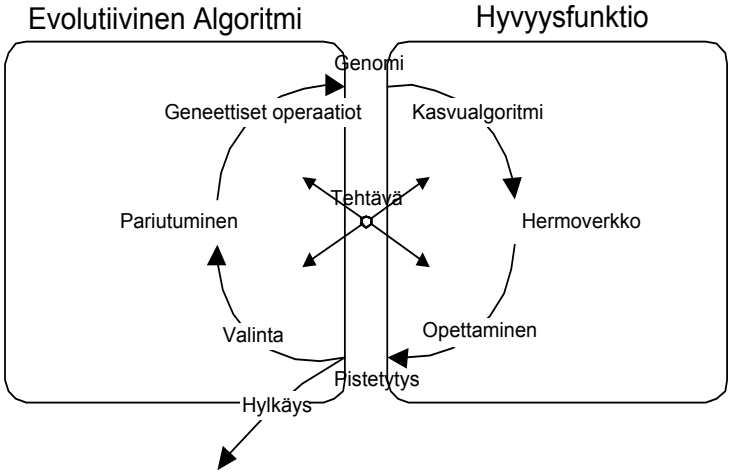
Geneettisessä algoritmissa ratkaisuehdotukset esitetään bittijonoina, joten ratkaisuehdotukset on koodattava bittijonoina, mikä voi olla vaikeaa tai vähintäänkin työlästä. Toisaalta geneettisen algoritmiin liittyy joukko valmiita geneettisiä operaatioita, jotka ovat käytettävissä, eikä niitä tarvitse ongelmakohtaisesti kehittää.

Toinen evolutiivisen laskennan muunnelma, *geneettinen ohjelmointi*, käyttää ratkaisuehdotuksen esittämiseen ongelmalle luonteenomaista tietorakennetta. Tällöin joudutaan määrittelemään tietorakenteelle soveltuvat geneettiset operaatiot. Ongelmalle räätälöidyt geneettiset operaatiot tekevät algoritmista *vahvan* ja algoritmin yleiskäyttöisyys kärsii. Geneettistä ohjelmointia on sovellettu esimerkiksi ”automaattiseen” ohjelmointiin. Koza tutkii LISP-ohjelmien generointia evolutiivisen laskennan avulla. Genomin tietorakenteena toimii jäsennetty LISP-koodi, joka on esitetty puuna. [Koza]

```
begin
        t ← 0
        alusta P^t
        evaluoi P^t
        while (ei lopetusehto) do
        begin
                t ← t + 1
                valitse P^t joukosta P^(t-1)
                rekombinoi P^t
                evaluoi P^t
        end
end
```

Kuva 1. Evolutiivinen algoritmi [Michalewicz, 2]

Evolutiivisen algoritmin tavanomainen rakenne on esitelty kuvassa 1. Kuvan 1 mukainen neuroverkon etsintäalgoritmi on vain malli, jota voidaan tapauskohtaisesti muunnella. Yleistäen voidaan sanoa, että evolutiivisen laskennan versioita on yhtä paljon kuin algoritmin tutkijoita. Kuvassa 2 on muunnelma biologista mukautumista esittävästä mallista.[Vaario, 1995] Malli kuvaa evolutiivisen algoritmin soveltamista neuroverkon etsintään. Evoluution abstraktio soveltuu siis suoraan kuvaamaan neuroverkon etsintäalgoritmia.



Kuva 2. Tutkimuksen teoreettinen viitekehys. Evolutiivinen laskenta sovellettuna neuroverkon etsintään [Vert. Vaario, 1995]

Kuvassa 2 vasemmalla ovat evolutiiviseen laskentaan kuuluvat operaatiot. Oikealla ovat neuroverkon evaluointiin liittyvät operaatiot, joita käsitellään luvuissa 3 ja 4. Kuvasta nähdään, kuinka neuroverkon etsintäprosessi voidaan jakaa seuraaviin osaongelmiin:

- genomin tietorakenteen määrittely (kohta 6.1),
- kasvualgoritmin kehittäminen (luvut 4 ja 5),
- neuroverkon rakenteen ja solmufunktioiden määrittely,
- neuroverkolle soveltuvan nopean opetusalgoritmin kehittäminen (kohta 3.2),
- opetetun neuroverkon hyvyyttä mittaavan algoritmin kehittäminen (kohta 3.3),
- valintafunktion kehittäminen,
- risteytyvien genomien valinnan eli pariutumisen kehittäminen (kohta 6.3) ja
- genomin tietorakenteelle soveltuvien geneettisten operaatioiden kehittäminen (kohta 6.2).

Tehtävä eli ratkottava ongelma voi vaikuttaa prosessiin useissa kohdissa, mutta tavallisesti se vaikuttaa vain neuroverkon testi- ja opetusdataan sekä pisteytykseen. Se voi vaikuttaa myös kasvualgoritmiin sekä evolutiivisen prosessin muihin osiin. Tehtävä vastaa biologista *ympäristön* käsitettä. Populaation muut jäsenet voidaan myös nähdä osana ympäristöä.

Hyvyysluku ja *genomi* ovat evolutiivisen algoritmin ja hyvyysfunktion väliset rajapinnat. Evolutiivinen algoritmi saa hyvyysfunktiolta tiedon genomin hyvyydestä. Hyvyysfunktio (evaluoi P^t kuvassa 1, oikea puolisko kuvassa 2)

määrittelee *populaation* yksilöille hyvyysluvut, joten joukkoa P^t vastaa siis hyvyysfunktion määrittelemä reaalilukujoukko H^t.

Määritelmä 1. Ongelman hakuavaruus A on geneettisessä algoritmissa tyyppiä $\{0,1\}^*$ oleva avaruus, jossa ratkaisuehdotukset $x^* \in \{0,1\}^*$ ovat bittijonoja. *Hyvyysfunktio h*: $A \subseteq \Omega \rightarrow \mathbf{R}$ on kuvaus joukosta Ω hyväksyttäviä ongelman ratkaisuehdotuksia reaalilukujen joukolle **R**. Kun $h(x_1) < h(x_2)$, niin x_1 on parempi kuin x_2. Ongelman *optimaalinen* ratkaisu on sellainen joukon A alkio x*, johon pätee $h(x^*) \leq h(x)$, $\forall x \in A$. Koska kyseessä on minimointiongelma, niin $h(x^*)$ on myös funktion *globaali minimi*. Ongelman optimaalinen ratkaisu on siis jokin *piste* x hakuavaruudessa A. [vert. Viitanen, 4]

Hyvyysfunktio määritellään ongelmakohtaisesti erikseen. Neuroverkon hyvyyden mittausfunktion ei tarvitse olla kovinkaan tarkka. Prosessiin kulutettu aika onkin lähes suoraan $t_i * k$, missä t_i on yhden neuroverkon evaluointiin käytetty keskimääräinen aika ja k evaluointien määrä. Koska resursseja on aina rajoitetusti käytettävissä, tulee yksittäiseen neuroverkkoon käytetty aika minimoida. Hyvyysfunktiota voidaan muunnella prosessin kuluessa. Prosessin edetessä erot populaation jäsenten välillä pienenevät ja hyvyysfunktion tarkkuudelle asetetaan suuremmat vaatimukset.

Taulukko 2. Evolutiivisen algoritmin parametrien vaikutus solmufunktion evaluointien määrään

Asetus	Pienehkö oletus	Suurehko oletus
Sukupolvien lukumäärä	50	2 000
Populaation koko	50	400
Populaatioiden lukumäärä	1	100
Hyvyysfunktion evaluointeja	2 500	80 000 000
Opetus ja valinta, testikertojen lukumäärä	100	10 000
Verkon evaluointeja	250 000	800 000 000 000
Opetuksen hinta solmua kohti	n	n^2
Verkossa solmuja	10	100
Solmufunktion evaluointeja	2 500 000	8 000 000 000 000 000

Evolutiivinen algoritmi vaatii toimiakseen useiden neuroverkkojen testaamisen. Neuroverkon opetusalgoritmit ovat raskaita ja jo yhden verkon kouluttaminen vaatii aikaa. Laskentaresurssien rajat tulevat nopeasti vastaan, jos neuroverkon opettamiseen käytetään standardiopetusalgoritmia. Neuroverkkoja käsittelevässä kirjallisuudessa esitetyt standardiopetusalgoritmit on suunniteltu yhden neuroverkon opettamiseen. Esimerkiksi yleisellä backpropagation-opetusalgoritmilla neuroverkon opetus maksaa $O(e^3)$, missä e on särmien määrä verkossa [Zeidenberg]. Taulukko 2 osoittaa, kuinka evolutiivisen algoritmin

parametrit vaikuttavat hyvyysfunktion evaluointien määrään. Evolutiivisen algoritmin parametrisointi on tutkijan omassa harkinnassa.

3. Neuroverkko

Neuroverkko on hajautettu informaatiota prosessoiva rakenne, jonka muoto on suunnattu graafi [Hecht - Nielsen].

Määritelmä 2. *Suunnattu graafi* G = (V,E) on pari, jossa V on ei-tyhjä äärellinen joukko *solmuja* ja E on joukko järjestettyjä solmupareja eli *särmiä*. Solmupari (u,v) *yhdistää* solmut u ja v. Särmä e = (u,v) on solmun u *lähtösärmä* ja solmun v *tulosärmä*. Solmun u *tuloaste* $|u|_{tulo}$ on solmuun tulevien särmien määrä; vastaavasti solmun u *lähtöaste* $|u|_{lähtö}$ on solmusta lähtevien särmien määrä. [Buckley and Harary]

3.1. Neuroverkon määritelmä

Neuroverkko (N) koostuu *solmuista* (node) ja *särmistä* (edge). Lisäksi jokaiselle särmälle (e) on määritelty *paino* (w) ja jokaiselle solmulle on määritelty *siirtofunktio* (f). Siirtofunktio muodostaa solmuun tulevista signaaleista lähtevän signaalin. Lähtösärmien signaalien arvot saadaan laskemalla solmun siirtofunktio.

Siirtofunktiona käytetään usein *signaalien* lineaarista summaa tai erityistä sigmoidifunktiota. Neuroverkon solmuja on kirjallisuudessa kutsuttu prosessoiviksi elementeiksi, neuroneiksi ja yksiköiksi. [Hecht-Nielsen, 23] Tässä työssä käytetään ainoastaan yksinkertaisia siirtofunktioita, jotka ovat muotoa $y = f(x^k)$, missä x^k on solmun tulosärmien määräämä k:n pituinen bittivektori ja f joko looginen “ja”- tai “tai”- operaatio. Täten verkon tulo hetkellä t määrää täysin verkon lähdön hetkellä t.

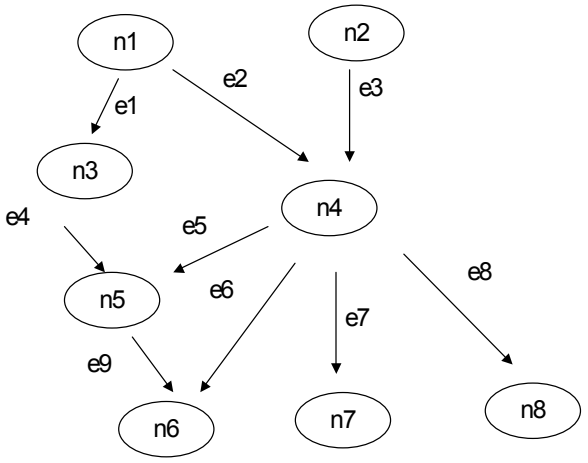
Määritelmä 3. Tavoitteena on neuroverkko, joka on 4-alkioinen järjestelmä M = (V, F, E, W), jossa [vert. Judd, 15]

- V on joukko solmuja $v \subseteq V$, $|V| = n$, $n > 0$.
  - Solmujoukko V voidaan jakaa kolmeen osajoukkoon
    - I on joukko verkon *tulosolmuja*, $I = \{ v_1,...,v_s \} \subseteq V$,
    - S on joukko verkon *sisäsolmuja*, $S = \{ v_{s+1},...,v_k \} \subseteq V$,
    - O on joukko verkon *lähtösolmuja*, $O = \{ v_{k+1},...,v_n \} \subseteq V$.
- F on verkon *konfiguraatio*, joka määrittelee joukon V solmujen siirtofunktiot, $F = \{f_1,f_2,\ldots,f_n\}$.

- E on joukko suunnattuja särmiä: $E \subseteq \{(v_i, v_j) : v_i, v_j \in V, i < j, v_i \in (I|S), v_j \in (S|O)\}$, $|E| = m$.
- W sisältää joukon E särmille määrätyt painot $W = \{w_1, w_2, \ldots, w_m\}$.

Joukon E rajoitteet varmistavat, että verkko on suunnattu graafi. Verkko ei saa sisältää syklejä.

Kuvassa 3 on esimerkinä 8 solmuinen neuroverkko A(2,3), ja jossa $I = \{ n_1, n_2\}$, $S = \{n_3, n_4, n_5\}$, $O = \{n_6, n_7, n_8\}$, $E = \{(n_1, n_3),(n_1, n_4),(n_2, n_4),(n_3, n_5),(n_4, n_5),(n_4, n_6),(n_4, n_7),(n_4, n_8),(n_5, n_6)\}$.



Kuva 3. Kahdeksansolmuinen neuroverkko

Kun kuvan 3 verkkoon määritellään konfiguraatio (F) ja särmien painot (W), voidaan verkon hyvyys mitata.

3.2. Neuroverkon opettaminen

Neuroverkon *opettaminen* tarkoittaa neuroverkolle tehtäviä peräkkäisiä operaatioita, joilla muutetaan verkon särmien painokertoimia. Verkon opettamiseen käytetään erityistä opetusalgoritmia. Opettamisen tavoitteena voi olla esimerkiksi saada verkko approksimoimaan tavoitefunktiota. Opettamisen alkaessa verkon särmät on alustettu joko satunnaisilla tai kasvumallin tuottamilla painoilla. Opetusalgoritmin rooli on pienempi, jos kasvumalli tuottaa särmille aloituspainot.

Neuroverkoille on olemassa useita vaihtoehtoisia opetusalgoritmeja, jotka voidaan luokitella neuroverkon saaman vasteen mukaan ohjattuun (supervised) opetukseen, ja vahvistavaan (graded) tai ohjaamattomaan oppimiseen (self-organization) [Kohonen]. Opetusalgoritmin lisäksi opettamisen onnistumiseen vaikuttavat myös verkon rakenne, särmien aloituspainot sekä opetusdata.

Verkon rakenne voidaan muodostaa siten, että kasvualgoritmi määrittelee osan särmistä kiinteiksi (fixed). Useimmat opetusalgoritmit eivät muokkaa verkon rakennetta. Sopiva rakenne on perinteisesti löydetty ad hoc –menetelmällä, sillä sopivan rakenteen löytäminen verkolle vaatii monimutkaisen algoritmin. Yleisesti on käytetty verkkoa, joka koostuu täysin toisiinsa kytketyistä tasoista (layers).

Opetusalgoritmi tavallaan muokkaa verkon topologiaa, kun se muuntaa särmän painon lähelle nollaa, jolloin solmujen välillä ei ole vuorovaikutusta. Solmujen välille jää kuitenkin vaikutusmahdollisuus, joka tulee erottaa verkon todellisesta rakenteesta, josta tämä vaikutusmahdollisuus puuttuu kokonaan. Särmän poistaminen pienentää opetusalgoritmin hakuavaruutta ja vähentää verkon koulutukseen tarvittavia resursseja.

Itseoppiva verkko tunnistaa sille annetusta syötteestä lainalaisuuksia. Verkon solmut kilpailevat keskenään siitä, mikä niistä edustaa syötettä parhaiten. Verkon koulutuksen avulla neuroverkko voi oppia luokittelemaan saamiaan syötteitä. [Kohonen]

Ohjatussa opettamisessa (supervised learning) neuroverkko saa santamansa vastauksen laadusta tietoa. Opetusalgoritmi perustuu yksinkertaisiin, solmujen lokaaleihin sääntöihin. Opetusalgoritmin kannalta on olennaista, annetaanko verkolle yksinkertainen oikein/väärin palaute vai kerrotaanko sille lisäksi oikea vastaus. Oppiminen ilman selvää palautetta on biologisesti lähinpänä todellisuutta oleva vaihtoehto.

Opetusta ajan hetkellä t merkitään $x^t = (s^t, o^t)$, jossa s^t on verkon syötevektori ja o^t palautesignaali tai “haluttu palaute”. Tätä voidaan kutsua *tapahtumaksi*. [Golden].

Yleensä ohjattu opettaminen tapahtuu seuraavan algoritmin mukaan, (ns. *perceptron* –algoritmi):

1. Olkoon t = 0. Verkko alustetaan satunnaisilla painoilla.
2. Valitaan opetusdatasta (satunnainen) tapahtuma.

3. Lasketaan palaute r(t) laskemalla signaalin kulku tulosärmistä sisäsolmujen kautta lähtösärmiin.
4. Muutetaan painoja alkaen lähtösärmistä päätyen tulosärmiin.
5. Tarkistetaan lopetuskriteeri. Verkon opettaminen lopetetaan tai palataan kohtaan 2.

Hieman parempi koulutusalgoritmi on backpropagation-algoritmi, joka käyttää oppimisnopeusparametria (learning rate). Yksittäisen solmun muunnosfunktio on esitelty alla. Kun α = oppimisnopeus ($1 >> \alpha > 0$) ja w on solmun tulojen painovektori, saadaan uusi painovektori w_{uusi}:

$$w_{uusi} = w_{vanha} - \alpha * (tavoite\_lähtösignaali - mitattu\_lähtösignaali)*F(w).$$

Neuroverkon opetustehtävän NP-täydellisyys tulee vastaan jo 2 solmun syvyisessä verkossa [Judd, 9]. Voimme opettaa paljon nopeammin miljoona viiden solmun verkkoa, kuin yhden syvän ja monia kytkentöjä sisältävän sata solmuisen verkon. Ei ole olemassa satunnaiselle neuroverkolle ja satunnaiselle solmufunktiolle käytettävää opetusalgoritmia, joka takaisi oppimistuloksen [Judd,9]. Ei siis voida taata, että neuroverkon opettaminen tuottaa toimivan verkon.

Neuroverkon rakenteen vaikutusta verkon oppimisnopeuteen voidaan kuvata seuraavasti [Boers and Kuiper, 18] :

- Verkko konvergoituu aina ja tulos on riippumaton satunnaisista asetuksista ja alkupainoista.
- Verkko konvergoituu niin nopeasti kuin mahdollista.
- Verkko ei osoita häiriöitä.
- Verkko yleistää.

Evolutiivisen algoritmin ajossa yksilöiden evaluointiin kulunut aika kertaantuu, joten yhteen yksilöön käytetty aika on minimoitava. Jotta koulutusalgoritmia kannattaa käyttää, on sen tuotettava geneettisiä operaatioita todennäköisemmin parannuksia verkon rakenteeseen. [Gruau, 72]. Evolutiivisen algoritmin avulla hakuavaruutta haravoidaan laajemmin kuin pelkän opetusalgoritmin avulla. Toisaalta opetusta käytetään nopeuttamaan evolutiivista algoritmia [Gruau, 71]. Opetusalgoritmi vie valtaosan prosessin ajasta, joten evolutiivisen algoritmin otosten määrä pienenee, eikä evolutiivinen algoritmi käy verkkoa yhtä laajasti läpi kuin ilman paikallista optimointia. Ainoastaan hyvin toimiva optimointi kannatta.

Prosessin nopeuttamiseksi voidaan käyttää myös ns. “Lamarckin evoluutiota” eli opetusalgoritmin muokkaamat painot voidaan siirtää takaisin osaksi evoluutiota. Tällöin koulutetusta neuroverkosta siirretään painojen muutokset genomiin, joita sitten käytetään seuraavan sukupolven neuroverkon aloituspainoina. Tämä on erittäin voimakas ominaisuus ja vaikuttaa ratkaisevasti prosessin kulkuun. Taulukossa 3 on esitelty opetuksen käytön mahdollisuudet.

Taulukko 3. Opetuksen vaikutus neuroverkkoon

Opetuksen tyyppi	Genotyypin muutos	Fenotyypin muutos
Ei opetusta	Ei	Ei
Baldwin (tyypillinen)	Ei	Kyllä
Lamarck	Kyllä	Kyllä

Liiallinen verkon opetus vaikuttaa heikentävästi verkon kykyyn yleistää. Tätä kutsutaan ns. ylioppimiseksi (over training). Oletetaan, että verkko opetetaan antamalla sille sisääntuloina kuvia henkilön kasvoista, ja verkon tehtävänä on tunnistaa tietyn henkilön kasvot. Verkko on ylikoulutettu, jos se tunnistaa kasvot vain kuvista, joita sille on näytetty koulutusvaiheessa. Koulutetun verkon tulisi tunnistaa kasvot myös uusista kuvista.

3.3. Neuroverkon hyvyyden mittaaminen

Hyvyysfunktio mittaa neuroverkon soveltuvuuden annettuun tehtävään. Tässä kohdassa esitellään neuroverkolle soveltuva hyvyysfunktio.

Neuroverkon toiminnallisuutta testataan joukolla tulo-lähtö -pareja. Tuloa voidaan kutsua myös ärsykkeeksi (stimulus) ja lähtöä reaktioksi (response), eli lyhyesti SR-pariksi, joka on ns. *tapahtuma*. Tapahtumajoukkoa kutsutaan yleensä *tehtäväksi* tai *valintadataksi*. Merkitään, että tulot σ ovat s:n pituisia vektoreita, $\sigma \subseteq \{0,1\}^s$, ja lähdöt ρ ovat r:n pituisia vektoreita, $\rho \subseteq \{0,1\}^r$.

Neuroverkon hyvyysfunktio voidaan kuvata minimifunktiolla $F(w) = \int_A |f(x) - G(x,w)|^2$ p(x)dV(x), jossa f(x) on tavoitefunktio, G(x,w) graafin toiminta painoilla w, ja p(x) painotettu todennäköisyysjakauma. Tavoitefunktion ja mitatun funktion erotuksen neliöstä lasketaan siis kertymä. Verkon tominnallisuudelle voidaan asettaa monia, jopa ristiriitaisia, tavoitteita. Yleensä tavoitteiden tärkeyttä painotetaan ja ne yhdistetään hyvyysluvuksi skalaarisella funktiolla.

Valintadatan lisäksi tarvitaan riippumatonta testidataa etsintäalgoritmin tuloksien testaamiseen. Etsintäalgoritmin tuottamaan verkon hyvyyteen ei voida käyttää opetuksessa käytettyä dataa, vaan tuloksien mittaamiseen tarvitaan

riippumatonta testidataa. Valintadataa käytetään simulaation aikana useita kertoja, joten yhteen verkkoon ei voida käyttää kovin laajaa otosta. Valintadataa ei voida myöskään soveltaa tuotetun neuroverkon hyvyyden mittaamiseen. Saamme neuroverkon toiminnasta liian myönteisen kuvan, ellei lopullisia tuloksia mitata erityisellä vertailudatalla. Poikkeuksena ovat ongelmat, joissa verkon hyvyys voidaan mitata kaikilla mahdollisilla tuloilla.

Neuroverkon toimiessa kaikissa sille annetuissa tehtävissä toivotulla tavalla hyvyyden mittaukseen käytetään muita mittareita kuin verkon toiminnallisuutta. Tavoitteeksi voidaan asettaa sekä verkon toiminnallisuus että verkon rakenteelliset ominaisuudet. Neuroverkon hyvyyteen vaikuttavat solmu- ja särmäjoukon koko tai verkon muoto. Verkon oppimisnopeus on usein tärkeä kriteeri tai esimerkiksi testipatterin täydellisen oppimisen todennäköisyys. Myös verkon yksinkertaisuus, kuten mallissa käytettyjen solmutyyppien vähyys, on sopiva kriteeri.

Evolutiivisen algoritmin osana myös muu populaatio voi vaikuttaa yksilön hyvyyteen. Verkon erilaisuus populaation muihin yksilöihin verrattuna voitaneen katsoa eduksi. Erilaisuus ei yleensä ole vahvuus sinänsä, vaan erilaisuudesta seuraava väite “neuroverkko osaa luokitella jotkut asiat paremmin kuin muut populaation yksilöt”.

Tässä työssä universaali approksimaattori eli evolutiivinen algoritmi etsii approksimoijaa eli neuroverkkoa. *Universaali approksimaattori* on laite, joka pyrkii löytämään määrätystä kuvauksesta $f:X \rightarrow Y$ mahdollisimman hyvän approksimaation, kun $X \subseteq \{0,1\}^n$ ja $Y \subseteq \{0,1\}^m$. Tässä työssä tämä laite on neuroverkko.

Kun neuroverkko opetusalgoritmin käytön jälkeen testataan, geneettinen algoritmi saa tietää, onko rakenne kehityskykyinen, vaikkei se aloituspainoilla toimisikaan. Hyvyysluku on sekä perityn että opitun käyttäytymisen mitta. Hyvyysluvun laskentatapa “tasoittaa” hakuavaruutta, jolloin piikin sijasta voimme löytää optimin myös kauempaa [Gruau, 73].

3.4. Biologinen hermoverkko

Tässä kohdassa käydään läpi kasvualgoritmin mallintamisen kannalta merkittäviä biologisen hermoverkon ominaisuuksia.

Biologinen neuroverkko koostuu soluista ja solujen välisestä vuorovaikutuksesta. Golden [Golden] luettelee useita tärkeitä neuroverkon ominaisuuksia, jotka ovat hermoverkoissa oleellisia, mutta eivät kasvualgoritmin kannalta merkittäviä:

1. Neuronit generoivat sähköimpulsseja, jotka välittyvät aksonia pitkin. Impulssien taajuutta voidaan karkeasti mallintaa neurosolun aktivaatiopotentiaalilla eli lähtösignaalin voimakkuudella. Toisaalta hermosolulla voi tulla useita tuloja, niin kutsututtuja dentriititejä, jotka vaikuttavat hermosolun aktivaatiopotentiaaliin.

2. Assosiatiivinen oppiminen. Useat tutkimukset ovat osoittaneet, että hermosolujen väliset yhteydet voivat muuttua, kun molemmat hermosolut ovat yhtä aikaa aktiivisia.

3. Suuri tila-avaruus. Jo yksittäinen hermosolu on hyvin monimutkainen mallintaa. Lisäksi hermosolukossa on useita hermosoluja, esimerkiksi ihmisellä on noin 10^{12} hermosolua, joilla on noin 10^{15} mahdollisesti muokattavaa kytkentää.

4. Hermosolujen paikallinen vuorovaikutus. Hermosolukolla on fyysiset rajoitteensa ja hermoverkkojen kytkennät rajoittuvat yleensä läheisiin soluihin. Viimeaikainen tutkimus viittaa myös solujen oppimisprosessin lokaalisuuteen.

5. Nopea tietojenkäsittely tapahtuu hitaiden neuronien avulla. Neuronit operoivat millisekunttien aikaskaaloilla; kuitenkin ihmisen kognitiiviset prosessit tapahtuvat sekunnin osissa, joten vuorovaikutusketjut eivät voi olla kovin pitkiä.

6. Hajautettu prosessointi ja kontrolli. Hermoverkostossa tapahtuu paljon rinnakkaisprosessointia.

7. Epälineaariset siirtofunktiot. Dentriittien vaikutus hermosolun aktivaatio-taajuuteen on usein mallinnettu erityisen sigmoidifunktion avulla.

8. Yhdenmukaisuus. Vaikka aivoissa on todettu useita erilaisia prosesseja, ovat solukon kaikki solut muodostuneet yhdestä kantasolusta ja yleisesti oletetaan, että samankaltaiset mekanismit vaikuttavat hermosolujen oppimisprosessissa kaikkialla aivoissa.

9. Hermosolukon toiminnallisuus sietää hyvin paikallisia epätarkkuuksia.

10. Hermoverkko on analogista tietojenkäsittelyä. Digitaalisilla prosessoreilla vastaavan toiminnallisuuden mallintaminen on resursseja suuresti tuhlaavaa.

Biologisella hermoverkolla on edellisten ominaisuuksien lisäksi merkittäviä ominaisuuksia, kuten:

a) Solukko on täysin kytketty, eli verkon jokaiseen solmuun tulee ainakin yksi särmä.

b) Koska verkko on eliöllä koko kasvun ajan toimiva hermosto, tulee verkon olla koko kasvun ajan täysin kytketty.

c) Hermosolukko kasvaa luomalla uusia soluja ja muuntamalla vanhoja. [Niemi et al.]

d) Aikuisessa yksilössä hermosolut eivät lisäänny. [Niemi et al.] Genomi määrää verkon rakenteen pääpiirteet, toisin sanoen opetusalgoritmi ei lisää uusia solmuja verkon rakenteeseen.

e) Mitä erilaistuneempi solu on, sitä huonommin se osaa jakaantua. [Niemi et al.] Verkon “tasainen” rakenne voitaneen varmistaa kasvumallin ominaisuudella, jossa erilaistumisaskelten määrä on pieni.

f) Hermosto toimii useiden erilaisten välittäjäaineiden avulla. Hermostossa on sekä ionikanavien kautta vaikuttavia nopeita välittäjäaineita että kanavien ulkopuolisia välittäjäaineita. Näiden kahden luokan nopeusero on noin 100-kertainen. Välittäjäaineiden vaikutusten erilaisuus mahdollistanee useilla erilaisilla aikaskaaloilla toimivat prosessit kuten esimerkiksi oppiminen ja toiminnallisuus. Sama välittäjäaine voi toimia sekä vahvistavana että heikentävänä riippuen sensorista.

Eliön kehityksessä on kaksi perustapahtumaa, solujen kasvu ja niiden tehtävien jako eli erilaistuminen. Biologista kasvua ohjaavat genomi, ravinto ja

hermoston käyttö. Biologisen hermosolun kasvu jaetaan neljään vaiheeseen [Kalat]:

- Jakautumisvaihe eli proliferaatio, jossa solujen lukumäärä lisääntyy. Valtaosa ihmisen neuroneista profiloituu ennen syntymää
- Solujen vaellus eli migraatio, jonka aikana hermoston rakenne muodostuu
- Erikoistuminen – aksonit erikoistuvat ensin ja sen jälkeen dentriitit. Erikoistuminen voi tapahtua jo solujen vaelluksen aikana (differation), jolloin solutyypit ja kytkentöjen voimakkuudet muodostuvat. Solujen erikoistuminen on peruuttamaton prosessi.
- Nopea tiedon kulku mahdollistuu tukisolujen ympäröidessä hermosolun (myelination). Tukisolukon muodostuminen alkaa ensin selkärangasta, jatkuu sitten pikkuaivoihin ja väliaivoihin sekä lopuksi isoihin aivoihin. Opetusalgoritmi voisi mallintaa tätä siten, että ulkomaailman kontrollereina toimivat uloimman tason mekanismit täsmentyvät ensin minkä jälkeen muodostuvat muut kontrollointimekanismit.

Osa neuroneista kasvaa ja matkustaa päämääräänsä jättäen aksonin jälkeensä kuin pitkän hännän. Neuroni kytkeytyy lihaksiin satunnaisesti, ja lähettävät sarjan erilaisiin lihaksiin sopivia viestejä. Aksonilla on aluksi monia vaihtoehtoisia kytkentöjä, mutta ajan myötä se hylkii osaa kykennöistä ja vahvistaa toisia. [Kalat]

Keskushermosto tuottaa enemmän neuroneita kuin se tarvitsee. Ylimääräiset neuronit mahdollistavat solujen valikoitumisen. Ylituotanto mahdollistaa “survival of the fittest” –periaatteen, esimerkiksi 85% sympaattisen neuroston soluista kuolee. [Kalat]

Tukisolukon merkityksen ja tehtävän ymmärtäminen mahdollistanee biologisesti orientoituneen neuroverkon kasvualgoritmin mallintamisen. Koska tukisolukko on dynaamisempi kuin neurosolukko, on se mahdollisesti tärkeässä asemassa kasvu- ja oppimistapahtumassa.

4. Neuroverkon kasvattaminen

4.1. Kasvualgoritmin vaihtoehdot

Koulutuksen avulla neuroverkosta muokataan tavoitefunktion approksimaattori. Ennen koulutusta on kuitenkin valittava verkon topologia. Verkon topologian valintaan ei ole mitään nyrkkisääntöjä tarkempia ohjeita. Vaikka emme tiedä tarvitsemamme verkon optimaalista rakennetta, voidaan topologia valita kokemuksen perusteella, kuten Boers ja Kuiper:

- Lisää sisäsolmuja yhtä paljon kuin tulo- ja lähtösolmuja on keskimäärin.
- Jos verkko ei opetettaessa konvergoidu, lisää sisäsolmuja.
- Jos verkko ei vieläkään konvergoidu, lisää tasoja.
- Jos verkko ei yleistä hyvin, vähennä sisäsolmuja. [Boers and Kuiper]

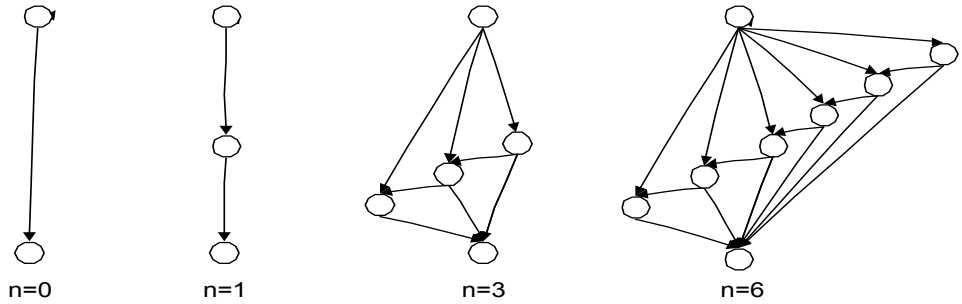
Verkon topologian ollessa ennalta tiedossa, tai jos tyydymme täysin kytkettyyn verkkoon, soveltuu evolutiivinen algoritmi neuroverkon etsintään. Verkon painot voidaan koodata vektoriin, jota evolutiivinen algoritmi käyttää genomina. Geneettisten operaatioiden toteutus on vaivatonta, kun vektorin pituus on etukäteen kiinnitetty. Erillistä muuntoalgoritmia genomista neuroverkoksi ei tarvita vaan voidaan käyttää *suorakoodausta* (direct encoding).

Suorakoodauksessa verkon esitystapa ja ilmentymä eivät eroa toisistaan ja suorakoodaus soveltuu vain kiinteän kokoisille rakenteille. Suorakoodauksessa verkon maksimikoko ei muutu, ellei verkkoa esittävän merkkijonon pituus muutu. Suorakoodaus ei ole biologisesti orientoitunut malli: eliöt eivät synny aikuisten yksilöiden osista kokoamalla. Biologisesti mielekkäitä geneettisiä operaatioita voidaan kohdistaa vain genomiin, ei genomin ilmentymään.

Yksinkertaista heuristiikkaa voidaan käyttää evolutiivisen algoritmin kanssa, mutta se soveltuu parhaiten avuksi suoraviivaiseen kokeiluun, jossa verkkoa opetetaan ja uutta rakennetta kokeillaan, jos opetettu verkko ei toimi toivotusti eli approksimoi funktiota riittävän hyvin.

Parametrisoitu koodaus [Gruau, 3] soveltuu rajoitettuihin erityisongelmiin. Parametrisoidussa koodauksessa genotyyppi määrittelee joukon funktion parametreja. Funktio määrittelee verkon rakenteen jollain kasvualgoritmista poikkeavalla tavalla. Tarkastellaan esimerkkinä seuraavanlaista funktiota, joka

määrittelee kuvan 4 mukaisen verkkoperheen. Funktio saa parametrinaan verkon sisäsolmujen määrän.



Kuva 4. Esimerkki parametrisoidun kasvumallin tuottamasta verkosta arvoilla 0,1,3 ja 6

Jos haemme yhteen ongelmaan soveltuvaa approksimaattoria, sekä yksinkertainen heuristiikka että parametrisoitu koodaus ovat hyviä vaihtoehtoja, mutta jos tavoitteena on kehittää työkalu automaattiseen neuroverkkojen generointiin, on kasvualgoritmi paras vaihtoehto.

4.2. Kasvualgoritmi

Kasvualgoritmi on kielioppi, jonka avulla kasvatetaan evolutiivisen algoritmin käyttämästä tietorakenteesta neuroverkko. Kasvualgoritmia käytetään, koska risteytysoperaation soveltaminen täysikasvuisille neuroverkoille on ongelmallista.

Evolutiivisen algoritmin käyttämästä tietorakenteesta, $x \in \Omega$, kasvualgoritmi muodostaa verkon G, joka tässä työssä on syklitön suunnattu graafi. Kasvualgoritmi, joka lukee syötteenään merkkijonon, voi muunnnella tuloksena syntyvän verkon kokoa merkkijonon sisältämän informaation perusteella [Kitano 1990].

Vaikka kasvualgoritmi määrittelee verkon topologian, sen ei tarvitse tuottaa valmista funktion approksimoivaa verkkoa. Kasvumallin vastuulla on tavallisesti solmujen toimintafunktioiden määrittely sekä neuroverkon kytkentöjen alustus. Kytkentöjen säätäminen jätetään yleensä erillisen *opetusalgoritmin* vastuulle.

Ympäristön vaikutus neuroverkon kasvuun on tärkeä osaongelma pyrittäessä biologisesti orientoituneeseen kasvualgoritmiin. Biologisesti orientoituneessa kasvumallissa solun jakaantuminen on keskeinen metafora, ja kasvualgoritmi voidaan tarkentaa solunjakaantumismalliksi.

Hyvältä kasvualgoritmilta vaaditaan useita ominaisuuksia, jotta muuntaminen genomista verkoksi sujuisi järkevällä tavalla:

Täydellisyys: Kaikki mahdolliset neuroverkon arkkitehtuurit on voitava koodata, jotta etsintäalgoritmi voi löytää mahdollisen ratkaisun. [Gruau, 17]

Tiiviys: Kasvualgoritmin on hyvä olla topologisesti kompakti. Mitä pienempiä genomit ovat, sitä pienempi hakuavaruus on, ja sitä vähemmän työtä etsintäalgoritmin täytyy tehdä. [Gruau, 17] Olen asiasta hieman toista mieltä (ks. 6.1).

Sulkeutuvuus: Kasvualgoritmi kehittää vain ongelman kannalta järkeviä rakenteita. Se tuottaa joko syklittömiä tai rekursiivisia arkkitehtuureja. Kun etsintäalgoritmin tuottamat genotyypit tuottavat mielenkiintoisia neuroverkkoja, ei evaluointeja tarvitse hylätä. Tämä lisää tehokkuutta. [Gruau, 19]

Modulaarisuus: Neuroverkko on voitava jakaa aliverkkoihin ja aliverkkojen kopioihin. Evolutiivinen algoritmi voi näin löytää oikeat rakennuspalikat kuvaamaan aliverkkoja ja aliverkkojen välisiä suhteita. Arkkitehtuurin säännöllisyydet tekevät verkon selväksi ja tulkinnalliseksi. Koodi on myös tiiviimpi, kun toistettavat rakenteet kuvataan vain kertaalleen. Modulaarisuus voi myös parantaa neuroverkkojen tehokkuutta [Boers and Kuiper, 2; Gruau, 22].

Skaalattavuus: Solukoodaus koodaa ongelmakohtaisen neuroverkkoperheen. Ongelma ratkaistaan ensin kahden tai kolmen kokoiselle tapaukselle. Koska koodi vastaavalla neuroverkolla on sama, vain koon ilmaiseva parametri muuttuu. [Gruau, 28] Gruau on saavuttanut hyviä tuloksia helposti skaalautuvissa ongelmissa. Hyvä esimerkki on pariteettiongelma, jossa neuroverkko laskee, onko verkon tulossa parillinen vai pariton määrä ykkösiä. Tällöin ongelman luonne pysyy erilaisilla tulosolmujen määrillä samankaltaisena.

Abstrahoitavuus: Massiivinen hakuavaruus voidaan muuntaa pienempiin osaongelmien hakuavaruuksiin muuntamalla ohjelma ensin aliohjelmiksi. Se auttaa etsintäalgoritmia löytämään neuroverkkoja ongelmille, joita voidaan ratkaista parametrisoidulla koodauksella. [Gruau, 29]

Ilmaisuvoima: Kasvumallilla on kyky tiivistää tietoa. [Gruau, 27]

Kielioppi: Mahdollisuus määritellä kasvumalli kieliopin avulla on kaikkien muiden ominaisuuksien perusta. [Gruau, 29]

4.3. Biologinen L-järjestelmä

Lindemayerin järjestelmä, eli L-järjestelmä, on biologista kasvua kuvaava malli. L- järjestelmää voidaan soveltaa merkkijonojen sekä graafien kasvattamiseen. L- järjestelmät eroavat tavanomaisesta kieliopista (Chomskyn merkityksessä), kahdella tavalla: 1) jokaisella aika-askeleella merkkijonon kaikki osat muuntuvat samanaikaisesti, 2) terminaaleja ja apumerkkejä ei erotella toisistaan. [Rozenberg and Salomaa]

Koska L-järjestelmiä voidaan soveltaa sekä merkkijonoille että graafille, voidaan muunnos merkkijonosta graafiksi tehdä joko ennen kasvualgoritmin evaluointia tai kasvun jälkeen. Voimme siis kasvattaa joko merkkijonoja tai verkkoja. Jos biologiaa käytetään mallina, on syytä käyttää osa genomista idun määrittelemiseen ja käyttää genomia *graafin* kasvun ohjaamiseen. Boers ja Kuiper [Boers and Kuiper] käyttävät merkkijonoja käsittelevää L-järjestelmää.

L-järjestelmät ovat diskreettejä matemaattisia rakenteita kolmessa merkityksessä:

- tilasiirtymät on määritelty diskreeteille soluille,
- kukin solu esitetään yhtenä äärellisenä tilojen joukkona,
- tilasiirtymät toteutetaan diskreetteinä aikasiirtyminä.

Solun toiminnan ohjauksen mallintaminen vain geenien avulla tekee mallista yksinkertaisemman. Biologiassa geenit kontrolloivat monisoluisen eliön kehitystä yhdessä solunesteen ja solukalvon kanssa. Vaikka solun kaikki komponentit ovat jossain muodossa koodattuna DNA:han, eivät kaikki solun prosessit ole sen suorassa kontrollissa. DNA:han on hyvin stabiili rakenne, joka ei reagoi yksilötasolla tapahtuneisiin muutoksiin. Biologisesti mielenkiintoisen mallin tulee ottaa huomioon myös kasvun rakenteellinen kontrolli, eli L-järjestelmän on oltava ns. kontekstinen.

Geenien *aktiivisuudella* on tärkeä osuus eliön kehityksen kontrolloinnissa. Kun solu jakaantuu, ns. *tytärsolujen* aktiiviset geenit voivat olla samoja kuin äitisoluissa, tai ne voivat erillaisia kahden tytärsolun välillä ja/tai äitisolun välillä. Kehityksen kontrollointi voidaan jaotella geenien aktiivisuuteen solulinjassa (lineages) ja solujen vuorovaikutuksena tapahtuvaan kontrolliin. [Latchman, 129]

L-järjestelmä, jossa kehitys on *kontekstitonta,* ainoastaan solulinjat eli geenien aktiivisuus, ohjaa yksilön kehitystä. Tätä voidaan mallintaa “merkkijono-D0L-

järjestelmällä", jossa ´merkkijono´ tarkoittaa solujen tilaa ja ´D0L-järjestelmä´ tarkoittaa solujen välisten vuorovaikutusten puuttumista. Solujen tiloja ei tässä erotella aktiivisten geenien, solunesteen ja solujen seinämien tiloihin.

D0L-järjestelmä on kolmikko (Σ,P,α), jossa Σ on solujen tiloja merkitsevä äärellinen aakkosto, P on äärellinen joukko tilojen siirtymäsääntöjä (productions) ja α on aksiooma eli aloitusmerkkijono. [Rozenberg and Salomaa]

Tilojen siirtymäsäännöt ovat muotoa $a \rightarrow w$, missä $a \in \Sigma$ ja $w \in \Sigma^*$; sääntö $a \rightarrow w$ merkitsee siirtymää solun tilasta a merkkijonoon w solujen tiloja. Säännöt mahdollistavat seuraavat siirtymät:

- solu säilyttää tilansa ($a \rightarrow a$),
- solu muuttaa tilaansa ($a \rightarrow b$),
- solu jakaantuu annetussa järjestyksessä useisiin määrätyissä tiloissa oleviin soluihin ($a \rightarrow a_1a_2 \ldots a_n$),
- solu katoaa ($a \rightarrow \lambda$, missä λ on tyhjä merkkijono).

Derivaatioaskel merkkijonolle w sisältää kunkin w:n merkin korvaamisen uudella merkkijonolla. *Johtaminen* (derivation) sisältää mielivaltaisen määrän toisiaan seuraavia askelia. Jos jokaiselle merkille on vain yksi sääntö, syntyvä merkkijono on yksiselitteinen. Tällöin järjestelmää kutsutaan deterministiseksi 0L-järjestelmäksi (D0L-järjestelmä). Kaikki järjestelmällä tuotettavat merkkijonot muodostavat järjestelmän määräämän kielen. [Rozenberg and Salomaa]

Yksilönkehitys on biologinen prosessi, johon liittyy monia mallintamisen kannalta mielenkiintoisia tekijöitä. Solulla on fyysinen ulottuvuutensa, jota rajoittaa vain pieni määrä muita soluja. Naapurisoluilla on tärkeä vaikutus solun kasvuun. Lisäksi solujen tilojen syklinen muuttuminen eli solujen sisäinen kello vaikuttanee kasvuun. Elävä organismi muodostuu soluista, joista jokainen sisältää saman kromosomiston, DNA:n, joten geneettinen koodi on sama yksilön kaikilla soluilla.

5. Neuroverkon kasvattaminen evolutiivisen algoritmin avulla

5.1. Boersin ja Kuiperin malli

Boers ja Kuiper [1992] käyttävät kasvumallina L-järjestelmää, joka tulkitaan kiinteän pituisesta bittijonosta. L-järjestelmän sääntöjen avulla kasvatetaan merkkijonoa kunnes muutaman iteroinnin jälkeen merkkijonosta tulkitaan neuroverkon rakenne. L-järjestelmän avulla voidaan kasvattaa eri suuruisia verkkoja tarpeen mukaan. Genomista saadaan neuroverkko seuraavasti:

- geneettisen algoritmin genomista (bittijono) muodostetaan muunnostaulukon (taulukko 4) ja erityisen muunnosalgoritmin (kuva 5) avulla joukko L-järjestelmän sääntöjä (taulukko 5),
- saadun L-järjestelmän avulla *aksioomasta* kasvatetaan neuroverkon kuvaava merkkijono (taulukko 6).

Boersin ja Kuiperin käyttämä aakkosto on S = {A-H, 1-5, [,]} ∪ {,} jossa A-H merkitsevät solmujen tyyppejä, 1-5 merkitsevät hyppyjä ja hakasulut rajaavat moduuleja. He ovat valinneet hyppyjen maksimipituuden ja solmujen tyyppien määrän malliin satunnaisesti. [Boers and Kuiper, 52]

Taulukko 4. Boersin ja Kuiperin käyttämä käännös taulukko bittijonosta L-järjestelmän sääntöjen merkeiksi. Esimerkiksi 011110 tulkitaan merkiksi ']' A = 01, B = 11, C = 10.

A	B 00	01	10	11	C
00	3	[	D	]	00
	3	[	D	]	01
	*	[	2	2	10
	*	[	2	5	11
01	*	1	E	]	00
	*	1	E	]	01
	*	1	F	]	10
	*	1	F	]	11
10	2	A	G	[	00
	2	A	G	[	01
	2	A	H	]	10
	4	A	H	]	11
11	,	B	*	C	00
	,	B	*	C	01
	,	B	[	C	10
	,	B	[	C	11

Muunnos genomina toimivasta bittijonosta merkkijonoksi suoritetaan kuvan 5 algoritmin mukaisesti. [Boers and Kuiper, 55]

1. Aloita satunnaisesta kohdasta tai määrätyn katkomerkin (*) kohdalta. Aloita lukemalla säännön a vasen konteksti.
2. Lue kuusi bittiä kerrallaan. Hae vastaava merkki taulukon 5 mukaan.
3. Jos merkki ei ole katkomerkki (*), lisää se sääntöön.
4. Lisää luettu säännön osa (voi olla tyhjä) kyseiseen paikkaan sääntöä (V,E,O,S). Etene seuraavaan säännön osaan. Jos osa oli S, aloita seuraavan säännön lukeminen.
5. Toista kohtia 2 - 4 kunnes kromosomin kaikki bitit on luettu. Poista kesken jäänyt sääntö.

Kuva 5. Genomin tulkinta L-järjestelmäksi Boersin ja Kuiperin mallin mukaan [Boers and Kuiper, 55]

Kun sääntöjoukko on generoitu, siitä poistetaan kelvottomat säännöt ja ylimääräiset hakasulut sekä tyhjät '[ ]' moduulit [Boers and Kuiper, 53].

Taulukko 5. Bittijonosta [10000001111000001] saadut säännöt.

	Vasen konteksti (V)	Edeltäjä	Oikea konteksti	Seuraaja
1:		A		BBB
2:		B	B	[C,D]
3:	C	B		C
4:		D		C
5:		D	D	C1

Säännössä jokainen kirjain kuvastaa yhtä moduulia. Peräkkäiset moduulit kytketään automaattisesti yhteen. Esimerkiksi BBB muodostaa kuvassa 6 kohdan 2 mukaisen verkon. Pilkulla erotettujen kirjainten välille ei muodosteta kytkentää. Yksinkertaisin moduuli on solmu. Säännön vasen puoli voi sisältää vain täydellisiä moduleja tai solmuja. Tästä syystä vasempien ja oikeiden hakasulkujen tulee olla yhtä suuria ja oikeassa järjestyksessä. Moduuli ovat täysin kytkettyjä. Numerot merkitsevät hyppymerkkiä. Hypyt mahdollistavat kytkentöjen luomisen muidenkin kuin peräkkäisten moduulien välille. Esimerkiksi A2 merkitsee, että moduuli A on seuraavan moduulin lisäksi kytketty kolmen kirjaimen päässä sijaitsevaan moduuliin.

Koska mallissa moduulit ovat aina suunnattuja verkkoja, kuuluu kuhunkin moduuliin joukko solmuja, ns. tulosolmuja, joihin ei tule särmiä moduulin muista

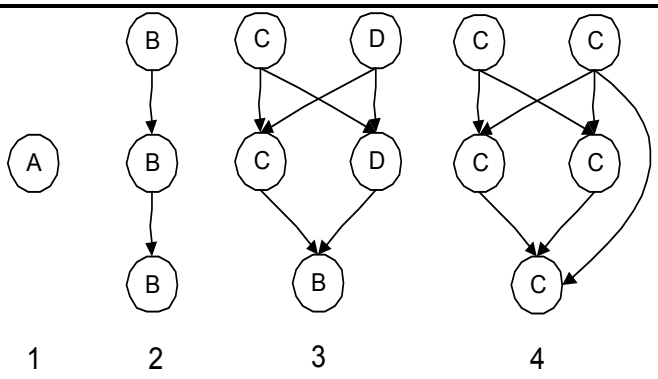
solmuista. Vastaavasti lähtösolmuja ovat kaikki ne solmut, joista ei lähde särmiä moduulin muihin solmuihin. [Boers and Kuiper, 47] Moduuliin tulevat särmät liitetään aina tulosolmuihin ja vastavaasti moduulista lähtevät särmät on liitetty moduulin lähtösolmuihin. Katkomerkit (*) vastaavat RNA:n transkriptiossa käytettyjä start- ja stop-merkkejä.

Luonnossa transkriptio merkitsee toimintoa, jossa DNA:sta tuotetaan RNA-ketjuja, joita tässä työssä kutsutaan säännöiksi. Sääntöjoukon valmistuttua aksioomasta johdetaan sääntöjoukon avulla merkkijono (taulukko 5), josta tulkitaan modulaarinen neuroverkko (kuva 6).

Taulukko 6. Taulukossa 5 esitetyn sääntöjoukon soveltaminen.

Iteraatio	
1	A
2	BBB
3	[C,D][C,D]C
4	[C,C1][C,C]C

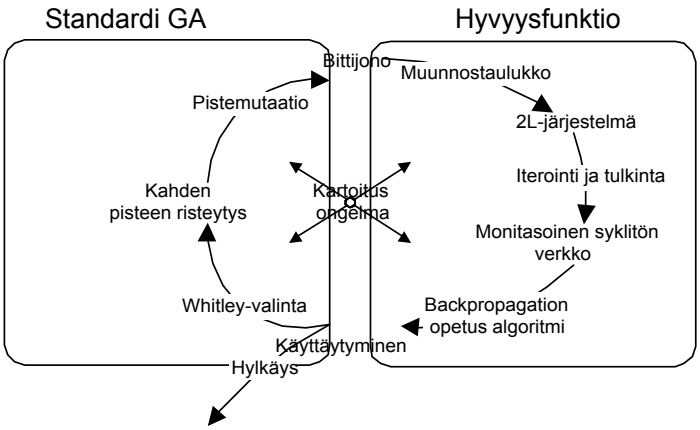
Kuvassa 7 esitellään neuroverkon topologian muuttuminen verkon kasvaessa. Peräkkäiset solmut kytketään automaattisesti yhteen. Hakasuluilla erotettujen moduulien välillä ei ole muita yhteyksiä kuin lähtösolmujen ja tulosolmujen väliset yhteydet.



Kuva 6. Boersin ja Kuiperin kasvumalin neuroverkko iteraatioilla 1 – 4.

Boersin ja Kuiperin malli määrittelee vain verkon topologian eikä alusta särmiä. Neuroverkon opetukseen käytetään backpropagation-opetusalgoritmia. Tämä "peruuttava" opetusalgoritmi tarvitsee toimiakseen syklittömän verkon. Prosessi on kokonaisuudessaan esitelty kuvassa 7.

Kuten kuvasta 7 nähdään, käyttävät Boers ja Kuiper geneettistä algoritmia tyypillisessä muodossa. Bittijononoon he soveltavat kahden pisteen risteytystä ja pistemutaatioita.



Kuva 7. Boersin ja Kuiperin malli

Prosessiin liittynee monia ongelmia: backpropagation-opetusalgoritmi on hyvin aikaavievä, ja merkkijonon tulkinta neuroverkoksi on suhteellisen keinotekoinen. Mielenkiintoisinta antia mallissa on bittijonon moninkertainen tulkinta, jossa lyhyestä bittijonosta saadaan useita L-järjestelmän sääntöjä. Lienee biologista realismia, että myös DNA:n tulkinta voi alkaa monista eri kohdista.

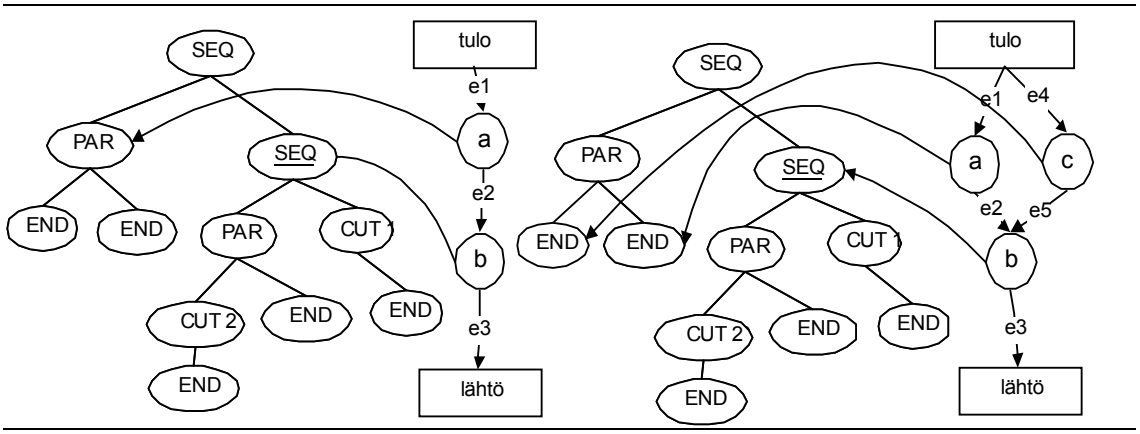
5.2. Gruaun malli

Gruau [1994] käyttää *solukoodausta* modulaaristen neuroverkkojen luontiin. Genomi on Gruaun mallissa erityinen kielioppipuu (grammar tree, ks. kuva 8), joka on evolutiiviselle algorimille sopiva tietorakenne. Kielioppipuun solmut on merkitty ohjelmasymboleilla [Gruau, 7]

Gruaun mallissa neuroverkko kasvaa solmusta verkoksi ollen koko kasvun ajan toimiva neuroverkko. Neuroverkon kaikilla soluilla on *osoitin* vastaavaan kielioppipuun solmuun. Solu lukee kielioppipuun solmusta tarvitsemansa symbolin. Symboli esittää ohjeet solun seuraavalle *askeleelle*. Solu voi joko jakaantua tai muokata kytkentöjään. [Gruau, 7] Kuva 8 havainnollistaa Gruaun kasvumallin käyttöä symmetriaongelman ratkaisemiseen.

Gruau kokeilee useita erilaisia kielioppeja. Esimerkkinä esitellään yksinkertainen versio Gruaun täydelliseksi osoittamasta kieliopista, jonka avulla voidaan kasvattaa kaikki mahdolliset verkon topologiat. Kielioppi L on puu, jonka solmuina ovat merkit {END | PAR | CUT}. Verkko kasvatetaan joko syklisestä tai syklittömästä alkutilanteesta. Verkon solmuilla on osoitin kielioppipuun merkkeihin, joiden mukaan solmu joko jakaantuu tai siitä poistetaan särmä:

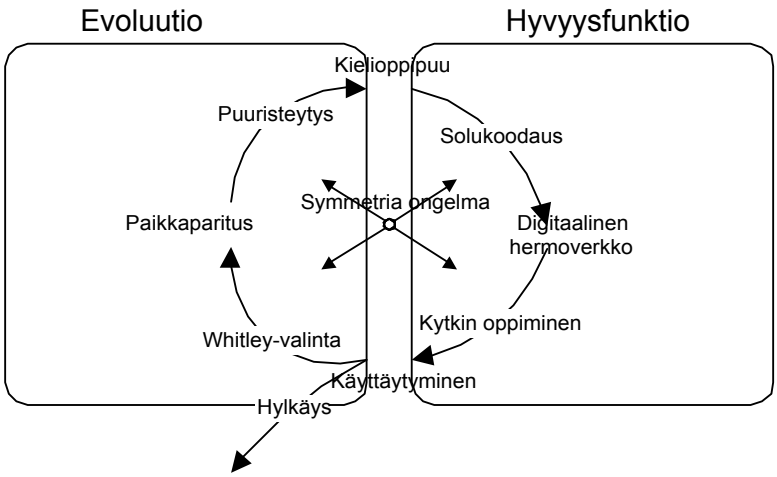
- CYC tai ACYC merkitsevät kahta erilaista aksioomaa, joista kasvaa joko suunnattu verkko (CYC) tai suuntaamaton verkko (ASYC),
- PAR jakaa solun, (ks. kuva 8)
- END lopettaa solun aktiviteetin,
- CUT saa parametrina kokonaisluvun k ja merkitsee k:nnen tulosärmän poistoa [Gruau, 16].



Kuva 8. Gruaun mallin par-komennon soveltaminen 'a'- solmuun

Solukoodaukseen voidaan liittää myös särmien painojen koodaus, joka toimii kuten CUT-merkki, mutta ei poista särmää vaan muuntaa särmän painoa. Rekursiivisia elementtejä verkkoon saadaan, kun END-merkin paikalla käytetään JMP-merkkiä. JMP-merkin kohdalla solun osoitin hyppää juureen, jolloin solmun tilalle kasvaa aliverkko [Gruau, 24].

Solukoodausen etuna on, että opetuksen aiheuttamien särmien painojen muutokset voidaan koodata takaisin kielioppipuuhun [Gruau, 71].



Kuva 9. Gruaun malli

Gruau käyttää solukoodausta vain digitaalisten verkkojen luontiin, mutta se sopinee kaikkien neuroverkkojen luontiin. Malli on kokonaisuudesssaan esitetty kuvassa 9. Gruau todistaa solukoodaukselle teoreettisia ominaisuuksia, ja hänen mielestään “solukoodaus määrittelee neuroverkon elegantilla ja kompaktilla tavalla”. Solukoodaus yhdessä geneettisen algoritmin kanssa soveltuu hyvin skaalautuviin ongelmiin, kuten pariteettiongelmaan. Pariteettiongelman ratkaisevassa verkossa sama rakenne toistuu useita kertoja, kun XOR-portti toistetaan jokaiselle verkon tulolle.

5.3. Dave-kasvumalli

Käytännön simulaatioita (7.1) varten on toteutettu kasvumalli, jonka ainoana tavoitteena on ollut yksinkertaisuus. Kasvumalli, jolle on annettu nimi ”Dave”, mahdollistaa kaikki verkon rakenteet. Se muistuttaa Gruaun [Gruau] mallia sikäli, että verkon jokaiseen solmuun liittyy yksi sääntö. Dave-kasvumallissa myös jokaisella särmällä on kasvua ohjaava sääntö, joka määrittelee särmän käyttäytymisen solmun jakaantuessa.

Dave-kasvumallia voidaan käyttää kun tavoitteena on graafi, jonka muodosta tunnetaan tulo- ja lähtösolmujen lukumäärä. Näin muodostetaan erityinen *alkutila* tai *aksioomagraafi*, josta graafin kasvattaminen aloitetaan. Aksioomagraafi sisältää yhden sisäsolmun. Aksioomagraafin sisäsolmuun tulee särmä jokaisestan tulosolmusta, ja sisäsolmusta lähtee särmä jokaiseen lähtösolmuun. Jokaisella aksioomagraafin särmällä ja sisäsolmulla on osoitin kasvumallin sääntöön siten, että solmu saa säännön 0, ensimmäinen tulosärmä seuraavan säännön jne. kunnes kaikilla alkutilan särmillä on sääntö. Mallin alkutilasta kasvatetaan sääntöjen avulla graafi, jonka jokaisella särmällä ja solmulla on ensimmäisen käytetyn säännön määräämä tyyppi.

Kasvumallia esittää kaksikko [aksioomagraafi, sääntö*]. Kasvumallin säännön ominaisuudet on esitelty taulukossa 7.

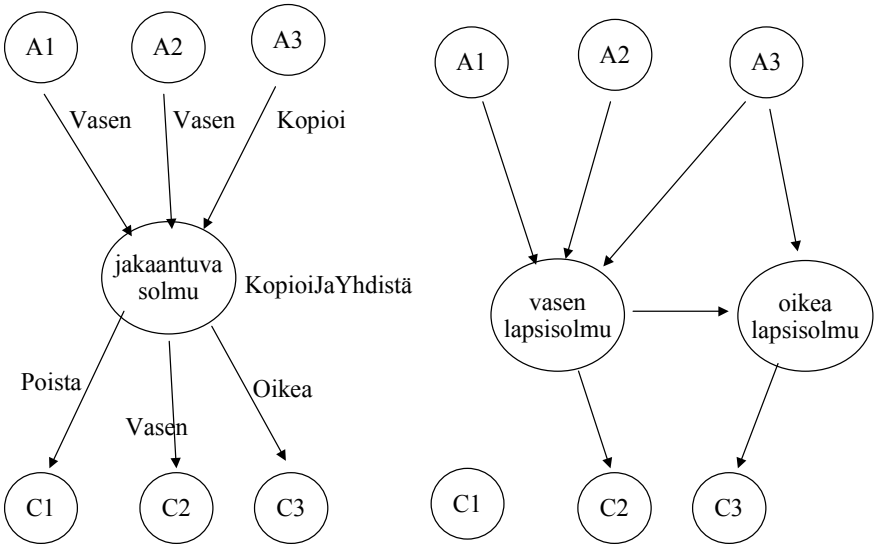
Taulukko 7. Kasvumallin sääntö

Säännön ominaisuus	Solmu	Särmä
Tyyppi	Solmun funktionaalinen tyyppi: ja, tai.	Särmän paino: negaatio, ei muutosta.
Vasen lapsi	Vasemman lapsisolmun säännön indeksi.	Vasemman lapsisärmän säännön indeksi.
Oikea lapsi	Oikean lapsisolmun säännön indeksi.	Oikean lapsisärmän säännön indeksi.
Uuden särmän sääntö	Vasemman ja oikean solmun väliin luodun särmän säännön indeksi.	-
Kopiotyyppi	Jakaantuuko solmu, ja tuleeko solmujen väliin särmä: Kopioi, KopioiJaYhdistä, Pysähdy.	Kopioidaanko särmä solmun jakaantuessa: Poista, Kopioi, Vasen, Oikea.

Solmun jakaantumisen yhteydessä jakaantuvat myös solmun tulo- ja lähtösärmät. Solmun jakaantuessa voi syntyneiden solmujen väliin syntyä uusi särmä, kuten kuvassa 10 tapahtuu. Solmu jakaantuu ainoastaan jos verkko säilyy jakaantumisen tuloksena ”täysin” kytkettynä, eli verkon jokaisen sisäsolmun tulo-

ja lähtöaste, jokaisen tulosolmun lähtöaste, ja jokaisen lähtösolmun tuloaste, on vähintään yksi. Esimerkiksi kuvan 10 mukaisessa solmun jakaantumisessa, jossa jakaantuvan solmun ja C1-solmun välinen särmä poistuu, on solmun C1 tuloaste oltava vähintään kaksi, muuten solmun jakaantumista ei sallita.

Jos solmun tyyppejä tarvitaan eri määrä kuin särmän tyyppejä, käytetään jakojäännöstä.



Kuva 10. Solmu jakaantuu Dave-kasvumallin mukaan

Yhden kasvumallin *iteraation* aikana kysytään jokaiselta solmulta, voidaanko se jakaa. Solmun jakaantuessa niin solmun kuin siihen kytkettyjen särmienkin säännöt vaihtuvat. Jos sisäsolmu ei onnistu jakaantumaan, sen osoitin muutetaan osoittamaan säännön vasemman lapsen indeksin mukaiseen sääntöön. Solmu voi siis olla iteraation jakaantumatta ja jakaantua seuraavalla iteraatiolla. Solmut jakaantuvat kunnes maksimimäärä solmuja ylittyy, jos yhtään solmua ei jaettu iteraation aikana tai sallittujen iteraatioiden lukumäärä täyttyy.

Solmutyypit ja särmätyypit vastaavat mallin muunneltavasta osuudesta. Tavoitteena oleva neuroverkko määrittelee tyyppien lukumäärän ja semantiikan. Sääntöjen on tarkoitus vastata biologista solun tilaa.

Dave-kasvumallia voidaan muunnella esimerkiksi sallimalla verkon ”rikkoutuminen” kasvun aikana tai erottaa solmujen ja särmien säännöt omiksi ryhmikseen. Mallin muunnoksien toimivuutta ei ole toistaiseksi tutkittu.

Kasvumallia käytetään esimerkkisimulaatiossa osana etsintäalgoritmin hyvyysfunktiota (kohta 7.1).

6. Biologisesti orientoituneen evolutiivisen laskennan kehittäminen

Tämän luvun tavoitteena on biologisia mekanismeja mallintamalla kehittää evolutiivista laskentaa.

6.1. Genomin rakenteellisten ominaisuuksien mallintaminen

Biologisen solun genomi (eli perimä) eroaa perinteisen geneettisen algoritmin käyttämästä merkkijonosta siten, että biologinen genomi sisältää runsaasti informaatiota, joka ei vaikuta solun toimintaan. Määrittelen genomin sisältämän informaation kahteen luokkaan seuraavasti:

Määritelmä 4. *Funktionaalinen informaatio* vaikuttaa eliön toimintaan.

Määritelmä 5. *Rakenteellinen informaatio* on geneettisten operaatioiden toimintaa ohjaavaa informaatiota.

Genomin funktionaalisen informaation perusteella johdetaan eliön fenotyyppi, josta saadaan hyvyysfuktion palauttama hyvyysluku. Perinteinen geneettisen algoritmin genomi sisältää ainoastaan funktionaalista informaatiota. On kuitenkin biologista realismia, että genomin tulee sisältää sekä funktionaalista informaatiota että rakenteellista informaatiota. Genomin rakenteellinen informaatio määrittelee, kuinka geneettiset operaatiot muuttavat genomia. Rakentellinen informaatio ei poista prosessista satunnaisuutta, mutta vähentää sitä mutaatioiden ja risteytysten toimintaan vaikuttamalla. Funktionaalisen ja rakenteellisen informaation ero on biologisesti epätarkka, mutta evolutiivisessa laskennassa käytettynä se on riittävän tarkka.

Seuraavaksi listataan biologisia tosiasioita, joiden katson olevan osa genomin rakenteellista informaatiota:

- Kromosomit. Aitotumallisen eliön genomi jakaantuu kromosomeihin. Kromosomit risteytyvät pareittain, eivät satunnaisesti. [Lokki et al.] Kromosomista muuntelua tapahtuu enemmän hyönteisillä ja muilla yksinkertaisilla eliöillä kuin nisäkkäillä. [Niemi et al.]
- Geenit. Kromosomi koostuu joukosta geenejä, joista jokainen geeni koodittaa tiettyä proteiinia. Kromosomin sisältämiä vaihtoehtoisia paikkoja kutsutaan

lokuksiksi. Kukin lokus voi sisältää erilaisia geenejä: vaihtoehtoisia geenejä kutsutaan *alleeleiksi*. [Lokki et al.]

- Tripletti. Geeni koostuu joukosta triplettejä, joista jokainen geeni koodittaa tiettyä aminohappoa. Triplettejä on 24 erilaista. Tripletti koostuu kolmesta emäsparista. [Niemi et al.]
- Emäspari. Emäsparit ovat fyysikaalisia kappaleita. Emäspareja on luonnossa kahdenlaisia: G-C ja A-U. Emäsparin G-C -liitos on kymmenen kertaa A-U -liitosta lujempi. [Niemi et al., 12]
- Solussa toimivat *korjausmekanismit* korjaavat aktiivisesti osan mutaatioista. [Niemi et al.]
- DNA:lla on myös fyysinen rakenteensa. DNA katkeaa toisista kohdista keskimääräistä helpommin (ns. kuumat pisteet), eikä ole siis fyysiseltä vahvuudeltaan tasalaatuinen. A-T -emäspareja on runsaammin paikoissa, joissa DNA-ketjujen tulee irrota toisistaan. [Niemi et al., 116].

Lisäksi listataan vielä joukko vähemmän merkityksellisiä, mutta mielenkiintoisia tosiasioita mallintamisen perustaksi. Näiden avulla pitäisi jo voida hahmotella biologisesti orientoitunutta genomia.

- Geenit sijoittuvat kromosomeihin melko tasaisesti kromosomin pituuden suhteessa. [Niemi et al., 236]
- Rakenteelliset kromosomimutaatiot ovat varsin yleisiä, mutta suurin osa katkenneista kromatideista korjautuu ennalleen DNA:n korjausmekanismien avulla. [Niemi et al., 230]
- Geneneettisessä koodissa ei ole ”välimerkkejä”, vaan sitä luetaan tauotta alkukohdasta päätekoodiin asti. Yhdenkin emäksen lisääminen tai poistaminen muuttaa koodin merkityksen perusteellisesti. [Niemi et al., 123]
- DNA:han sisältyy itse asiassa kahden tyyppistä informaatiota: geenit, jotka määrittelevät proteiinit, ja ohjeet, joiden mukaan solu lukee geenejään. [Niemi et al., 116]
- Useimmissa saman lajin solutumissa DNA:n määrä on vakio. [Niemi et al., 116]
- Prokaryoottien (bakteerien) kaksoiskierre on suljettuna renkaana. [Niemi et al.]
- Kolmen emäksen tripletit koodittavat 20 erilaista aminohappoa. Siis kutakin aminohappoa vastaa 1-6 koodisanaa. [Niemi et al., 123]

- Useimmat eukaryoottien geenit eivät ole yhtenäisiä, vaan koodittavien eksonijaksojen välissä on intronijaksoja. [Niemi et al., 126]
- 10% DNA:sta on useasti jopa 10^6 kertaan toistuvia DNA-sekvenssejä. Kyseisiä sekvenssejä käytetään kromosomien pariutumisessa. Yksittäisten geenien osuus on noin 60%. [Niemi et al., 131]

Rakenteellinen informaatio tulisi mallintaa siten, että evolutiivinen laskenta voi mukautua ongelmaan sopivaksi heuristiikaksi. Seuraavaksi käydään läpi genomin rakenteita siten, että poimitaan etsintäalgoritmin kannalta olennainen.

Emäspari. Emäspari voitanee tulkita atomaariseksi merkiksi, jolle liitetään joukko ominaisuuksia. Atomaarisistä merkeistä kootaan käytetty aakkosto. Toisaalta voidaan antaa myös rakenteellisen merkistön muuttua evolutiivisen laskennan aikana, tällöin merkistö muodostetaan joukosta ominaisuuksia. Rakenteellisen aakkostosta muodostuvat funktionaaliset merkit, esimerkiksi 1, 2 tai 3 merkkiä muodostaa funktionaalisen aakkoston merkin.

Tripletit. Geenit muodostuvat joukosta funktionaalisia merkkejä, jotka eivät ole rakenteellisesti tasalaatuisia. Sama tripletti eli funktionaalisen aakkoston merkki, voi esiintyä useana rakenteellisesti erilaisena versiona. Tripletit voivat olla rakeenteeltaan erilaisia, vaikka ovat hyvyysfunktion kannalta synonyymejä. DNA:ssa jokaista funktionaalisen aakkoston merkkiä koodittaa 1-6 erilaista triplettiä.

Geenit. Biologisesti orientoitunutta evolutiivista laskentaa on luonnollisinta soveltaa ongelmiin, joiden ratkaisuehdotukset voidaan kuvata sääntöjoukkona. Tällöin geenit rinnastetaan sääntöihin. Tämä ei mielestäni rajoita liikaa evolutiivisen laskennan sovellutusaluetta. Geenejä tai sääntöjä voitanee pitää yksikköinä, jotka usein risteytyvät atomäärisinä, ilman muutoksia. Sääntöjen määrä ei yleensä muutu risteytymisen yhteydessä.

Kromosomi. Genomiin on voitava muodostua rakenteita, jotka risteytyvät erillisinä kokonaisuuksina: sääntöjoukkojen on voitava risteytyä erikseen. Evolutiivinen laskenta voi sallia rakenteita, jotka ovat eliön toiminnan kannalta vähämerkityksisiä, jotta kromosomien lukumäärän vaihtelu mahdollistuu.

Seuraavaksi esitellään joukko perinteisen geneettisen algoritmin ongelmia, joihin voitanee vaikuttaa rakenteellisen informaation avulla:

- *Vaiheistus* voi perinteisellä geneettisellä operaatiolla epäonnistua täysin. Geneettisen algoritmin soveltaminen vaihtelevan mittaisten merkkijonojen aikaan saamiseksi on sekvenssien poistot sallittava. Oletetaan esimerkiksi, että aakkosto on koodattu kolmella bitillä {00 = ”a”, 01 = ”b”, 10 = ”c”, 11 = ”_”}. Jos nyt merkkijonosta ”ca_abb” poistetaan ensimmäinen bitti, muuttuu se merkkijonoksi ”abcac”, joka ei muistuta edellistä merkkijonoa. Vaiheistuksen muutosta ei voida siis sallia ainakaan kovin yleisenä tapahtumana, korkeintaan harvinaisena poikkeuksena.
- *Kombinaatiot* aiheuttavat ongelmia, jos tavoitteena on oikea kombinaatio. Kun tarkastellaan genomia A = ”12345” ja genomia B = ”54321”, saadaan yhden kohdan risteyksen tuloksena esimerkiksi ”12321”, joka ei ole sallittu merkkijono.
- Perinteinen geneettinen algoritmi ei tue aakkoston muodostumista prosessin kuluessa. Geneettinen algoritmi ei sisällä mekanismia geneettisten operaatioiden yhteydessä tapahtuvan rakennuspalikoiden rikkoutumisen estämiseksi [vert. Mitchell, 42]. Kuten skeemateoria todistaa [Holland, 1975], se tukee pienten skeemojen muodostumista, mutta ei tue suurten skeemojen muodostumista. Holland todisti geneettisen algoritmin toimivuuden populaation ollessa ääretön ja kun geenien välillä ei ole riippuvuuksia. Geenien väliset riippuvuudet ovat osoittautuneet kuitenkin isoksi ongelmaksi.

Määritelmä 6. Rakenteiden tasot. Genomi koostuu joukosta rakennuspalikoita, jotka voivat koostua joukosta alemman tason rakennuspalikoita. Saman tason rakennuspalikat muistuttavat toisiaan. Genomi on ylimmän tason rakennuspalikka, alimman tason muodostaa ns*. rakenteellinen aakkosto*.

Rakenteellisen merkin ominaisuuksia voivat olla esimerkiksi stabiilisuus pistemutaatiota vasten tai se, kuinka helposti merkkijono katkeaa merkin kohdalta. Normaalisti geneettisen algoritmin merkeillä on vain funktionaalinen ominaisuus, joka vaikuttaa genomista tuotetun fenotyypin (esim. graafin) ilmiasuun. Rakenteellisessa aakkostossa merkeillä on siis muitakin ominaisuuksia, jotka vaikuttavat evolutiivisen algoritmin kulkuun. Aakkostoon lisättyjen ominaisuuksien on tarkoitus toimia prosessin evolutiivisena muistina.

Rakenteellinen aakkosto voidaan mallintaa esimerkiksi seuraavasti: aakkoston merkit mallinnetaan käyttäen moniulotteista vektoria, jonka jokainen bitti ilmentää yhtä aakkoston ominaisuutta. Aakkoston merkki voi esimerkiksi muodostua kolmen pituisesta vektorista, $v = \{a_1,b_2,c_3\}$. Siihen liitetään

ominaisuudet toiminta(v) = a_1 (a_1 yleensä {0|1}), stabiilisuus(v) = b_2 sekä kiinnittyvyys(v) = c_3 (c_3 yleensä {0|1}). Genomi jaetaan säännöksi kiinnittyvyys-ominaisuuden avulla. Kiinnittyvyys-ominaisuuden ollessa peräkkäisillä merkeillä ”00” aloitetaan uuden säännön lukeminen. Sääntöjen lukumäärä muodostuu ”00”-merkkijonojen lukumäärästä. Stabiilisuudella (b) tarkoitan geenin pistemutaation todennäköisyyttä. Stabiilisuuden ollessa 0 mutaatio sallitaan vain stabiilisuudelle. Stabiilisuuden ollessa 1, pistemutaatio voi tapahtua myös muille ominaisuuksille. Merkin toiminta-ominaisuus vaikuttaa sääntöjen sisältöön. Kiinnityvyysominaisuus vaikuttaa risteytykseen siten, että genomit katkaistaan ainoastaan kiinnittyvyyskohdista ”000”.

Epistasia eli geenien välinen vuorovaikutus. Niin sanotut säätelygeenit ohjaavat muiden geenien aktiivisuutta. Genomi voi sisältää geenejä, jotka aktivoivat tai passivoivat isompia geenijoukkoja. Epistasian mallintamista ei tässä käsitellä; aihetta käsittelee esimerkiksi Kauffman [Kauffman]. Rakenteellisen aakkoston tavoitteena on että, vaikka geenien välistä vuorovaikutusta ilmenee, risteytysoperaatio voi siitä huolimatta, sekä toteuttaa monipuolista tekijäinvaihtoa, että tuottaa toimivia genomeja.

Biologinen genomi sisältää siis useita rakenteellisia rakennuspalikoita, jotka on otettava huomioon evolutiivista laskentaa kehitettäessä. Tässä kohdassa ohitettiin useita tärkeitä tosiasioita kuten, kromosomiston diploidisuus, kuinka DNA voi muodostaa 3-ulotteisia rakenteita, solun jakaantuminen, solun risteytyminen. Näistä tarkemmin biologian oppikirjoissa [Niemi et al.]. Seuraavassa kohdassa tarkastellaan lähemmin geneettisiä operaatioita.

6.2. Geneettiset operaatiot

Tässä kohdassa esitellään luonnossa tapahtuvat geneettiset operaatiot. Lisäksi määritellään muutamia evolutiiviseen laskentaan soveltuvia geneettisiä operaatioita, jotka käyttävät rakenteellista informaatiota. Tavoitteena on laadukas geneettinen operaatio.

Määritelmä 7. Geneettisen operaation *laatu*. Laadukas operaatio synnyttää genomin, joka täyttää tehtävän rajoitteet, ja syntynyt fenotyyppi säilyttää useimmat käytettyjen genomien piirteet.

Perinteinen geneettinen algoritmi käyttää mutaatiotaajuutta p_m bittiä kohti, jossa p_m on simulaation ajan vakio ja yleensä $p_m \in [0.001, 0.01]$, eli yhden sukupolven aikana tapahtuu yhdestä kymmeneen mutaatiota tuhatta bittiä kohti

[Bäck]. Geneettisen algorimin mutaatiofunktio on adaptiivisesti suuntautumaton; mutaatio ilmaantuu riippumatta siitä, onko se genomille hyödyllinen vai haitallinen. On kuitenkin huomattava, että biologiset rakenteet muodostuvat useista matalamman tason komponenteista, joten niiden ”mutaatiotaajuuksissa” on suuria vaihteluita.

Kun biologi sanoo, että mutaatiot ovat satunnaisia, hän ei tarkoita, että ne olisivat satunnaisia, hän tarkoittaa että ne ovat eliön *tarpeiden kannalta* satunnaisia. Biologisen DNA:n mutaatiot eivät ole täysin satunnaisia, vaan tripletin todennäköisyys mutatoitua riippuu sen rakenteesta, naapureista, sijainnista ja jopa ympäristö olosuhteista (em. säteily) [Lokki et al.].

On suorastaan mahdotonta toteuttaa genomi monimutkaiselle järjestelmälle, jossa jokaisen merkin muutos aiheuttaa yhtä suuren muutoksen järjestelmän toimintaan ja jokaisen merkin mutaatiotaajuus sukupolvea kohti on vaikkapa 10^{-5} genomin pituuden ollessa 10^5 merkkiä. Jos esimerkiksi oletetaan, että osa genomin informaatiosta käytetään koodittamaan rakenteita, joiden avulla genomin muut osat tulkitaan, niin tällöin pienikin muutos näissä osissa aiheuttaa suuria muutoksia genomin fenotyyppiin. Koneiston stabiilisuuden on oltava useita kertaluokkia suurempi kuin genomin muiden osien.

Mielestäni geneettisten operaatioiden on toimittava siten, että suhteellisen todennäköiset muutokset genomiin aiheuttavat pieniä muutoksia yksilön toimintaan. Jopa muunnosten määrä suhteessa eliön lisääntymisstrategiaan, ympäristöön ja toisaalta informaation määrään voidaan huomioida. Jos informaatiota on paljon, aiheuttaa pienikin mutaatiotaajuus emäsparia kohti liikaa mutaatioita.

Mutaatiotaajuuden lisääminen populaation konvergoitumisen estämiseksi ei ole biologisesti perusteltua. Populaation monimuotoisuuden ylläpitoon on tarjolla biologisesti luontevia ratkaisuja, kuten koevoluutio, kilpailevat populaatiot, dynaaminen hyvyysfunktio, erilaisten kromosomien risteyttämättä jättäminen ja erilaisuuden suosiminen, joita käsitellään kohdissa 6.3 ja 6.4. Mielestäni geneettisten operaatioiden tulee olla ennen kaikkea laadukkaita, ja populaation pysähtyminen lokaaliin optimiin estetään dynaamisen hyvyysfunktion avulla, eikä geneettisten operaatioiden avulla.

DNA:han kohdistuneita mutaatioita voi olla neljänlaisia: korvautumia, poistumia, enentymiä ja kääntymiä. [Lokki, 16]. Geenin monistuminen on yksi evoluution aikana tapahtuvista geenimuutoksista. [Niemi et al., 126] Monistuneet

geenit ovat toisinaan pysyneet samanlaisina, mutta usein ne ovat muuntuneet alkuperäisestä ja eriytyneet joihinkin tehtäviin. [Niemi et al., 134] Monistumista voinee tapahtua sekä risteytettäessä että mutaation seurauksena solun jakautumisen yhteydessä.

Yksilöllisen mutaatiotaajuuden, siis jokaisella merkillä on oma mutaatiotaajuus, on todettu toimivan vähintään yhtä hyvin kuin yleensä käytetyn matalan mutaatiotaajuuden. [Stanhope and Daida]

Perinteinen risteytysoperaatio risteyttää genomit katkaisemalla molemmat risteytettävät genomit joko yhdestä tai kahdesta kohdasta, tai se voi rekombinoida jokaisen merkin erikseen. [Mitchell] Mielestäni perinteinen risteytysoperaatio on joko tekijäinvaihdoltaan riittämätön tai rikkoo liiaksi genomin sääntöjen välisiä riippuvuuksia. Risteytymisoperaatiota tulisi kehittää niin, että se sallii sääntöjen välisiä riippuvuuksia rikkomatta tapahtuvan monipuolisen tekijäinvaihdon.

Luonnossa tapahtuva eukaryoottien genomin risteytysoperaatio on hyvin tarkasti ohjautuva prosessi, jossa tapahtuu kahdenlaista risteytymistä. Kromosomit rekombinoituvat hyvin satunnaisesti ja jo pelkkien kromosomikombinanttien määrä 2^{46} (ihmisen genomi) on suuri. Kromosomeja rekombinoiva risteytys on helppo toteuttaa myös evolutiivisessa laskennassa rekombinoimalla satunnaisesti genomin ”kakkostason” rakennus palikoita

Luonnossa tapahtuu myös kromosomin ja kromosomin vastinparien välistä risteytymistä, vaikka kromosomit eivät aina risteydy. Kromosomien risteyttämättä jättämisen voisi tulkita siten, että jos kromosomit (tai genomit tai geenit) ovat hyvin erilaisia, niitä ei risteytetä. Kromosomien risteytymisen edellytyksenä on, että homologiset kromosomit löytävät meioosin aikana toisensa ja pariutuvat tarkalleen rinnakkain niin, että kunkin geenin paikka vastaa tarkalleen homologisen geeninsä paikkaa. [Niemi et al.] Kun kromosomi katkeaa, katkeaa se yleensä monesta kohtaa [Lokki 218, 225].

Jos kromosomissa ja sen vastinparissa on eri määrä geenejä, on vaarana, että eri tehtävää koodittavat geenit risteytyvät. Esimerkiksi yksittäisesti esiintyvien geenien osuus on valtava, joten risteytysoperaatio ei voi ”satunnaisesti” poistaa geenejä. Toisaalta risteytysoperaatio ei voi vain lisätä geenien määrää. Luonto kuitenkin risteyttää myös kromosomeja, eli evolutiivisen laskennan on mahdollistettava kromosomien risteytyminen. Biologisesti orientoituneen risteytysoperaation on myös estettävä suuresti erilaisten kromosomien risteytyminen.

Kromosomien, jotka eivät ole vastinpareja, välinen tekijäinvaihto on luonnossa niin yleistä, että se on luultavasti populaation evoluution kannalta toivottavaa. Se on kuitenkin paljon harvinaisempaa kuin kromosomin sisäinen tekijäinvaihto, jota tapahtunee lähes jokaisessa risteytystapahtumassa. Ylimääräisten kromosomien tai deleetioiden esiintyminen johtaa yleensä spontaaniin aborttiin; eliö on elinkykyinen vain poikkeus tapauksissa. Luonnon risteytysoperaatio on erittäin laadukas huolimatta siitä, että sen täytyy operoida suurilla tietomäärillä.

Tulkitsen biologista risteytymiseen liittyvät faktat seuraavaksi perusperiaatteeksi. Risteytä samankaltaiset genomit, ja toista tämä kaikilla genomin rakenteellisilla tasoilla itsenäisenä tapahtumana.

Kromosomien samankaltaisuutta voitanee mallintaa geenien lukumäärällä. Voidaan myös määritellä aakkostoon rakenteellinen ominaisuus, joka sallii kromosomin katkeamisen, ja näin ollen vaatia, että risteytyvien kromosomien ”katkokohtien” määrä on sama. Katkokohdat mahdollistavat geenijoukkojen synnyn kromosomeihin, jotka taas tulkitaan omina itsenäisinä rakenteinaan siten, että ne joko risteytyvät kromosomin risteytyessä tai vain rekombinoituvat yhtenäisinä rakenteina.

Alleelien määrät ajautuvat kohti tasapainoa, jos geenit pääsevät kombinoitumaan vapaasti keskenään, mutta tasapainon saavuttaminen vie sitä kauemmin mitä lähempänä geenit ovat. Siis mitä lähempänä tarkasteltavat lokukset ovat sitä tiukemmin kytkeytyneitä ne ovat. [Lokki et al., 70]. Mielestäni mallinnettavan rakenteellisen aakkoston ja käytetyn risteytysoperaation tulee toimia juuri näin. Rakenteellisen merkkijonon avulla tavoitellaan tilannetta, jossa evolutiivinen laskenta ryhmittelee ne säännöt peräkkäin, joiden on hyvä risteytyä yhdessä. Lähekkäiset säännöt ovat samassa matalan tason rakennuspalikassa ja pysyvät risteytymisen yhteydessä usein yhdessä. Peräkkäisistä säännöistä voi siis syntyä isompia rakennuspalikoita. Edellä hahmoteltua evolutiivista laskentaa tulisi soveltaa ongelmiin, joissa sääntöjen järjestyksellä on merkitystä ainoastaan risteytysoperaation kannalta: esimerkiksi kromosomien ei tulisi olla numeroituja, vaan kromosomien pariutuminen tulisi suorittaa rakenteiden samankaltaisuuden perusteella.

Skeemateorian soveltaminen genomiin, joka on muodostettu monesta tasosta, vaatinee hieman kehittelyä, mutta periaatteessa skeemateorian soveltaminen hierarkkiseen malliin on melko yksinkertaista. Evolutiivisen

laskennan on voitava muodostaa myös skeemoja, jotka muodostuvat muista skeemoista, kuten luonnossa.

Genomin ylemmän tason rakenteet voivat muistuttaa geneettisessä ohjelmoinnissa käytettyä puurakennetta. Aakkoston rakenteelliset ominaisuudet ovat kuitenkin eri asia kuin tavanomainen geneettinen ohjelmointi. Geneettisessä ohjelmoinnissa ei myöskään yleensä oteta kantaa lajiutumiseen. Kun sallitaan tai kielletään osaa kromosomeista risteytymästä keskenään, voi se olla alku mahdolliselle lajiutumisen mallintamiselle.

6.3. Populaatiot ja lajiutumisen mallintaminen

Miten luonnossa esiintyvät lajit, ja lajien väliset vuorovaikutukset tulisi evolutiivisen laskennan mallintaa? Mitä ilman evolutiivinen laskenta jää, jos populaatioiden dynamiikkaa ei mallinneta? Näitä ja muita populaatioihin liittyviä kysymyksiä käsitellään tässä kohdassa.

Pariutuminen, jolla risteytyvät yksilöt valitaan, tulee mallintaa. Luonnossa samaa *lajia* olevat yksilöt voivat risteytyä keskenään. Genomi määrittelee lajin, eli siis genomien on oltava riittävän samankaltaisia. Luonnossa fenotyyppi valitsee fenotyypin, seikka jolla voi olla merkitystä, mutta voitanee jättää evolutiivisen laskennan mallinnuksesta pois. Lopullisen valinnan tekevät kuitenkin genomit, joiden risteytys joko onnistuu tai sitten ei.

Laji on keskenään lisääntyvien, muista populaatioista lisääntymisisolaatiossa olevien populaatioiden joukko [Lokki et al., 325]. Lajiutuminen on yksinkertaista tuottaa mahdollistamalla populaation maantieteellinen eriytyminen. Mutta jos tavoitteena on biologisesti realistisempi malli, myös sympatrista (ilman maantieteellistä eristymistä tapahtuvaa) [Lokki et al., 212] lajiutumista on mahdollistettava.

Luonnossa lajiutuminen ei aluksi ole luonteeltaan geneettistä siten, että kahden lajiutumassa olevan populaation yksilöt eivät voisi saada yhteisiä jälkeläisiä, vaan lajiutuminen ilmenee käyttäytymisen muutoksena. Lajiutuminen saa lopullisuuden tuntua viimeistään paritteluun liittyvän käyttäytymisen muuttumisen yhteydessä. Esimerkiksi kaksi kalalajia, jotka luonnossa ovat eri lajia, saavat usein akvaario-olosuhteissa lisääntymiskykyisiä jälkeläisiä. Tällöin lajien käyttäytymistä erotteleva tekijä (lämpötila) on poistettu ympäristöstä. Lämpötilaan (ympäristön muuttuja) perustuva lisääntymiseristäytyminen voi aikaan saada melko täydellisen lisääntymiseristäytymisen kalalajien välille [Lokki

et al.]. Kaksi yksilöä voivat olla siis eri lajia, vaikka ne periaatteessa voisivat risteytyä. Lopullista lajiutuminen on, kun genomien risteytysoperaation tuottamat jälkeläiset eivät ole elinkelpoisia. On biologista realismia, että genomi ei voi risteytyä minkä tahansa genomin kanssa.

Mielestäni biologisesti orientoitunut pariutumisoperaatio, joka siis valitsee risteytettävät yksilöt, pitää huolen siitä, että yksilöt ovat mahdollisimman samanlaisia. Ainakin kromosomien määrän tulisi olla sama. Risteytymis-operaation on silti oltava laadukas, sen on tuotettava elinkelpoisia yksilöitä jopa toivottoman erilaisista genomeista. Pariutumisoperaatio tavallaan määrittelee simulaatiossa esiintyvien lajien runsauden.

Biologista realismia simulaatioon tuo yksilöihin lisättävä paikkaominaisuus, joka vaikuttaa pariutumiseen. Vain yksilöiden ollessa lähellä toisiaan voi risteytyminen tapahtua. Geneettisessä algoritmissa, jossa käytetään vain yhtä populaatiota, risteytyvät yksilöt valitaan yleensä hyvyyslukujen perusteella; yleensä pariutumiseen lisätään hiukan satunnaisuutta. Paikkaominaisuuden käyttäminen geneettisen algoritmin kanssa on varsin yleistä. [Gruau].

Populaation koolla on ratkaiseva vaikutus populaation evolutiivisille ominaisuuksille. Eristyneet pikkupopulaatiot ovat alttiita geneettisen ajautumisen vaikutuksille [Lokki et al., 281]. Populaatio on altis suuntaamaan uuteen ratkaisuun, jos osa populaation vaikuttavista voimista on poissa. Pieni populaatio voi viedä erikoitumisensa pitemmälle, toisaalta suppeamman ympäristön vaatimukset ovat terävämpiä, kun useita mahdollisuuksia on poissa. Toisaalta saatavilla oleviin mahdollisuuksiin on mukauduttava entistä paremmin. Myös geneettisen aineksen supistuminen, jättää tilaa satunnaisuudelle ja tiettyjen alleelien tai ominaisuuksien korostuminen on osa geneettistä ajautumista.

Populaation on sopeuduttava sekä ilmastoon että ravintoon, mutta tämä ei vielä riitä, sillä populaation on myös sopeuduttava elämään yhdessä ekosysteemin muiden lajien kanssa. Kyseessä on siis koevoluutio [Lokki et al.], ja populaatiot sopeutuvat vastavuoroisesti toisiinsa adaptoitumalla. Koadaptaatio merkitsee sitä, että populaatiot mahdollisimman vähän pyrkivät toistensa alueelle. Näin ne välttävät kovaa kilpailua, ja populaatiot lokeroituvat ekologisesti. Koevoluutio (coevolution) on lajien vuorovaikutusta. Se poikkeaa usean riippumattomien yhtäaikaisten hakuprosessin rinnakkaisesta ajosta. Jos lajien välillä ei ole vuorovaikutusta, rinnakkainajolla ei saada tuntuvaa hyötyä. Toki kaksi

ajoa tuottaa todennäköisemmin yhtä ajoa parempia ratkaisuehdotuksia, mutta ajojen lisäämisen hyöty pienenee kohti nollaa.

Populaatioiden ja lajiutumisen mallintaminen tavoitteena on valjastaa yhä kasvavat prosessoritehot yhä monipuolisemman etsintäalgoritmin käyttöön. Tavoitteena on useiden populaatioiden dynamiikan mallinnuksen avulla saada etsintäalgoritmi toimimaan satunnaishakua paremmin, yhä pitempien simulaatioiden ajan. Rinnakkaiset lajit mahdollistanevat hakuavaruuden ”leveämmän” läpikäynnin. On kuitenkin luultavaa, ettei lajien erilaistumista tapahdu ilman biologisesti orientoitunutta hyvyysfunktiota.

6.4. Biologisesti orientoitunut hyvyysfunktio

Evolutiivisen laskennan vastine ympäristön käsitteelle on hyvyysfunktio. Ympäristön muuttuminen on ehkä tärkein mallinnettava asia, joka biologisesti orientoituneen hyvyysfunktion on otettava huomioon. Tavoitteena on ajallisesti ja paikallisesti vaihtelevan hyvyysfunktion avulla saada aikaan etsintäalgoritmi, joka pystyy löytämään yhä parempia vastauksia yhä vaikeampiin hakutehtäviin.

Luonnossa ympäristö mittaa genotyypistä kasvaneen fenotyypin elinkelpoisuuden. Genomin hyvyys on siis aina sidoksissa ympäristöön, tämä evolutiivisessa laskennassa luontoa ilmeisempää: se onko ”0101” parempi kuin ”1010”, on tietysti hyvyysfunktiosta riippuva.

Ajallisesti muuntuvan hyvyysfunktio on biologista realismia. Biologisessa luonnossa ympäristö on *aina* osa jotain suurempaa kokonaisuutta, joten vaikka ympäristö sinällään olisi muuttumaton, ei se sitä voi olla, koska se on osa isompaa kokonaisuutta, joka on jatkuvassa muutoksessa. Biologisesti orientoituneen hyvyysfunktion on siis muututtava ajan myötä.

Paikallisesti muuntuva hyvyysfunktio on myös biologinen tosiasia. Paikassa 1 resurssi A voi olla yleisempi kuin resurssi B. Paikassa 2 taas voi olla ainoastaan resurssia B; tällöin yksilön hyvyyteen ei vaikuta resurssin A hyödyntäminen. Jos resurssien A ja B hyödyntäminen suosivat erilaista fenotyyppiä, muodostunee paikkaan 1 erilainen populaatio kuin paikkaan 2. Kun jokaisella yksilöllä on oma paikkansa avaruudessa, mahdollistuu geneettisesti erilaisten alipopulaatioiden synty ja etäisten populaatioiden eriytyminen. Jos evolutiivisen laskennan yksilöille määritellään paikkaominaisuus, on yksilöille määriteltävä tapa liikkua. Liikkuminen voitanee mallintaa satunnaiseksi tapahtumaksi tai resurssien ohjaamaksi.

Populaation muiden yksilöiden vaikutus hyvyysfunktioon tuo oman lisänsä evolutiiviseen laskentaan. Koska populaation yksilöt ovat samankaltaisia, ne käyttävät samoja resursseja. Resurssien niukkuus tulee mallintaa, jotta estetään yhden lajin dominointi ja tuetaan lajiutumista. Ainoastaan laji, joka osaa hyödyntää kaikkia resursseja tehokkaammin kuin muut lajit, voi ehkä dominoida simulaatiota väliaikaisesti.

Geneettisen algoritmin suurin ongelma on populaation liiallinen *konvergoituminen*, kun populaation genotyyppien varianssi pienenee risteytysten seurauksena. Prosessi käyttäytyy tällöin normaalisti, mutta jos konvergoituminen tapahtuu liian aikaisin, on etsintä epäonnistunut. Populaatio on tällöin jäänyt ns. lokaaliin optimiin. Kun populaation genomit muistuttavat toisiaan, ei globaalin optimin löytäminen välttämättä onnistu ja prosessi voi pysähtyä paikoilleen. Mallintamalla lajiutuminen tai vain populaatioiden välinen dynamiikka, voidaan ehkäistä prosessin pysähtymistä: biologinen evoluutio ei toistaiseksi ole pysähtynyt.

Biologisesti orientoituneen hyvyysfunktion ei tarvitse vaatia ongelmalta liikaa erityisominaisuuksia, sen sijaan, monimuotoisuutta voidaan tukea ryhmittelemällä verkon testidata ja painottaa tapauksia lajikohtaisesti. Toisaalta vaikeisiin ongelmiin liittyy usein useita ristiriitaisia tavoitteita, joita voidaan käyttää hyväksi erilaisten populaatioiden tukemiseksi.

Biologisesti orientoitunut hyvyysfunktio voidaan mallintaa esimerkiksi seuraavasti: jokaiseen paikkaan liittyy tehtäväfunktio. Paikka antaa yksilölle energiaa, jos se vastaa tehtäväfunktion esittämään kysymykseen oikein. Onnistuneet käynnit paikalla vähentävät tehtäväfunktion energiapotentiaalia. Tehtäväfunktion energiapotentiaali kasvaa tasaisesti. Yksilön on saatava riittävästi energiaa toimintojensa ylläpitämiseen ja suvun jatkamiseen.

6.5. Evoluution hierarkkisen rakenteen mallintaminen

Geneettinen algoritmi voidaan nähdä karkeana evoluution mallinnuksena ja evolutiivisen laskennan esi-asteena. Kuten edellisissä kohdissa todettiin, sisältää luonnossa tapahtuva evoluutio useita etsintäalgorimin kannalta merkittäviä ilmiöitä, joita geneettinen algorimi ei tue. Taulukoon 8 on koottu biologisen evoluution perusasioita yhteen.

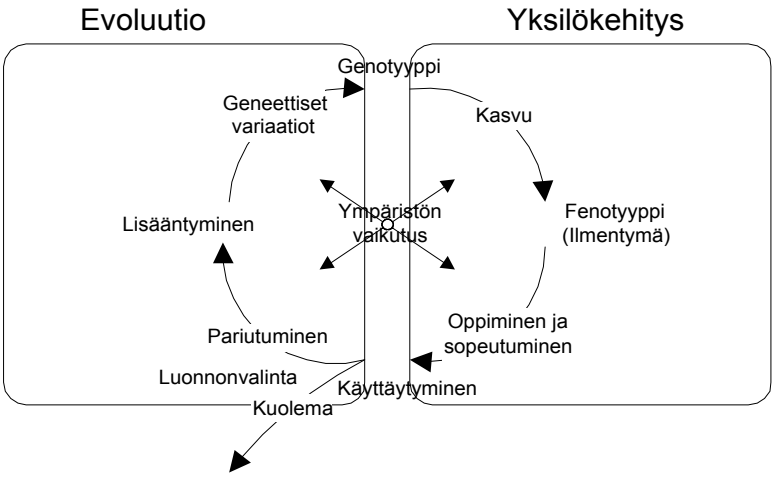
Taulukko 8. Evolutiivisen laskennan tasot.

Ilmentymä	Informaatio	Merkintä	Tulkinta	Mallinnettavaa
ekosysteemi		$E = P^{x > 1}$	ongelma	ravintoketju niukat resurssit
populaatio	geenipooli	$P = Y^{x > 0}$	hakustrategia	lajiutuminen
yksilö	genotyyppi	$Y = K^{1-30}$	ratkaisuehdotus, sääntöjoukko	pariutuminen, seksuaalivalinta, risteytyminen, liikkuminen, fenotyypin kasvaminen
kromosomi	geenijoukko	A^{0-100}	osaratkaisu, toisistaan riippuvien sääntöjen joukko	sukupuoli, resessiivinen, diploidi, risteytyminen
proteiini	geeni	$P^{100-1000}$	sääntö, rakennuspalikka	alku(merkki)*loppu, kopiomutaatiot
aminohappo	tripletti	M^3	funktionaalinen aakkosto	pistemutaatio
emäs	{U|C|G|A}	M	rakenteellinen aakkosto	rakenteelliset ominaisuudet

Evolutiivisen laskennan aakkoston kehittäminen voidaan tehdä joko siten, että luonnosta mallinnetut rakenteet, kromosomit, tripletit, geenit ovat prosessin alkuvaiheesta lähtien läsnä, tai kehitetään malli, joka mahdollistaa rakenteiden muodostumisen prosessin kuluessa.

Mahdollisia suuntaviivoja mallille, jossa rakenteet muodostuvat prosessin kuluessa, tarjoaa Maynard ja Szathmáry [Maynard and Szathmáry]. He jakavat evoluution historiallisen kehityksen kahdeksaan pääsiirtymään, joista useimmissa on yhteisiä piirteitä. Yksilöt, jotka aikaisemmin kykenivät yksilölliseen kopioitumiseen (replikoitumiseen), kopioituvat siirtymän jälkeen ainoastaan osana suurempaa kokonaisuutta. Myös työn jakaminen eli erikoistuminen on ollut huomattavaa. Huomattava on myös muutokset kielessä, informaation varastoinnissa ja siirrossa. Evolutiivisen laskennan kannalta olisi mielenkiintoista mallintaa tapa, jolla kaksi ratkaisuehdotusta yhdistetään yhdeksi niin, että ne yhdistymisen jälkeen lisääntyvät yhtenäisenä yksikkönä.

Evolutiivista laskentaa edelleen kehitettäessä tulee myös fenotyypin kasvu (kuva 11) mallintaa. Tällöin tulee määritellä kuinka rakenteellinen genomi ohjaa kasvua, tällöin lienee epistasian mallintaminen tarpeen.



Kuva 11. Biologinen evoluutio

Geneettinen algoritmi toteutetaan usein vain yhdellä populaatiolla; populaation konvergoituminen katsotaan tällöin ongelmaksi. Rakenteellinen aakkosto (6.1) yhdessä toimivan risteytysoperaation (6.2) ja muuntelevan hyvyysfunktion kanssa (6.4) mahdollistanee prosessin, jossa populaation konvergoituminen on luonnollisessa tasapainossa mutaatiotaajuuden kanssa. Esitetyt parannusehdotukset ovat prosessin kuormitukseen nähden edullisia, ja ne voidaan toteuttaa kuormittamatta prosessia merkittävästi.

Evolutiivista laskentaa tulee kehittää, koska evoluutio on todistettavasti tuottanut erittäin laadukkaita järjestelmiä. Geenien (tai sääntöjen) välillä on suuria riippuvuuksia. Geneettiset operaatiot eivät näitä riippuvuuksia voi tietää, silti geneettiset operaatiot toimivat luonnossa, ja ne on saatava toimimaan yhtä tehokkaasti myös evolutiivisen laskennan apuna.

7. Dave-simulointiympäristö

7.1. Daven toteutus

Esimerkkisimulaatiota varten olen toteuttanut Javalla luokkakirjaston evolutiivisen laskennan tutkimuksen mahdollistamiseksi. Se on suunniteltu tukemaan evolutiivisten algoritmien tutkimusta. Kirjasto ei toistaiseksi ole vapaasti saatavilla. Suosittelen tutkimusta aloittavalle esimerkiksi ECJ-Java kirjastoa, joka on saatavilla Internetissä. Simulaation satunnaislukugeneraattorina käytetään ECJ-java kirjaston MersenneTwisterFast -luokkaa.

7.2. Esimerkkinä pariteettiongelma

Esimerkkiongelmana toimii pariteettiongelma: onko verkon tulossa parillinen vai pariton määrä ykkösiä? Pariteettiongelmaa on käytetty aikaisemminkin kasvualgoritmin testiongelmana [Gruau]. Verkon hyvyydeksi lasketaan oikein vastattujen testien lukumäärä.

Ongelmafunktio on muotoa $y^1 = f(x^5)$, missä y ja x ovat bittivektoreita. Tavoitteena on siis luoda boolean-verkko, joka laskee bittijonon pariteetin, eli muuntaa syötteenä saadun bittijonon joko ykköseksi tai nollaksi.

7.3. Ajetun simulaation esittely

Dave-kasvumallia on testattu pariteettiongelman ratkaisuun. Kasvumallia on toistaiseksi käytetty ainoastaan yksinkertaisen etsintäalgoritmin yhteydessä. Kasvumallin toteutuksen toimivuutta ja ongelmia käsitellään tässä kohdassa.

Simulaatiossa luodaan boolean-verkko, joka toteuttaa pariteettiongelman. Verkon solmut toteuttavat joko ”ja”-operaation tai ”tai”-operaation. Verkon särmät toteuttavat negaation tai eivät muuta signaalia. Hyvyyslukuna käytetään oikein vastattujen testien lukumäärää. Kun kaikkiin testeihin tulee oikea vastaus, hyvyyslukuna käytetään verkon rakennetta: solmujen ja särmien summaa.

Käytetty etsintäalgoritmi on määritelty kuvassa 12. Aluksi bittijono alustetaan satunnaisilla arvoilla. Kullakin iteraatiolla bittijonoa mutatoidaan ja siitä tulkitaan Dave-kasvumallin mukainen sääntöjoukko. Kasvumallin mukainen verkko kasvatetaan ja testataan. Jos verkko on parempi kuin ennen mutaatioita,

mutaatiot säilytetään. Näin jatketaan niin kauan kun löydetään parannuksia verkkoon.

```
begin
        alusta(a);
        h_max := evaluoi(a );
        k  := 0;
        while ( h_max < 1 )  do
        begin
                a_mut := mutatoi( a );
                h_mut := evaluoi(a_mut );
                if (h_mut >= h_max )
                        if ( h_mut > h_max )
                                k = 0;
                        h_max  := h_mut;
                        a := a_mut;
                else
                        k := k + 1;
                if ( k  > 100000 )
                        k = 0;
                        a := mutatoi( a );
        end
end
```

Kuva 12. Käytetty etsintäalgoritmi

Simulaation aluksi pyritään löytämään toimiva verkon rakenne. Kun toimiva verkko on löytynyt, etsitään rakenteeltaan pienempää verkkoa. Etsintäalgoritmi hyväksyy myös ratkaisuehdotukset, jotka ovat yhtä hyviä kuin paras löydetty ratkaisuehdotus. Tämä mahdollistaa etsintäalgoritmin ”vaeltamisen” ja parantaa etsintäalgoritmin toimivuutta.

Taulukko 9. Simulaation parametrit

Parametri	Arvo
Sääntöjen lukumäärä	24, (504 bittiä, 21 bittiä kutakin säätöä kohden)
Kasvuiteraatioiden lukumäärä	5
Verkon tulosolmuja	5
Verkon lähtösolmuja	1
Verkon testikertoja eli tapauksia	32 (kaikki mahdollisuudet)
Pistemutaatioiden lukumäärä	10

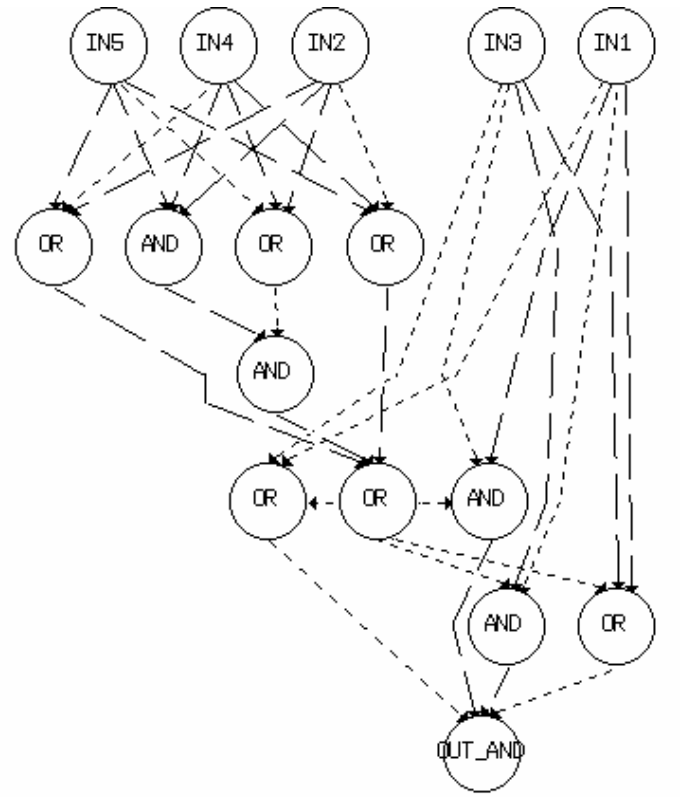
Lisäksi simulaatiossa käytettin muutamia ad hoc -parametereja, kuten ”toivottu solmujen määrä”. Kasvua tuetaan simulaation alussa, koska solmusääntö ei salli jakaantumista 50% todennäköisyydellä.

Taulukko 10. Ajetusta simulaatiosta mitattiin seuraavat tiedot.

Mitattu	Arvo
Testattuja verkkoja (iteraatioita)	1 851 776
Hakua mahdollisesti edistävien verkkojen lukumäärä (vähintään yhtä hyvä, kuin edellinen)	82 107
Verkon sisäsolmujen keskimääräinen lukumäärä	19
Paras löydetty verkko	5 tulosolmua, 18 sisäsolmua, 1 lähtösolmu, 40 särmää
Paras hyvyysluku	1.0
Parhaan verkon käytettyjen sääntöjen lukumäärä (kuinka moni säännöistä vaikuttaa tuotettuun verkkoon)	21
Ensimmäinen funktion toteuttava verkko löytyi iteraatiolla	304 228
Simulaation kesto (PC 500Mhz, 500Mt RAM)	1 tunti

Etsintäalgoritmi löytää ongelmafunktion toteuttavan verkon, mutta onnistuu huonosti verkon rakenteen optimoinnissa. Verkkoon jää turhia rakenteita. Löydetyn verkon ilmeisin ongelma ovat solmut, joiden tulo- ja lähtöaste on yksi, tällöin ”särmä-solmu-särmä”-kytkentä toteuttaa särmän toiminnallisuuden. Tarpeettomat solmut voidaan poistaa yksinkertaista heuristiikkaa käyttämällä, mutta olisi kuitenkin parempi, jos etsintäalgoritmi löytäisi edellä mainitun muutoksen automaattisesti.

Esitelty etsintäalgoritmi ei löytänyt 12 tunnin simulaation aikana toimivaa verkkoa 7-pariteettiongelmaan. Paras etsintäalgoritmin tuottama verkko on esitelty kuvassa 13; verkossa on 10 sisäsolmua ja 33 särmää.



Kuva 13. 5-pariteettiongelman ratkaiseva verkko.

8. Lopuksi

Kasvumalli on luonteva tapa verkon rakenteen määrittämiseksi. Dave-kasvumalli määrittelee verkon tiiviillä tavalla. Tämän työn perusteella ei Dave-kasvumallin toimivuutta tai hyvyyttä suhteessa muihin kirjallisuudessa esiteltyihin kasvumalleihin voida päätellä.

Valitettavasti en pystynyt tämän työn puitteissa todentamaan, miten genomin rakennuspalikoiden risteyttämättä jättäminen vaikuttaa lajiutumiseen. Pidän asiaa tutkimisen arvoisena. Tuntemani geneettisten algoritmien tutkimukset eivät ole asiaa käsitelleet.

Työn tulokset ilmenivät lähinnä evolutiivisen laskennan kehitysehdotusten muodossa, kirjallisuudessa evolutiivinen laskenta mainitaan usein ”muistittomana” etsintäalgoritmina. Tämä ei mielestäni ole oikein:

1. Etsintäalgoritmilla on paremmat edellytykset toimia, jos sillä on muisti ympäristössä, jossa muutokset eivät ole täysin satunnaisia.
2. Biologisella evoluutiolla on ”muisti”, joka tulee mallintaa evolutiiviseen laskentaan.
3. Aakkoston rakenteelliset ominaisuudet toimivat evolutiivisen laskennan muistina.

Mahdollinen jatkotutkimus keskittyy rakenteellisen informaation mallintamiseen ja evolutiivisen laskennan kehittämiseen. Tarkoituksena on toteuttaa evoluutiota mallintava etsintäalgoritmi ja todistaa etsintäalgoritmin toimivuus. Kasvualgoritmi toiminee ongelmallisen genotyyppi-fenotyyppi koodauksen toteuttajana. Jos jatkotutkimukseen ryhdytään, on myös konkreettinen sovellutusalue valittava. Tavoitteena on saada tuloksia jonkun ”yksinkertaisen” NP-täydellisen ongelman parissa. Tulevaisuuden tavoitteena on etsintäalgoritmin soveltaminen go-pelin tekoälyn kehittämiseen.

Viiteluettelo

[Aho, Mäkinen ja Poranen, 2001] Isto Aho, Erkki Mäkinen ja Timo Poranen, Algoritmien Suunnittelu ja analyysi. Tampereen yliopisto, Tietojenkäsittelytieteiden laitos, Raportti C-2000-1.

[Boers and Kuiper, 1992] Egbert J.W. Boers and Herman Kuiper, Biological metaphors and the design of modular artificial neural networks, Master's thesis, Dep. CS and Exp. and Theor. Phych., Leiden University, Netherlands. 1992.

[Buckley and Harary, 1990] Fred Buckley and Frank Harary, Distance in Graphs. Addison-Wesley, 1990.

[Bäck, 1993] Thomas Bäck, Optimal Mutation Rates in Genetic Search, Proceedings of the fifth international conference on genetic algorithms. Morgan Kaufmann Publishers, 1993.

[Dawkins, 1993] Richard Dawkins, Geenin itsekkyys. Alkuteos: The Selfish Gene, Kimmo Pietiläinen (suom.), 2. p., Gummerus, 1993.

[Golden, 1996] Richard M Golden, Mathematical Methods for Neural Network Analysis and Design, MIT Press, 1996.

[Gruau, 1994] Frédéric Gruau, Neural network synthesis using cellular encoding and the genetic algorithm. Ph.D. Thesis, Ecole Normale Supérieure de Lyon, France, 1994.

[Hecht-Nielsen, 1989] Robert Hecht-Nielsen, Neurocomputing, Addison-Westley, 1989.

[Holland, 1975] J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[Judd, 1990] J. Stephen Judd, Neural Network Design and the Complexity of Learning. MIT Press, 1990.

[Kalat, 1995] James W. Kalat, Biological Psychology, 5th ed., Brooks, 1995.

[Kauffman, 1993] Stuart Kauffman, The Origins of Order: Self-Organization and Selection in Evolution, Oxford University Press, 1993.

[Kitano, 1990] Hiroaki Kitano, Designing neural networks using genetic algorithms with graph generation system, Complex Systems 4, 461 – 476, 1990.

[Kohonen, 1995] Teuvo Kohonen, Self-Organizing Maps, Springer, 1995.

[Koza, 1992] J. R. Koza, Genetic Programming: On the programming of computers by means of natural selection, MIT Press, Cambridge, 1992.

[Latchman, 1995] David S. Latchman, Eukaryotic Transcription Factors, 2th ed., Academic Press, 1995.

[Lokki et al., 1986] J. Lokki, A. Saura ja P. Tigerstedt, Evoluutio ja populaatiot, WSOY, 1986.

[Maynard and Szathmáry, 1995] Smith John Maynard and Eörs Szathmáry, The Major Transitions in Evolution, Freeman, 1995.

[Michalewicz, 1992] Zbigniew Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1992.

[Mitchell, 1996] Melanie Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1996.

[Moriarty and Miikkulainen, 1996] David E. Moriarty and Risto Miikkulainen, Efficient Reinforcement Learning through Symbiotic Evolution. Machine Learning 22, 11 –32, 1996.

[Niemi et al., 1994] Mikko Niemi, Kalevi Korhonen, Ismo Virtanen, Solu- ja molekyylibiologia. WSOY, 1994.

[Rozenberg ja Salomaa, 1980] Grzegorz Rozenberg and Arto Salomaa, The Mathematical Theory of L Systems, Academic Press, 1980.

[Sean and Lee, 1996] Luke Sean and Spector Lee, Evolving Graphs and Networks with Edge Encoding: Preliminary Report, 1996 Conference Standford University, July 28-31, 1996.

[Stanhope and Daida] Stephen A. Stanhope and Jason M. Daida, An Invidually Variable Mutation-Rate Strategy for Genetic Algorithms. Lecture Notes in Computer Science 1213, Evolutionary Programming VI, 1997.

[Vaario, 1995] Jari Vaario, Tekoelo - synteettinen lähestymistapa elämän ymmärtämiseen. Teoksessa: Eero Hyvönen ja Jouko Seppänen (toim.), Keinoelämä - Artificial Life: Tekniikkaa, luonnontiedettä, filosofiaa ja taidetta, Kari Paatelainen Oy, 70 – 85, 1995.

[Viitanen, 1997] Sami Viitanen, Some New Global Optimization Algorithms. Report, Åbo Akademi, 1997.

[Zeigberg, 1991] Matthew Zeidenberg, Neural Networks in Artifical Intelligence, 1991.

Liitteet

Liite 1 Lyhenteet

Suomeksi	Englanniksi	Lyhenne
Evolutionäärinen algoritmi	Evolutionary Algorithm	EA
Evolutionäärinen laskenta	Evolutionary Computing	EC
Evolutionäärinen ohjelmointi	Evolutionary Programming	EP
Geneettinen algoritmi	Genetic Algorithm	GA
Genotyyppi (DNA:n sisältämä informaatio)	Genotype	
Genomi (DNA, fyysisenä, perimä)	Genome	
Geneettinen ohjelmointi	Genetic Programming	GP
Fenotyyppi (eliön ilmentymä)	Fenotype	
Itseorganisoiva, itsejärjestäytyminen, Itseoppiva	Self-organizing	
Keinoelämä	Artificial Life	AL
Keinotekoinen neuroverkko	Artificial Neural Network	ANN
Neurolaskenta	Neural Computing	NC
Neuroverkko, Neuraaliverkko	Neural Network	NN
Tekoäly	Artificial Intelligence	AI

Liite 2 Kuvat