

Arja Lavonen

**NeuroHaku – monikerroksisen perceptron-neuroverkon
epälineaarinen optimointi**

Tietotekniikan pro gradu -tutkielma
Tieteellinen laskenta
4. syyskuuta 2005

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Arja Lavonen

Yhteystiedot: email: arhelavo@cc.jyu.fi

Työn nimi: NeuroHaku – monikerroksisen perceptron-neuroverkon epälineaarinen optimointi

Title in English: NeuroSearch – nonlinear optimization of multilayer perceptron neural network

Työ: Tietotekniikan pro gradu -tutkielma

Sivumäärä: 74

Linja: Tieteellinen laskenta

Tiivistelmä: Resurssinhaku vertaisverkossa on haastavaa, koska verkossa ei ole keskitettyä hakemistoa, josta hakuja voisi tehdä. Resurssin löytämiseksi on kehitetty monia hakualgoritmeja. Yksi tällainen hakualgoritmi on NeuroHaku, joka käyttää monikerrosperceptron-neuroverkkoja. Tässä tutkielmassa käsitellään lyhyesti vertaisverkkoja, neuroverkkoja ja NeuroHaun toimintaa. Tutkimuksen käytännön osassa selvitetään, miten eri optimointimenetelmät sopivat monikerrosperceptron-neuroverkon optimointiin. Käytettyjä menetelmiä ovat suora hakumenetelmät ja evoluutioalgoritmit.

English abstract: Finding a resource in Peer-to-Peer (P2P) network is challenging because there isn't any index where to search from. Many search algorithms have been developed to find the resources. One of these algorithms is NeuroSearch, which uses multilayer perceptron neural networks. In this thesis Peer-to-Peer networks, neural networks and the function of NeuroSearch are described. In the practical part of thesis it is studied how different optimization methods suit for optimizing multilayer perceptron neural networks. Direct search methods and evolutionary algorithms are used for optimization.

Avainsanat: neuroverkko, vertaisverkko, optimointi, Powellin menetelmä, evoluutioalgoritmit, geneettiset algoritmit, evoluutio-ohjelmointi, evoluutiostrategiat, MLP

Keywords: neural network, peer-to-peer, optimization, Powell's method, evolutionary algorithms, genetic algorithms, evolutionary programming, evolution strategies, MLP

Copyright © 2005 Arja Lavonen

All rights reserved.

Sisältö

1	Johdanto	1
1.1	Tutkimusongelma	1
1.2	Cheese Factory -tutkimusryhmä	1
1.3	Tutkielman rakenne	2
2	Vertaisverkoista	3
2.1	Asiakas-palvelin -malli	3
2.2	Hybridi vertaisverkko	4
2.2.1	Bit Torrent -verkko	5
2.3	Puhdas vertaisverkko	7
2.4	Vertaisverkot tulevaisuudessa	8
3	Neuroverkoista	9
3.1	Neuroverkon rakenne	9
3.2	Neuroverkon oppiminen	10
3.2.1	Ohjattu oppiminen	11
3.2.2	Mukautuva oppiminen	11
4	Optimoinnista	12
4.1	Mitä optimointi on?	12
4.2	Termejä ja määritelmiä	13
4.2.1	Objektifunktio, rajoitefunktio ja muuttujat	13
4.2.2	Globaali ja lokaali optimi	13
4.2.3	Minimointi ja maksimointi	13
4.2.4	Unimodaalisuus	14
4.3	Erilaisia optimointitehtäviä	14
5	Optimointimenetelmiä	16
5.1	Epälineaarinen optimointi	17
5.1.1	Powellin menetelmä	17
5.1.2	Powellin toinen menetelmä	19

5.1.3	Zangwillen menetelmä	20
5.2	Evoluutioalgoritmit	21
5.2.1	Mutaatio	23
5.2.2	Rekombinaatio	24
5.2.3	Valinta	25
5.2.4	Geneettiset algoritmit	26
5.2.5	Evoluutiostrategiat	28
5.2.6	Evoluutio-ohjelmointi	29
6	Resurssien hakuongelma	31
6.1	Resurssien hakuongelman formalisointi	31
6.1.1	Sisääntuloparametrit ja funktio	31
6.1.2	Muut muuttujat ja funktiot	32
6.1.3	Algoritmi	33
6.1.4	Ongelman rajoitteita	34
7	NeuroHaku	35
7.1	Sisääntuloparametrit	35
7.2	Rakenne	40
7.3	Painokertoimien määrittäminen	40
7.4	Aiemmat tutkimukset	42
8	Tutkimusympäristö	43
8.1	Mitattavat suureet	43
8.2	Java-luokat	43
8.3	Käytetyt menetelmät	44
8.3.1	Rekombinaatio	44
8.3.2	Mutaatio	44
8.3.3	Valinta	44
8.4	Käytetyt laitteet	45
8.5	Neuroverkon optimointi suora hakumenetelmällä	45
9	Tutkimustuloksia	47
9.1	Yksipisteristeytys	48
9.1.1	Elitistinen valinta	48
9.1.2	Todennäköisyyteen perustuva valinta	49

9.1.3	Turnajaisvalinta	51
9.2	Kaksipiste-risteytys	52
9.2.1	Elitistinen valinta	52
9.2.2	Todennäköisyyteen perustuva valinta	55
9.2.3	Turnajaisvalinta	56
9.3	Mutaation voimakkuuden ja risteytysasteen vaikutus	57
9.3.1	Yksipisteristeytys	57
9.3.2	Kaksipisteristeytys	59
9.4	Yhteenveto tutkituista menetelmistä	60
9.4.1	Yhteenveto yksipisteristeytyksestä	60
9.4.2	Yhteenveto kaksipisteristeytyksestä	62
9.5	Johtopäätökset	63
10	Loppupäätelmät	65
10.1	Yhteenveto	65
10.2	Tulevaisuus	65
	Lähteet	67

1 Johdanto

1.1 Tutkimusongelma

Tässä tutkimuksessa selvitetään, miten eri optimointimenetelmät sopivat monikerroksisen perceptron (engl. *multi-layer perceptron*, *MLP*)-neuroverkon optimointiin. Tutkimustapauksena käytetään MLP:hen perustuvaa NeuroHaku-neuroverkon optimointia. NeuroHakua käytetään vertaisverkoissa päättämään, mitä kautta kyselyt leviävät vertaisverkossa, perustuen vertaisverkkosolmun sisältämään lokaaliin tietämykseen.

NeuroHaussa on käytössä evoluutio-ohjelmointiin pohjautuva optimointialgoritmi. Optimointi vie melko kauan aikaa ja se voi joissain tapauksessa pysähtyä myös kesken, joten evoluutio-ohjelmointi ei näyttäisi olevan paras mahdollinen optimointialgoritmi NeuroHaun tapauksessa. Optimointi haluttaisiin saada toimimaan nopeammin säilyttäen silti riittävä fitness-arvo. Tehtäväni oli pyrkiä löytämään NeuroHakuun sopivampi optimointialgoritmi, jotta laskenta-aikaa saataisiin huomattavasti vähennettyä.

1.2 Cheese Factory -tutkimusryhmä

Pro gradu -aiheen sain Jyväskylän yliopiston vertaisverkkoja tutkivalta Cheese Factory [8] tutkimusryhmältä. Ryhmä tutkii vertaisverkkojen tietoliikennettä ja vertaisverkkojen käytäytymistä resurssien hajautetussa haussa sekä vertaisverkkojen tehokasta käyttöä.

Projektin WWW-sivuilla hanketta kuvataan seuraavasti:

”Hankkeessa tutkitaan laskennan hajauttamista useille, tasa-arvoisille laskentayksiköille. Tämän tyyppistä laskentaa kutsutaan vertaisverkkolaskennaksi. Vertaisverkkolaskennassa ei ole lainkaan Master-yksikköä, joka huolehtisi järjestelmän toiminnasta, vaan myös kaikki ylläpitoon tarvittavat toiminnallisuudetkin (esim. tehtävien jako) on hajautettu. Ratkaisun etuna on vikasietoisuus, skaalautuvuus sekä ylläpidon helppous. Haittana on taas hallinta-algoritmien monimutkaisuus ja suurempi resurssien tarve.” [1]

1.3 Tutkielman rakenne

Luvussa kaksi käsitellään vertaisverkkoja ja eri verkkoarkkitehtuureita. Luvussa kolme käydään läpi neuroverkkoja, niiden rakennetta ja oppimista. Lukuun neljä on koottu perustietoa optimoinnista: termejä, määritelmiä ja erilaisia optimointitehtäviä. Luvussa viisi esitellään tutkimuksessa käytetyt optimointimenetelmät. Luvussa kuusi käydään läpi resurssien hakuongelma vertaisverkoissa. Luvussa seitsemän käsitellään NeuroHakua, mitä se tekee, miten se on rakentunut ja miten se toimii. Luvussa kahdeksan on kuvattu tutkimusympäristö. Lukuun yhdeksän on koottu tutkimuksessa saadut tulokset. Viimeisessä luvussa esitetään loppupäätelmät.

2 Vertaisverkoista

Vertaisverkot (engl. *Peer-to-Peer*, *P2P*) tunnetaan pääasiassa tiedostojen, etenkin musiikki-tiedostojen, jakamisesta. Tunnetuimpia vertaisverkkoja käyttäviä sovelluksia ovat Napster [25], Gnutella [12] ja KaZaA [17]. Vertaisverkkoja voidaan kuitenkin käyttää myös esimerkiksi hajautettuun laskentaan [19]. Myös monet pikaviestinohjelmista perustuvat vertais-tekniikkaan [32]. Skype-vertaisverkon [30] internet-puhelut on myös ala, joka lisää jatkuvasti suosiotaan vertaisverkoissa.

Vertaisverkko on vanha keksintö. Ensimmäinen dokumentti, jossa käsiteltiin tietojen vaihtamista kahden tietokoneen välillä, julkaistiin jo vuonna 1969. Vuosien saatossa vertaisverkot kuitenkin unoituivat, sillä asiakas-palvelin -malli sopi hyvin lähes kaikkiin tarpeisiin. Vuonna 1999 nykymuotoiset vertaisverkot nousivat puheenaiheeksi Napsterin myötä. Napster oli vertaisverkko, joka mahdollisti käyttäjille tehokkaan ja ilmaisen tavan kopioida tiedostoja toinen toisiltaan. Etenkin mp3-musiikin jako teki siitä merkittävän sovelluksen, jonka myötä tiedostojen jakamisen ajatus levisi nopeasti ja synnytti monia kilpailevia ohjelmia. [32]

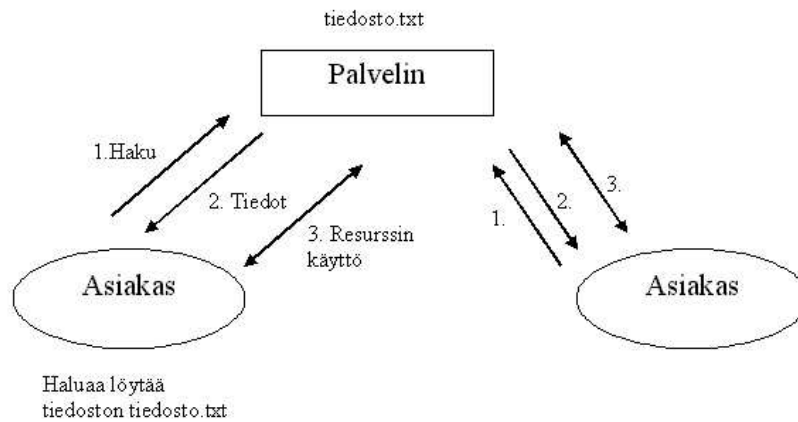
Seuraavassa esitellään olemassa olevia vertaisverkkoarkkitehtuureita, joista puhtaassa vertaisverkossa toimii NeuroHaku.

2.1 Asiakas-palvelin -malli

Asiakas-palvelin arkkitehtuurissa verkko rakentuu palvelimista ja asiakaskoneista. Palvelin on verkon keskeinen osa. Se tarjoaa kaiken verkon sisällön ja palvelut. Asiakas voi vain pyytää palvelimelta jotain tiedostoa tai palvelua. Asiakas ei pysty tarjoamaan muille verkon koneille mitään omia resurssejaan. [29]

Rajanveto asiakkaan ja palvelimen välille ei välttämättä kuitenkaan ole ihan yksinkertaista. Asiakas-palvelin -mallissa asiakkaan ja palvelimen roolit voivat vaihtua tilanteen mukaan. Esimerkiksi postin luvun aikana sähköpostijärjestelmä toimii palvelimena ja työasema on asiakas. Sähköpostipalvelin muuttuu kuitenkin asiakkaaksi, kun se välittää postia eteenpäin ja keskustelee toisen postipalvelimen kanssa. [32]

Kuvassa 2.1 on esitetty asiakas-palvelin arkkitehtuurin rakennetta. Ensinnäkin asiakas tekee haun palvelimelle, minkä jälkeen palvelin lähettää tiedon siitä, löytyykö haettua resurssia.



Kuva 2.1: Asiakas-palvelin arkkitehtuuri

Jos resurssi löytyi, asiakas pyytää sen käyttöön palvelimelta.

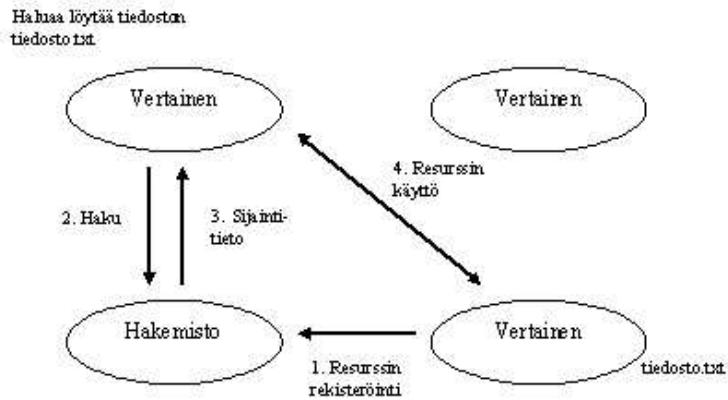
2.2 Hybridi vertaisverkko

Vertaisverkolla tarkoitetaan verkkoa, joka muodostuu hajautetuista vertaissolmuista. Jokainen solmu toimii sekä palvelimena että asiakkaana. Toisin kuin asiakas-palvelin -mallissa solmu voi nyt tarjota palveluja muille solmuille sekä kuluttaa muiden solmujen tarjoamia palveluita. Vertaissolmuja kutsutaan usein vertaisiksi.

Hybridin vertaisverkon erottaa puhtaasta vertaisverkosta se, että hybridissä vertaisverkos-
sa on olemassa keskussolmu, joka pitää kirjaa verkon sisällöstä tai palveluista. Jos tämän
solmun poistaa, osa verkon resurssien tiedoista häviää sen mukana. [29]

Resurssin haku hybridissä vertaisverkossa on esitetty kuvassa 2.2. Ensimmäisenä ver-
taisen kirjautuessa verkkoon, tiedot hänen resursseistaan rekisteröidään hakemistoon. Hakemis-
to pitää siis yllä tietoa kaikista verkossa olevista resursseista. Kun vertainen haluaa etsiä
jonkin tietyn resurssin, lähettää se haun hakemistoon. Hakemistolta tulee vastaus, löytyykö
resurssia ja jos löytyy, niin missä se sijaitsee. Tämän jälkeen resurssia voidaan käyttää suo-
raan kahden vertaisen välillä.

Hybridissä mallissa on useita teknisiä heikkouksia. Ensinnäkin keskuspalvelin on vält-
tämätön järjestelmän toiminnalle, mikä tekee siitä todella vikaantumisalttiin. [32] Keskus-
palvelin on myös verkon pullonkaula, sillä suurin osa liikenteestä kulkee sinne. Lisäksi
hybridissä vertaisverkossa palvelimen hankkiminen on kallis laitteistoinvestointi.



Kuva 2.2: Resurssin hakeminen hybridissä vertaisverkossa

1990-luvun lopulla kehitetty Napster oli hybridiä vertaisverkkoa käyttävä sovellus. Verkossa oli keskussolmu, joka toimi hakemistona. Käyttäjän kirjautuessa verkkoon, kaikista hänen jakamistaan tiedostoista meni tieto hakemistoon. Tiedoston sijaintitiedon hakeminen verkossa tapahtui siis hakemiston kautta ja tiedoston lataaminen suoraan vertaisten välillä. [15]

Nykyisin Napster on maksullinen sovellus, jonka kautta voi ostaa musiikkia. Napster on yhteistyössä viiden suuren levymerkin ja satojen itsenäisten levyntekijöiden kanssa. [25]

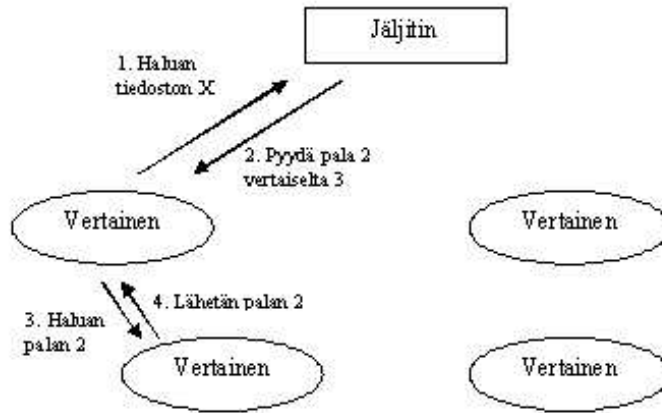
Pikaviestinohjelmat käyttävät myös vertaisverkoissaan keskitettyjä hakemistoja, joiden avulla käyttäjät löytävät toisensa. Viestimien käyttämä toimintamalli onkin hyvin samantapainen kuin Napsterin. Poikkeuksena on kuitenkin Internet Relay Chat (IRC), joka käyttää asiakas-palvelin arkkitehtuuria. [32]

2.2.1 Bit Torrent -verkko

Bit Torrent -verkko [3] on hybridimalliin luokiteltava vertaisverkko, jonka erityispiirteenä on monilähdelataus. Monilähdelataus toimii kuitenkin myös puhtaissa vertaisverkoissa, esimerkiksi KaZaAssa.

Tiedostojen kopioinnin nopeuttamiseksi kehitetyssä monilähdelatauksessa samaa tiedostoa voidaan ladata yhtäaikaan monesta eri lähteestä. Yksinkertainen tapa toteuttaa tämä on etsiä verkosta useita samaa tiedostoa jakavia käyttäjiä ja pyytää heiltä eri pala kyseisestä tiedostosta. [32]

Kehittynein muoto monilähdelatauksesta on Bit Torrent -toteutus. Bit Torrent -verkossa



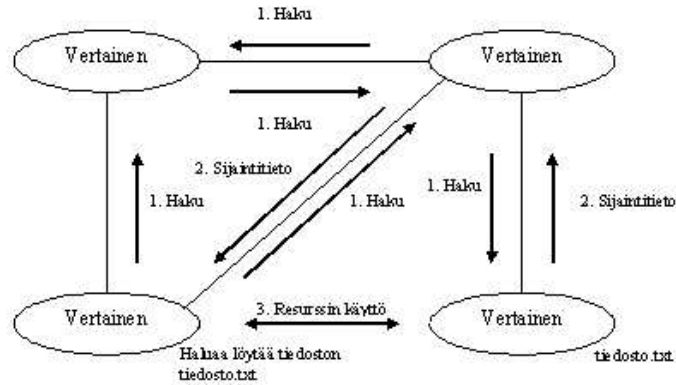
Kuva 2.3: Resurssin hakeminen Bit Torrent -verkossa

on kolme roolia: siemen (engl. *seed*), jäljitin (engl. *tracker*) ja parvi (engl. *swarm*). Siemen, eli yksi verkon vertainen, jakaa koko tiedoston. Jäljitin, joka on keskitetty palvelin, taas pitää kirjaa siementiedostosta: moneenko osaan se on jaettu ja kuka käyttäjistä kopioi mitäkin palasta. Asiakas on käyttäjä, joka kopioi tiedoston itselleen. Parveksi taas kutsutaan samasta tiedostosta kiinnostuneiden asiakkaiden ja siementen joukkoa. Parveen kuuluu siis monta vertaista. [32]

Kuvassa 2.3 on esitetty resurssin hakua Bit Torrent -verkossa. Ensin vertainen kertoo jäljittimelle haluavansa jonkin tiedoston. Jäljittimellä on tieto siitä, moneenko palaan tiedosto on jaettu ja keillä nuo palat on. Jäljitin palauttaa vertaiselle tiedon, keneltä se voi pyytää palan tiedostoa. Tämän jälkeen vertainen pyytää tiedostonpalan toiselta vertaiselta ja kyseessä oleva vertainen lähettää palan. Vertainen ilmoittaa myös jäljittimelle, minkä palan se on ladannut. Tätä jatketaan niin kauan, kunnes kaikilla on täydellinen kopio tiedostosta.

Asiakkaan ottaessa yhteyttä jäljittimeen, hänelle kerrotaan, mikä on harvinaisin pala tiedostosta ja missä se sijaitsee. Tämän jälkeen asiakas ottaa yhteyttä toiseen asiakkaaseen ja pyytää siltä tiedoston palasta. Näin siirrettävä tiedosto kootaan pala palalta. [32]

Jokainen Bit Torrent -parven asiakas sekä kopioi tiedostoa itselleen että lähettää sitä samalla muille. Näin ollen tiedostojen siirto tapahtuu tavallista vertaisverkkoa huomattavasti tehokkaammin. [32]



Kuva 2.4: Resurssin haku puhtaassa vertaisverkossa

2.3 Puhdas vertaisverkko

Puhtaassa vertaisverkossa kaikki solmut ovat samanarvoisia ja ne voivat jakaa resurssejaan toisille solmuille. Määritelmän mukaan puhdas vertaisverkko on kyseessä silloin, kun vertaisverkosta voidaan poistaa mikä tahansa solmu ilman, että verkon toiminta siitä mitenkään kärsii. [29]

Puhtaassa vertaisverkossa verkon kriittinen piste on siis poistettu. Tietoliikenne on myös mahdollista salata, mutta sitä ei juurikaan käytetä. Nyt ongelmaksi tulee kuitenkin hakutoimintojen vaikeus yhtenäisen hakemiston puuttuessa. Tähän on haettu ratkaisua verkon tulvituksen perustuvilla algoritmeilla. Verkon solmun hakiessa jotain resurssia, lähettää se kyselyn kaikille naapureilleen, jotka lähettävät kyselyn edelleen eteenpäin omille naapureilleen. Tulvitus on hyvin raskas verkon toiminnan kannalta, minkä takia kehittyneemmille vaihtoehtoisille olisi selvästi tarvetta. [32]

Resurssin hakua puhtaassa vertaisverkossa on esitetty kuvassa 2.4. Solmu lähettää ensin haun etsimästään resurssista naapurisolmulleen, joka lähettää sitä edelleen eteenpäin. Jos haettu resurssi löytyy, lähettää resurssin omistava solmu resurssin sijaintitiedot takaisin. Kun tieto resurssin sijainnista on saatu, voidaan resurssia käyttää.

Tiedostonjakamiseen tarkoitetuista ohjelmista Gnutella [12] on puhdas vertaisverkkosovellus. Kun käyttäjä liittyy Gnutellan verkkoon, hän tuo verkkoon mukanaan haluamansa resurssit jaettaviksi. Verkossa käyttäjä voi anonymisti etsiä ja käyttää muiden käyttäjien jakamia resursseja. Myös KaZaA [17] ja Skype [30] ovat puhtaita vertaisverkko-ohjelmia.

2.4 Vertaisverkot tulevaisuudessa

Vertaisverkkojen kehittyessä kiinnitetään entistä enemmän huomiota salaukseen. Liikenteen salaamisen lisäksi anonyymiin reitittämiseen perustuvat ratkaisut ovat herättäneet kiinnostusta. Tarkoituksena on kehittää verkko, jonka käyttäjien tunnistaminen olisi lähes mahdotonta tai vaatisi ainakin valtavia ponnistuksia ja kansainvälistä yhteistyötä. [32]

Internetin käyttöä anonyymisti on tutkittu pitkään. Freenet [11] on ohjelmisto, joka on kehitetty internetissä liikkumisen anonyymiteetin turvaamiseksi. Freenet mahdollistaa kenen tahansa julkaista materiaalia, kommunikoida ja etsiä tietoa internetistä täysin anonyymisti.

Tietokone-lehden [32] arvion mukaan yhteiskunta tulee puuttumaan vertaisverkkoihin jossain vaiheessa. Tämä on varmasti aivan totta, sillä piratismi on vertaisverkkojen yksi huolestuttavimmista piirteistä. Vertaisverkot ovat kuitenkin saaneet niin valtavan suosion, että niiden muuttaminen ei tule olemaan helppoa. Piratismi ja tekijänoikeudet ovat kuitenkin vain pieni osa vertaisverkkoja. Piratismin kitkemiseksi tehdään jatkuvasti työtä. Tutkitaan, miten esimerkiksi musiikkia voitaisiin jakaa kontrolloidusti, jolloin tekijänoikeusongelmia ei tulisi, mutta jakelu olisi edelleen helppoa.

Lisäksi vertaisverkoissa on paljon muita hyviä ominaisuuksia. Esimerkiksi Linuxin jakelu vertaisverkossa on hyödyllinen käytötapa. Myös uusia käyttökohteita vertaisverkoille kehitetään jatkuvasti. Internet-puhelut ovat jo nyt arkipäivää ja tulevaisuudessa tulee varmasti mahdolliseksi TV-lähetysten streaming-jako. Myös laskentagridit tulevat todennäköisesti toimimaan tulevaisuudessa P2P-perustaisesti.

3 Neuroverkoista

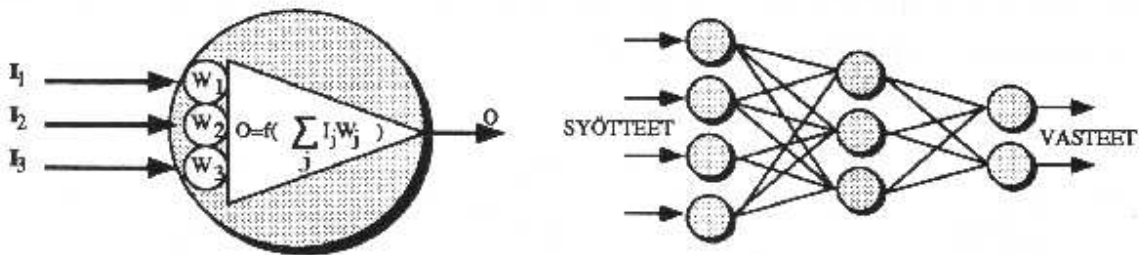
Neuroverkkojen (engl. *neural network*) perusideana on matkia ihmisen aivojen neuronien rakennetta ja toimintaa matemaattisesti. Neuroverkko on siis aivojen matemaattinen malli, joka jäljittelee matemaattisesti ihmisen oppimisprosessia. [36]

3.1 Neuroverkon rakenne

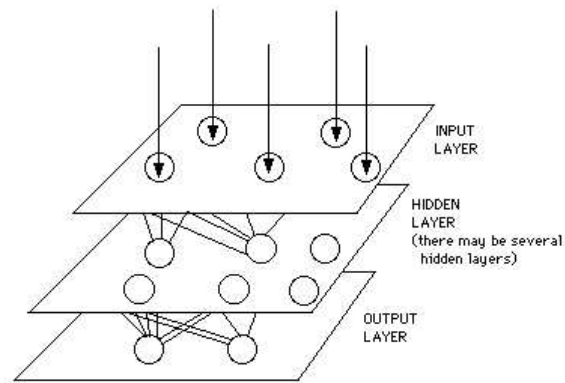
Neuroverkko koostuu joukosta yksinkertaisia solmuja (neuroneja), joiden välillä on liitoksia. Vasemmalla kuvassa 3.1 on kuvattu neuronin rakennetta. Neuroniin tulee syötteitä, jotka on kuvassa esitetty matemaattisin symbolein I_1, I_2, I_3 . Jokainen syöte kerrotaan omalla painokertoimellaan W_n . Tämän jälkeen neuroni tekee luvuilla joitain operaatioita, yksinkertaisimmillaan laskee ne vain yhteen, kuten kuvassa 3.1. Kuvan tapauksessa syötteiden ja painokertoimien tulojen summa on suoraan neuronin ulostulo O .

Neuronissa voi kuitenkin olla myös aktivaatiofunktio, joka muodostaa tuloksen ja ulostulo saadaan vasta sitten. [14] Aktivaatiofunktion tarkoituksena on muuttaa tulos halutunlaiseksi. Neuroverkolta voidaan haluta ulostulona vaikka vain luku väliltä $[0,1]$. Tällöin aktivaatiofunktiona voisi olla esimerkiksi tapauksessa $\frac{1}{1+Exp[-sum]}$, missä sum tarkoittaa syötteiden ja painokertoimien tulojen summaa. [10]

Neuroni ei vielä yksinään ole tehokas laskentayksikkö. Neuroneja yhdistämällä saadaan aikaiseksi neuroverkko, kuten kuvassa 3.1 oikealla näkyy. Jotta neuroverkko toimisi halutulla tavalla, tulee neuronit ryhmitellä ja yhdistää niitä toisiinsa. Ryhmittely toteutetaan luomalla



Kuva 3.1: Yksittäisen neuronin rakenne ja neuroneista muodostettu neuroverkko. [31]



Kuva 3.2: Neuroverkon kerrokset [10]

neuroverkkoon kerroksia, jotka sitten yhdistetään toisiinsa. Se, miten kerrokset yhdistetään, on tapauskohtaista ratkaistavan ongelman kanssa. Periaatteessa neuroverkolla on kuvan 3.2 kaltainen rakenne. Osa neuroneista on rajapintana ulkomaailmaan saamalla sieltä syötteitä. Osa taas toimii rajapintana antamalla ulostulon. Loput neuroneista ovat näkymättömissä. [28]

Neuroverkko voidaan toteuttaa yksi- (vain sisään- ja ulostulokerros) tai useampikerroksisena (sisään- ja ulostulokerros sekä piilokerros). Piilokerroksia voi olla myös useampia kuin yksi. Yksikerroksisella neuroverkolla voidaan kuvata vain lineaarisia funktioita ja tämän takia käytännön sovellukset vaativatkin usein vähintään kaksikerroksisen verkon. Kerroksien lisääminen lisää verkon laskentakapasiteettia. Myös neuronien lukumäärä kerroksilla ja aktiivaatiofunktioit valitaan aina tehtäväkohtaisesti. [36]

Monikerrosperceptron-neuroverkolla tarkoitetaan useampikerroksista neuroverkkoa, jonka neuronit ovat nimeltään Perceptroneja. Alunperin Perceptronilla tarkoitettiin neuronina, joka oletti syötteiden olevan binäärilukuja ja joka antoi ulostulona vain nolliä tai ykkösiä. [31] Nykyisin MLP-verkko tarkoittaa kuitenkin neuroverkkoa, jonka neuronit antavat jatkuvia arvoja, jossa on useita kerroksia ja jossa ei hyväksytäkään takaisinkytkentöjä [28]. Toinen neuronityyppi, Adaline, on lineaarinen sisältäen yhden kerroksen painokertoimia ja sallien myös jatkuva-arvoiset syötteet. [31]

3.2 Neuroverkon oppiminen

Kun neuroverkko on saatu rakennettua, voidaan aloittaa sen opettaminen. Ensimmäiset painokertoimet valitaan sattumanvaraisesti, minkä jälkeen alkaa opetus-oppimis-prosessi.

Neuroverkon opettamiseen on kaksi tapaa: ohjattu oppiminen (engl. *supervised learn-*

ing) ja mukautuva oppiminen (engl. *unsupervised learning* tai *adaptive learning*). Ohjatussa oppimisessa voidaan käyttää hyväksi neuroverkon ulostuloa. Mukautuvassa oppimisessa neuroverkko toimii pelkkien sisääntuloparametrien avulla. Suurin osa neuroverkoista käyttää ohjattua oppimista. NeuroHausta löytyy piirteitä sekä ohjatusta että mukautuvasta oppimisesta.

3.2.1 Ohjattu oppiminen

Neuroverkon ohjattu oppiminen perustuu verkon neuronien välisten painokertoimien aseteluun. Ohjatussa oppimisessa tiedetään sekä sisääntuloparametrit että ulostulo. Opettaminen aloitetaan syöttämällä sisääntuloparametrit ja asettamalla painokertoimet, minkä jälkeen verkolle syötetään opetusdata. Opetusdatan perusteella saadaan ulostulo, jota verrataan ennalta päätettyyn tai haluttuun ulostuloon. Tämän jälkeen käydään neuroverkko läpi takaperin ja painokertoimien arvoja muutetaan ulostulojen ja haluttujen tavoitearvojen virheen perusteella. Opetusta toistetaan niin kauan, kunnes ulostulo on tarpeeksi lähellä haluttua ulostuloa. [36]

3.2.2 Mukautuva oppiminen

Mukautuvassa oppimisessa neuroverkolle annetaan vain sisääntuloparametrit. Ulostulosta ei siis ole mitään tietoa etukäteen. Verkon on itse päätettävä, minkä piirteiden mukaan se ryhmittelee dataa. [28]

Mukautuvassa oppimisessa ei käytetä hyväksi mitään ulkopuolista tietoa ulostulosta, vaan neuroverkko tarkkailee sisäisesti toimintaansa. Verkko etsii sisääntuloparametreista säännöllisyyksiä tai trendejä ja tekee muutoksia neuroverkon funktion mukaan. [28]

Mukautuvaa oppimista käytetään huomattavasti vähemmän kuin ohjattua oppimista. On olemassa kuitenkin paljon tilanteita, joissa täsmällistä opetusdataa ei ole olemassa: esimerkiksi peleissä pelitilanteet muuttuvat jokaisen vuoron seurauksena. Näissä tilanteissa ei voida käyttää ohjattua oppimista.

Tärkein mukautuvan oppimisen menetelmä on professori Teuvo Kohosen kehittämä itseorganisoituva piirrekartta (engl. *Self-organizing Feature Map, SOM*). SOMia voidaan käyttää moniin tehtäviin kuten datan klusterointiin, ryhmittelyyn tai korrelaatioiden etsimiseen. [31]

4 Optimoinnista

Luvussa käydään läpi optimoinnin peruskäsitteitä.

4.1 Mitä optimointi on?

”The fundamental problem of optimization is to arrive at the best possible decision in any given set of circumstances.” [37]

Perusajatuksena optimoinnissa on, että ongelmalle löydetään paras mahdollinen ratkaisu annetuissa olosuhteissa. Yleensä tämä halutaan tehdä vielä mahdollisimman nopeasti. Optimointiongelmia löytyy kaikkialta: talouselämässä pyritään maksimoimaan yrityksen saamaa voittoa, liikenteessä pyritään löytämään lyhin tai nopein reitti kahden kaupungin välillä, jopa lentokoneen siiven muodon suunnittelussa tarvitaan optimointia.

Alettaessa ratkaista optimointiongelmaa, pyritään se ensin esittämään matemaattisena mallina. Aina matemaattisten mallien käyttö ei kuitenkaan ole suoraan mahdollista. Päätöksentekotilanteessa inhimillinen tekijä voi olla niin voimakas, että matemaattisesta mallista saatu ratkaisu ei sovikaan ratkaisuksi.

Matemaattisesti muotoiltuna optimointiongelmassa on yleensä funktio, jota minimoidaan tai maksimoidaan tiettyjen rajoitteiden suhteen.

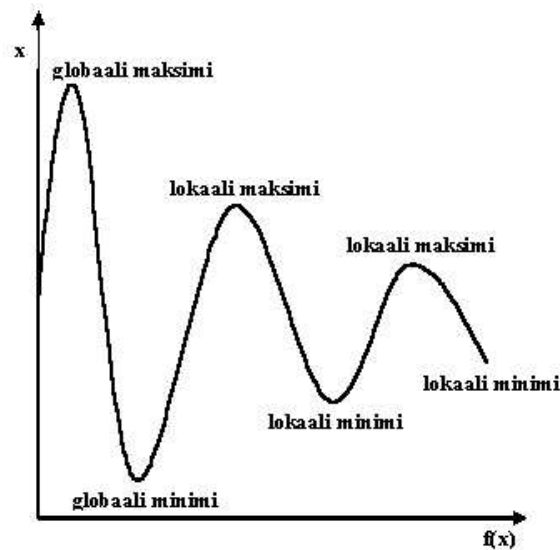
Optimointitehtävä on yleisesti muotoa:

$$\begin{cases} \text{minimoi} & f(\mathbf{x}) \\ \text{rajoittein} & g_i(\mathbf{x}) \leq 0 \quad \text{kaikille } i = 1, \dots, m \\ & h_i(\mathbf{x}) = 0 \quad \text{kaikille } i = 1, \dots, l \end{cases} \quad (4.1)$$

missä f , g_i ja h_i ovat funktioita $\mathbf{R}^n$:stä $\mathbf{R}$:ään.

Optimointitehtävässä (4.1) on tavoitteena löytää sellainen piste $\mathbf{x}$, jossa funktio f saa mahdollisimman pienen arvon (minimiarvon), kun rajoitteet g_i ja h_i ovat voimassa.

Seuraavaksi käydään tarkemmin läpi optimointiin liittyviä termejä ja määritelmiä.



Kuva 4.1: Lokaaleja ja globaaleja optimeja.

4.2 Termejä ja määritelmiä

4.2.1 Objektifunktio, rajoitefunktio ja muuttujat

Optimointitehtävän (4.1) optimoitavaa funktiota f kutsutaan *objektifunktioksi* ja funktioita g_i ja h_i *rajoitefunktioiksi* tai *rajoitteiksi*. Rajoitteet g_i ovat *epäyhtälörajoitteita* ja rajoitteet h_i *yhtälörajoitteita*. Vektorin $\mathbf{x}$ komponentit $\mathbf{x} = (x_1, x_2, \dots, x_n)$ ovat tehtävän *muuttujia*.

4.2.2 Globaali ja lokaali optimi

Piste $\mathbf{x}^*$ on optimointitehtävän *globaali optimi*, jos $f(\mathbf{x}^*) \leq f(\mathbf{x})$ kaikilla $\mathbf{x} \in S$.

Piste $\mathbf{x}^*$ taas on *lokaali optimi*, jos on olemassa $\delta > 0$ siten, että $f(\mathbf{x}^*) \leq f(\mathbf{x})$ kaikilla $\mathbf{x} \in S$, joilla $\|\mathbf{x} - \mathbf{x}^*\| \leq \delta$.

Molemmissa tapauksissa S on sallittu alue eli rajoitteiden leikkausjoukko. Kuvassa 4.1 on esitelty lokaaleja ja globaaleja optimeja.

4.2.3 Minimointi ja maksimointi

Minimointi tarkoittaa objektifunktion pienimmän arvon etsimistä rajoitteiden ollessa voimassa. Maksimoidessa etsitään suurinta arvoa. Tässä työssä tarkastellaan tästä eteenpäin vain maksimointitehtäviä. Minimointitehtävän voi muuttaa maksimointitehtäväksi objektifunk-

tion etumerkkiä vaihtamalla. Eli jos tehtävänä on minimoida f , niin tehtävä on maksimoimitehtävänä maksimoida funktio $-f$.

4.2.4 Unimodaalisuus

Yhden muuttujan funktion f sanotaan olevan *unimodaalinen* välillä $[a, b]$, jos jollekin pisteelle $x^* \in (a, b)$ pätee: $f(x)$ aidosti vähenevä välillä $[a, x^*)$ ja aidosti kasvava välillä $(x^*, b]$.

Funktion unimodaalisuutta tarkasteltavalla välillä vaaditaan monessa optimointimenetelmässä oletuksena.

4.3 Erilaisia optimointitehtäviä

Optimointitehtäviä voidaan jaotella niiden ominaisuuksien perusteella. Seuraavassa on lueteltu erilaisia optimointitehtäviä Miettisen [23] mukaan.

1. Lineaarinen optimointi (engl. *linear programming*) on tehtävä, jonka objektifunktio ja kaikki rajoitteet ovat lineaarisia.
2. Epälineaarinen optimointi (engl. *mathematical programming, nonlinear programming*) tarkoittaa tehtävää, jossa joko objektifunktio tai ainakin yksi tehtävän rajoitteista on epälineaarinen.
3. Kvadraattinen optimointi on epälineaarinen ongelma, jossa on kvadraattinen (toisen asteen yhtälöön tai polynomiin perustuva) objektifunktio ja lineaariset rajoitteet.
4. Geometrisella optimoinnilla tarkoitetaan tehtävää, jonka objektifunktio ja rajoitteet ovat posynomeja eli muotoa

$$\sum_{i=1}^k c_i \prod_{j=1}^n x_j^{p_{ij}}$$

5. Kokonaislukuoptimoinnin tehtävän muuttujat rajoitetaan siten, että ne voivat saada vain kokonaislukuja.
6. Jatkuva optimointi on kokonaislukuoptimoinnin vastakohta, eli muuttujat voivat saada myös desimaaliarvoja.
7. Kombinatorisesta optimoinnista puhutaan silloin, kun tehtävällä on sallittuja ratkaisuja äärellinen määrä.

8. Monitavoiteoptimoinnin tehtävässä on useampi kuin yksi objektifunktio. Objektifunktiot voivat olla keskenään ristiriidassa, mutta siitä huolimatta niitä kaikkia optimoidaan yhtä aikaa. Objektifunktiot ja rajoitteet voivat olla mitä tahansa edellä kuvattuja tyyppisiä.
9. Epäsileällä optimoinnilla tarkoitetaan sitä, että jokin ongelman funktioista ei ole differentioituva.
10. Globaalissa optimoinnissa etsitään tehtävän globaalia optimia.
11. Dynaamisesta optimoinnista puhuttaessa tarkoitetaan pikemminkin ratkaisuperiaatetta kuin mitään erityistä optimointitehtävää. Dynaaminen optimointi soveltuu monivaiheisiin päätöksentekoprosesseihin, jos tehtävä on jaksollinen ja joissa sama rakenne toistuu eri vaiheissa.
12. Optimisäätötehtävää kuvataan kahdenlaisilla muuttujilla: säätömuuttujilla ja tilamuuttujilla. Säätömuuttuja määrittelee systeemin kulun vaiheesta toiseen, kun taas tilamuuttuja määrää systeemin käyttäytymisen eri vaiheissa. Tavoitteena on löytää säätömuuttujille sellaiset arvot, että objektifunktio minimoituu monen vaiheen jälkeen tiettyihin tila- ja säätömuuttujille asetettuihin rajoitteisiin.
13. Stokastisessa optimoinnissa ainakin osa parametreista sisältää satunnaisuutta.
14. Deterministinen optimointi on stokastinen vastakohta, eli satunnaisuutta ei ole ollenkaan.

NeuroHaun tapauksessa optimointiongelma kuuluu luokkiin 2, 6, 8, 9, 10 ja 13. Objektifunktiona on neuroverkon fitness (hyvyys), joka on määritelty luvussa 7.3 ja joka käyttäytyy epälineaarisesti. Objektifunktio ei ole myöskään differentioituva, joten kyseessä on epäsileä optimointitehtävä. Ongelman muuttujina ovat painokertoimet, jotka voivat saada desimaaliarvoja. Monitavoiteoptimointitehtävän NeuroHausta tekee se, että fitness koostuu kahdesta tavoitteesta: löydettyjen resurssien määrää pyritään maksimoimaan ja hakuun käytettyjä paketteja minimoimaan. Ongelman fitness-funktio käyttäytyy stokastisesti, eli eri evaluatiokierroksilla samat muuttujat voivat saada eri fitness-arvoja. Haastavuutta tehtävään tuo myös se, että NeuroHaulla halutaan löytää globaali optimipiste.

5 Optimointimenetelmiä

Optimointimenetelmät voidaan jakaa lineaarisiin ja epälineaarisiin menetelmiin. Optimointitehtävän ratkaisuun voidaan käyttää myös valmiita ohjelmia kuten LINDO [22] ja CPLEX [9] lineaaristen tehtävien ratkaisuun, GINO [13] ja LINGO [22] epälineaarisille tehtäville tai WWW-NIMBUS [38] monitavoitetehtäville.

Lineaarisessa optimointitehtävässä sekä objektifunktio että kaikki rajoitteet ovat siis lineaarisia. Lineaariset optimointitehtävät ovat melko yksinkertaisia. Mikäli rajoitteita ei ole monta, voi lineaarisen kaksimuuttujaisen tehtävän ratkaista jopa graafisesti. Monimutkaisempiin tehtäviin voidaan käyttää erilaisia valmiita algoritmeja, esimerkiksi Simplex-algoritmia [33].

Epälineaarinen optimointi on monimutkaisempaa kuin lineaarinen, koska objektifunktio tai ainakin yksi rajoitteista on epälineaarinen. Myös epälineaariselle optimoinnille on kehitetty omia ratkaisumenetelmiä. Epälineaarisia menetelmiä ovat mm. eliminointi-, interpolointi-, suorahaku-, derivaattoja käyttävä- ja sakkofunktiomenetelmä. Menetelmät on kehitetty erilaisille epälineaarisille tehtäville. [23]

Eliminointimenetelmät soveltuvat yhden muuttujan funktion minimointiin ja niissä on periaatteena etsiä hakuavaruudesta ensin funktiolle unimodaalinen väli, jota sitten eliminoidaan esimerkiksi puolittamalla väliä. Interpolointimenetelmien ideana taas on approksimoida tehtävän objektifunktiota polynomilla, jonka minimikohta on helppo löytää. Myös interpolointimenetelmissä tulee funktiolle ensin etsiä unimodaalinen väli, jota tarkastellaan. Myös interpolointimenetelmät sopivat yhden muuttujan funktion minimointiin. [23]

Suorahakumenetelmiä ja derivaattoihin perustuvia menetelmiä voidaan käyttää usean muuttujan rajoittamattomaan optimointiin. Suorahakumenetelmät generoivat suuntia, joihin optimointi etenee. Alkeellisimmat suorahakumenetelmät, kuten satunnaisten suuntien menetelmä, ovat melko tehottomia, mutta kehittyneimmät menetelmät, esimerkiksi Powellin menetelmä, pystyvät jopa yhtä hyvin tuloksiin kuin derivaattoja käyttävät menetelmät. Derivaattoihin perustuvat menetelmät käyttävät optimoinnissa nimensä mukaan apuna objektifunktion derivaattaa. Derivaatta helpottaa optimointia, mutta sen laskeminen on usein hyvin työläs ja aikaavievä operaatio. [23]

Sakkofunktiomenetelmissä on periaatteena, että objektifunktiota sakotetaan, jos se poistuu sallitulta alueelta tai edes pyrkii sen reunalle. Tämä tehdään lisäämällä objektifunktioon

sakkotermi. Sakkofunktiomenetelmät soveltuvat usean muuttujan funktion rajoitteiseen tehtävään. [23]

Evoluutioalgoritmit ovat toiminnaltaan melko erilaisia kuin lineaariset- tai epälineaariset optimointimenetelmät. Evoluutioalgoritmit ovat saaneet vaikutteita biologiasta ja ne jäljittelevät toiminnaltaan evoluutiota perustuen populaatioihin, mutaatioihin, risteytykseen ja jälkeläisiin.

Tässä työssä keskitytään evoluutioalgoritmeihin ja epälineaarisista menetelmistä suorahakumenetelmiin, koska ne vaikuttivat soveliaimmilta NeuroHaun optimointiongelmaan. Evoluutioalgoritmeista ja suorahakumenetelmistä kerrotaan lisää seuraavissa luvuissa.

5.1 Epälineaarinen optimointi

Seuraavassa esitellään kolme suorahakumenetelmää, joita tutkimuksessa käytettiin NeuroHaun optimointiin.

5.1.1 Powellin menetelmä

Powellin menetelmä on lokaali suorahakumenetelmä. Suorahakumenetelmissä ei käytetä apuna funktion derivaattainformaatiota. Powellin menetelmä on kuitenkin kehittynein ns. pattern search -menetelmä, ja tehokkuudessaan se pystyy kilpailemaan derivaattoja käyttävien menetelmien kanssa.

Menetelmä pohjautuu Powellin [26] kahteen teoreemaan:

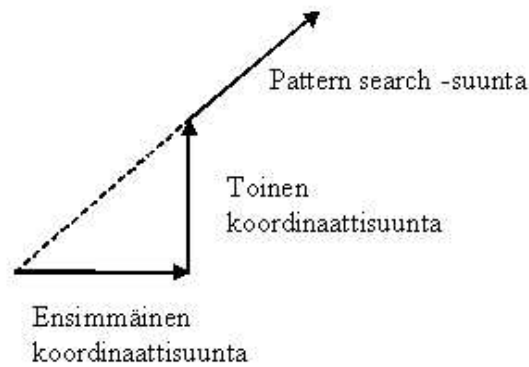
Olkoon f kvadraattinen siten, että $f(x) = xAx + bx + c$.

Teoreema 5.1 Minimoidaessa toisen asteen kvadraattista funktiota $f(x)$ peräkkäin n :ssä konjugaattisessa suunnassa minimi löydetään viimeistään n :llä askeleella riippumatta aloituspisteestä.

Teoreema 5.2 Jos x_0 on kvadraattisen funktion $f(x)$ minimi avaruudessa, joka sisältää suunnan q , ja x_1 on myös saman avaruuden minimi, niin suunta $(x_1 - x_0)$ on konjugaattinen suunnan q kanssa.

Todistukset löytyvät Powellin artikkelista. Näiden teoreemien pohjalta saadaan seuraava algoritmi [39]:

1. Olkoot $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_n$ koordinaattisuunnat ja $\mathbf{p}_0$ aloituspiste.
2. Laske λ_r siten, että se minimoi funktion $f(\mathbf{p}_{r-1} + \lambda_r \mathbf{d}_r)$, kun $r = 1, 2, \dots, n$ ja määritä $\mathbf{p}_r = \mathbf{p}_{r-1} + \lambda_r \mathbf{d}_r$.



Kuva 5.1: Pattern search- suunnan muodostaminen.

3. Aseta $\mathbf{d}_r = \mathbf{d}_{r+1}$, $r = 1, \dots, n - 1$ ja aseta $\mathbf{d}_n = (\mathbf{p}_n - \mathbf{p}_0)$.
4. Valitse λ siten, että se minimoi funktion $f(\mathbf{p}_n + \lambda(\mathbf{p}_n - \mathbf{p}_0))$. Aseta $\mathbf{p}_0 = \mathbf{p}_0 + \lambda(\mathbf{p}_n - \mathbf{p}_0)$ ja aloita uusi iteraatio kohdasta 2.

Algoritmiin pitää vielä lisätä lopetusehto. Algoritmi voidaan joko lopettaa generaatioiden lukumäärän perusteella tai sitten, kun algoritmilla saadut kaksi peräkkäistä optimipistettä ovat ennalta valitun $\epsilon > 0$ päässä toisistaan.

Powellin menetelmä aloitetaan minimoimalla objektifunktiota koordinaattiakselien suuntiin. Aloituspisteestä lähdettäessä parannetaan siis yhtä muuttujan arvoa kerrallaan niin paljon kuin mahdollista. Seuraavasta pisteestä lähdettäessä parannetaan seuraavan muuttujan arvoa ja niin edelleen. Näin jatketaan, kunnes kaikki muuttujat on käyty läpi. Tätä kutsutaan kokeiluvaiheeksi. Kokeiluvaiheen jälkeen seuraa ns. pattern search -vaihe. Tällöin muodostetaan uusi etenemissuunta kokeiluvaiheen perusteella ja mennään uuteen pisteeseen. Pattern search -suunnan muodostumista kaksidimensioisessa tapauksessa on esitetty kuvassa 5.1. Jokaisen pattern search -vaiheen jälkeen yksi koordinaattisuunnista korvataan pattern search -suunnalla.

Menetelmä löytää pisteen $\mathbf{x}$, jota minimoi objektifunktion f . Powell väittää, että tämä tapahtuu n kierroksen kuluessa, jos n hakusuuntaa ovat konjugaattisia ja kun objektifunktio on kvadraattinen.

Väite ei kuitenkaan pidä paikkaansa, kuten Zangwill [39] on osoittanut. Zangwillen artikkelissa esitetään vastaesimerkki, jossa Powellin menetelmä ei konvergoi minimiin ikinä.

Jos funktioksi valitaan $f(x, y, z) = (x - y + z)^2 + (-x + y + z)^2 + (x + y - z)^2$ ja aloituspisteeksi $(1/2, 1, 1/2)$ ei Powellin menetelmällä päästä funktion f minimiin, joka olisi pisteessä $(x, y, z) = (0, 0, 0)$. [39]

Powellin menetelmän suurimpana ongelmana on, että yksiulotteiset minimoinnit konjugaattisiin suuntiin tulisi tehdä tarkasti, mikä on mahdotonta numeeristen menetelmien äärellisyydestä/epätarkkuudesta johtuen. Powellin menetelmä edellyttää myös, että hakusuuntien tulisi olla lineaarisesti riippumattomia. Tämä ei kuitenkaan täyty, jos optimaalinen askelpituus johonkin hakusuuntaan on 0. [23]

5.1.2 Powellin toinen menetelmä

Powell kehitti toisen menetelmän, joka ottaa huomioon ensimmäisessä menetelmässä havaitut virheet. Ensimmäinen menetelmä saattoi joskus muodostaa lineaarisesti riippuvat suunnat, jotka eivät viritä koko avaruutta. Korjatakseen tämän Powell esittää, että ei ole viisasta hyväksyä jokaista menetelmän antamaa suuntaa uudeksi hakusuunnaksi.

Zangwill on muotoillut Powellin toisesta menetelmästä yksinkertaistetun version [39]:

1. Olkoot koordinaattisuunnat $\mathbf{d}_1^1, \mathbf{d}_2^1, \dots, \mathbf{d}_n^1$, aloituspiste $\mathbf{p}_0^1$ ja kerroin $\epsilon, 1 \geq \epsilon > 0$ annettu. Oletetaan myös, että suunnat on normalisoitu siten, että $\|\mathbf{d}_r^1\| = 1, r = 1, \dots, n$. Aseta $\delta^1 = 1$. Mene iteraatioon k arvolla $k = 1$.
2. Iteraatio k :
 - (a) Laske λ_r^k siten, että se minimoi funktion $f(\mathbf{p}_{r-1}^k + \lambda_r^k \mathbf{d}_r^k)$, kun $r = 1, 2, \dots, n$ ja määritä $\mathbf{p}_r^k = \mathbf{p}_{r-1}^k + \lambda_r^k \mathbf{d}_r^k$.
 - (b) Määritä $\alpha^k = \|\mathbf{p}_n^k - \mathbf{p}_0^k\|$ ja $\mathbf{d}_{n+1}^k = (\mathbf{p}_n^k - \mathbf{p}_0^k)/\alpha^k$. Laske λ_{n+1}^k funktion $f(\mathbf{p}_n^k + \lambda_{n+1}^k \mathbf{d}_{n+1}^k)$ minimiksi ja aseta $\mathbf{p}_0^{k+1} = \mathbf{p}_{n+1}^k = \mathbf{p}_n^k + \lambda_{n+1}^k \mathbf{d}_{n+1}^k$.
 - (c) Olkoon $\lambda_s^k = \max\{\lambda_r^k \mid r = 1, \dots, n\}$.
Tapaus (i): Jos $\lambda_s^k \delta^k / \alpha^k \geq \epsilon$, olkoon $\mathbf{d}_r^{k+1} = \mathbf{d}_r^k$, kun $r \neq s$, $\mathbf{d}_s^{k+1} = \mathbf{d}_{n+1}^k$ ja aseta $\delta^{k+1} = \lambda_s^k \delta^k / \alpha^k$.
Tapaus (ii): Jos $\lambda_s^k \delta^k / \alpha^k < \epsilon$, olkoon $\mathbf{d}_r^{k+1} = \mathbf{d}_r^k$, kun $r = 1, \dots, n$ ja aseta $\delta^{k+1} = \delta^k$.
3. Jos $k = n$, lopeta. Muuten mene askeleeseen 2 arvolla $k = k + 1$.

5.1.3 Zangwillen menetelmä

Zangwille on myös kehittänyt oman menetelmänsä poistaakseen Powellin ensimmäisessä menetelmässä havaitut epäkohdat. Algoritmi on seuraava [39]:

1. Olkoot $\mathbf{c}_r$, $r = 1, \dots, n$ koordinaattisuunnat ja oletetaan, että ne on normalisoitu.

Alkuaskel: Olkoon lähtöpiste $\mathbf{p}_n^0$ ja n normalisoitua suuntaa $\mathbf{d}_r^1$, $r = 1, \dots, n$ annettu.

Olkoon λ_n^0 tehtävän minimoi $f(\mathbf{p}_n^0 + \lambda_n^0 \mathbf{d}_n^1)$ optimiratkaisu ja $\mathbf{p}_{n+1}^0 = \mathbf{p}_n^0 + \lambda_n^0 \mathbf{d}_n^1$.

Aseta $t = 1$ ja mene iteraatioon k arvolla $k = 1$.

2. Iteraatio k : $\mathbf{p}_{n+1}^{k-1}$, $\mathbf{d}_r^k$, $r = 1, \dots, n$ ja t on annettu.

- (a) Askel (i): Olkoon α tehtävän minimoi $f(\mathbf{p}_{n+1}^{k-1} + \alpha \mathbf{c}_t)$ optimiratkaisu. Päivitä t .

$$t \leftarrow \begin{cases} t + 1 & \text{jos } 1 \leq t < n \\ 1 & \text{jos } t = n \end{cases}$$

Jos $\alpha \neq 0$. Olkoon $\mathbf{p}_0^k = \mathbf{p}_{n+1}^{k-1} + \alpha \mathbf{c}_t$. Jos $\alpha = 0$, toista askel(i). Jos askel(i) toistetaan n kertaa peräkkäin, lopeta. Tällöin piste $\mathbf{p}_{n+1}^{k-1}$ on optimi.

- (b) Askel (ii): Kun $r = 1, \dots, n$, laske λ_r^k siten, että se minimoi tehtävän $f(\mathbf{p}_{r-1}^k + \lambda_r^k \mathbf{d}_r^k)$. Aseta $\mathbf{p}_r^k = \mathbf{p}_{r-1}^k + \lambda_r^k \mathbf{d}_r^k$.

Olkoon $\mathbf{d}_{n+1}^k = (\mathbf{p}_n^k - \mathbf{p}_{n+1}^{k-1}) / \|\mathbf{p}_n^k - \mathbf{p}_{n+1}^{k-1}\|$.

Määritä λ_{n+1}^k tehtävän minimoi $f(\mathbf{p}_n^k + \lambda_{n+1}^k \mathbf{d}_{n+1}^k)$ optimiratkaisuksi ja aseta $\mathbf{p}_{n+1}^k = \mathbf{p}_n^k + \lambda_{n+1}^k \mathbf{d}_{n+1}^k$.

Aseta $\mathbf{d}_r^{k+1} = \mathbf{d}_{r+1}^k$, $r = 1, \dots, n$.

Mene kohtaan (2) k :n arvolla $k + 1$.

Menetelmässä askeleessa (i) edetään jaksollisesti koordinaattisuunnat läpi. Joka kerta, kun palataan askeleeseen (i) valitaan uusi koordinaattisuunta palaamalla $\mathbf{c}_1$:een $\mathbf{c}_n$:n jälkeen. Joka $n + 1$:llä kerralla, kun käytetään askelta (i) käytetään myös samaa koordinaattisuuntaa. Algoritmissa t osoittaa käytettävää koordinaattisuuntaa. Jos askel (i) toistetaan n kertaa peräkkäin, niin kaikki n koordinaattisuuntaa on käyty läpi ja piste on pysynyt samana. Tällöin optimipiste on löydetty. [39]

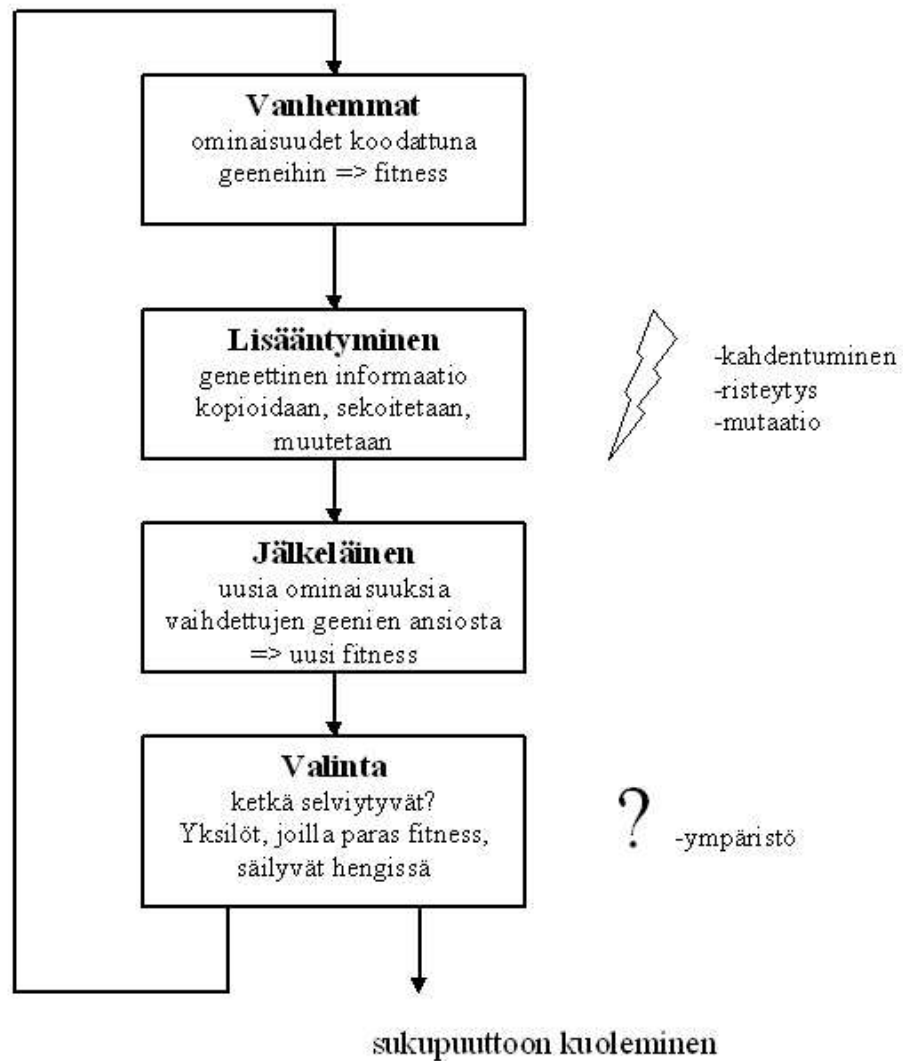
5.2 Evoluutioalgoritmit

Evoluutioalgoritmit (engl. *evolutionary algorithms, EA*) ovat saaneet vaikutteita biologiasta ja evoluutioteoriasta ja ne perustuvat populaatioihin, mutaatioihin, risteytykseen ja jälkeläisiin. Evoluutioalgoritmit jäljittelevät toiminnaltaan evoluutiota. Yksilöt lisääntyvät ja saavat jälkeläisiä, joista heikoimmat kuolevat ja parhaat selviävät. Nämä jälleen lisääntyvät ja uusista jälkeläisistä taas osa kuolee ja osa säilyy hengissä. Evoluutioalgoritmeissa on käytössä "evoluutiokierrokset", joita kutsutaan myös sukupolviksi (engl. *generation*) tai generaatioiksi. Evoluutiokierroksilla tapahtuu mm. lisääntymistä ja heikoimpien karsintaa.

Evoluutioalgoritmeissa yksilön sanotaan koostuvan geeneistä, jotka ovat optimointitehtävän muuttujat. Yksilö on yleensä kuvattu vektorina. Geenien perusteella yksilölle lasketaan fitness- eli hyvyysarvo, jonka perusteella evoluutio etenee. Vanhemmat ovat yksilöitä, joiden geenejä muuttamalla saadaan aikaan jälkeläisiä. Seuraavalla evoluutiokierroksella jälkeläiset ovat vanhempia. Evoluutiokierrosten aikana yksilöt ja niiden fitness-arvot muuttuvat ja lähestyvät tehtävän optimipistettä.

Evoluutioalgoritmien perusidea selviää kuvasta 5.2. Vanhempien geeneissä on koodattuna jotain ominaisuuksia, joiden perusteella saadaan laskettua niiden fitness (hyvyys). Vanhemmat lisääntyvät, jolloin heidän geeninsä sekoittuvat, kopioituvat ja muuntuvat. Tässä vaiheessa voi tapahtua myös eri evoluutioalgoritmista riippuen mutaatiota, risteytymistä tai kahdentumista. Muutettujen geenien ansiosta jälkeläisillä on hieman erilaiset ominaisuudet kuin heidän vanhemmillaan. Jälkeläisille lasketaan myös fitness, joka voi erota paljonkin vanhempien fitnessseistä. Jälkeläisistä vain osa selviytyy ja pääsee jatkamaan evoluutioketjua. Se, miten valinta jälkeläisistä tehdään, riippuu jälleen käytettävästä evoluutioalgoritmista. Biologiassa myös ympäristöllä on vaikutusta yksilöiden hengissäsäilymiseen. Aina paras yksilö ei säilykään hengissä, vaan se voi kuolla jonkin ennalta arvaamattoman tapauksen seurauksena.

Suurin osa evoluutioalgoritmien toteutuksista on peräisin kolmesta toisiaan muistuttavasta, mutta itsenäisesti kehitetyistä menetelmistä: geneettisistä algoritmeista (engl. *genetic algorithms, GA*), evoluutiostrategioista (engl. *evolutionary strategies, ES*) ja evoluutio-ohjelmoinnista (engl. *evolutionary programming, EP*). Nämä menetelmät perustuvat erilaisiin geneettisiin operaatioihin, kuten mutaatioon (engl. *mutation*), risteytykseen (engl. *crossover*) ja valintaan (engl. *selection*). Seuraavassa esitellään ensin yleisesti geneettiset operaatiot ja sen jälkeen niiden käyttöä eri evoluutioalgoritmeissa.



Kuva 5.2: Yleinen evoluutioalgoritmi. [2]

5.2.1 Mutaatio

Mutaatiolla tuotetaan uusia jälkeläisiä muuttamalla vanhempien geenejä satunnaisesti. Mutaation avulla optimointialgoritmi etsii funktion lokaalia optimipistettä.

Mutaation toteutukseen on olemassa monia erilaisia menetelmiä, esimerkiksi Gaussin-mutaatio [16], Cauchy-mutaatio [18] ja Lévy-mutaatio [20]. Lisäksi mutaatiomenetelmiä voidaan yhdistellä. Periaatteena kaikissa on, että otetaan jokin satunnaisluku, jolla kertomalla geenien arvoja muutetaan. Gaussin-mutaatiossa käytetään, kuten nimikin kertoo, Gaussin normaalijakauman avulla tuotettua satunnaislukua. Cauchyn-mutaatiossa normaalijakauma on korvattu Cauchyn jakaumalla. Jälkeläiset x'_i saadaan siis luotua Gaussin tapauksessa seuraavasti:

$$x'_i = x_i + N(0, 1) \quad (5.1)$$

Ja Cauchyn-mutaatiossa:

$$x'_i = x_i + C(0, 1) \quad (5.2)$$

Kaavoissa $N(0, 1)$ on normaalijakautunut satunnaisluku, $C(0, 1)$ on Cauchyn-jakaumasta arvottu satunnaisluku ja $i = 1, \dots, n$.

Gaussin- ja Cauchyn-mutaatioihin voidaan lisätä myös strategiaparametri, joka määrittelee, kuinka isoja hyppyjä kullakin iteraatiokierroksella tehdään. Strategiaparametri on usein itsemukautuva. Tällöin mutaation kaavoiksi saadaan:

$$\sigma'_i = \sigma_i \exp(\tau' N_i(0, 1) + \tau N(0, 1)) \quad (5.3)$$

$$\mathbf{x}'_i = \mathbf{x}_i + \sigma_i \delta \quad (5.4)$$

missä $\forall i \in \{1, \dots, \mu\}$. Parametrivektori σ on nyt siis itsemukautuva. Kaavoissa δ on satunnainen luku, joka arvotaan jokaiselle populaation yksilölle mutaatiomenetelmän jakauman perusteella: normaalijakauma Gaussin-mutaatiossa ja Cauchyn-jakauman mukaan Cauchyn-mutaatiossa. $\tau = (\sqrt{2\sqrt{n}})^{-1}$ ja $\tau' = (\sqrt{2n})^{-1}$. Populaatiovektori $\mathbf{x} \in \mathbf{R}^n$ ja $N(0, 1)$ on normaalijakautunut satunnaisluku. [18]

Lévy-mutaatiossa tuotetaan neljä jälkeläistä käyttämällä neljää eri Lévy todennäköisyyttä, minkä jälkeen paras jälkeläinen valitaan populaatioon. Myös Lévy-mutaatiossa on käytössä strategiaparametri. Jälkeläiset saadaan seuraavilla kaavoilla [18]:

$$\mathbf{x}'_{1i} = \mathbf{x}_{1i} + \sigma_i L(\alpha = 1.0) \quad (5.5)$$

$$\mathbf{x}'_{2i} = \mathbf{x}_{2i} + \sigma_i L(\alpha = 1.3) \quad (5.6)$$

$$\mathbf{x}'_{3i} = \mathbf{x}_{3i} + \sigma_i L(\alpha = 1.7) \quad (5.7)$$

$$\mathbf{x}'_{4i} = \mathbf{x}_{4i} + \sigma_i L(\alpha = 2.0) \quad (5.8)$$

$$\sigma'_i = \sigma_i \exp(\tau' N_i(0, 1) + \tau N(0, 1)) \quad (5.9)$$

missä $\forall i \in \{1, \dots, \mu\}$, jossa $L(\alpha)$ on satunnaisluku vastaten Lévy-jakaumaa arvoilla 1.0, 1.3, 1.7 ja 2.0. Parametrin arvo $\alpha = 1.0$ vastaa Cauchyn- ja $\alpha = 2.0$ Gaussin-mutaatiota. Muut muuttujat ovat samoja kuin kaavoissa (5.3) ja (5.4).

Evoluutiostrategioissa ja evoluutio-ohjelmoinnissa mutaatioon käytetään yllä mainittujen kaltaisia menetelmiä. Geneettisissä algoritmeissa sen sijaan käytetään hieman erilaista mutaatiota, josta on kerrottu tarkemmin luvussa 5.2.4.

5.2.2 Rekombinaatio

Rekombinaatio eli risteytys on mutaation ohella toinen tapa lisääntyä. Rekombinaatiossa vanhempien geenit sekoitetaan risteyttämällä. Toisin kuin mutaatioissa, nyt lisääntymiseen tarvitaan siis vähintään kaksi vanhempaa.

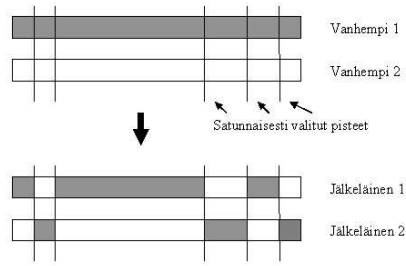
Risteytykseen on olemassa monia erilaisia tapoja, yksinkertaisin niistä on yksipisteristeytys (engl. *one-point crossover*). Yksipisteristeytyksessä valitaan satunnaisesti kohta j , josta lähtien vanhempien geenit vaihdetaan. Olkoot $\mathbf{x}$ ja $\mathbf{y}$ vanhemmat. Tällöin risteytys kohdasta j tapahtuu seuraavasti [4]:

$$\begin{aligned} &< x_1 x_2 \dots x_j x_{j+1} \dots x_n > + < y_1 y_2 \dots y_j y_{j+1} \dots y_n > = \\ &< x_1 x_2 \dots x_j y_{j+1} \dots y_n > \text{ ja } < y_1 y_2 \dots y_j x_{j+1} \dots x_n > \end{aligned}$$

Kaksipisteristeytyksessä (engl. *two-point crossover*) taas valitaan satunnaisesti kaksi pistettä, joiden väliset geenit vaihdetaan. Olkoon satunnaisesti valitut pisteet i ja j , jolloin risteytys näyttää seuraavalta:

$$\begin{aligned} &< x_1 x_2 \dots x_i x_{i+1} \dots x_{j-1} x_j \dots x_n > + < y_1 y_2 \dots y_i y_{i+1} \dots y_{j-1} y_j \dots y_n > = \\ &< x_1 x_2 \dots x_i y_{i+1} \dots y_{j-1} x_j \dots x_n > \text{ ja } < y_1 y_2 \dots y_i x_{i+1} \dots x_{j-1} y_j \dots y_n > \end{aligned}$$

Pisteiden lukumäärää voidaan tästäkin vielä kasvattaa, jolloin kyseessä on monipisteristeytys (engl. *multi-point crossover*). Tällöin valitaan satunnaisesti $z \geq 1$ pistettä ja risteytyksessä vaihdetaan joka toinen pisteiden väliin jäävä segmentti vanhempien kesken. [5] Tästä on esimerkki kuvassa 5.3, jossa on esitetty 5-pisteristeytys.



Kuva 5.3: 5-pisteristeytys.

Lisäämällä pisteitä tästäkin, päästään tasaiseen risteytykseen (engl. *uniform crossover*). Nyt ei enää vaihdeta segmenttejä, vaan tarkastellaan jokaista geeniä erikseen. Jokaisen geenin kohdalla arvotaan, vaihdetaanko se vanhempien kesken vai ei. Matemaattisesti ilmaistuna:

$$a'_i = \begin{cases} a_{M,i}, & \chi > 1/2 \\ a_{F,i}, & \chi \leq 1/2 \end{cases}$$

missä M (mother) ja F (father) ilmaisevat satunnaisesti valittuja vanhempia ja $\chi \in [0, 1]$ on satunnaisluku. [5]

5.2.3 Valinta

Valinnan tarkoituksena on valita populaatiosta ne, jotka selviävät seuraavalle evoluutiokierrokselle. Valitut ovat siis seuraavassa generaatiossa vanhempia. Valinta tehdään yksilöiden fitnessien perusteella.

Valintaan on olemassa monia erilaisia menetelmiä. Evoluutiostrategialla on selkeästi omanlaisensa valintamenetelmät, kun taas geneettisissä algoritmeissa ja evoluutio-ohjelmoinnissa käytetään pääasiassa todennäköisyyteen perustuvia menetelmiä.

Geneettisillä algoritmeilla valintaan on olemassa kaksi hyvin suosittua toteutustapaa: rulettivalinta (engl. *roulette selection*) ja deterministinen kokeilu (engl. *deterministic sampling*). Rulettivalinnassa lasketaan jokaiselle yksilölle todennäköisyys seuraavasti:

$$F_i = \frac{f_i}{\sum_j f_j}$$

missä f_i on yksilön i fitness-arvo. Tämän jälkeen vanhempien populaatio valitaan satunnaisesti tämän todennäköisyyden perusteella. Deterministisessä kokeilussa taas lasketaan jokaiselle yksilölle i arvo:

$$C_i = RND(F_i n) + 1$$

missä n on yksilöiden lukumäärä populaatiossa ja RND tarkoittaa pyöristämistä kokonaisluvuksi. Tämän jälkeen valintaoperaatio varmistaa, että jokainen yksilö tulee valittua vanhempien populaatioon juuri C_i kertaa. [4]

Myös turnajaisvalinta (engl. *tournament selection*) perustuu todennäköisyyteen. Periaatteenä on valita populaatiosta satunnaisesti q yksilöä, joista paras selviää seuraavalle kierrokselle. Yleisimmin käytetty turnajaiskoko on $q = 2$, mutta myös muita voidaan käyttää. [5]

Toinen tapa turnajaisvalinnan toteuttamiselle on valita populaatiosta järjestyksessä kaikki yksilöt yksi kerrallaan ja yksilölle satunnaisesti populaatiosta vastustajat. Tämän jälkeen tutkitaan, montako vastustajaa yksilö voittaa, eli moneen vastustajaan verrattuna sillä on parempi fitness. Jatkoon pääsevät ne yksilöt, jotka ovat menestyneet turnajaisissa parhaiten, eli voittaneet eniten vastustajia. Tässäkin tapauksessa turnajaiskoko q voi vaihdella, mutta yleisimmin käytetään arvoa $q = 2$. [5]

Kaikille todennäköisyyksiin perustuvilla valintaoperaatioilla ainoa vaatimus on se, että todennäköisyyksien summan tulee olla 1. [5]

Evoluutiostратегioissa käytetään ns. elitistisiä valintamenetelmiä, jolloin ei käytetä todennäköisyyksiä, vaan parhaat yksilöt pääsevät automaattisesti jatkoon. EA:ssa on olemassa kaksi valintatapaa: ”+”-valinta ja ”,”-valinta, joita merkitään yleisesti (μ^+, δ) . Ero (μ, δ) - ja $(\mu + \delta)$ -valinnalla on siinä, ketkä kaikki valintaan osallistuvat. (μ, δ) -valinnassa vain evoluutiossa syntyneet jälkeläiset (joita merkitään δ) ovat mukana. $(\mu + \delta)$ -valinnassa taas valitaan sekä syntyneistä jälkeläisistä että heidän vanhemmistaan (joita merkitään μ). Tällöin vanhemmat voivat säilyä hengissä siis useamman evoluutiokierroksen ajan. Molemmilla tavoilla on sekä hyvät että huonot puolensa. (μ, δ) -valintaa suositellaan reaality-luku-parametrien optimointiin ja $(\mu + \delta)$ -valintaa kombinatorisiin ongelmiin. [2]

5.2.4 Geneettiset algoritmit

Geneettiset algoritmit (engl. *genetic algorithms*, GA) otti ensimmäisen kerran käyttöön Holland 1960-luvulla. GA:ta esitettiin alunperin mukautuvan prosessin yleiseksi malliksi, mutta nykyisin suurin osa sitä käyttävistä sovelluksista on optimoinnin alalla. [6]

GA:ssa yksilön sanotaan koostuvan kromosomeista .

$$\langle x_1, x_2, \dots, x_n \rangle$$

jotka taas koostuvat geeneistä. Kromosomi on n :n pituinen vektori, jonka komponentit x_i

ovat siis geenejä. Geeneiksi valitaan ratkaistavan ongelman parametrit. GA:ssa geenit kuvataan usein binäärilukuina $\{0,1\}$. Kromosomi voi tällöin näyttää esimerkiksi tältä: $\langle 100010011 \rangle$. Geenit voidaan kuvata kuitenkin myös reaalilukuina, kuten Whitley, Dominic ja Das toteavat Genetic Algorithms teoksessa [4].

Kuten muidenkin evoluutioalgoritmien, myös GA:n toiminta perustuu geneettisiin operaatioihin. GA:ssa nämä ovat: rekombinaatio, mutaatio ja valinta. [4] Geneettiset algoritmit eroaa muista evoluutioalgoritmeista siinä, että mutaatio tehdään geeneittäin ja mutaatiossa ei käytetä strategiaparametria. Evoluutiostrategioissa ja evoluutio-ohjelmoinnissa mutatoidaan kerralla koko yksilö ja mutaatiosta käytetään hyvin usein strategiaparametrin sisältäviä versioita.

Ensimmäisenä tehdään rekombinaatio, jossa vanhempien-populaation (eli edelliseltä generaatiokierrokselta jatkoon päässeiden yksilöiden) geenejä risteytetään ja näin saadaan aikaan uusia jälkeläisiä. Risteytyksessä vanhempien populaatiosta otetaan kaksi vanhempaa (äiti ja isä), joiden geenit risteytetään. GA:ssa risteytykseen voidaan käyttää kaikkia luvussa 5.2.2 kuvattuja menetelmiä. Käytetyin risteytystapa lienee kaksipisteristeytys. [27]

Geneettisten algoritmien mutaatio eroaa hieman evoluutiostrategioissa ja evoluutio-ohjelmoinnissa käytettävästä mutaatiosta. Koska GA:ssa yksilö on tavallisesti kuvattu binäärilukuektorina, riittää mutaatioksi pelkkä geenien vaihtaminen 1:stä 0:aan tai päinvastoin. GA:ssa ei myöskään mutatoida kerralla koko yksilöä kuten muissa evoluutioalgoritmeissa, vaan mutaatio tehdään geeneittäin siten, että geeneillä on tietty todennäköisyys tulla muutetuksi.

Valintaan GA:ssa käytetään pääasiassa todennäköisyyteen perustuvia menetelmiä. Luvussa 5.2.3 kuvatuista menetelmistä suosituimpia ovat rulettivalinta ja turnajaisvalinta. [27]

Geneettisten algoritmien perusalgoritmi on seuraavanlainen:

1. $t = 0$
2. Alusta populaatio satunnaisesti
3. Arvioi populaation yksilöiden fitness
4. **while** ($t < max_iteraatio$) **do**
 - Risteytä
 - Mutato geeneittäin
 - Arvioi populaation yksilöiden fitness
 - Valitse todennäköisyyteen perustuvalla valinnalla seuraavalle evoluutiokierrokselle selviytyvät yksilöt
 - $t++$

Geneettistä algoritmia voi säätää ratkaistavan ongelman mukaan. Grefenstette mainitsee [4]:ssä, että algoritmin toimintaan vaikuttavat mm. populaation koko, kuinka usein risteytetään, mutaation voimakkuus, kuinka monta prosenttia populaatiosta saa vaihtua yhdessä generaatiossa ja valintaoperaatio.

5.2.5 Evoluutiostrategiat

Evoluutiostrategiat (engl. *evolution strategies*, *ES*) ovat kehittäneet Rechenberg ja Schwefel 1970-luvun puolivälissä. Alunperin ES kehitettiin ratkomaan vaikeita diskreettejä sekä jatkuvia parametrien optimointiongelmia. [6]

ES:ssä käytettävät geneettiset operaatiot ovat: valinta, mutaatio ja rekombinaatio. [2] Näistä ensimmäisenä tehdään rekombinaatio, johon ES:ssä käytetään luvussa 5.2.2 esitellyjen risteytysmenetelmien sijasta usein diskreettiä rekombinaatiota (engl. *discrete recombination*) tai keskiarvorekombinaatiota (engl. *intermediate recombination*). Diskreetissä rekombinaatiossa valitaan satunnaisesti jokaisen geenin kohdalla, kummalta vanhemmalta se kopioidaan jälkeläiselle. Keskiarvorekombinaatiossa jälkeläisen geeni saadaan laskemalla vanhempien vastaavien geenien keskiarvo. ES:ssä yhden jälkeläisen luomiseen voidaan käyttää myös koko populaation kaikkia yksilöitä, jolloin puhutaan panmictic-tapauksesta. [5]

Rekombinaation jälkeen tehdään mutaatio, joka on valinnan lisäksi tärkein ES:n operaatiosta. Mutaatio-operaatiot ovat ongelmariippuvaisia ja niiden oikeanlaisella suunnittelulla on kriittinen vaikutus koko algoritmin toimintaan. Mutaatioon käytetään yleisesti luvussa 5.2.1 kuvatuista menetelmistä strategiaparametrin sisältäviä versioita. [2]

Lopuksi valintaoperaatio valitsee yksilöistä osan jatkoon. Evoluutiostrategioiden valintamenetelmät eroaa geneettisten algoritmien ja evoluutio-ohjelmoinnin valintamenetelmistä. ES:ssä käytetään todennäköisyyteen perustuvien menetelmien sijasta elitististä valintaa, jolloin parhaat yksilöt pääsevät siis suoraan jatkoon. ES:ssä valintamenetelmiä merkitään $(+)$.

ES:n perusalgoritmi on seuraava [2]:

1. $t = 0$
2. Alusta populaatio satunnaisesti
3. Arvioi populaation yksilöiden fitnessit
4. **while** ($t < max_iteraatio$) **do**
 Rekombinaatio
 Mutaatio

```
Arvioi populaation yksilöiden fitnessit
Valitse (+)-valinnalla yksilöt seuraavalle kierrokselle
t++
```

Evoluutiostrategiat eroaa siis muista evoluutioalgoritmeista risteytyksen ja valinnan osalta. ES:ssä risteytys toteutetaan diskreettinä tai keskiarvorekombinaationa, joita ei GA:ssa ja EP:ssä käytetä. Valintaan ES:ssä käytetään vain elitistisiä menetelmiä, kun taas geneettisissä algoritmeissa ja evoluutio-ohjelmoinnissa voidaan käyttää myös todennäköisyyteen perustuvia valintamenetelmiä.

5.2.6 Evoluutio-ohjelmointi

Evoluutio-ohjelmointi (engl. *evolutionary programming, EP*) syntyi yrityksestä luoda tekoälyä. EP:n on luonut L. J. Fogel 1960-luvun alussa. Evoluutio-ohjelmoinnilla pystyttiin alunperin ratkomaan vain diskreettejä ongelmia. [6] 1980-luvun lopulla hänen poikansa D. B. Fogel laajensi EP:tä toimimaan myös jatkuvien parametrien optimointiin. EP:llä on paljon yhtäläisyyksiä ES:n ja GA:n kanssa, mutta tulee kuitenkin muistaa että ne kaikki on kehitetty täysin erillään. [5]

Evoluutio-ohjelmointi koostuu mutaatiosta ja valinnasta. Siinä ei siis käytetä ollenkaan rekombinaatiota, kuten ES:ssä ja GA:ssa, vaan se etenee täysin mutaation varassa. Mutaatio onkin EP:ssä keskeinen geneettinen operaatio. Mutaatioon voidaan käyttää kaikkia luvussa 5.2.1 esitellyjä menetelmiä, mutta yleisimmin käytössä on Gaussin-tyyppinen mutaatio, jolla saadaan sekä isoja että pieniä hyppyjä. [5]

Mutaation jälkeen EP:ssä valitaan seuraavalle kierrokselle selviytyvät yksilöt. Valinta tehdään luvussa 5.2.3 kuvatuilla todennäköisyyteen perustuvilla menetelmillä.

Algoritmi on seuraava:

1. $t = 0$
2. Alusta populaatio satunnaisesti
3. Arvioi populaation yksilöiden fitnessit
4. **while** ($t < max_iteraatio$) **do**
 Mutatoi
 Arvioi populaation yksilöiden fitnessit
 Valitse todennäköisyyteen perustuvalla valinnalla yksilöt seuraavalle kierrokselle
 $t++$

Evoluutio-ohjelmoinnin erityispiirre muihin evoluutioalgoritmeihin verrattuna on siis rekombinaation puuttuminen. EP:ssä jälkeläiset muodostetaan pelkästään mutaation avulla, kun taas GA:ssa ja ES:ssä käytetään myös rekombinaatiota.

6 Resurssien hakuongelma

Vertaisverkkojen keskeinen ongelma on resurssien etsiminen verkosta. Kuten aiemmin luvussa 2.3 kerrottiin, puhtaassa vertaisverkossa ei ole keskitettyä hakemistoa, josta hakuja voisi tehdä. Ongelma voidaan ratkaista käyttämällä hakualgoritmia, jossa kysely lähetetään aina solmun kaikille naapureille, jotka lähettävät kyselyn jälleen eteenpäin kaikille omille naapureilleen jne. Kun kysely saapuu solmulle, jolta haluttu resurssi löytyy, palauttaa se tiedot resursseista, minkä jälkeen kyselyn aloittanut solmu voi käyttää resurssia.

Edellä kuvattu hakualgoritmi on kuitenkin todella raskas, koska kyselyjä lähetetään niin paljon. Solmu, jolla on paljon naapureita, voi saada saman kyselyn useampaan kertaan. Hakualgoritmi olisi optimaalinen, jos kysely välitettäisiin vain niille naapureille, joilla haluttu resurssi on tai jotka voivat välittää kyselyn lyhyintä reittiä halutun resurssin omaavalle solmulle.

Vertaisverkoissa halutaan harvoin löytää kaikki verkosta löytyvät resurssit. Usein riittää, että löydetään esimerkiksi puolet verkon resursseista. Tällöin hakualgoritmin kyselyjen määrä vähenee huomattavasti, mikä on tärkeää verkon toiminnan kannalta.

Hakualgoritmin toimivuutta kuvataan fitness (hyvyys) -funktiolla. Fitnessiin vaikuttavat lähetettyjen kyselyjen lukumäärä ja löydettyjen resurssien lukumäärä. Mitä vähemmän paketteja lähetetään ja mitä enemmän resursseja löydetään, sitä parempi fitness saadaan.

6.1 Resurssien hakuongelman formalisointi

Seuraavassa esitetään esimerkki algoritmista, joka laskee fitness-arvon. Algoritmi koostuu useammasta funktiosta ja sisääntuloparametrista, jotka on esitelty seuraavassa.

6.1.1 Sisääntuloparametrit ja funktio

Algoritmi saa seuraavat sisääntuloparametrit:

- Yhtenäinen graafi $G = (V_G, E_G)$, joka sisältää vertaisverkon topologian. $V_G = \{v_1, v_2, \dots, v_n\}$ G :n solmut ja E_G on joukko suuntaamattomia yhteyksiä (v_i, v_j) , $v_i, v_j \in V_G, i \neq j$
- $L \subseteq V_G$ resursseja sisältävät solmut

- r = juurisolmu, joka aloittaa kyselyn
- k = löydettävien resurssien lukumäärä
- p = palkinto, joka lisätään fitnessiin jokaisesta löydetyistä resurssista

Sisääntulofunktiona on: $f(v, A, \Gamma)$ funktio, joka päättää, mille solmuille kysely lähetetään. Funktio vaihtuu käytettävän algoritmin mukaan. Funktio saa syötteinä kyselyä lähettävän solmun v , puun $A = (V_A, E_A)$ joka sisältää kyselypolut, joista v tietää ja joukon naapurisolmuja Γ , joille v voi lähettää kyselyn eteenpäin. V_A on joukko solmuja, jotka on määritelty kolmikkona $(index_A, index_G, arrivaltime)$, jossa $index_A$ kuvaa solmun uniikkia indeksiä puussa A , $index_G$ kuvaa solmun uniikkia indeksiä puussa G ja $arrivaltime$ kertoo ajan, jolloin solmu vastaanotti kyselyn. E_A on joukko suuntaamattomia yhteyksiä (v_i, v_j) , $v_i, v_j \in V_A, i \neq j$. Funktio palauttaa joukon solmuja, jotka sisältyvät Γ :aan ja joille kysely välitetään.

6.1.2 Muut muuttujat ja funktiot

Algoritmin käyttämät muut muuttujat ovat:

- $t_c \in \mathbb{N}$ nykyinen aika (current time), joka kuvaa hetkeä, jolloin solmu lähettää kyselyn eteenpäin
- $t_a \in \mathbb{N}$ saapumisaika (arrival time), joka merkitsee aikaa, jolloin solmu vastaanottaa kyselyn
- Puu $S = (V_S, E_S)$, joka sisältää kaikki kyselyssä käytetyt hakupolut. V_S on joukko solmuja kuvattuna kolmikkona $(index_S, index_G, arrivaltime)$, missä $index_S$ on solmun uniikki indeksi puussa S , $index_G$ on solmun uniikki indeksi puussa G ja $arrivaltime$ määrittää ajan, jolloin solmu sai kyselyn. E_S on joukko suuntaamattomia yhteyksiä (v_i, v_j) , $v_i, v_j \in V_S, i \neq j$.

Myös seuraavia funktioita tarvitaan:

- $\Gamma(G, v)$ palauttaa v :n naapurisolmut graafissa $G = (V_G, E_G)$. Solmu w on v :n naapurisolmu $\Leftrightarrow \exists(w, v) \in E_G \vee \exists(v, w) \in E_G$.
- $\text{Select}(V_S, t_a)$ palauttaa sen solmun $index_S$:n, jolla $arrivaltime = t_a$ joukosta V_S tai $\emptyset$, jos ei löydy solmua, jolla $arrivaltime = t_a$.

- $\text{Routes}(S, v)$ kokoaa alipuun $T \subseteq S$, joka sisältää kaikki polut juurisolmusta solmuihin, joilla $\text{index}_G = v$ ja ja jokaiselle T :n solmulle funktio lisää vastaavan lapsisolmun S :stä T :hen. Funktio palauttaa T :n.

Ulostulona algoritmista saadaan fitness, joka kertoo kuinka hyvin annettu sisääntulofunktio f selviää resurssienhaku ongelmasta.

6.1.3 Algoritmi

Itse algoritmi on seuraava:

- (1) $t_c = 0$
- (2) $t_a = 0$
- (3) $V_S = (1, r, t_a)$
- (4) $\text{current} = \text{Select}(V_S, t_c)$
- (5) **while** $\text{current} \neq \emptyset$ **do**
- (6) $A = \text{Routes}(S = (V_S, E_S), \text{current})$
- (7) $\text{Receivers} = f(\text{current}, A, \Gamma(G, \text{current}))$
- (8) **for** $i = 1$ **to** $|\text{Receivers}|$ **do**
- (9) $t_a = t_a + 1$
- (10) $E_S = E_S \cup \{\text{current}, |V_S| + 1\}$
- (11) $V_S = V_S \cup (|V_S| + 1, \text{Receivers}[i], t_a)$
- (12) **end for**
- (13) $t_c = t_c + 1$
- (14) $\text{current} = \text{Select}(V_S, t_c)$
- (15) **end while**
- (16) $\text{fitness} = 0$
- (17) **for** $i = 1$ **to** $|L|$ **do**
- (18) **if** $k > 0$ **and** $L[i] \in V_S$ **then do**
- (19) $\text{fitness} = \text{fitness} + p$
- (20) $k = k - 1$
- (21) **end if**
- (22) **end for**
- (23) $\text{fitness} = \text{fitness} - |E_S|$

6.1.4 Ongelman rajoitteita

Ongelman määrittelyssä on joitain rajoitteita. Se ei esimerkiksi ota huomioon informaatiota, joka voitaisiin viedä vastauspakettien mukana. Ongelma ei myöskään sovi dynaamiselle topologialle, jossa topologia vaihtuu kyselyn aikana. Tämä ei kuitenkaan ole vakava puute, sillä yhden kyselyn suorittaminen on huomattavasti nopeampaa kuin topologian muuttuminen.

Ongelma ottaa huomioon myös vain yhden yksikön pituiset yhteydet. Verkon solmut eivät myöskään muista edellisiä kyselyjä, joten ongelma pätee vain yksittäisiin kyselyihin. Ongelma on kuitenkin mahdollista laajentaa myös useampaa kyselyä koskevaksi.

Ongelman muotoilusta puuttuu lisäksi osassa hakualgoritmeista käytettävästä informaatiosta. Esimerkiksi NeuroHausssa tiedetään $\Gamma(G, v)$:hen kuuluvan naapurisolmun naapureiden lukumäärä.

7 NeuroHaku

Vertaisverkkojen pääasiallisena ongelmana on se, miten löytää verkosta esimerkiksi jokin tietty tiedosto, kun ei ole olemassa keskitettyä hakemistoa, josta etsiä. Tämä voidaan tehdä käyttämällä resurssien löytämiseen tarkoitettuja hakualgoritmeja. Eräs tällainen algoritmi on NeuroHaku (engl. *NeuroSearch*) [35], joka on kehitetty Jyväskylän yliopiston Cheese Factory-tutkimusprojektissa [8].

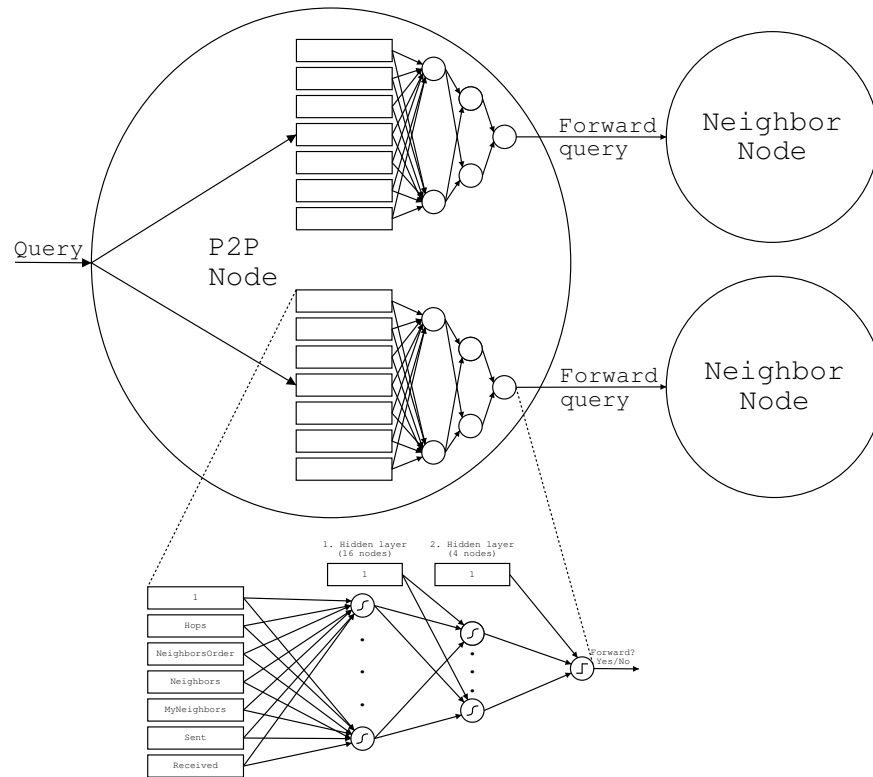
NeuroHaku on mukautuva resurssien hakuun suunniteltu algoritmi, joka tekee päätöksen, mille vertaissolmun naapureista kysely lähetetään eteenpäin. Jokaisen solmuparin, jonka välillä on yhteys, on myös neuroverkko. Neuroverkolle annetaan sisääntuloparametreina kyselyn siinä vaiheessa olevat tiedot, jotka neuroverkko prosessoi ja laskee niiden perusteella ulostulon. Jos ulostulo oli 1, kysely lähetetään tätä yhteyttä pitkin eteenpäin. Jos taas ulostulo oli 0, kyselyä ei jatketa.

Algoritmi käy läpi kaikki solmun naapurit (joita kutsutaan vastaanottajiksi) yksi kerrallaan ja laskee neuroverkon ulostulon, joka määrää mille vastaanottajille kysely lähetetään. Prosessia on kuvattu kuvassa 7.1. Jos kysely löytää halutun resurssin solmusta, vastaus lähetetään takaisin samaa reittiä kuin mitä kysely tuli. Kuvassa 7.2 on esitetty vertaisverkko ja NeuroHauin siinä tuottama hakupolku. Kuvan haku on lähtenyt solmusta, joka on merkitty ?-llä. !:t taas tarkoittavat solmuja, joilla on etsittävä resurssi.

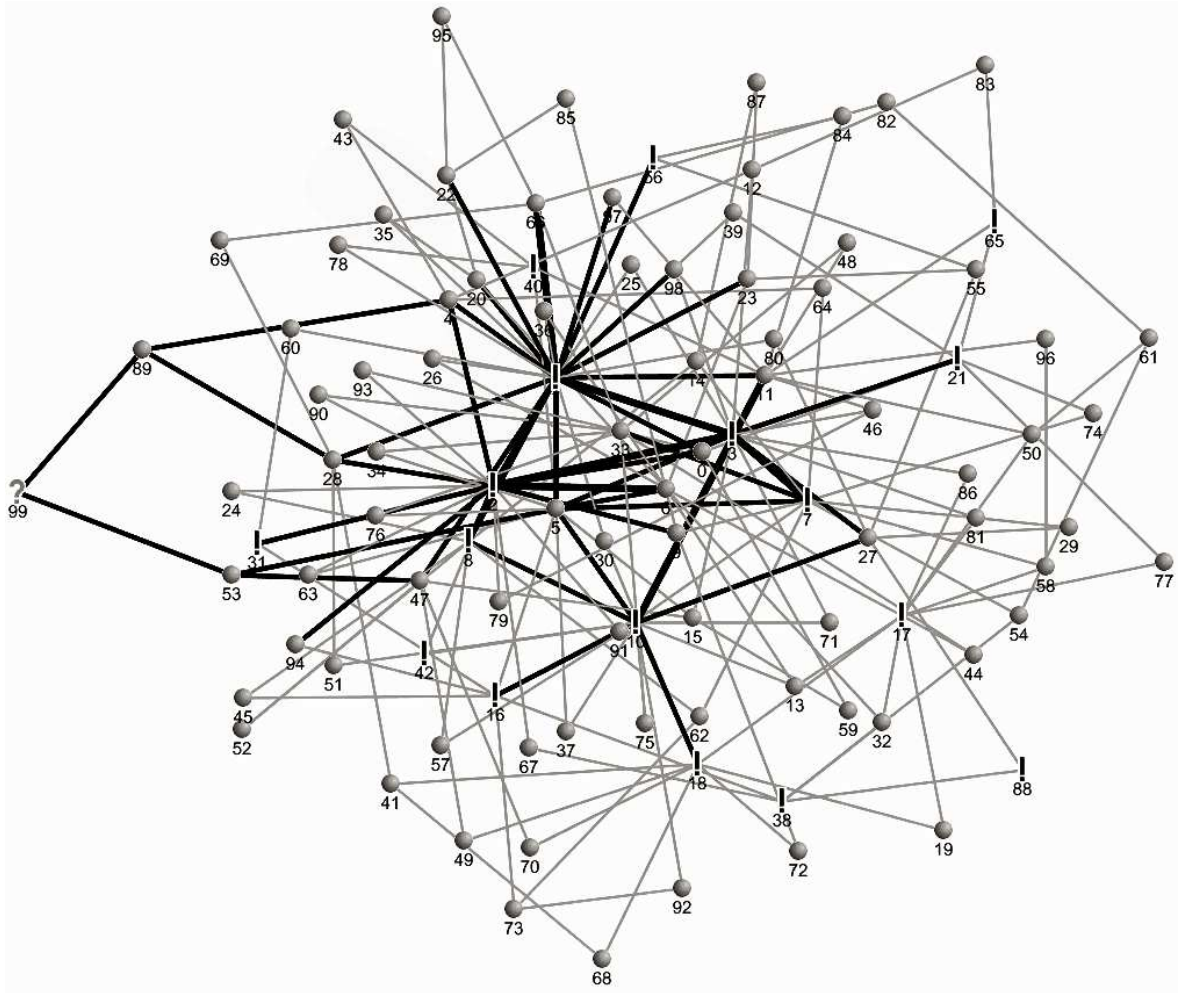
7.1 Sisääntuloparametrit

Algoritmissa käytetään neuroverkolle seuraavia sisääntuloparametreja:

- *Bias = 1* on bias termi. Bias eli kynnyсарvo mahdollistaa neuronin ulostulon poikkeamisen nollatasosta neuronin sisääntulojen summan ollessa nolla.
- *Hops* on hyppyjen lukumäärä kyselyssä. Hypyt ilmoittaa, kuinka montaa linkkiä pitkin kysely on tähän mennessä kulkenut. Neuroverkko voi päättää lopettaa sellaisen kyselyn lähettämisen, joka on kulkenut liian pitkään.
- *ToNeighborsRank* kertoo solmun naapurien lukumäärän verrattuna muihin vastaanottajiin. Vastaanottajalla, jolla on eniten naapureita, *NeighborsOrder* = 0, ja seuraavaksi eniten naapureita olevalla *NeighborsOrder* = 1 jne.



Kuva 7.1: NeuroHaku-neuroverkon toiminta vertaisverkossa. [35]



Kuva 7.2: NeuroHaun tuottama hakupolku vertaisverkossa ja löydetyt resurssit.

- *ToNeighborsAmount* on vastaanottajan naapurien lukumäärä.
- *CurrentNeighborsAmount* on solmun naapurien lukumäärä.
- *Sent* on 1, jos kysely on jo lähetetty vastaanottajalle tätä linkkiä pitkin. Muuten arvo on 0.
- *From* on 1, jos viesti tuli solmulle vastaanottajalta, muuten arvo on 0.
- *ToFreeNeighborsAmount* kertoo, kuinka monta sellaista naapurisolmua, joilta resurssia ei vielä ole etsitty, vastaanottajalla arvellaan olevan. Jos kysely ei ole aikaisemmin käynyt vastaanottajalla, arvo on sama kuin *ToNeighborsAmount*. Muuten kysely rekisteröi edellisellä kerralla solmussa käydessään, kuinka monelta naapurilta resurssia ei oltu vielä etsitty. Jos nykyisessä solmussa ei ole käyty aikaisemmin, *ToFreeNeighborsAmount* on yhtä suuri kuin rekisteröityjen solmujen lukumäärä -1. Jos solmussa on vierailtu aiemmin, niin *ToFreeNeighborsAmount* on yhtä suuri kuin rekisteröityjen solmujen lukumäärä.
- *InitiatorNeighborAmount* on kyselyn aloittaneen solmun naapureiden lukumäärä.
- *RepliesNow* kertoo, kuinka monta vastausta kysely on löytänyt kyselypolullaan.
- *PacketsNow* kertoo, kuinka monta kyselypakettia kyselypolulla on tähän mennessä käytetty.
- *RepliesToGet* kertoo, kuinka paljon resursseja tulee löytää.
- *ToVisited* on 1, jos kysely on jo käynyt vastaanottajalla, muuten arvo on 0.
- *RandomNeighbor* on 1 nykyisen solmun satunnaisesti valituille vastaanottajille ja 0 muille vastaanottajille.
- *FromNeighborsAmount* kertoo, kuinka monta naapuria on solmulla, jolta kysely saatiin. Jos *Hops* on 0 eli nykyinen solmu on lähettäjä, on *FromNeighborsAmount* arvo sama kuin *CurrentNeighborsAmount*.
- *CurrentVisited* on 1, jos kysely on aikaisemmin käynyt nykyisessä solmussa, muuten 0.
- *Forwarded* kertoo, kuinka monta kertaa nykyinen solmu on päättänyt lähettää kyselyn eteenpäin saatuaan nykyisen kyselyviestin, jota ollaan lähettämässä eteenpäin.

- *NotForwarded* kertoo, kuinka monta kertaa nykyinen solmu on päättänyt olla lähettämättä kyselyä eteenpäin saatuaan nykyisen kyselyviestin, jota ollaan lähettämässä eteenpäin.
- *DecisionsLeft* kertoo, kuinka monelle naapurille nykyinen solmu vielä tekee lähettämispäätöksen.
- *EnoughReplies* on 1, jos kyselyviestin kulkeman hakupolun kautta on löydetty saman verran tai enemmän resurssivastauksia kuin mitä *RepliesToGet* on. Muuten *EnoughReplies* on 0.

Kolmea viimeistä sisääntuloparametria varten määritellään valkoinen, harmaa ja musta naapurisolmu seuraavasti. Naapurisolmu on valkoinen, jos nykyinen hakupolku ei ole käynyt naapurisolmussa. Naapurisolmu on harmaa, jos nykyinen hakupolku on käynyt naapurisolmussa, mutta jolla on olemassa valkoisia naapurisolmuja. Naapurisolmu on musta, jos nykyinen hakupolku on käynyt naapurisolmussa ja josta lähdettäessä ei ollut olemassa valkoisia naapurisolmuja.

- *White* kertoo vastaanottajan naapureiden lukumäärän rankin suhteessa muihin nykyisen solmun valkoisiin naapureihin. Jos vastaanottaja ei ole valkoinen, on *Whiten* arvo ääretön. Jos vastaanottaja on valkoinen ja sillä on enemmän naapureita kuin nykyisen solmun muilla valkoisilla naapureilla, on *Whiten* arvo 0. Jos vastaanottaja on valkoinen ja sillä on vähemmän naapureita kuin nykyisen solmun yhdellä valkoisella naapurilla, mutta enemmän kuin muilla nykyisen solmun valkoisilla naapureilla, on *White* = 1 jne.
- *Grey* kertoo vastaanottajan naapureiden lukumäärän rankin suhteessa muihin nykyisen solmun harmaisiin naapureihin. Jos vastaanottaja on musta tai valkoinen, on *Greyn* arvo ääretön. Jos vastaanottaja on harmaa ja sillä oli eniten valkoisia naapureita suhteessa muihin nykyisen solmun harmaisiin naapureihin, on *Grey* = 0 jne.
- *Black* kertoo vastaanottajan naapureiden lukumäärän rankin suhteessa muihin nykyisen solmun mustiin naapureihin. Jos vastaanottaja on valkoinen tai harmaa, on *Blackin* arvo ääretön. Jos vastaanottaja on musta ja sillä on eniten naapureita suhteessa muihin nykyisen solmun mustiin naapureihin, on *Blackin* arvo 0 jne.

Hops, *ToNeighborsRank*, *Forwarded*, *NotForwarded*, *DecisionsLeft*, *RepliesNow*, *PacketsNow*, *RepliesToGet*, *White*, *Grey* ja *Black* skaalataan funktiolla $f(x) = \frac{1}{x+1}$, ja *ToNeighborsAmount*, *CurrentNeighborsAmount*, *ToFreeNeighbors*, *InitiatorNeighborAmount* ja *From-*

NeighborsAmount funktiolla $f(x) = \frac{1}{x}$ ennen neuroverkkoon syöttämistä. Skaalaus tehdään, jotta varmistuttaisiin siitä, että kaikki sisääntuloparametrit ovat välillä 0-1. [35]

7.2 Rakenne

Neuroverkko on kolmikerroksinen perceptron neuroverkko, jossa on kaksi piilokerrosta. Ensimmäisessä kerroksessa on 15 solmua ja bias ja toisessa kerroksessa kolme solmua sekä bias. Piilokerroksilla käytetään aktivaatiofunktiona *tanh*:ta:

$$t(a) = \frac{2}{1 + e^{-2a}} - 1.$$

Ulostulosolmun aktivaatiofunktiona on kynnysfunktio:

$$s(a) = \begin{cases} 0, & a < 0 \\ 1, & a \geq 0 \end{cases}$$

Jos tämän ulostulona on 1, viesti lähetetään naapurille, muuten ei. [35]

7.3 Painokertoimien määrittäminen

Ennenkuin NeuroHaku-algoritmia voidaan käyttää, täytyy neuroverkon painokertoimet optimoida. Cheese Factory -projektin [8] kehittämässä P2P-simulaattorissa [34] tämä tehdään evoluutioalgoritmeilla, tarkemmin sanottuna evoluutio-ohjelmoinnilla. Palaute, joka ohjaa optimointiprosessia, saadaan laskemalla verkossa lähetettyjen kyselypakettien määrän ja löydettyjen vastausten määrä.

Neuroverkon hyvyys (fitness) määritellään kaksiosaisesti. Jokainen kysely j pisteytetään neuroverkolle h ja fitness lasketaan summaamalla kaikki pisteet (engl. *score*) n kyselyn jälkeen:

$$fitness_h = \sum_{j=1}^n score_j.$$

Pisteet taas määräytyy sen mukaan, kuinka paljon resursseja on löydetty ja kuinka paljon niiden löytämiseksi piti lähettää paketteja. Pisteet määrätään seuraavien sääntöjen mukaan, missä *packets* on lähetettyjen kyselypakettien määrä, *replies* on löydettyjen vastausten määrä ja *availableResources* tarjolla olevien vastausten maksimimäärä:

1. Jos $packets > 300$ niin $score = 0$: algoritmi, joka lopulta pysähtyy on parempi kuin algoritmi, joka ei pysähdy. 300 pidetään rajana, jolloin kyselyn oletetaan jatkavan ikuisesti etenemistä verkossa.

2. Jos $replies = 0$ niin $score = 1 - \frac{1}{packets+1}$: jos ei löydetä yhtään resurssia, neuroverkon tulisi lähettää enemmän paketteja.
3. Jos $replies < \frac{availableResources}{2}$ ja $replies > 0$ niin $score = 50 \times replies - packets$: jos ei löydetä tarpeeksi resursseja, niin neuroverkko kehittyy, jos se löytää enemmän resursseja.
4. Jos $replies \geq \frac{availableResources}{2}$ niin $score = 50 \times \frac{availableResources}{2} - packets$: jos löydetään puolet olemassa olevista resursseista, hyvyys arvo ei kasva, vaikka neuroverkko löytäisi enemmän paketteja. Ainoa keino parantaa $scorea$ on tällöin kuluttaa vähemmän paketteja.

Optimointiprosessi aloitetaan neuroverkolla, jonka painokertoimet on määrätty satunnaisesti. Jokainen neuroverkko testataan P2P-simulaattorissa usealla kyselyllä, minkä jälkeen lasketaan fitness-arvot. Kun kaikki neuroverkot on testattu, valitaan puolet (arvoltaan parhaat) neuroverkoista mutaatioon ja muodostetaan niistä uusi sukupolvi. Tuloksena saadaan saman verran neuroverkkoja kuin oli ensimmäisellä testauskierroksella. Uusi sukupolvi voidaan jälleen testata simulaattorissa. Optimointiprosessi lopetetaan ennalta määrätyn generaatiokierroksen jälkeen.

Neuroverkon mutaatio tarkoittaa, että painokertoimia muutetaan hieman alkuperäisestä. Muutos tehdään satunnaisesti käyttämällä normaalijakaumaa. Satunnainen variaatiofunktio on:

$$\sigma'_i(j) = \sigma_i(j) \exp(\tau N_j(0, 1)), j = 1, \dots, N_w \quad (7.1)$$

$$w'_i(j) = w_i(j) + \sigma'_i(j) N_j(0, 1), j = 1, \dots, N_w \quad (7.2)$$

missä N_w on painokertoimien ja bias-termien lukumäärä neuroverkossa, $\tau = \frac{1}{\sqrt{2\sqrt{N_w}}}$, $N_j(0, 1)$ on standardi Gaussin satunnainen muuttuja jokaiselle j , σ on itsemukautuva parametrivektori, joka määrittelee askelpituuden uuden painokertoimen löytämiseksi, i on neuroverkon indeksi $1 \leq i \leq 30$ ja $w'_i(j)$ on uusi painokerroin. [35]

NeuroHaussa käytetään siis evoluutioalgoritmeista EP:tä. Risteytystä ei ole lainkaan, vaan jälkeläisten populaatio saadaan aikaan mutaatiolla. Myös ES:n piirteitä löytyy, sillä NeuroHaussa käytetään itsemukautuvaa skaalausparametriä. Mutaatioon NeuroHaussa käytetään Gaussin-mutaatiota. Valinta taas on toteutettu elitistisenä, jolloin arvoiltaan parhaat yksilöt pääsevät jatkokoon.

Itse asiassa kyseessä on monitavoiteoptimointitehtävä, sillä fitness yrittää kuvata useampaa kriteeriä.

Neuroverkon kehittäminen vaatii paljon laskentatehoa ja NeuroHaun tapauksessa tavalliselta PC-koneelta menee 100000 generaation läpikäymiseen noin viikko. Algoritmi voi myös pysähtyä kesken, jos skaalauskerroimet σ_i menevät pieniksi, jolloin uusi generaatio on kopio edellisestä, eikä optimointi enää etene.

7.4 Aiemmat tutkimukset

NeuroHakua on tutkittu aiemmin mm. populaation koon, sisääntuloparametrien lukumäärän ja neuronien lukumäärän osalta. Töyrylä on gradussaan [34] todennut, että 16 neuroverkon populaatio on riittävä määrä NeuroHaun vakaalle toiminnalle. Verkkojen lisääminen varmistaa, että saavutettu tietämyksentaso säilyy, mutta se myös lisää laskenta-aikaa.

Töyrylä [34] on tutkinut myös sisääntuloparametrien lukumäärän vaikutusta. Hän on havainnut, että kuusi parametria on minimi NeuroHaun toiminnalle. Lisäämällä parametrien lukumäärää, voidaan algoritmin toimintaa parantaa, mutta laskentaan kuluva aika myös kasvaa.

Töyrylän gradussa [34] on neuronien lukumäärän osalta päästy tulokseen, että mitä enemmän neurononeita on käytössä sitä parempiin tuloksiin päästään. Toisaalta jälleen myös laskenta aika kasvaa neuronien lukumäärän kasvaessa. NeuroHaku tutkimuksissa on käytetty 16:4-neuroverkkoa (neuroverkkoa, jossa ensimmäisellä piilokerroksella on 16 neuronia ja toisella 4) ja viime aikoina on tehty testiajoja myös 30:20-verkoilla. Jälkimmäisellä rakenteella on päästy parempiin fitness-arvoihin kuin 16:4-neuroverkoilla.

8 Tutkimusympäristö

8.1 Mitattavat suureet

Mitattaviksi suureiksi valittiin:

- Keskiarvo-fitness n :lle ajolle: n :n ajon maksimi-fitnessien keskiarvo
- Maksimi-fitness n :lle ajolle: n :n ajon paras fitness
- Minimi-fitness n :lle ajolle: n :n ajon huonoin maksimifitness
- Keskiarvoajoaika: n :n ajon ajoaikojen keskiarvo
- Maksimi ajoaika: n :n ajon suurin ajoaika
- Minimi ajoaika: n :n ajon pienin ajoaika.

Lisäksi tarkasteltiin lähetettyjen pakettien lukumäärää, saatujen vastausten lukumäärää ja kyselyjen käyttämiä keskimääräistä maksimihyppyjen määrää. Maksimihypyillä tarkoitetaan, kuinka monta hyppyä kysely enimmillään käytti.

8.2 Java-luokat

Ohjelman pääluokka on NetSimulator.java, joka lukee simulaation parametrit, esimerkiksi sisääntuloparametrit, käytettävän mutaatiomenetelmän ja iteraatioiden lukumäärän config-tiedostosta.

Evaluator.java luokassa opetetaan neuroverkkoa. Evaluator luodaan config-tiedostosta luettujen muuttujien perusteella.

NeuroSearchGenerator.java luo neuroverkkopopulaation, jota simulaatiossa käytetään. Luokasta löytyy metodit keskiarvo-fitnessin laskemiseen ja parhaan fitnessin etsimiseen. NeuroSearchGenerator luodaan ja alustetaan Evaluatorissa.

NeuralNet.java on luokka, jossa neuroverkko on määritelty. Luokka sisältää paino- ja skaalauskerroimet sekä metodit mutaatiolle ja risteytykselle.

Java-luokkiin voi tutustua tarkemmin Joni Töyrylän pro gradu -tutkielmassa [34].

8.3 Käytetyt menetelmät

NeuroHaun optimoinnin toteutukseen otettiin uusia piirteitä evoluutioalgoritmeista.

8.3.1 Rekombinaatio

Täysin uutena piirteenä NeuroHakuun otettiin mukaan risteytys. Risteytys toteutettiin yksi- ja kaksipisteristeytyksenä. Näin voidaan vertailla niiden tehokkuutta NeuroHaun tapauksessa.

8.3.2 Mutaatio

Cauchyn, Lévy- ja Gaussin-mutaatioiden vaikutusta NeuroHakuun on tutkittu aikaisemmin ja Gaussin-mutaatio on todettu tehokkaimmaksi. Näin ollen mutaatiomenetelmä päätettiin pitää ennallaan. Mutaatioon tehtiin kuitenkin pieni muutos siten, että mutaatio vaihdettiin geeneittäin tehtäväksi.

8.3.3 Valinta

Valintamenetelmiksi valittiin elitistinen valinta, turnajaisvalinta ja oma kehittämäni todennäköisyyteen perustuva valinta.

Elitistinen valinta toteutettiin, kuten se NeuroHaussa alunperin oli. Parhaat neuroverkot pääsivät siis suoraan jatkoon, jolloin saatiin puolet populaatiosta. Populaation toinen puolisko muodostettiin mutaation avulla.

Turnajaisvalinnasta otettiin käyttöön versio, jossa jokainen populaation yksilö pääsee ”turnajaisiin” ja saa vastaan kaksi populaatiosta satunnaisesti valittua yksilöä. Jatkoon valittiin puolet populaatiosta sen perusteella, mitkä olivat voittaneet eniten vastustajia. Mikäli useampi yksilö oli voittanut yhtä monta vastustajia, jatkoon valittiin se, jolla oli parempi fitness. Toinen puoli populaatioon saatiin mutaatiolla.

Omassa valinnassa ideana oli muokata elitististä valintaa siten, että kaikilla neuroverkoilla olisi mahdollisuus päästä jatkoon, mutta parhaat verkot pääsisivät suuremmalla todennäköisyydellä. Rulettivalintahan tekee juuri tämän, mutta NeuroHaun tapauksessa ongelmaksi muodostui se, että huonoimmat verkot saivat niin pienen valituksi tulemisen todennäköisyyden, että käytännössä valittiin lähes aina vain parhaat verkot. Omassa valinnassa neuroverkot järjestettiin ja viidelle huonoimmalle annettiin todennäköisyys 1/105, seuraaville viidelle 2/105 ja niin edelleen siten, että parhaat viisi neuroverkkoa saivat todennäköisyyden 6/105. Tämän jälkeen neuroverkoista valittiin satunnaisesti puolet jatkoon to-

dennäköisyyksien perusteella.

Kaikissa valinnoissa lisäksi varmistettiin se, että sama verkko ei voinut tulla valituksi kahdesti.

8.4 Käytetyt laitteet

Tutkimuksessa käytettiin testitapausten ajoissa PC-tietokoneita, joissa oli käyttöjärjestelmänä Windows XP. Suurin osa koneista oli AMD Athlon 3000 2.09 GHz, jossa muistia oli 1GB. Yksi koneista oli hieman tehottomampi, AMD Athlon 1600 1.41GHz, jossa myös muistia oli hieman vähemmän, 256MB. Tehottomampi kone oli yliopistolta tutkimusta varten käyttöön annettu tietokone, muut koneet olivat mikroluokkien koneita. Koneiden tehokkuuden erot huomattiin vasta testiajojen jälkeen.

8.5 Neuroverkon optimointi suorahakumenetelmillä

Suorahakumenetelmät (engl. *direct search methods*) ovat suosittuja optimointimenetelmiä. Tähän on syynä mm. se, että ne toimivat hyvin käytännössä, ne sopivat lähes kaikkiin epälineaarisiin optimointiongelmiin ja ne ovat melko yksinkertaisia toteuttaa ja helppoja käyttää. [21]

Suorahakumenetelmät eivät myöskään tarvitse funktion derivaattaa apuna optimoinnissa. Edellisessä luvussa esitelty Powellin-, Powellin toinen- ja Zangwillen menetelmä ovat suorahakumenetelmiä. Myös mm. Hooken ja Jeevesin menetelmä [37] ja Rosenbrockin menetelmä [7] kuuluvat suorahakumenetelmiin.

Päätin kokeilla suorahakumenetelmiä neuroverkon optimointiin niiden yksinkertaisuuden, joustavuuden ja luotettavuuden takia. Powellin menetelmä käytiin läpi tietotekniikan laitoksen Epälineaarinen optimointi -kurssilla ja se vaikutti parhaalta suorahakumenetelmältä kyseiseen ongelmaan. Powellin menetelmää on myös käytetty menestyksellä neuroverkon opettamisessa Nakanishin, Kohdan ja Inouen tutkimuksessa [24].

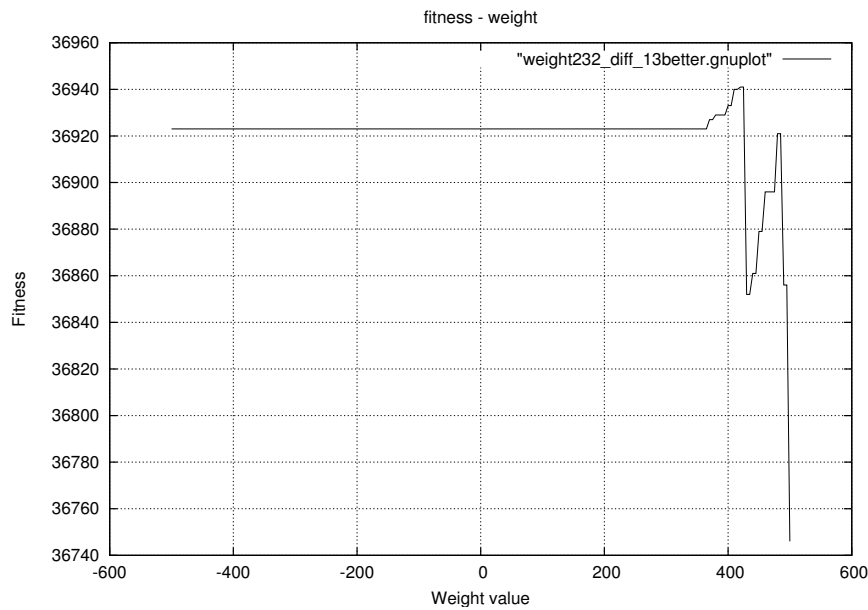
Ratkaistava ongelma oli siis muotoa

$$\text{maksimoi } f(\mathbf{x}), \text{ missä } \mathbf{x} = \{x_1, x_2, \dots, x_{292}\}$$

on NeuroHaun painokerroinvektori, $x_i \in \mathbb{R}, i = 1, \dots, 292$,

$$f(\mathbf{x}) = \text{fitness, joka on määritelty luvussa 7.3.}$$

Ongelma osoittautui kohinaiseksi, jolloin suorahakumenetelmät eivät toimineet ollenkaan. Tämä onnistuttiin kuitenkin kiertämään kiinnittämällä kyselyn aloitussolmut ja kyseltävät



Kuva 8.1: Suorahakumenetelmällä saadun neuroverkon yhden painokertoimen muutoksen vaikutus fitnessiin.

resurssit. Tämän jälkeen Powellin menetelmällä saatiin jo parempia tuloksia. Ongelma oli kuitenkin lähes 300 dimensioinen, joten Powellin menetelmä ei toiminut siinä parhaalla mahdollisella tavalla. Ongelmaksi muodostui hakusuuntien riippuvuus toisistaan. Algoritmi löysi kyllä lokaalin optimipisteen, mutta sitä ei voinut pitää luotettavana. Myöskään Zangwillen tai Powellin toinen menetelmä eivät päässeet hyviin tuloksiin optimoitavassa tapauksessa. Suorahakumenetelmät eivät tässä tapauksessa toimi hyvin lähes 300 dimensioisiin ongelmiin.

Suorahakumenetelmillä saatiin parhaalla ajolla fitnessiksi 36923. Tutkittaessa tämän verkon painokertoimia huomattiin, että 292:sta painokertoimesta 14 oli sellaisia, jotka eivät olleet optimiarvoissaan. Kuvassa 8.1 on esimerkki yhdestä tällaisesta painokertoimesta. Suorahakumenetelmällä saatu tulos on x-akselilla kohdassa $x = 10$. Kuvasta nähdään, että esimerkiksi arvolla $x = 425$ olisi päästy parempaan fitnessiin. Menetelmän löytämä piste ei siis ollut globaali optimi.

9 Tutkimustuloksia

Tutkimuksessa tutkittiin eri optimointimenetelmien vaikutusta NeuroHaussa. Suorahakumenetelmien osoittauduttua huonoksi vaihtoehdoksi, siirryttiin evoluutioalgoritmeihin, joista valittiin uusia piirteitä NeuroHaun optimoinnin toteutukseen. Optimointiin otettiin mukaan risteytys ja valintamenetelmää vaihdeltiin. Käytetyt menetelmät on kuvattu tarkemmin luvussa 8.3.

Kaikissa tutkimustapauksissa käytettiin seuraavia parametreja:

- Käytetty mutaatio: Gaussin mutaatio
- Kolmikerroksinen neuroverkko, jossa ensimmäisellä piilokerroksella 16 neuronia ja toisella 4.
- 50 % resursseista löydettävä
- Kyselyjen lukumäärä: 10
- Sisääntuloparametrien lukumäärä: 23
- Mutaation voimakkuus: 0.001
- Risteytysaste: 0.6
- Populaation koko: 30
- Generaatioiden lukumäärä: 20000

Mutaation voimakkuus määrittelee, millä todennäköisyydellä geenille tehdään mutaatio. Risteytysaste määrittelee, millä todennäköisyydellä uusi jälkeläinen tehdään risteyttämällä.

Testiajoissa käytettiin eri verkkotopologioita neuroverkon opetuksessa ja analysoinnissa. Näin neuroverkko ei voinut mukautua vain yhteen topologiaan, vaan sen täytyi pystyä löytämään resursseja monenlaisista verkoista. Verkoissa resurssien lukumäärä vaihteli, ollen kaikissa kuitenkin 400 tietämällä.

Kyselyissä kyselyn aloitussolmu valittiin joka kyselyn alussa sattumalta. Tämän johdosta samoilla painokeroimilla voidaan saada eri fitness arvoja.

Jokaisella tutkimustapauksella ajettiin kolme ajoa. Koska ajettavia tapauksia oli melko paljon ja yhteen ajoon meni aikaa aina lähes 20 tuntia, päätettiin kukin tutkimustapaus ajaa

kolmesti. Ajamalla testitapauksia useampaan kertaan, olisi saatu enemmän tutkimusaineistoa ja luotettavampia tuloksia, mutta myös aikaa olisi tutkimuksenn mennyt tällöin huomattavasti enemmän.

9.1 Yksipisteristeytys

Tässä luvussa kerrotaan, millaisia tuloksia NeuroHaulla saatiin, kun optimoinnissa käytettiin mukana yksipisteristeytystä.

9.1.1 Elitistinen valinta

NeuroHaussa alunperin olleella elitistisellä valinnalla päästiin yksipisteristeytyksessä tasaisiin tuloksiin, maksimi- ja minimifitnessien eroksi tuli vain 184 pistettä. Maksimi-fitnessiksi saatiin 36320 ja minimi-fitnessiksi 36134. Keskiarvoksi kolmelta ajolta tuli 36200.

Taulukkoon 9.1 on kerätty tietoja ajojen suureista. Pakettien lukumäärä (packets) tarkoittaa verkossa lähetettyjen kyselypakettien määrää, vastaukset (replies) on kyselyn löytämien resurssien lukumäärä, joista kyselyä on palkittu ja hypyt (hops) on hyppyjen lukumäärä kyselyssä.

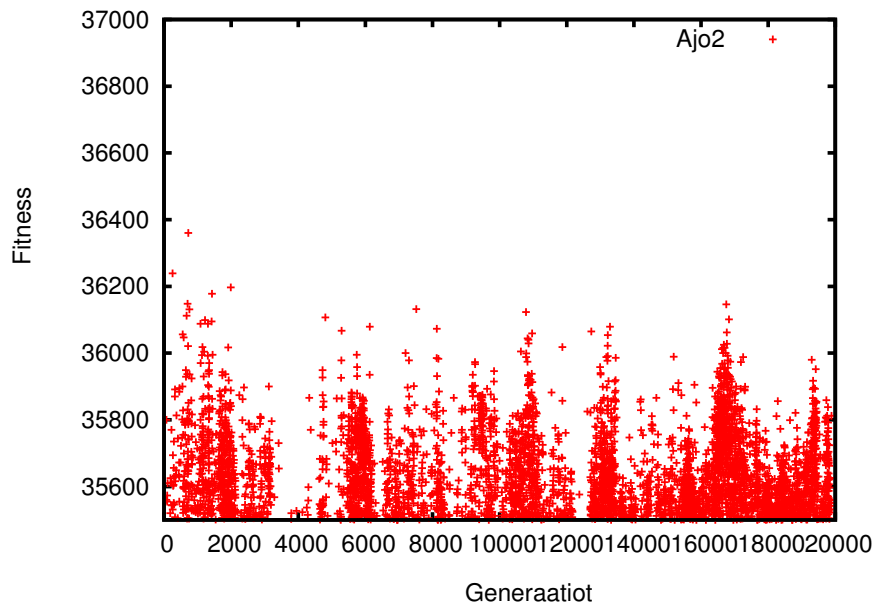
Paras fitness saatiin siis ajolla 1. Taulukosta nähdään, että kaikilta kolmelta ajolta saadut tulokset ovat tasaisia lähes kaikkien muuttujien osalta. Lähetettyjen pakettien määrässä on jonkin verran eroa. Eniten paketteja on lähettänyt ajo 3, joka sai myös heikoimman fitnessin. Selviä eroja näkyy ajoihin kuluneissa ajoissa. Kauimmillaan ajoon 1 meni 30h 52min, kun taas ajo 2 oli valmis jo 16h 32min kuluttua. Tässä tapauksessa tulee kuitenkin ottaa huomioon, että ajo 1 on ajettu tehottomimmalla koneella, mikä vaikuttaa ajoon kuluneeseen aikaan. Keskiarvoksi kolmesta ajosta saatiin 22h 49min.

	Maksimi-fitness	Pakettien lkm	Vastaukset	Hypyt	Aika
Ajo1	36320	4880	824	13.23	30h 52min
Ajo2	36146	4404	811	13.33	16h 32min
Ajo3	36134	5615	835	9.68	21h 4min

Taulukko 9.1: Tilastotietoja elitistisen valinnan testiajojen parhaista neuroverkoista.

Ajon 2 kohdalla ei ole valittu parasta fitnessiä, koska se on saavutettu jo generaatiolla 722 kuten kuvasta 9.1 nähdään. Näin alussa generaatioita hyvä fitness on saatu ennemminkin sattumalta, sillä neuroverkko ei ole siihen mennessä ehtinyt kehittyä ja mukautua ongelmaan.

Tarkasteltavaksi on valittu generaation 16752 antama neuroverkko. Parhaan verkon fitness olisi ollut 36360. Paketteja siinä käytettiin 4840 ja vastauksia saatiin 824. Hyppyjä kysely käytti 17.21. Eroa parhaan ja tarkasteltavaksi valitun verkon välillä ei siis ole paljon. Hypyt ovat hieman laskeneet generaatioiden myötä, samoin lähetettyjen pakettien lukumäärä.

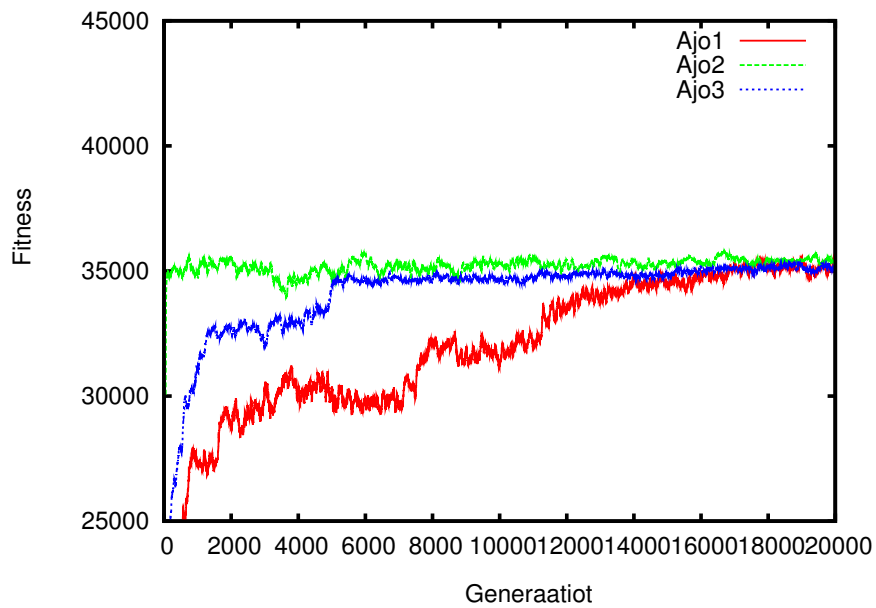


Kuva 9.1: Yksipisteristeytyksen elitistisen valinnan ajon 2 yli 35500 fitnesssit.

Kuvassa 9.2 on kaikkien ajojen fitness-käyrät. Käyriä piirtäessä on käytetty liikkuvaa 50 keskiarvoa, koska muuten käyristä tulisi liian sotkuisia lukea. Tästä eteenpäin fitness-käyrät on aina piirretty näin, ellei toisin ole mainittu. Kuvasta huomataan, että ajojen fitnessit on kehittyneet melko erilailla. Parhaan fitnessin saanut ajo 1 on saanut selvästi huonoimpia fitnesssejä aina 12000 generaatioon asti, kehittyen kuitenkin koko ajan tasaisesti ja päästen ajon loppupuolella tasoihin ajojen 2 ja 3 kanssa. Ajo 2 taas on saanut melko tasaisia fitness-arvoja aina ensimmäisistä generaatioista lähtien. 14000 generaatiosta eteenpäin kaikkien kolmen ajon fitness-käyrät ovat lähes tasoissa.

9.1.2 Todennäköisyyteen perustuva valinta

Itse kehittämälläni todennäköisyyteen perustuvalla valinnalla saatiin yksipisteristeytyksessä maksimi-fitnessiksi 37319, mikä on tutkimuksen paras fitness-arvo. Minimi-fitness jäi kuitenkin alhaiseksi päästen vain 35449:ään. Keskiarvo nousi kuitenkin korkealle: 36578. Todennäköisyyteen perustuva valinta pärjäsikin hyvin muihin yksipisteristeytyksen tapauksiin verrattuna.



Kuva 9.2: Yksipisteristeytyksen elitistisen valinnan ajokäyrät.

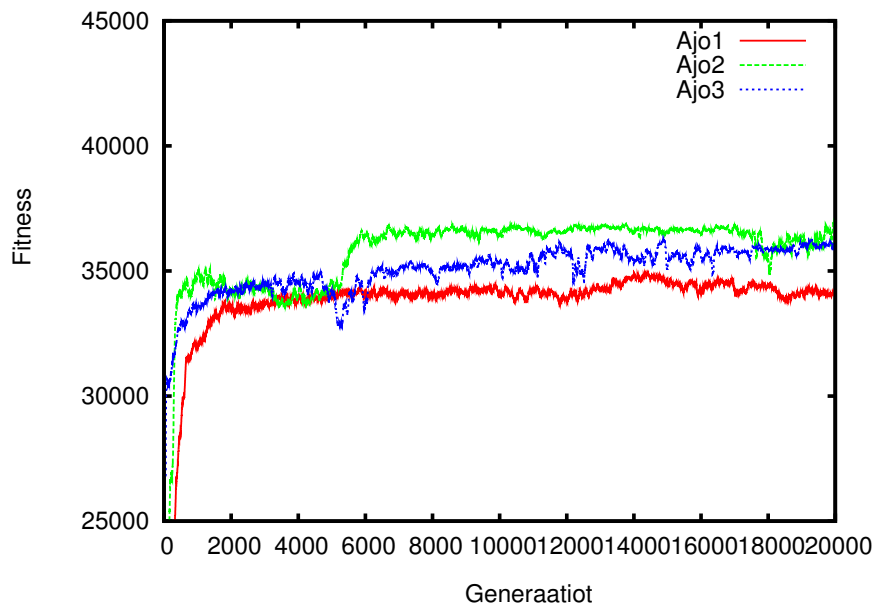
	Maksimi-fitness	Pakettien lkm	Vastaukset	Hypyt	Aika
Ajo1	35449	6101	831	9.95	17h 27min
Ajo2	37319	4781	842	21.73	13h 37min
Ajo3	36965	4135	822	20.19	15h 16min

Taulukko 9.2: Tilastotietoja todennäköisyyteen perustuvan valinnan testiajojen parhaista neuroverkoista.

Taulukosta 9.2 on mielenkiintoista tarkastella ajojen muuttujia. Taulukosta havaitaan, että heikoimman maksimi-fitnessin saanut ajo 1 on käyttänyt selvästi vähemmän hyppyä kuin ajot 2 ja 3. Ajon 1 hypyt on alle kymmenen, kun taas ajot 2 ja 3 ovat käyttäneet yli 20 hyppyä. Ajo 1 on joutunut lähettämään myös paljon kyselypaketteja, mikä tietenkin vähentää sen fitnessiä.

Myös ajoajoissa on jonkin verran eroa. Parhaan fitnessin saanut ajo 2 on käyttänyt vähiten aikaa, 13h 37minuuttia. Ajolla 1 kesti lähes neljä tuntia kauemmin, eli 17h 27min. Kolmen ajon keskiarvoksi saatiin 15h 27min. Myös aikoja vertailtaessa todennäköisyyteen perustuva valinta sai yksipisteristeytyksessä parhaat tulokset.

Kuvan 9.3 fitness-käyristä näkee hyvin, että ajo 2 on menestynyt parhaiten ja ajo 1 heikoiten. Ajon 3 fitnessit ovat kasvaneet läpi koko ajon tasoittuen 20000 generaatiota lähestyttäessä. Ajo 2 ei ole kehittynyt enää 6000 generaation jälkeen.



Kuva 9.3: Yksipisteristeytyksen todennäköisyyteen perustuvan valinnan ajokäyrät.

	Maksimi-fitness	Pakettien lkm	Vastaukset	Hypyt	Aika
Ajo1	36026	4824	817	11.82	19h 17min
Ajo2	35082	6218	826	10.2	17h
Ajo3	35392	6308	834	9.69	19h 43min

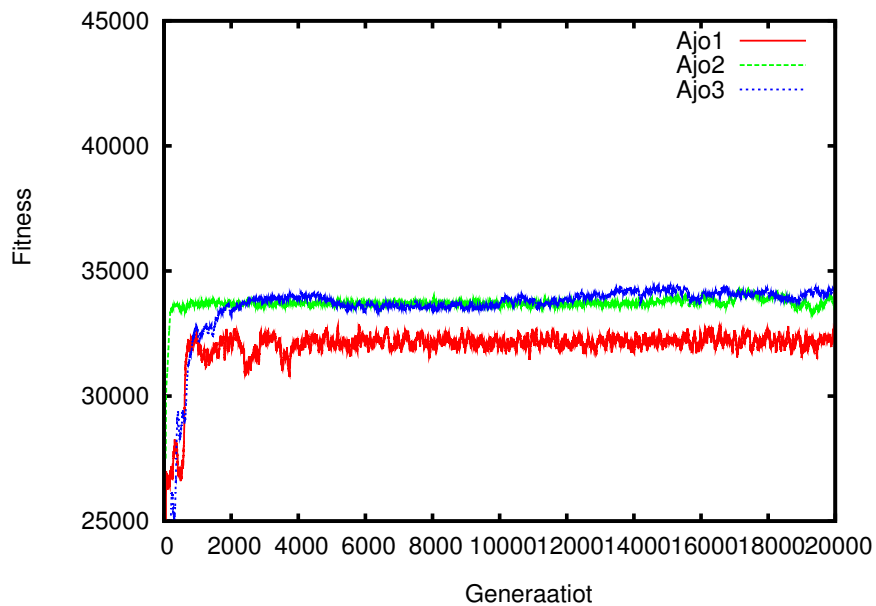
Taulukko 9.3: Tilastotietoja turnajaisvalinnan testiajojen parhaista neuroverkoista.

9.1.3 Turnajaisvalinta

Turnajaisvalinta jäi yksipisteristeytyksessä alhaisiin fitnessseihin. Maksimi-fitness pääsi juuri ja juuri yli 36000, nimittäin 36026:een. Minimi-fitness jäi vain 35082:een. Keskiarvokin jäi näin ollen matalaksi ollen 35500. Fitnessillä mitattuna turnajaisvalinta on yksipisteristeytyksen tapauksista selvästi heikoin.

Taulukkoon 9.3 kootuista muuttujista nähdään, että ajojen välillä ei ole ollut suuria eroja. Ainoastaan lähetettyjen pakettien lukumäärissä on eroavaisuuksia. Ajo 1 on lähettänyt vähiten paketteja ja on näin ollen saavuttanut myös parhaan fitnesssin.

Ajojen tasaisuus näkyy myös ajoajoista, joissa ei ole parhaan ja huonoimman välillä kuin alle kolmen tunnin ero. Nopeimmillaan algoritmi pyörähti läpi tasan 17 tunnissa. Kauiten aikaa kului ajolla 1, 19h 53min. Keskiarvoksi saatiin 18h 43min. Tasaisista ajoajoista huolimatta turnajaisvalinta ei pärjää todennäköisyyteen perustavalle valinnalle, jonka ajoaikojen



Kuva 9.4: Yksipisteristeytyksen turnajaisvalinnan ajokäyrät.

keskiarvoksi tuli 15h 27min.

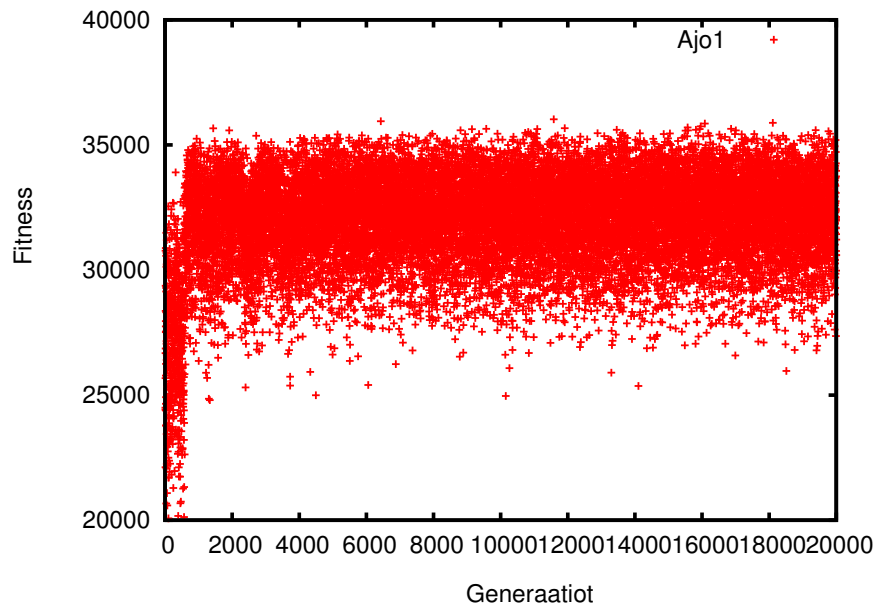
Kuvassa 9.4 piirretyistä ajojen fitness-käyristä nähdään, että parhaan maksimi-fitnessin saanut ajo 1 on saanut keskimäärin kuitenkin alhaisia fitness-arvoja. Sen fitness-käyrä kulkee selvästi alempana kuin ajojen 2 ja 3 käyrät. Tämä selittyy kuvasta 9.5, josta nähdään, että ajo 1 on saanut koko generaatioiden ajan fitnesssejä hyvin suurelta väliltä: 27000-36000. Ajon 3 fitnesssit on kuvassa 9.6. Verrattaessa ajojen 1 ja 3 fitnesssejä, huomataan että ajon 3 fitness arvot ovat keskittyneet huomattavasti pienemmälle välille.

9.2 Kaksipiste-risteytys

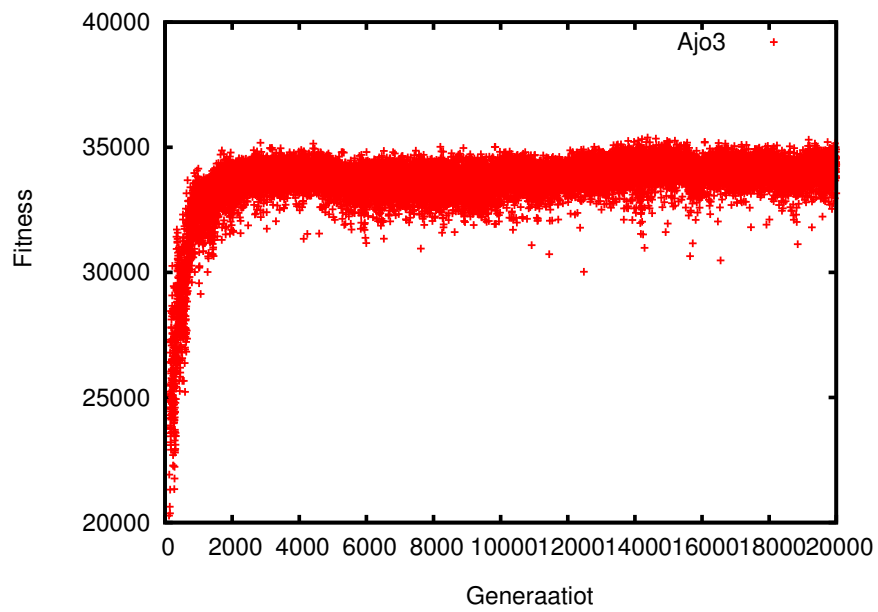
Tässä luvussa kerrotaan, millaisia tuloksia NeuroHaululla saatiin, kun optimoinnissa käytettiin mukana kaksipisteristeytystä.

9.2.1 Elitistinen valinta

Elitistinen valinta pärjasi kaksipisteristeytyksessä paremmin kuin yksipisteristeytyksen tapauksessa. Maksimi-fitness lähenteli jo 37000:tta päästen 36917:sta. Myös minimi-fitness ylsi reilusti päälle 36000:nen, nimittäin 36205:een. Keskiarvoksi kolmesta ajosta tuli 36480. Elitistinen valinta pärjasi kaksipisteristeytyksen tapauksista parhaiten.



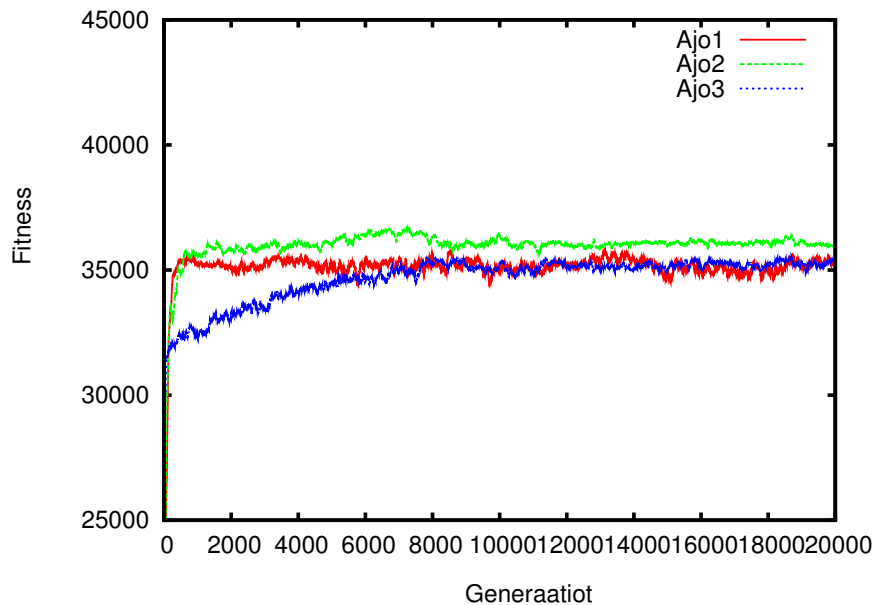
Kuva 9.5: Yksipisteristeytyksen turnajaisvalinnan ajon 1 fitnesssejä.



Kuva 9.6: Yksipisteristeytyksen turnajaisvalinnan ajon 3 fitnesssejä.

	Maksimi-fitness	Pakettien lkm	Vastaukset	Hypyt	Aika
Ajo1	36317	4533	817	3.64	15h 35min
Ajo2	36917	4833	835	20.17	14h 35min
Ajo3	36205	4745	819	5.1	17h 33min

Taulukko 9.4: Tilastotietoja elitistisen valinnan testiajojen parhaista neuroverkoista.



Kuva 9.7: Kaksipisteristeytyksen elitistisen valinnan ajokäyrät.

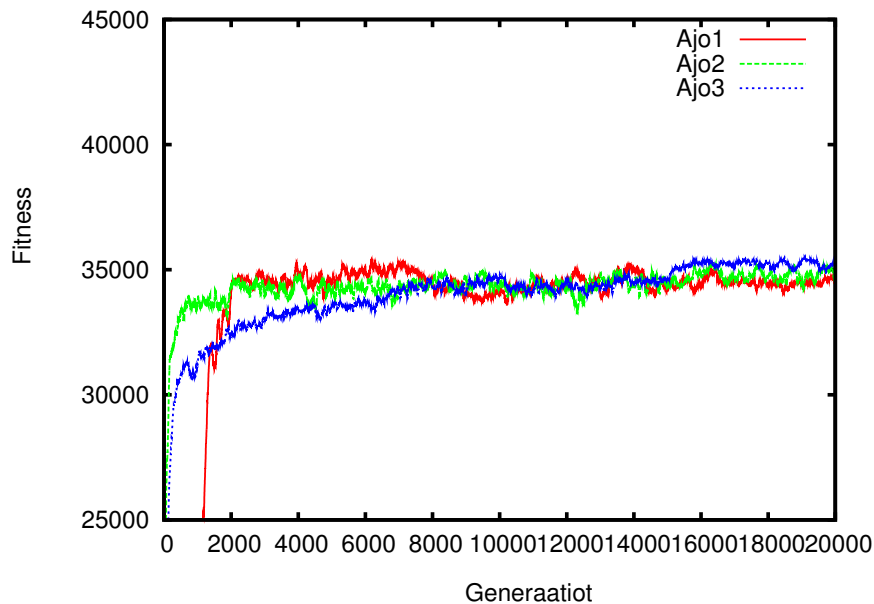
Taulukkoon 9.4 on jälleen koottu ajojen muuttujia. Paras fitness on saatu ajolla 2 ja heikoin ajolla 3. Ajolla 2 hyppyjen lukumäärä on selvästi suurempi kuin ajoilla 1 ja 3. Jälleen 37000:n tietämille päässyt fitness on käyttänyt noin 20 hyppyä, kun taas ajoilla 1 ja 3, joilla fitness jäi alhaiseksi, hypyt on jäänyt selvästi alle kymmenen. Muuten ajojen välillä ei ole suuria eroja.

Myös ajoajat ovat tasaiset. Nopeimman ja hitaimman ajon välillä on ajoissa eroa vain kolme tuntia. Vähiten aikaa kului ajolla 2, 14h 35min. Ajolla 3 aikaa kului 17h 33min. Keskiarvoksikin tuli muihin valintamentelmiin verrattuna huippuaika, 15h 54min.

Kuvan 9.7 fitness-käyristä nähdään, että ajo 2 on saanut alusta asti parempia fitness-arvoja kuin ajot 1 ja 3. Ajon 3 fitness on kehittynyt melko tasaisesti noin 8000 generaatioon asti, minkä jälkeen se on pyörinyt noin 35000:ssa. Ajojen 1 ja 2 fitnessit ovat kehittyneet huomattavasti nopeammin.

	Maksimi-fitness	Pakettien lkm	Vastaukset	Hypyt	Aika
Ajo1	36340	4810	823	4.7	14h 40min
Ajo2	35876	5274	823	5.84	18h 14min
Ajo3	36037	4813	817	3.54	19h 8min

Taulukko 9.5: Tilastotietoja todennäköisyyteen perustuvan valinnan testiajojen parhaista neuroverkoista.



Kuva 9.8: Kaksipisteristeytyksen todennäköisyyteen perustuvan valinnan valinnan ajokäyrät.

9.2.2 Todennäköisyyteen perustuva valinta

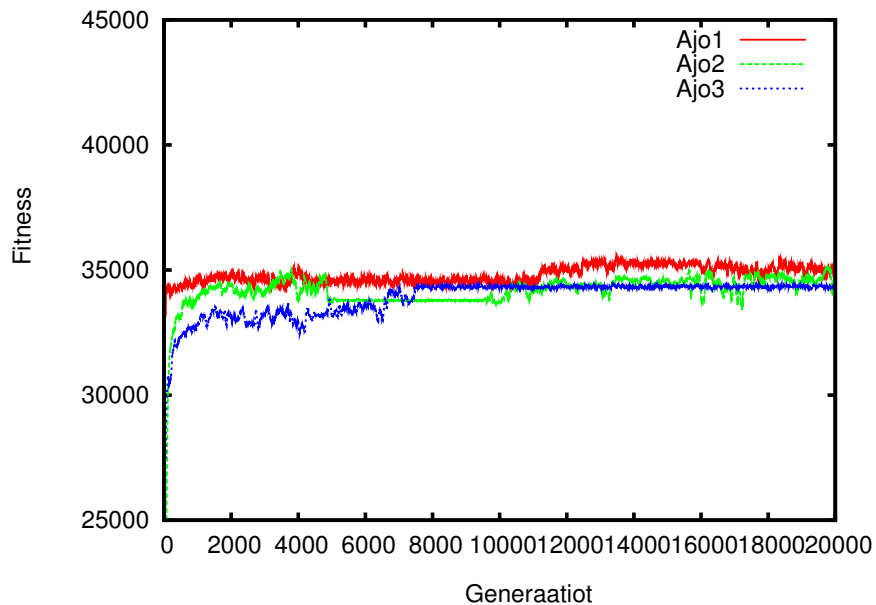
Todennäköisyyteen perustuvalla valinnalla ei fitnessin osalta päästy yhtä hyviin tuloksiin kuin elitistisellä valinnalla. Maksimi-fitness nousi 36340:een, mutta minimi-fitness jäi alle 36000:nen, päästen vain 35876:een. Keskiarvo ylsi hieman yli 36000, nimittäin 36084.

Todennäköisyyteen perustuvalla valinnalla saatiin hyvin tasaisia tuloksia. Sekä fitnessit että taulukkoon 9.5 kootut muuttujat pyörivät samoissa lukemissa. Tämä näkyy myös kuvan 9.8 ajojen fitness-käyristä. Mielenkiintoista on tarkastella ajon 3 fitness-käyrää, joka on noussut tasaisesti lähes melkein 20000 generaatioon saakka. Ajojen 1 ja 2 käyrät sen sijaan ovat melko tasaisia ja pyörivät hieman alle 35000:ssa noin generaatiosta 2000 alkaen.

Ajoajoissa sen sijaan on nähtävissä enemmän hajontaa. Nopeiten algoritmi pyörähti läpi 14 tunnissa 40 minuutissa. Kauimmillaan se vei aikaa 19h 8min. Keskiarvoksi tuli 17h

	Maksimi-fitness	Pakettien lkm	Vastaukset	Hypyt	Aika
Ajo1	36910	4840	835	15.38	17h 5min
Ajo2	35817	4383	804	17.7	16h 4min
Ajo3	35349	6301	833	10.47	21h 51min

Taulukko 9.6: Tilastotietoja turnajaisvalinnan testiajojen parhaista neuroverkoista.



Kuva 9.9: Kaksipisteristeytyksen turnajaisvalinnan ajokäyrät.

21min, joka on melko hyvä, mutta ei pärjää elitistisen valinnan alle 16 tunnille.

9.2.3 Turnajaisvalinta

Turnajaisvalinnalla saatiin kaksipisteristeytyksessä melko korkea maksimi-fitness: 36910, joka ylittää lähes tasoihin elitistisen valinnan kanssa. Minimi-fitness jäi kuitenkin heikoksi ja laski näin ollen myös keskiarvo-fitnessiä. Minimi-fitnessiksi tuli 35349 ja keskiarvo-fitnessiksi 36025.

Taulukkoon 9.6 on jälleen kerätty tilastotietoja ajojen suureista. Paras fitness saatiin ajolla 1 ja heikoin ajolla 3. Mitään isoja heittoja ajojen välillä ei ole havaittavissa. Ajolla 3 lähetettyjen pakettien lukumäärä on suurempi ja hypyt on pienempi kuin ajoissa 1 ja 2. Myös kuvan 9.9 perusteella ajot on helppo laittaa järjestykseen. Ylimpänä kulkevan ajon 1 fitness-käyrä tuotti myös parhaan maksimi fitnessin ja ajo 3 heikoimman.

Ajoaikoja tarkasteltaessa huomataan, että nopeimman ja hitaimman ajon ero on lähes kuusi tuntia, mikä on enemmän kuin muissa tutkimustapauksissa. Nopeimmillaan algoritmi vei aikaa 16h 4min, mutta kauimmillaan sillä kesti 21h 51min. Keskiarvoksi kolmelta ajolta tulee 18h 20min. Turnajaisvalinta ei pärjää aikoja tarkasteltaessa elitistiselle valinnalle eikä todennäköisyyteen perustuvalla valinnalla.

9.3 Mutaation voimakkuuden ja risteytysasteen vaikutus

Kirjallisuudessa [4] korostettiin, että mutaation voimakkuudella ja risteytysasteella on huomattava vaikutus evoluutioalgoritmien toimivuuteen. Tämän vuoksi selvitin vielä lopuksi, miten todennäköisyyteen perustuva valinta käyttäytyy yksi- ja kaksipisteristeyksessä, kun ko. arvoja muutetaan. Mutaation voimakkuudeksi vaihdettiin 0.025 ja risteytysasteeksi 0.66.

9.3.1 Yksipisteristeytys

Yksipisteristeyksessä muutetuilla mutaation voimakkuus - ja risteytysastearvoilla ei päästy aivan yhtä hyviin tuloksiin kuin alkuperäisillä arvoilla. Maksimi-fitness jäi nyt hieman alle 37000, kun alkuperäisillä arvoilla päästiin 37319:sta. Minimi-fitness nousi kuitenkin noin 800 pistettä korkeammalle. Muutetuilla arvoilla maksimi-fitnessiksi saatiin 36963, minimi-fitnessiksi 36267 ja keskiarvoksi kolmelta ajolta 36570.

Taulukkoon 9.7 kootuista suuresita nähdään, että ajot ovat olleet tasaiset millä tahansa suureella mitattuna. Suuria heittoja ei ole edes ajoajoissa.

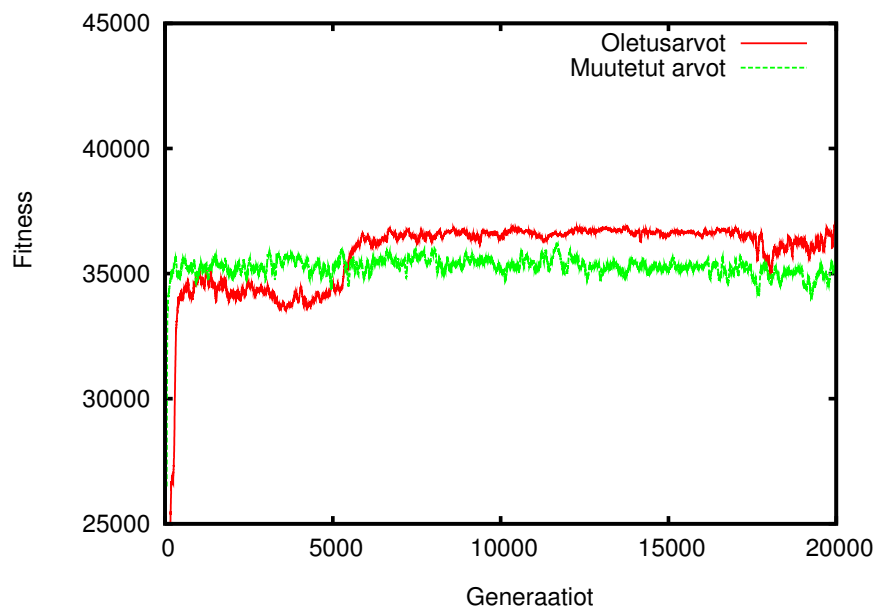
Alkuperäisillä arvoilla tehtyihin ajoihin verrattuna hyppyt ovat jääneet huomattavasti pienemmäksi. Alkuperäisillä arvoilla päästiin jopa yli 20 hyppyihin, kun taas nyt päästiin parhaimmillaan vain kuuteen.

Myös ajoajoissa on havaittavissa muutoksia alkuperäisillä arvoilla tehtyihin ajoihin verrattuna. Alkuperäisillä mutaation voimakkuudella ja risteytysasteella ajettaessa kolmen ajon keskiarvoksi saatiin 15h 27min ja kauimmillaan aikaa kului reilu 17 tuntia. Nyt muutetuilla arvoilla maksimi ajoaika tippui 25 minuuttia. Keskiarvoaika sen sijaan nousi tunnilla. Kolmen ajon keskiarvoksi tuli nimittäin 16h 58min.

Kuvaan 9.10 on piirretty oletusarvoilla ja muutetuilla arvoilla parhaat fitnessit saavuttaneet ajokäyrät. Kuvasta nähdään, että muutetuilla arvoilla 35000 fitness on saavutettu hyvin nopeasti, mutta tämän jälkeen fitness ei ole enää juuri kehittynyt. Alkuperäisillä arvoilla on fitnessiin tullut noin 5000 generaation kohdalla selvä hyppäys ja sen jälkeen se on pyörinyt korkeammissa lukemissa.

	Maksimi-fitness	Pakettien lkm	Vastaukset	Hypyt	Aika
Ajo1	36963	4437	828	6.18	16h 43min
Ajo2	36267	4933	824	3.63	17h 2min
Ajo3	36480	4670	823	4.42	16h 58min

Taulukko 9.7: Tilastotietoja muutetuilla mutaation voimakkuuden ja risteytysasteen arvoilla ajettujen todennäköisyyteen perustuvan valinnan testiajojen parhaista neuroverkoista yksipisteristeytyksen tapauksessa.



Kuva 9.10: Mutaation voimakkuuden ja risteytysasteen muutoksen vaikutus fitness-käyriin.

	Maksimi-fitness	Pakettien lkm	Vastaukset	Hypyt	Aika
Ajo1	37298	4702	840	21.34	21h 39min
Ajo2	36260	5290	831	3.93	28h 1min
Ajo3	36583	4517	822	7.04	27h 24min

Taulukko 9.8: Tilastotietoja muutetuilla mutaation voimakkuuden ja risteytysasteen arvoilla ajettujen todennäköisyyteen perustuvan valinnan testiajojen parhaista neuroverkoista kaksipisteristeytyksen tapauksessa.

9.3.2 Kaksipisteristeytytys

Kaksipisteristeytyksen tapauksessa muutetuilla mutaation voimakkuudella ja risteytysasteella päästiin selvästi parempiin fitnessseihin kuin alkuperäisillä arvoilla. Maksimi-fitness nousi 37298:een ja minimi-fitnesskin pääsi yli 36000, nimittäin 36360:een. Molemmat arvot ovat parempia kuin alkuperäisillä arvoilla saadut vastaavat. Näin ollen myös keskiarvo-fitness parani päästen 36711:sta.

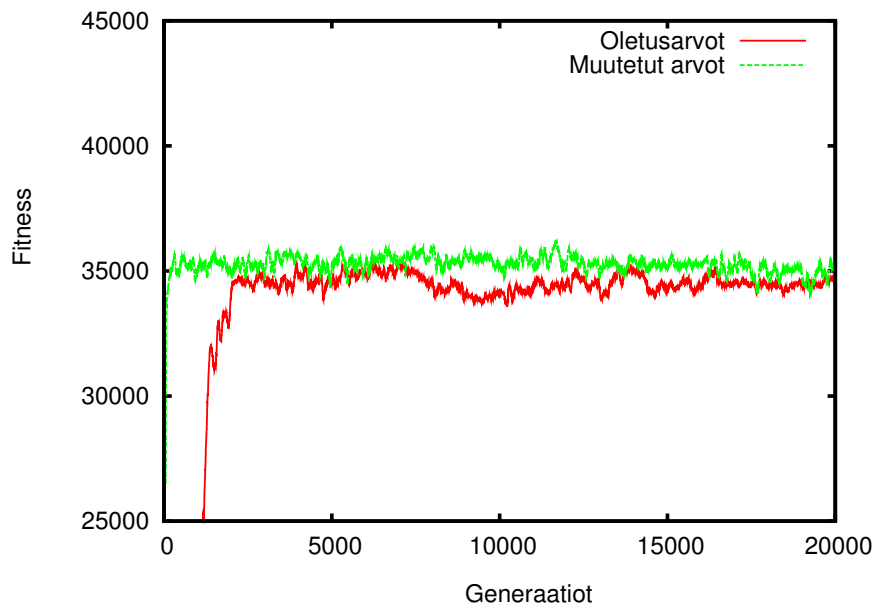
Taulukosta 9.8 nähdään, että ajojen välillä on eroa varsinkin hyppyissä. Ajolla 1, jolla päästiin parhaaseen fitnessiin, on käytetty myös eniten hyppyjä. Heikoimman fitnessin saanut ajo 2 taas käytti alle 4 hyppyä. Alkuperäisillä arvoilla ajettaessa hypyt jäivät kaikilla ajoilla alle kuuteen.

Myös ajoajoissa on eroa oletusarvoilla ajettuihin testeihin verrattuna. Alkuperäisillä arvoilla meni kauimmillaan reilu 19 tuntia ja keskiarvoaika oli hieman päälle 17 tuntia. Nyt ajoajat ovat selvästi suuremmat. Maksimijaksoiksi tuli nimittäin 28h 1min ja keskiarvoksikin 27h 58min. Ajoajat nousivat siis lähes kymmenen tuntia.

Ajoajat nousivat siis selvästi sekä yksipiste- että kaksipisteristeytyksen tapauksessa. Syynä on mitä ilmeisemmin se, että laskentaa on tullut niin paljon lisää, että se vaikuttaa oleellisesti myös ajallisesti algoritmin toimintaan.

Kuvassa 9.11 on jälleen parhaan fitnessin saaneiden ajojen fitness-käyrät. Fitness-käyristä nähdään, että muutetuilla arvoilla ajettu käyrä käyttäytyy samanlaisesti kuin yksipisteristeytyksen tapauksessa. Hyvä fitness-taso on saavutettu jo heti ensimmäisillä generaatioilla, minkä jälkeen selvää kehittymistä ei enää ole havaittavissa.

Sekä yksi- että kaksipisteristeytyksessä muutetuilla arvoilla tehdyistä ajoista saadut fitness-käyrät käyttäytyivät kaikki lähes samalla tavalla. Kaikilla ajoilla fitness-taso saavutettiin jo heti ensimmäisillä generaatioilla, minkä jälkeen fitness-käyrät pyörivät samoissa lukemissa kuin kuvien 9.10 ja 9.11 muutetuilla arvoilla piirretyt käyrät.



Kuva 9.11: Mutaation voimakkuuden ja risteytysasteen muutoksen vaikutus fitness-käyriin.

9.4 Yhteenveto tutkituista menetelmistä

9.4.1 Yhteenveto yksipisteristeytyksestä

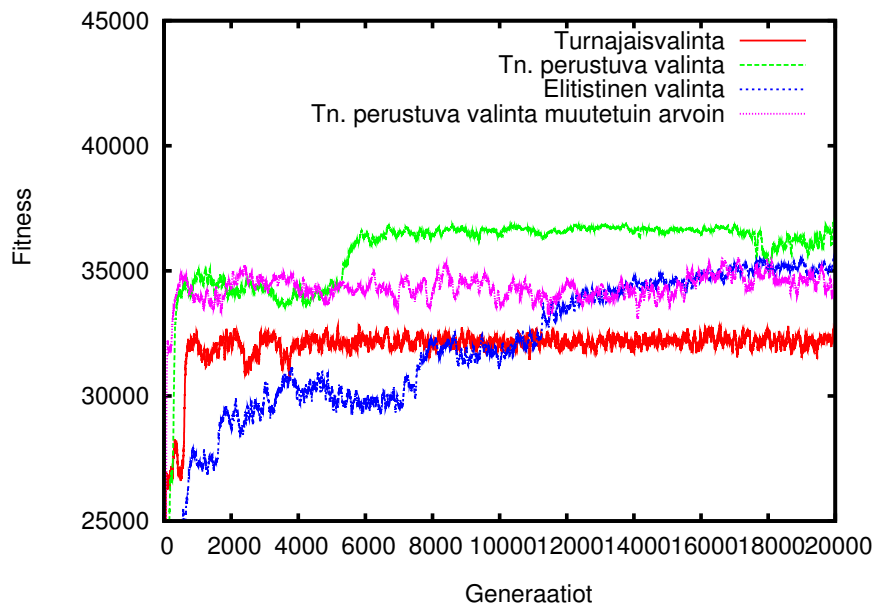
Fitnessillä mitattuna yksipisteristeytyksessä parhaiten pärjasi todennäköisyyteen perustuva valinta. Sillä saatiin paras maksimi-fitness sekä keskiarvo-fitness. Elitistinen valinta ylsi parhaaseen minimi-fitnessiin päästen 36134:ään. Sen maksimi-fitness ei kuitenkaan päässyt paljon minimiä korkeammalle, vain 36320:een, joten keskiarvokin jäi näin ollen melko alhaiseksi: 36200. Turnajaisvalinta menestyi yksipisteristeytyksessä heikoiten. Keskiarvo-fitness jäi ainoana alle 36000 päästen vain 35500:aan. Maksimi-fitnessiksi saatiin 36026 ja minimiksi 35082. Kaikkien menetelmien parhaiden ajojen fitnessit ja ajoajat on koottu taulukkoon 9.9.

Todennäköisyyteen perustuva valinta pärjasi parhaiten myös ajoaikoja tarkasteltaessa. Keskiarvoajaksi todennäköisyyteen perustuvalla valinnalla saatiin 15h 27min. Lähimmäs tätä pääsi todennäköisyyteen perustuva valinta muutetuilla arvoilla, mutta sekin jäi yli tunnin päähän ajalla 16h 58min. Heikoimmat ajoajat sai elitistinen valinta. Tulee kuitenkin muistaa, että yksi elitistisen valinnan ajoista ajettiin tehottomammalla koneella kuin kaikki muut testit, mikä vaikuttaa sekä maksimi- että keskiarvoaikaan. Elitistisessä valinnassa kesti kauimmillaan lähes 31 tuntia ja keskiarvokin on yli 20 tuntia, 22h 49min.

Mutaation voimakkuuden ja risteytysasteen muutoksilla ei näyttäisi yksipisteristeytyksen

	Maks-fitness	Min-fitness	Ka-fitness	Maks-aika	Min-aika	Ka-aika
Elitistinen valinta	36320	36134	36200	30h 52min	16h 32min	22h 49min
Tn. perustuva valinta	37319	35449	36578	17h 27min	13h 37min	15h 27min
Turnajaisvalinta	36026	35082	35500	19h 53min	17h	18h 43min
Tn. per. val. uusilla arvoilla	36963	36267	36570	17h 2min	16h 43min	16h 58min

Taulukko 9.9: Tilastotietoja yksipisteristeytyksen testiajojen parhaista neuroverkoista.



Kuva 9.12: Yksipisteristeytyksen parhaiden fitnessien ajokäyrät.

tapauksessa olevan suurta vaikutusta. Maksimi-fitness jäi alhaisemmaksi muutetuilla kuin alkuperäisillä arvoilla. Minimi-fitness sen sijaan nousi 35449:stä 36267:ään. Keskiarvoon tuli vain 8 pisteen ero. Suuria johtopäätöksiä arvojen muutosten vaikutuksesta ei kuitenkaan voi tehdä, koska sitä kokeiltiin vain yhteen menetelmään ja vain yhdellä muutoksella.

Kuvaan 9.12 on kerätty valintamenetelmien ajoista ne, joilla saatiin parhaat fitnessit. Kuten kuvasta nähdään, turnajaisvalinta osoittautui heikoimmaksi menetelmäksi. Myös parhaan menetelmän voi nimetä kuvasta katsomalla. Todennäköisyyteen perustuva valinta alkuperäisellä mutaation voimakkuudella ja risteytysasteella antoi parhaat tulokset.

Tutkimuksen perusteella voidaan sanoa, että näillä testitapauksilla yksipisteristeytyksessä todennäköisyyteen perustuva valinta oletusarvoilla menestyi parhaiten.

	Maks-fitness	Min-fitness	Ka-fitness	Maks-aika	Min-aika	Ka-aika
Elitistinen valinta	36917	36205	36480	17h 33min	14h 35min	15h 54min
Tn. perustuva valinta	36340	35876	36084	19h 8min	14h 40min	17h 21min
Turnajaisvalinta	36910	35349	36025	21h 51min	16h 4min	18h20min
Tn. per. val. uusilla arvoilla	37298	36260	36711	28h 1min	21h 39min	27h 24min

Taulukko 9.10: Tilastotietoja kaksipisteristeytyksen testiajojen parhaista neuroverkoista.

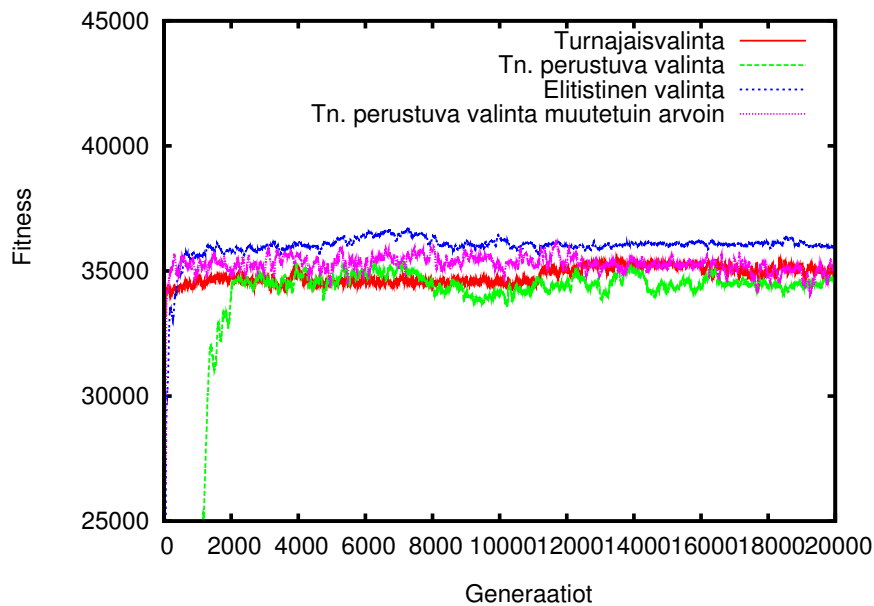
9.4.2 Yhteenveto kaksipisteristeytyksestä

Kaksipisteristeytyksessä parhaiten pärjasi todennäköisyyteen perustuva valinta muutetuilla mutaation voimakkuus ja risteytysaste arvoilla. Fitnessillä mitattuna se pärjasi parhaiten kaikista tutkimustapauksista, mukaanlukien yksipisteristeytyksessä. Maksimi-fitnessiksi saatiin 37298, minimi-fitnessiksi 36260 ja keskiarvoksi 36711. Lähimmäs todennäköisyyteen perustuvan valinnan muutetuilla arvoilla tehtyjä ajoja pääsi elitistinen valinta. Elitistisellä valinnalla maksimi-fitness nousi 36917:n ja keskiarvo-fitnessiksi saatiin 36480. Turnajaisvalinta ja todennäköisyyteen perustuva valinta alkuperäisillä arvoilla pyörivät keskiarvo-fitness-arvoissa lähes samoissa arvoissa, hieman yli 36000:ssa.

Mielenkiintoista oli havaita, että sekä yksi- että kaksipisteristeytyksessä hyvät, lähelle 37000 tai sen yli päässeet, fitnessit saatiin ajoilla, joilla hypyt nousivat 20 tietämille.

Taulukon 9.10 ajoaikoja tarkasteltaessa muutetuilla arvoilla ajettu todennäköisyyteen perustuva valinta ei pärjää enää läheskään yhtä hyvin. Itseasiassa sen ajat ovat kaikista heikoimpia jääden yli kymmenen tuntia parhaista. Sen sijaan elitistinen valinta menestyi hyvin myös ajoajoissa. Se sai parhaan tuloksen niin minimi-, maksimi- kuin keskiarvoajaksakin. Keskiarvoajaksi tuli 15h 54min. Lähimmäs tätä pääsi todennäköisyyteen perustuva valinta alkuperäisillä mutaation voimakkuudella ja risteytysasteella saaden keskiarvoksi 17h 21min.

Mutaation voimakkuuden ja risteytysasteen muutoksilla oli selvä vaikutus kaksipisteristeytyksessä. Niin minimi-, maksimi- kuin keskiarvo-fitness nousi selvästi, vähintään 500 pistettä. Ajoaikoihin muutoksella sen sijaan oli negatiivinen vaikutus. Ajoajat kasvoivat arvojen muutoksen myötä lähes kymmenen tuntia. Ajoaikojen nousu oli seurasta laskennan lisääntymisestä. Mutaation voimakkuuden ja risteytysasteen vaikutusta pitäisi tutkia enemmänkin ennenkuin mitään johtopäätöksiä voidaan tehdä. Näiden suppeiden testien valossa näyttäisi kuitenkin siltä, että löytämällä mutaation voimakkuudelle ja risteytysasteelle oikeat arvot fitness voi nousta vieläkin korkeammalle.



Kuva 9.13: Kaksipisteristeytyksen parhaiden fitnessien ajokäyrät.

Kuvaan 9.13 on koottu kaikista valintamenetelmistä parhaan fitnessin antanut ajo. Fitness-käyristä nähdään, miten menetelmät keskimäärin menestyivät. Kuvan perusteella elitistinen valinta olisi paras, seuraavana todennäköisyyteen perustuva valinta uusilla arvoilla ja kaksi heikointa olisivat turnajaisvalinta ja todennäköisyyden perustuva valinta alkuperäisillä arvoilla.

Otettaessa huomioon fitnessin lisäksi myös ajoihin kuluneet ajat, elitistinen valinta pärjasi selvästi parhaiten. Se vei vähiten aikaa ja pääsi silti toiseksi parhaisiin fitnessseihin. Parhaat fitnessit saanut todennäköisyyteen perustuva valinta muutetuilla mutaation voimakkuus ja risteytysaste arvoilla käytti aikaa yli kymmenen tuntia enemmän kuin elitistinen valinta.

9.5 Johtopäätökset

Tutkimuksen tarkoituksena oli selvittää, mitkä optimointimenetelmät sopivat monikerroksisen perceptron-neuroverkon optimointiin. Tässä vaiheessa keskityttiin kokeilemaan eri optimointimenetelmiä ja rajaamaan näin mahdollisesti epäsovivimmat pois.

Suorahakumenetelmistä käytettiin Powellin menetelmää, Powellin toista menetelmää ja Zangwillen menetelmää. Tehtävän suuren dimension vuoksi nämä menetelmät eivät sopineet ratkaisuksi ongelmaan.

Evoluutioalgoritmeista otettiin uutena piirteenä mukaan risteytys, mutaatio vaihdettiin tapahtuvaksi geeneittain ja eri valintamenetelmiä kokeiltiin. Yksipisteristeytyksessä parhaaksi valintamenetelmäksi osoittautui todennäköisyyteen perustuva valinta, joka ainoana pääsi yli 37000 fitnessin. Se pärjasi parhaiten myös ajoaikoja tarkasteltaessa. Kaksipisteristeytyksessä parhaisiin tuloksiin päästiin myös todennäköisyyteen perustuvalla valinnalla, jossa mutaation voimakkuus oli 0.024 ja risteytysaste 0.667. Tämä oli ainoa, jolla kaksipisteristeytyksessä päästiin yli 37000. Myös elitistinen valinta pärjasi kaksipisteristeytyksessä hyvin saaden maksimi-fitnessiksi 36917. Elitistinen valinta on aikoja tarkasteltaessa selvästi todennäköisyyteen perustuvaa valintaa parempi. Niiden välille tuli eroa yli kymmenen tuntia. Elitistisen ajon nopeuteen vaikuttaa todennäköisesti se, että siinä jatkoon pääsevät neuroverkot valitaan suoraan paremmuusjärjestyksessä, kun taas todennäköisyyteen perustuvassa valinnassa joudutaan tekemään laskentaa myös valintavaiheessa.

Mutaation voimakkuuden ja risteytysasteen muutosten vaikutusta tutkittiin vain todennäköisyyteen perustuvassa valinnassa. Näiden suppeiden testien perusteella näyttäisi siltä, että ko. arvoilla on vaikutusta optimoinnin tehokkuuteen. Tiukan aikataulun takia ko. arvojen vaikutusta ei ehditty tutkia enempää. Arvot optimoimalla päästäisiin todennäköisesti vieläkin parempiin tuloksiin.

Tutkimuksen tuloksia ei voi suoraan verrata Vapan ym. [35] tutkimukseen, sillä tutkimuksissa on käytetty eri määrä sisääntuloparametreja. Vapan ym. tutkimuksessa oli käytössä vain seitsemän sisääntuloparametria. Parametrien lisäys generoi lisää painokertoimia, mikä parantaa algoritmin toimintaa. Fitness siis näin ollen paranee. Toisaalta ajoon kuluva aika kuitenkin kasvaa, koska laskemista on myös enemmän.

Alkuperäisellä NeuroHaun optimointimenetelmällä ja 23:lla sisääntuloparametrilla on päästy noin 36500 fitnessiin, joten osa tutkimuksessa kokeilluista menetelmistä vaikuttaisivat hyviltä vaihtoehdoilta NeuroHaun optimointiin. Yksipisteristeytys todennäköisyyteen perustuvalla valinnalla ja kaksipisteristeytys elitistisellä valinnalla saivat fitnessiä 37000 tietämille, mikä on selvä parannus alkuperäiseen NeuroHakuun verrattuna. Em. menetelmät vaikuttaisivat myös ajallisesti nopeilta menetelmiltä.

10 Loppupäätelmät

10.1 Yhteenveto

Tutkimuksessa selvitettiin, mitkä optimointimenetelmät sopivat monikerroksisen perceptron-neuroverkon optimointiin. Tutkielman teoriaosuudessa luvussa kaksi käytiin lyhyesti läpi vertaisverkkoja ja niiden toimintaa. Tutkielma keskittyi kuitenkin neuroverkkoihin, joten niiden rakennetta ja oppimista käsiteltiin tarkemmin luvussa kolme. Luvut neljä ja viisi ovat tutkielman teorian kannalta ehkäpä tärkeimmät, sillä niissä käydään läpi optimointia ja eri optimointimenetelmiä.

Käytännön osuus työssä keskittyi optimointimenetelmien soveltamiseen NeuroHaun tapauksessa. Resurssien hakuongelma ja sen muotoilu algoritmina esiteltiin luvussa kuusi. Luku seitsemän esitteli resurssien hakuun ratkaisuksi kehitetyn NeuroHaun toimintaa ja rakennetta. Tutkimusympäristö, käytetyt menetelmät ja käytetyt laitteet esiteltiin luvussa kahdeksan. Saadut tulokset analysoitiin luvussa yhdeksän.

10.2 Tulevaisuus

Optimointimenetelmiä on olemassa paljon ja tässä tutkimuksessa ehdittiin käydä läpi vain muutama vaihtoehto. Tulevaisuudessa voidaan jatkaa jäljellä olevien menetelmien kokeilemistä tai keskittyä evoluutioalgoritmeihin. Esimerkiksi simuloity jäähdytys tai tabu-haku olisivat varmasti kokeilemisen arvoisia menetelmiä. Evoluutioalgoritmeistakin löytyy vielä paljon kokeiltavaa. Esimerkiksi oikeat arvot mutaation voimakkuudelle ja risteytysasteelle löytämällä olisi todennäköistä päästä entistä parempiin tuloksiin.

Painokertoimet jäivät tämän tutkimuksen testitapauksissa huomattavasti pienemmiksi kuin alkuperäisellä NeuroHaulla saadut painokertoimet. Tästä herää kysymys, kannattaako mutaatio toteuttaa geeneittäin vai sittenkin yksilöittäin kuten alkuperäisessä NeuroHausassa oli? Geeneittäin mutatoidessa mutaatiota tapahtuu harvemmin ja painokertoimien muutokset ovat näin ollen pienempiä. Entä riittääkö kaksipisteristeytys vai pitäisikö sekin viedä geenitasolle käyttämällä tasaista risteytystä?

Tutkimuksessa saatiin rohkaisevia tuloksia siitä, että parempia fitnesssejä on mahdollista saavuttaa ja että eri menetelmien välillä ajoajoissa on selviä eroja. Oikeat kombinaatiot va-

lintaan, risteytykseen ja mutaatioon löytämällä, saadaan NeuroHausta entistä tehokkaampi.

Lähteet

- [1] Agora Center, *Peer-to-Peer Communication/Computing*, <URL: <http://www.jyu.fi/agora-center/tutkimus/p2p.shtml>>, viitattu 11.6.2005.
- [2] Beyer H.G. *The Theory of Evolutionary Strategies*, Springer, Saksa, 2001.
- [3] BitTorrent, <URL: <http://www.bittorrent.com/>>, viitattu 13.8.2005.
- [4] Buckles B.P., Petry F.E., *Genetic Algorithms*, IEEE Computer Society Press, 1992.
- [5] Bäck T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, 1996.
- [6] Bäck T., Hammel U., Schwefel H-P. *Evolutionary Computation: Comments on the History and Current State*, Evolutionary Computation, vol. 1, no. 1, 1997, s. 3-17.
- [7] Céa J., *Optimisation théorie et algorithmes*, Dunod, Paris, 1971.
- [8] Cheese Factory, <URL: <http://tisu.it.jyu.fi/cheesefactory/>>, viitattu 13.6.2005.
- [9] CPLEX, <URL: <http://www.cc.jyu.fi/konkkola/atk/ilog/cplex80/>>, viitattu 4.7.2005.
- [10] DACS – Data & Analysis Center for Software, *Artificial Neural Networks Technology*, <URL: <http://www.thedacs.com/techs/neural/neural.title.html>>, viitattu 12.7.2005.
- [11] Freenet – *The Freenet Project*, <URL: <http://freenetproject.org/index.php?page=index>>, viitattu 6.8.2005.
- [12] Gene K., *Gnutella, Peer to Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly, 2001, s. 94-122.
- [13] GINO, <URL: <http://www-fp.mcs.anl.gov/otc/Guide/SoftwareGuide/Blurbs/gino.html>>, viitattu 18.7.2005.

- [14] Honkela T. *Neuroverkot: johdatus moderniin tekoälyyn*, Teknillinen korkeakoulu, saatavilla html-muodossa osoitteesta <URL: <http://www.cis.hut.fi/tho/stes/step96/honkela2/>>, viitattu 9.5.2005.
- [15] Howstuffworks, *How The Old Napster Worked*, <URL: <http://computer.howstuffworks.com/napster.htm>>, viitattu 20.6.2005.
- [16] Jain A., Fogel D. *Case Studies in Applying Fitness Distributions in Evolutionary Algorithms. II. Comparing the Improvements from Crossover and Gaussian Mutation on Simple Neural Networks*, Proceedings of First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, 2000, s. 91-97.
- [17] KaZaA, <URL: <http://www.kazaa.com>>, viitattu 4.7.2005.
- [18] Koenig A.C., *A Study of Mutation Methods for Evolutionary Algorithms*, University of Missouri-Rolla, 2002.
- [19] Kotilainen N., Vapa M., Weber M., Töyrylä J. and Vuori J., *P2PDisCo – Java Distributed Computing for Workstations Using Chedar Peer-to-Peer Middleware*, Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2005), Denver, Colorado, USA, 2005. Löytyy pdf-muodossa osoitteesta <URL: <http://tisu.it.jyu.fi/cheesefactory/documents/P2PDisCoChedar.pdf>>, viitattu 1.8.2005.
- [20] Lee C-Y., Yao X. *Evolutionary Algorithms with Adaptive Lévy Mutations*, Proceedings of the 2001 IEEE Congress on Evolutionary Computation, 2001.
- [21] Lewis R.M., Torczon V. ja Trosset M.W., *Direct search methods: then and now*, Journal of Computational and Applied Mathematics, Vol. 124, 2000, s. 191-207.
- [22] LINDO, <URL: <http://www.lindo.com/>>, viitattu 7.7.2005.
- [23] Miettinen Kaisa, *Epälineaarinen optimointi*, Jyväskylä 2003.
- [24] Nakanishi H., Kohda T., Inoue K. *A Design Method of Optimal Control System by Use of Neural Network*, In Proceedings of The 1997 IEEE International Conference on Neural Networks, 1997.

- [25] Napster, <URL: <http://www.napster.com>>, 29.6.2005.
- [26] Powell M.J.D., *An efficient method for finding the minimum of a function of several variables without calculating derivatives*, Computer Journal, Vol. 7, 1964, s. 155-162.
- [27] Quagliarella D., Périaux J., Poloni C., Winter G., *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science*, Wiley, 1998.
- [28] Reed R.D., Marks II R.J. *Neural Smoothing Supervised Learning in Feedforward Artificial Neural Networks*, A Bradford Book, London, England, 1998.
- [29] Schollmeier R., *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*, Proceedings of the First International Conference on Peer-to-Peer Computing, 2002.
- [30] Skype, <URL: <http://www.skype.com/intl/fi/>>, viitattu 19.7.2005.
- [31] Tekes, *Neurolaskennan mahdollisuudet*, saatavilla html-muodossa osoitteesta <URL: http://www.tekes.fi/julkaisut/raportit/43_94>, viitattu 26.5.2005.
- [32] Tietokone, 5/2005.
- [33] Tietotekniikan laitos, *Optimoitimenetelmät*, Jyväskylän yliopisto, Tietotekniikan laitos, kevät 2003.
- [34] Töyrylä J., *Building NeuroSearch – Intelligent Evolutionary Search Algorithm For Peer-to-Peer Environment*, Pro gradu -tutkielma, Jyväskylän yliopisto, 2004. Saatavilla pdf-muodossa osoitteesta <URL:<http://tisu.it.jyu.fi/cheesefactory/documents/MastersThesisJoniToyryla.pdf>>, viitattu 28.7.2005.
- [35] Vapa M., Kotilainen N., Auvinen A., Kainulainen H and Vuori J., *Resource Discovery in P2P Network Using Evolutionary Neural Networks*, International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA 2004), Luxembourg, 2004. Saatavilla pdf-muodossa osoitteesta <URL: <http://tisu.it.jyu.fi/cheesefactory/documents/NeuroSearchAISTA2004.pdf>>, viitattu 1.8.2005.

- [36] VTT elektronikka, *Vuorovaikutteiset järjestelmät - Älykkäät järjestelmät*, saatavilla html-muodossa osoitteesta <URL: http://www.vtt.fi/ele/tutkimus/ais_vanha/aly/neuroverkot.htm>, viitattu 28.7.2005.
- [37] Walsh G.R., *Methods of Optimization*, John Wiley & Sons Ltd., 1975.
- [38] WWW-NIMBUS, <URL: <http://nimbus.mit.jyu.fi/>>, viitattu 30.6.2005.
- [39] Zangwill W.I., *Minimizing a function without calculating derivatives*, Computer Journal, Vol 10, 1967, s. 293-296.