Teemu Keltanen

Vertaisverkkojen topologian hallinta neuroverkoilla

Tietotekniikan
pro gradu -tutkielma
20. joulukuuta 2006

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Teemu Keltanen
Yhteystiedot: teemukel@jyu.fi
Työn nimi: Vertaisverkkojen topologian hallinta neuroverkoilla
Title in English: Topology Management in P2P Networks using Evolutionary Neural Networks
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 120
Tiivistelmä: Resurssien tehokas löytäminen on vertaisverkkojen olennainen ongelma. Vertaisverkkojen tehokkuuteen voidaan vaikuttaa resurssien hakualgoritmien lisäksi muuttamalla itse vertaisverkon rakennetta. Tutkielman tavoitteena on analysoida neuroverkkojen soveltamista vertaisverkkojen topologian hallintaan.
English abstract: Resource discovery is an essential problem in peer-to-peer networks. In addition to search algorithms, the structure of the peer-to-peer network itself has influence on the efficiency of the network. This thesis analyses the adaptation of neural networks for topology management of peer-to-peer networks.
Avainsanat: vertaisverkko, topologian hallinta, neuroverkot, evoluutioalgoritmit
Keywords: peer-to-peer, topology management, neural networks, evolutionary algorithms

Sisältö

1 Johdanto

Vertaisverkot (engl. *Peer-to-Peer networks, P2P networks*) ovat tekniikan kehittyessä saavuttaneet koko ajan suurempaa suosiota erilaisten resurssien jakamisvälineenä. Vuosituhannen vaihteen jälkeen ne ovat saaneet julkisuutta lähinnä laittomien viihdetiedostojen jakamisen yhteydessä, mutta musiikkitiedostojen jakeluun kehitetyn Napsterin ja muiden vastaavien sovellusten valtava suosio kuitenkin osoitti, kuinka hyvin vertaisverkot soveltuvat koneiden verkottumiseen suuressa mittakaavassa. Tästä johtuen niin tiede- kuin yritysmaailmakin on kiinnostunut aiheesta ja tänä päivänä se onkin laaja tutkimuskohde ympäri maailmaa. Vertaisverkkojen ympärille kehitetyt sovellukset, kuten esimerkiksi internetpuheluja välittävä Skype [66], vertaisverkkoalustojen kehitys mobiililaitteille [77] sekä grid-laskennan kehitys vertaisverkkoalustalle [11], kertovat yritysmaailman kasvavasta kiinnostuksesta aihetta kohtaan. Vertaisverkkoja käytetään vielä pääasiassa viihdetiedostojen jakamiseen, jolloin ei yleensä aseteta korkeita vaatimuksia tiedostojen saatavuudelle. Monissa kaupallisissa sovelluksissa haluttu resurssi on kuitenkin löydettävä tietyllä varmuudella [61]. Tänä päivänä vertaisverkkojen ongelmia sekä kehitysmahdollisuuksia selvitetään yleensä laboratorioissa simuloimalla sen sijaan, että testaajina toimisivat loppukäyttäjät [33].

Erilaisia vertaisverkkoja tutkittaessa on syytä kiinnittää huomiota erityisesti niiden vikasietoisuuteen, skaalautuvuuteen sekä mahdollisimman tehokkaaseen resurssien löytämiseen. Hajautetusta luonteestaan johtuen vertaisverkot ovat vikasietoisempia kuin keskitettyihin palvelimiin perustuvat järjestelmät, sillä minkä tahansa koneen poistuminen vertaisverkosta käsitellään normaalina tapahtumana [50]. Erilaisten hakualgoritmien tutkimiseen on käytetty paljon aikaa ja niitä on yritetty tehostaa myös organisoimalla verkon rakennetta eli topologiaa eri tavoin. Tämän tutkimuksen näkökulmasta vertaisverkon tärkeimpänä ongelmana pidetään resurssien löytämistä puhtaassa järjestämättömässä vertaisverkossa

Tässä tutkimuksessa selvitetään vertaisverkon topologian vaikutusta resurssihakujen tehokkuuteen kontrolloimalla topologiaa neuroverkoilla (engl. *neural networks*). Raa'asti yksinkertaistettuna neuroverkkoa voidaan pitää mustana laatikkona, jolle annetaan sisääntuloarvoja ja laskutoimituksen jälkeen saadaan laatikosta jokin tulos eli neuroverkko on funktioapproksimaattori. Neuroverkkojen soveltuvuus tilanteisiin, joissa ongelmaan vaikuttavien tekijöiden suhde on monimutkainen eikä ongel-

ma ratkea perinteisin matemaattisin keinoin, on todettu jo 1940-luvun loppupuolella eli samaan aikaan kun nykyiset tietokoneet ovat syntyneet [22]. Neuroverkkojen suosio on lisääntynyt rajusti parina viime vuosikymmenenä tietokoneiden laskentatehon kasvaessa ja neuroverkkotutkimuksen edetessä.

Tässä tutkimuksessa neuroverkkoa käytetään oppivana luokittimena, jonka tehtävänä on kertoa vertaisverkon solmulle, kannattaako sen luoda yhteys johonkin toiseen solmuun siitä kerätyn tiedon perusteella. Neuroverkon opetukseen käytetään evoluutioalgoritmeja. Evoluutioalgoritmin etuna on hyvin opetettu neuroverkko, vaikkei tietyn ongelmaparametrin vaikutuksesta muihin parametreihin tai ongelman lopputulokseen olisi mitään tietoa [54]. Hyvin opetetulla neuroverkolla tarkoitetaan neuroverkkoa, jolla optimoinnissa päästään lokaalin optimin sijaan lähelle globaalia optimia.

1.1 Tutkimusongelma

Resurssien tehokas löytäminen on vertaisverkkojen olennainen ongelma, sillä niissä ei ole keskitettyä hakemistoa, josta resurssit voisi löytää. Vertaisverkkojen tehokkuuteen voidaan vaikuttaa resurssien hakualgoritmien lisäksi muuttamalla itse vertaisverkon rakennetta eli topologiaa [78]. Vertaisverkkojen topologia voidaan jakaa kahteen osaan siinä mielessä, että verkon looginen topologia eli päällysverkko muodostuu verkon fyysisen topologian päälle. Päällysverkko muodostuu vertaisverkkoon kuuluvista solmuista ja niiden välisistä yhteyksistä ilman, että otetaan kantaa siihen, miten yhteydet sijaitsevat fyysisessä verkossa. Tässä tutkimuksessa tutkitaan loogisen päällysverkon muuttamista simuloimalla eikä oteta kantaa taustalla olevaan fyysiseen verkkoon. Tutkielman neuroverkko toimii vuorovaikutuksessa resurssien hakualgoritmin kanssa siten, että suoritettujen hakujen yhteydessä kerätyn tiedon avulla muutetaan verkon rakennetta aina tietyin aikavälein. Toisin sanoen verkon rakenne perustuu sen solmujen ominaisuuksiin ja tapahtumiin eikä sattumanvaraisuuteen tai johonkin yhteen tiettyyn ominaisuuteen kuten useissa tutkimuksissa. Neuroverkot soveltuvat topologian hallintaan, jos niiden avulla saadaan aikaan mahdollisimman tehokas verkkotopologia kulloinkin käytettävälle hakualgoritmille. Tehokkaalla verkkotopologialla tarkoitetaan tässä tutkimuksessa sitä, että resurssien löytämiseen käytetään mahdollisimman vähän tietoliikennekapasiteettia. Tutkimuksen päähakualgoritmina käytettiin korkeimman asteen hakua (engl. *highest degree search, HDS*) [2], sillä sen tiedetään toimivan parhaiten potenssijakautuneessa verkossa [23]. Tätä tietoa voidaan käyttää hyväksi analysoitaessa tutkimuksessa tuotettuja topologioita. Myös muita hakualgoritmeja testattiin, jotta

nähdään, minkä muotoinen topologia kullakin algoritmilla saadaan ja miten muut hakualgoritmit toimivat jonkin toisen haun perusteella muodostetussa verkossa. Muut käytettävät hakualgoritmit olivat satunnaiskävelijä (engl. *random walker*) [40] ja leveyshaku (engl. *breadth-first search, BFS*) [41].

Vertaisverkon solmut etsivät resursseja hajautetusti. Toisin sanoen solmuilla on tietoa vain naapureista, naapureiden naapureista sekä niistä aiemmista resurssien hakutapahtumista, joihin se on itse osallistunut. Topologian muutosprosessissa kukin solmu kysyy opetetulta neuroverkolta, mitkä tiedossa olevat solmut sopisivat naapureiksi tällä hetkellä.

Ennen kuin tämä prosessi toimii kuvatulla tavalla, neuroverkot on opetettava. Tässä tutkimuksessa neuroverkkojen opetukseen käytetään evoluutioalgoritmia. Opetukseen tarvitaan hyvyysfunktio (engl. *fitness function*), joka tässä tutkimuksessa mittaa koko vertaisverkon liikennemäärää ja jota käytetään mittarina neuroverkkojen evolutionääriselle kehittymiselle. Toisin sanoen ensin muodostetaan neuroverkon päätösten avulla vertaisverkkotopologia, jonka tehokkuutta mitataan hyvyysfunktiolla. Mitä tehokkaampi vertaisverkko on, sitä parempi neuroverkko on. Hyvyysfunktion stabilisoituminen evoluution edetessä kertoo siitä, että evoluutioalgoritmi ei osaa enää merkittävästi parantaa neuroverkon toimintaa.

1.2 Cheese Factory -projekti

Jyväskylän yliopiston Cheese Factory -tutkimusprojekti [13] tutkii P2P-verkkoja ja erityisesti resurssien etsintää ja topologian hallintaa näissä verkoissa. Projektissa on toteutettu Chedar-vertaisverkkoalusta [4], jonka ominaisuuksiin kuuluu muun muassa ohjattu topologian hallinta. Lisäksi projektissa tutkitaan hajautettua laskentaa sekä mobiileja vertaisverkkoja.

Tämän tutkimuksen työkalut on niin ikään kehitetty Cheese Factory -projektissa. Vertaisverkon toimintaa tutkitaan P2PRealm-vertaisverkkosimulaattorilla [37], joka sisältää vertaisverkkomallin, neuroverkkototeutuksen ja algoritmit neuroverkkojen opetukseen. Lisäksi simulaattorista saadaan tulosteet P2PStudio-monitorointiohjelmaan [36], jolla saadaan kuva vertaisverkon topologiasta. P2PRealm-simulaattorilla on aiemmin tutkittu NeuroSearch-hakualgoritmia [72], jossa neuroverkot ohjaavat resurssikyselyiden etenemistä. Tämän tutkimuksen yhteydessä simulaattoriin lisättiin topologian hallinta neuroverkoilla.

1.3 Tutkielman rakenne

Toisessa luvussa esitellään vertaisverkkojen arkkitehtuureja sekä tutkimustuloksia niiden tehostamiseksi. Luvussa 3 käsitellään neuroverkkojen peruskäsitteet sekä esitellään niiden yleisimmät opetustavat. Luvussa 4 esitellään evoluutioalgoritmit, joita käytetään tässä tutkimuksessa neuroverkon opettamiseen. Luvussa 5 luodaan lyhyt katsaus olemassa oleviin vertaisverkkosimulaattoreihin ja esitellään tarkemmin tässä tutkimuksessa käytetty P2PRealm-simulaattori ja sen neuroverkkototeutus sekä simulaattoriin lisätty topologian hallinta -algoritmi NeuroTopologia. Luvussa 6 käydään läpi tutkimustapaukset sekä saadut tulokset. Yhteenveto tutkielmasta esitellään luvussa 7.

2 Vertaisverkot

Tässä luvussa esitellään vertaisverkon käsite ja yleisimmät vertaisverkkojen arkkitehtuurit. Lisäksi käydään läpi tärkeimmät puhtaan järjestämättömän vertaisverkon ominaisuudet ja joitakin tutkimuksia liittyen tämän vertaisverkkomallin hakualgoritmeihin ja topologian muutoksiin. Painotus puhtaaseen järjestämättömään vertaisverkkoarkkitehtuuriin johtuu siitä, että myös tämä tutkimus keskittyy puhtaisiin järjestämättömiin vertaisverkkoihin.

2.1 Asiakas–palvelin-arkkitehtuuri ja vertaisverkkoarkkitehtuurit

Hajautettu järjestelmä on kokoelma itsenäisiä koneita, jotka näkyvät käyttäjälle yhtenä järjestelmänä. Järjestelmän koneet kommunikoivat ja koordinoivat toimintansa ainoastaan viestien välityksellä [17]. Vertaisverkko voidaan määritellä jakamalla hajautetut järjestelmät kahteen osaan: asiakas–palvelin-arkkitehtuuriin (engl. *client-server architecture*) ja vertaisverkkoihin. Asiakas–palvelin-arkkitehtuuri on hajautettu järjestelmä, joka koostuu yhdestä tehokkaasta palvelimesta sekä asiakkaasta. Palvelin on keskitetty tiedon ja palvelujen tarjoaja, missä on yleensä tietokantapohjainen tiedonhallinta. Asiakkaat ovat palvelua pyytäviä ja käyttäviä osapuolia, jotka eivät voi tarjota palveluja. Järjestelmän kommunikointi perustuu ns. pyyntö-vastemalliin (engl. *request-response-model*), jossa käytetään kaikkien osapuolten ymmärtämää protokollaa [64].

Asiakas–palvelin-arkkitehtuuri soveltuu hyvin tilanteisiin, joissa verkon koko on suhteellisen pieni, sillä järjestelmän skaalautuvuus on heikko. Resurssien etsiminen ei ole ongelma, mutta paljon kysytty resurssi voi kuormitetussa palvelimessa olla heikommin saatavilla. Arkkitehtuurissa tiedon luotettavuus on arvioitavissa ja sen muutokset hallittavissa. Yksi tämän mallin suurista ongelmakohdista on sen alttius hyökkäyksille ja vikaantumisille, sillä palvelimen poistuminen verkosta lamaannuttaa koko järjestelmän.

Vertaisverkko on hajautettu järjestelmä, jossa toisistaan riippumattomat koneet käyttävät jaettuja resursseja ilman keskitettyä palvelinta. Resursseiksi voidaan ymmärtää esimerkiksi tiedostot, prosessoriteho, levytila tai verkon kaistanleveys [64]. Vertaisverkon koneita kutsutaan vertaisiksi (engl. *peer*) ja ne ovat samaan aikaan sekä palvelimia että asiakkaita. Vertaisverkosta on olemassa myös hybridejä malleja,

joissa haku tapahtuu keskitetyn palvelimen avulla, mutta resurssien jako on hajautettua. Näistä kerrotaan tarkemmin luvussa 2.2.

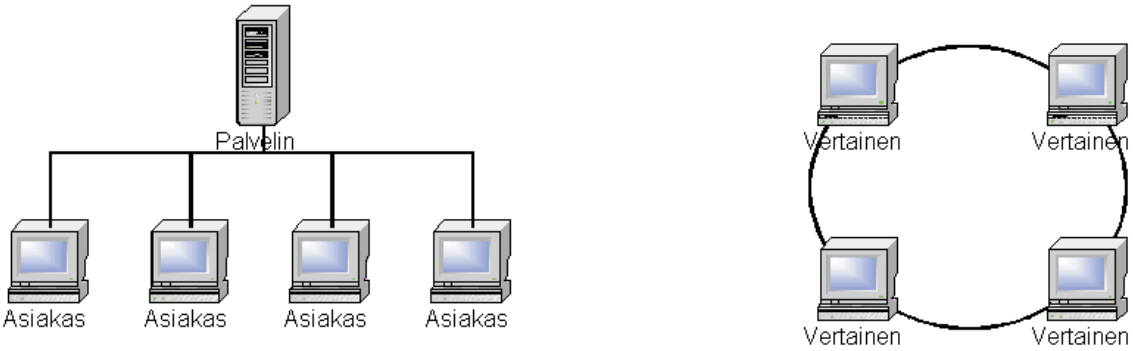
Vertaisverkoilla on onnistuttu välttämään joitakin asiakas–palvelin-järjestelmien vajavaisuuksia suurissa verkoissa [75]. Vertaisverkkojen käyttövarmuus on suurempi, sillä yhden tai muutamankaan solmun vikaantuminen ei vaikuta ratkaisevasti verkon toimintakykyyn. Sama palvelu löytyy useammasta paikasta. Lisäksi vertaisverkot ovat skaalautuvampia, sillä solmut voivat liittyä verkkoon usean eri solmun kautta. Hyvään skaalautuvuuteen vaikuttaa myös se, että vertaisverkko on hajautettu hallinnan, tallennuskapasiteetin ja kaistanleveyksien kannalta. Vertaisverkkojen nykyinen massasuosio perustuu niiden kykyyn jakaa suuria määriä tiedostoja tavalla, joka ei vaadi käyttäjältä muuta kuin halutun tiedoston nimen. Suuren suosion saavuttaneet vertaisverkkojärjestelmät, kuten esimerkiksi Napster [47] ja Gnutella [25], ovat käytännössä osoittaneet vertaisverkkojen käyttökelpoisuuden laajassa mittakaavassa, jos luovutaan vaatimuksesta, että kaikki resurssit ovat aina löydettävissä.

Toisaalta järjestelmässä, johon periaatteessa pääsee palvelimeksi mikä tahansa kone, on myös omat riskinsä. Vertaisten on luotettava toistensa palveluihin, ja esimerkiksi jaettavat tiedostot voivat olla tarkistamattomia. On osoitettu, että vertaisverkkojen tasa-arvoisuus jää usein toteutumatta ja usein vertaiset ottavat enemmän kuin antavat ja tämä heikentää sitä tehokkuutta, johon vertaisverkoilla olisi mahdollista päästä. Vertaisverkko ei myöskään ole homogeeninen: Joillakin verkon koneilla on paremmat liikenneyhteydet, muistikapasiteetti tai prosessointiteho kuin toisilla. Vertaisverkon tehokkuuden kannalta tietoturvaa ja epätasa-arvoisuutta ehkä tärkeämpi ongelma on löytää halutut resurssit tehokkaasti verkosta, jossa solmuilla on useimmiten tieto vain lähimmistä naapureistaan [75]. Asiakas–palvelin-arkkitehtuurin eroja vertaisverkkoarkkitehtuuriin on esitelty kuvassa 2.1.

2.2 Vertaisverkkojen kehityskaari

Vertaisverkkoja käytettiin jo ennen 1980-luvulla tapahtunutta asiakas–palvelin-mallin yleistymistä. Ensimmäinen vertaisverkko ARPAnet kehitettiin 1960-luvulla muutaman tutkimuslaitoksen yhdistämiseksi tietoverkolla toisiinsa. 1980-luvun lopulla verkkoon liitettiin yliopistojen lisäksi yrityksiä ja yhteisöjä. Käyttäjämäärien kasvu johti verkon muuttumiseen epäsymmetriseksi ja hierarkkiseksi asiakas–palvelin-arkkitehtuuriksi, sillä suurimmalle osalle käyttäjistä tiedon ja tiedostojen hakeminen oli ensisijaista eikä tiedon jakaminen ollut tärkeää. [50]

Vertaisverkkojen uusi tuleminen tapahtui 1990-luvun lopussa. Napster oli en-



Kuva 2.1: Asiakas–palvelin-arkkitehtuurissa asiakkaat kommunikoivat palvelimen kanssa, kun taas vertaisverkkoarkkitehtuurissa vertaiset ovat sekä asiakkaita että palvelimia.

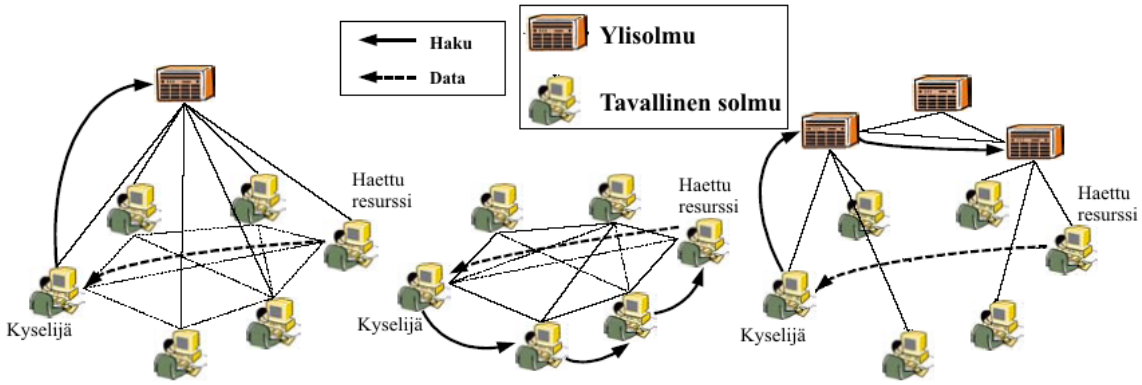
simmäinen Internetissä laajasti käytetty vertaisverkkosovellus. Sitä käytettiin lähinnä MP3-tiedostojen jakamiseen suoraan käyttäjältä toiselle. Napster oli tekniseltä toteutukseltaan ensimmäisen sukupolven vertaisverkkoprotokolla, jolle on ominaista keskitetty palvelin jaettavana olevan tiedon indeksoinnissa. Tiedostojen lataaminen suoritetaan kuitenkin suoraan toiselta vertaiselta. Protokollaa kutsutaan myös hybridiksi vertaisverkkoarkkitehtuuriksi. Tämän protokollan edut ovat haun nopeus ja solmujen alhainen kuormitus verkossa siirrettävään kokonaisdataan nähden. Ensimmäisen sukupolven arkkitehtuuri ei kuitenkaan skaalaudu hyvin käyttäjämäärän kasvaessa ja se on altis hyökkäyksille keskitetystä palvelimesta johtuen. [50]

Toisen sukupolven vertaisverkkoarkkitehtuurissa kaikki on hajautettua eli kaikki verkon solmut osallistuvat tasapuolisesti verkon ylläpidollisiin toimintoihin, kuten esimerkiksi hakujen suorittamiseen. Tunnetuin esimerkki tällaisesta puhtaasta järjestämättömästä vertaisverkosta on Gnutella. Verkon solmut eli serventit (engl. *server+client=servent*) toimivat nimensä mukaisesti sekä palvelimina että asiakkaina. Hybridiin arkkitehtuuriin verrattuna tämän etuna on parempi viansietokyky. Sen sijaan verkon skaalautumista rajoittaa sen kuormittuminen. Kun ensimmäisen sukupolven arkkitehtuurissa tiedoston löytää yhdellä haulla, niin nyt verkon koosta ja Gnutellan BFS-hakualgoritmin (engl. *breadth first search*, katso luku 2.4) TTL-elinaikaparametristä (engl. *time-to-live*) riippuen voi verkon liikenne kasvaa erittäin suureksi. Lisäksi ongelmaksi muodostuvat hakuaikojen kasvu sekä hakujen onnistumisprosentin lasku [60].

Toisen sukupolven vertaisverkoista on olemassa myös ryhmä, jossa resursseja indeksoidaan bittijonoilla ja vertaisten naapurit määritellään näiden indeksien perusteella. Näin voidaan taata resurssihakujen onnistuminen. Topologian ylläpito

parantaa vertaisverkon hakutehokkuutta, mutta kuluttaa enemmän koneiden prosessoritehoja [45]. Esimerkkejä puhtaista järjestetyistä vertaisverkoista ovat Freenet, CAN, Chord, Oceanstore ja PAST [60].

Verkkojen kasvaessa myös vertaisverkkoihin on tullut hierarkkisia piirteitä. Kolmannen sukupolven vertaisverkkoarkkitehtuurin verkoissa on yhdistelty kahden ensimmäisen parhaita piirteitä. Niissä ei ole varsinaisia palvelimia, mutta niissä on kuitenkin hierarkkisesti ylempiä ylisolmuja (engl. *super-peer*), jotka toimivat indeksointivarastoina. Ylisolmut muodostavat keskenään puhtaan vertaisverkon. Tavalliset solmut liittyvät verkkoon ylisolmun kautta ja ylisolmut tietävät niihin kytkeytyneiden solmujen resursseista. Haun saapuessa ylisolmu osaa välittää haun juuri sille solmulle, jossa haettu resurssi sijaitsee. Tämän tyyppinen arkkitehtuuri ei kuormita verkon solmuja samalla tavalla kuin toisen sukupolven arkkitehtuurit ja siinä on mahdollista siirtää myös reaaliaikaista dataa, josta esimerkkinä on Voice over P2P -ohjelma Skype [66]. Muita esimerkkejä super-peer vertaisverkoista ovat Kazaa, DC++ ja JXTA [75]. Vertaisverkkojen eri malleja on esitelty kuvassa 2.2.



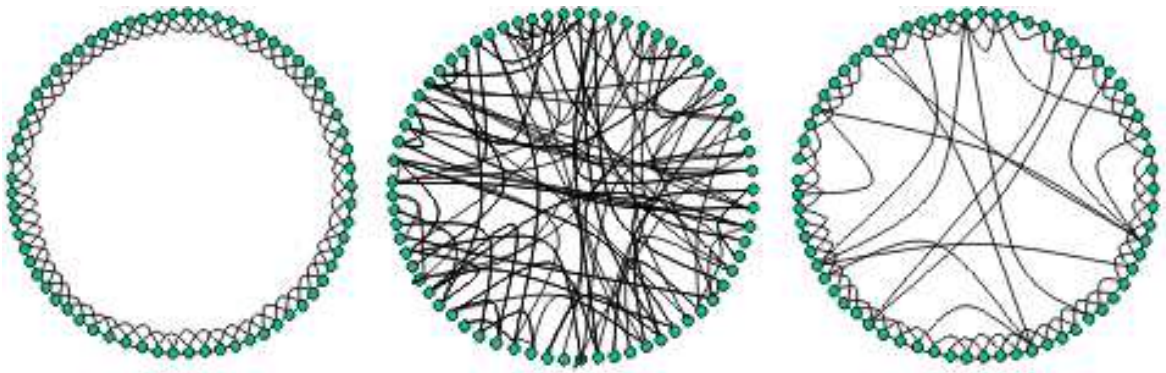
Kuva 2.2: Hybridi, puhdas ja super-peer vertaisverkko sekä niissä tapahtuvien hakujen kohdistuminen.

2.3 Puhtaan järjestämättömän vertaisverkon ominaisuuksia

Puhtaassa järjestämättömässä vertaisverkossa ei ole keskitettyä hakemistoa tai verkon ulkopuolelta tulevaa kontrollia verkon topologiaan. Verkon ylläpito ei vaadi työtä ja se muodostuu solmujen liittyessä siihen jotain säännöstöä noudattaen. Yleensä riittää tietää jonkin vertaisen osoite. Tällöin ei myöskään oteta huomioon verkkoon liittyvän solmun ominaisuuksia, kuten sen jakamia resursseja tai kaistanleveyttä. Tämä aiheuttaa verkossa epäyhtenäisyyttä, sillä samaan kohtaan verkkoa

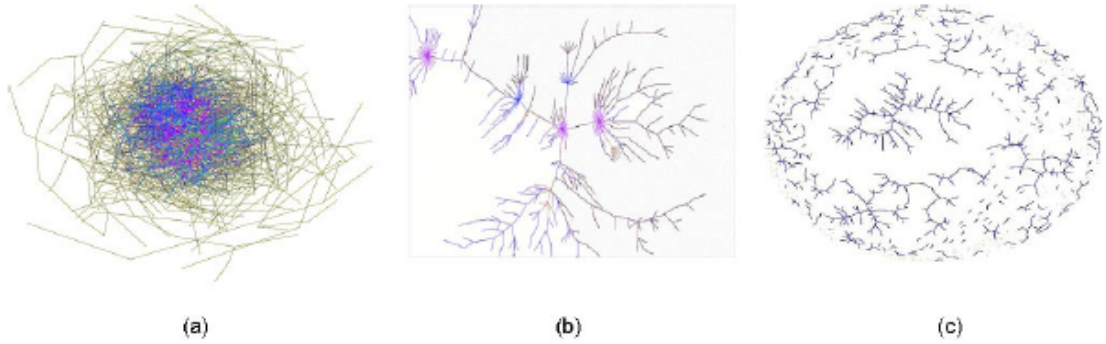
saattaa liittyä yhtä hyvin tehokas solmu kuin mahdollinen pullonkaulasolmukin [50]. Lisäksi verkossa voi olla runsaasti vapaamatkustajia (engl. *free riders*), jotka vain käyttävät palveluja eivätkä osallistu muuhun toimintaan. Esimerkiksi Gnutellassa vapaamatkustajien osuus on tutkimuksen mukaan jopa 27% [43].

Vertaisverkoilla on todettu olevan samoja piirteitä kuin small world -verkoilla. Small world -verkossa mitä tahansa kahta verkon solmua yhdistää lyhyt polku. Lisäksi verkko on voimakkaasti klusteroitunut eli kaikki solmut ovat vahvasti kytkeytyneitä erityisesti lähimpiin naapureihinsa. Kolmas vertaisverkoille ominainen piirre on sen yhteyksien potenssijakautuneisuus. Potenssijakautuneessa verkossa pienellä osalla solmuja on paljon naapureita ja suurella osalla solmuja on vähän naapureita. Samoja ominaisuuksia on havaittu monissa muissakin verkoissa, kuten esimerkiksi WWW-verkossa, Internetissä ja myös sellaisissakin yhteyksissä kuin maapallon lentokenttien reittiverkostossa. Kuvassa 2.3 verrataan potenssijakautunutta small-world -verkkoa säännölliseen ja satunnaiseen verkkoon. Potenssijakautunut verkko ei ole yhtä vikasietoinen kuin esimerkiksi satunnainen verkko, sillä hyvin kohdistetulla hyökkäyksellä muutamia parhaiten kytkeytyneitä solmuja vastaan on mahdollista lamaannuttaa verkkoa huomattavasti. Kuvassa 2.4 on havainnollistettu Saroiun tutkimuksen yhteydessä suoritettua Gnutellan verkon pirstaloitumista, kun 1771 solmun verkosta poistetaan ensimmäisessä testissä 30% solmuista satunnaisesti ja toisessa testissä 63 parhaiten kytkeytynyttä solmua. [62]



Kuva 2.3: Säännöllinen verkko, satunnainen verkko ja potenssijakautunut small-world -verkko.

Keskitetyn palvelimen puuttumisen välitön etu on verkon skaalautuvuuden paraneminen [45]. Skaalautuvuutta rajoittavat tehtävät, kuten palvelujen koordinointi, on jaettu verkossa kaikille. Teoriassa esimerkiksi Gnutella-protokollaa käyttävä verkko voi skaalautua rajattomasti, mutta siinä käytettävä hakualgoritmi aiheuttaa niin paljon liikennettä, että se muodostuu verkon kokoa rajoittavaksi tekijäksi [45].



Kuva 2.4: (a): Gnutella-verkon topologiaa helmikuussa 2001 (1771 solmua), (b): Verkon solmuista 30% on poistettu satunnaisesti, (c): Verkon solmuista 4% parhaiten kytkeytynyttä on poistettu, jolloin verkko on pirstaloitunut pienempiin osiin [62].

2.4 Vertaisverkkojen tehostaminen hakualgoritmeilla

Resurssien hakualgoritmit puhtaassa vertaisverkossa voidaan jakaa kahteen kategoriaan riippuen siitä, millä perustein solmut välittävät kyselyitä eteenpäin. Puhtaassa järjestämättömässä vertaisverkossa solmuilla ei ole mitään tietoa resurssien sijainnista verkossa ja solmujen sijainneistakin se tuntee vain naapurinsa. Puhtaassa järjestetyssä vertaisverkossa solmut valitsevat naapurinsa tiettyjen sääntöjen mukaan ja täten järjestävät verkon tehokkaaksi resurssien haulle. Ongelmana tässä on kuitenkin se, että vain resursseja, jotka voidaan identifioida bittijonolla, voidaan hakea verkosta. Tässä luvussa käsitellään vain puhtaan järjestämättömän vertaisverkon hakuja, sillä vain niitä käytetään tässä tutkimuksessa.

Resurssien haku ilman yleistä tietoa resurssien sijainnista on haastava operaatio ja takuita haun onnistumisesta ei ole, ellei käydä kaikkia verkon solmuja läpi. Nykyiset resurssien hakualgoritmit järjestämättömässä ympäristössä ovat tavalla tai toisella kompromisseja suurten liikennemäärien ja löydettyjen resurssien määrän välillä [74].

Yleinen ratkaisu ongelmaan on leveyshaku (engl. *breadth-first search, BFS*) [41], jossa kyselyä jatketaan kunkin solmun kaikille naapureille, kunnes viesti on kulkenut tietyn pituisen matkan. Algoritmia kutsutaan myös tulvimiseksi (engl. *flooding*). Algoritmissa viestin kulkemaa matkaa rajoitetaan TTL-arvolla (engl. *time to live*), joka pienenee aina yhdellä, kun viestiä lähetetään eteenpäin. Kysely lopetetaan TTL:n pienennettyä nollaan. Jos haluttu resurssi löytyy, lähetetään ilmoitus kyselyn aloittajalle samaa reittiä, kuin kysely tuli, ja jatketaan kyselyjä. Tämä algoritmi on osoittautunut Gnutellan tapauksessa käyttäjän kannalta tehokkaaksi sekä alituiseen muuttuvassa verkossa joustavaksi. Skaalautuva se ei kuitenkaan ole ja menetelmä on lii-

kennemäärän kannalta kovin epäedullinen. Havainnollistetaan esimerkiksi Gnutellalle melko yleinen tilanne, jossa viestiä välitetään eteenpäin, kunnes se on käynyt läpi seitsemän solmua eli TTL-parametrin arvo on 7 ja solmulla on keskimäärin neljä naapuria $C = 4$ (viesti välitetään kolmelle naapurille). Tällöin lähetettyjen viestien määrä on $2 * \sum_{i=0}^{TTL} C * (C-1)^i = 26240$ [1].

Välittämällä kyselyjä vain osalle naapureista voidaan BFS-algoritmin valtavaa viestimäärää pienentää huomattavasti ja samalla kuitenkin löytää tyydyttävä määrä resursseja. Yksi tällainen algoritmi on satunnaiskävelijä (engl. *random walker*) [40], jossa solmu valitsee satunnaisesti naapuriensa joukosta sen, jolle kysely lähetetään. Kukin naapuri jatkaa tätä, kunnes resurssi löytyy tai TTL-arvo pienenee nollaan. Satunnaiskävelijästä on johdettu korkeimman asteen haku (engl. *highest degree search, HDS*) [2], joka välittää kyselyn aina sille naapurille, jolla on eniten naapureita. Adamicin et al. tutkimuksessa [2] on todettu, että hakutehokkuus parantuu huomattavasti satunnaiskävelijäalgoritmia ja erityisesti HDS-algoritmia käytettäessä. Haun skaalautuvuutta kuitenkin rajoittaa se, että suurin osa liikenteestä painottuu verkon parhaille solmuille ja näin ollen potenssijakautuneessa verkossa pieni osa solmuista joutuu kantamaan vastuun lähes kaikista kyselyistä. HDS-algoritmia voidaan käyttää lähinnä vain tutkimustarkoituksiin, sillä oikeassa P2P-verkossa se ei ole käytännöllinen. Algoritmi vaatii kuljetun polun pitämisen viestissä ja suuressa verkossa polku voi kasvaa liian pitkäksi.

Yksi tapa tehostaa edellä mainittuja kävelijäalgoritmeja (satunnaiskävelijä ja HDS) on tuottaa resursseista kopioita (engl. *replication*) kyselyjen yhteydessä. Lv:n et al. tutkimuksessa [40] kävelijäalgoritmeja tutkittiin Grid- ja Gnutella-verkoissa sekä potennssijakautuneessa ja satunnaisessa verkossa. Tutkimuksessa huomattiin merkittäviä parannuksia kaikissa verkoissa, kun haetusta resurssista tuotettiin kopioita suhteessa hakutiheyden neliöjuureen. Tiedoston kopiointi voidaan suorittaa myös solmuun, joka ei ole resurssia hakenut. Tutkimuksen yhteydessä saatiin myös mielenkiintoinen tulos, jossa satunnaisen verkon todettiin olevan kävelijäalgoritmeille tehokkaampi alusta kuin potenssijakautunut verkko tai Grid-verkko. Tässä tutkimuksessa saadaan erilaiset tulokset. Replikointi ei onnistu kaikille resursseille. Esimerkiksi laskentatehon tapauksessa se on mahdotonta. Lisäksi pelkkien viittausten replikointi on usein turhaa, jos resurssi muuttuu usein tai vertaiset vaihtuvat verkossa toistuvasti.

Yangin ja Garcia-Molinan tutkimuksessa [74] käytettiin TTL-arvoa niin, että hauille suoritettiin useampia kierroksia, joille kullekin oli ennalta määrätty TTL-arvo. Ensimmäisellä kierroksella kyselyjä suoritettiin pienemmällä TTL-arvolla ja näin ollen etsittiin resursseja lähinaapureilta. Jos ensimmäinen kierros ei antanut haluttua tu-

losta, niin suoritettiin uusi kierros kyselyjä suuremmalla TTL-arvolla ja resursseja käytiin etsimässä kauempaa. TTL-arvon säätämisen lisäksi hakuja pyrittiin tehostamaan antamalla solmuille tehtäväksi muistaa aiempien hakujen tuloksia, joihin kuuluivat muun muassa vastausten määrä ja lähetettyjen kyselyiden määrä. Näiden tietojen perusteella solmuja jaettiin eri alaluokkiin ja kyselyt ohjattiin näiden tietojen perusteella. Yangin ja Garcia-Molinan menetelmä pienensi liikennemääriä verrattuna BFS-algoritmiin, mutta vastaavasti kyselyiden suoritusaika oli suurempi varsinkin isommissa vertaisverkoissa.

On kehitetty myös joukko algoritmeja, joissa solmut pitävät yllä tietoa verkon resursseista varsinaisten resurssihakujen lomassa. Rohrs [55] ehdottaa informoitua hakua tulvimisen sijaan QRP-protokollan (engl. *Query Routing Protocol*) avulla. Verkon solmut pitävät yllä omaa reititystaulukkoa (engl. *routing table*), jotka sisältävät binääriset avainsanat tiedostoista eli avaimen hajautusarvot (engl. *hashed keywords*), jotka yleensä tiivistetään Bloomin suodattimella [9] sopivaan muotoon. Näitä reititystaulukoita vaihdellaan muiden solmujen kanssa säännöllisesti ja niitä yhdistellään oman reititystaulukon kanssa eteenpäin lähetettäväksi. Hakujen tehostamiseksi reititystaulukoita tulisi vaihdella mahdollisimman kaukana olevien solmujen kanssa, mutta tämä aiheuttaa paljon liikennettä ja solmujen kaistanleveydet ovat rajoittavana tekijänä. Algoritmin ongelmana on myös Bloomin suodattimen tekemät virheet sekä vertaisverkon dynaaminen luonne, jolloin reititystaulukoiden tiedot vanhenevat nopeasti. Lisäksi jos verkossa on paljon eri nimisiä resursseja, tulee osumia reititystauluun vähän.

Crespon ja Garcia-Molinan RI-indeksointi (engl. *Routing Indices*) [18] käyttää samaa ideaa kuin QRP, mutta siinä ei käytetä Bloomin suodatinta tiivistämään naapurien tietoja, vaan tiedostot jaetaan eri kategorioihin. Jokainen solmu ylläpitää reititystaulukkoa siitä, kuinka monta tietyn kategorian resurssia voidaan löytää kunkin naapurin kautta. Solmun luodessa uuden yhteyden, se luo uuden reititystaulukon omista ja naapuriensa resursseista ja lähettää sen uudelle naapurilleen. Resurssihakujen ohjaus perustuu naapurien hyvyysarvoon. Naapurin hyvyysarvo määräytyy sen mukaan, kuinka monta saman kategorian resurssia kuin kysytty resurssi, löytyy sen kautta. Solmun tulee jatkuvasti tiedottaa muuttuneesta reititystaulukostaan naapureilleen tiedostojen tai naapurien vaihtuessa. Se muodostaakin tämän toteutuksen suurimman pullonkaulan, sillä vertaisverkossa muutokset ovat jatkuvia.

Tsoumakosin ja Roussopouloksen APS-hakualgoritmissa (engl. *Adaptive Probabilistic Search*) [70] [71] solmut pitävät QRP:n ja RI:n tapaan yllä tietoja naapureistaan, mutta niitä päivitetään vain resurssihakujen yhteydessä. Solmun tehdessä kyselyjä se tiedottaa siitä myös naapureilleen. Naapuri tallentaa kysytyn resurssin tiedot ja

myös sen, onnistuiko haku vai ei. Tämän tiedon perusteella solmu voi käyttää hyväkseen todennäköisyyksiä valitessaan naapuria oman kyselynsä yhteydessä. Tämän algoritmin etuna on, että sen rekisterien ylläpitoon tarvittava liikennemäärä on pieni ja samalla se sopeutuu paremmin dynaamiseen vertaisverkkoympäristöön. Algoritmi löytää suositut resurssit nopeasti, mutta muita resursseja sen on hankala löytää. Algoritmin seurauksena liikenne painottuu monesti siihen solmuun, josta haettu resurssi on löydetty ensimmäisenä ja mahdollisesti nopeamman yhteyden päässä oleva solmu jää huomioimatta.

2.5 Vertaisverkkojen tehostaminen topologiaa muuttamalla

Vertaisverkkojen hakualgoritmien tehostamiseksi on myös paljon tutkittu niiden toiminta-alustan eli verkon topologian vaikutusta hakujen tehokkuuteen. Tässä luvussa esitellään joitain tärkeimpiä periaatteita ja niiden pohjalle kehiteltyjä tekniikoita. Myöskään tässä luvussa ei käsitellä muita kuin puhtaita järjestämättömiä vertaisverkkoja.

Yksi periaate järjestellä verkon topologiaa on klusteroida solmut sen mukaan, mistä ne ovat aikaisempien resurssihakujen perusteella olleet kiinnostuneita (engl. *interest-based locality*). Ramanathan, Kalogeraki ja Pryne etsivät tutkimuksessaan [53] hyviä naapureita muodostamalla yhteyksiä niihin solmuihin, jotka toistuvasti antavat hyviä tuloksia ja luopumalla niistä yhteyksistä, jotka eivät anna hyviä tuloksia. Tämän seurauksena solmuilla on yleensä vähän yhteyksiä, mutta ne ovat klusteroituneena samanaiheisiin solmuihin. Ongelmia seuraa, jos solmu hakee paljon resursseja eri aihepiireistä tai vaihtaa kiinnostuksen kohteita, sillä klusterien välillä yhteyksiä on vähän. Sripanidkulchai, Maggs ja Zhang [67] muodostavat Gnutella-verkossa oikopolkuja solmuihin, joissa aiempien hakujen perusteella on samanaiheisia resursseja. Oikopolkujen verkosto muodostaa oman loogisen topologian Gnutellan päälle eikä näin ollen vaikuta Gnutellan skaalautuvuuteen. Resurssihauissa käytetään ensin oikopolkuverkkoa, mutta jos sieltä ei haluttua resurssia löydy, suoritetaan BFS-haku Gnutella-verkossa normaaliin tapaan. Condie, Kamvar ja Garcia-Molina [15] kehittivät protokollan, jossa Gnutellan kaltaisen vertaisverkon solmu muodostaa yhteyksiä suoraan solmuihin, joista se tulevaisuudessa todennäköisesti löytää haluamansa resurssit. Tämä perustuu solmujen pisteyttämiseen aiempien hakujen pohjalta. Lisäksi protokollassa pisteytettiin yhteyksien luotettavuutta, millä pyrittiin pienentämään vapaamatkustajien ja pahansuopien solmujen vaikutusta. Simulaatiossa vertaisverkko sisälsi hyviä solmuja, jotka jakoivat ja etsivät resursseja normaalisti, sekä huonoja solmuja, jotka jakoivat korruptoituneita tiedostoja.

Normaalien kyselytapahtumien lisäksi simulaatioissa huonot solmut pyrkivät rikkomaan vertaisverkon rakenteen. Huonojen solmujen sekä vapaamatkustajien todettiin siirtyvän verkon laidalle. Vaikkakaan tutkimuksen tarkoituksena ei ollut luoda klustereita, joissa samasta aiheesta kiinnostuneet solmut ovat samassa, näin kuitenkin tapahtui.

Toinen samantyyppinen periaate kuin intressiperustainen klusterointi, on määrätä ennalta sisältöryhmiä, joihin solmut liittyvät (engl. *content-based locality*). Crespo ja Garcia-Molina [19] ehdottavat globaalia luokitteluhierarkiaa, jolla puhtaan järjestämättömän vertaisverkon solmut voitaisiin klusteroida yhteen tai useampaan semanttiseen päällysverkkoon SON:iin (engl. *Semantic Overlay Network*). Solmu arvioi verkkoon liittyessään siinä olevat SON-verkot ja omien resurssiensa sisältötyypit. Tämän jälkeen se valitsee SON-verkot, joihin se liittyy. Kaikki tähän operaatioon liittyvä liikenne tehdään tulvimalla. Tutkimuksessa jätetään mainitsematta, kuinka päällysverkkojen indeksiä pidetään yllä täysin järjestämättömässä vertaisverkossa, mutta siinä on käsitelty klusteroinnissa tärkeää luokittelemisen ongelmaa, joka on jäänyt monessa klusterointia ehdottavassa tutkimuksessa vähemmälle. Khambatti, Ryu ja Dasgupta huomasivat tutkimuksessaan [35], että solmut löytävät useita samaan sisältöryhmään kuuluvia solmuja kyselemällä niitä vain naapureilta ja naapurien naapureilta.

Kolmas periaate topologian hallintaan on ottaa huomioon vain solmujen liikennemäärät tai tekniset ominaisuudet. Näissä tekniikoissa siis jätetään huomioimatta solmujen intressit ja niiden jakamien resurssien määrä ja laatu. Cooperin ja Garcia-Molinan tutkimuksessa [16] ylikuormittunut solmu katkaisee jonkin yhteyden. Katkaistava yhteys on joko eniten kuormittava linkki tai eniten kuormittavat linkit, jotka ylittävät jonkin määrätyn liikennerajoitteen. Voidaan myös valita, katkaistaanko yhteyksiä tyypin mukaan. Tutkimuksen verkkoalustaan on lisätty hakuja helpottava paikallinen indeksointijärjestelmä, joka vaatii ylläpitoonsa liikennettä normaalien resurssihakujen lisäksi. Ylikuormittuneiden solmujen auttamisen sijaan voidaan myös keskittyä pelkästään tehokkaiden koneiden suosimiseen. Chawathe et al. halusivat tutkimuksessaan [12] sijoittaa solmut siten, että suorituskykyisille solmuille annetaan heikompia solmuja naapuriksi siten, että vahvemmille solmuille tulisi suhteessa enemmän naapureita. Kontrollointi perustuu tyytyväisyystasoon, joka puolestaan perustuu suorituskykyyn ja astelukuun. Tavoiteltava topologia muistuttaa paljon super-peer-topologiaa ja tutkimuksen tavoitteena olikin luoda järjestämätön verkko, joka on dynaamisempi ja joustavampi kuin super-peer-verkko. Topologian hallinta oli vain osa tutkimuksen toteutusta eikä suoraa vertailua esimerkiksi super-peer-topologian suorituskykyyn tehty.

Yllä mainituissa tutkimuksissa on kautta linjan valittu jokin vertaisverkon ominaisuus tai ongelma, johon on pyritty kehittämään jokin tekniikka ratkaisuksi. Ongelma vain on siinä, että sisältöklusteroinnin seurauksena solmut, joilla on suosittuja resursseja, joutuvat kovan liikennerasituksen alaiseksi. Vastaavasti pelkästään liikennerasituksia mittaamalla saatetaan syrjiä sisällöltään rikkaita solmuja ja huonontaa hakuaikoja. Sakaryan ja Unger ovat tutkimuksessaan [59] yhdistäneet osia aiemmin mainituista periaatteista ja mittaavat simuloimalla topologian kehitystä, kun siihen vaikutetaan sekä liikennerasituksen että resurssien sisältöklusteroinnin keinoin. Verkon solmuilla on kerättyä tietoa muistakin solmuista kuin naapureistaan. Verkossa kulkeviin viesteihin tarttuu tietoa esimerkiksi niiden solmujen sisältötyypeistä, osoitteista ja kestoista, joissa viesti on äskettäin kulkenut. Liikennemääriin reagoiva algoritmi herää solmussa, kun saapuvien viestien jono kasvaa yli kriittisen arvon. Tällöin lähetetään varoitusviestejä myös niille solmuille, joiden osoitteita on mainittu jonossa olevissa viesteissä. Näitä tietoja hyväksi käyttäen solmut luovat kiertoteitä viesteille ruuhkan välttämiseksi. Sisällön klusterointi perustuu myös viestien mukana kulkevaan lisätietoon, joiden perusteella solmut pitävät tietovarastoa verkon rakenteesta. Resurssien sisältöön reagoiva algoritmi herää, jos kyselyjen hakuajat tai viestien kulkema matka kasvavat yli sallitun rajan. Tutkimuksen mukaan paikallisesti toimivat algoritmit voivat tehokkaasti vaikuttaa myös Internetin laajuudella toimivissa vertaisverkoissa ja saavuttaa globaalisti hyödyllisiä tuloksia.

Iles ja Deugo [32] ovat kehittäneet metaprotokollan puhtaisiin vertaisverkkoihin, joissa käytetään BFS-hakualgoritmia. Tätä metaprotokollaa ohjataan geneettisen ohjelmoinnin avulla ja sen ilmentyminä saadaan vertaisverkkoprotokollia, joiden säännöt määräävät vertaisverkon rakenteen. Heidän tutkimuksensa tavoitteena oli luoda metaprotokolla, jolla saadaan optimi yleislähettävä vertaisverkko kulloinkin määritettyyn ympäristöön. Metaprotokollan evolutionääriseen ohjaukseen käytettiin kahta tekijää: CONN (engl. *connections*), joka ilmaisee tavoitteen kunkin vertaisverkon solmun naapurien määräksi sekä RANK (engl. *ranking*), joka on vertailuluku kullekin mahdolliselle naapurille. CONN-tekijän arvoihin vaikuttavat esimerkiksi nykyisten naapurien määrä sekä erilaiset statistiikat solmun liikennemääristä. RANK-tekijään vaikuttavat enimmäkseen tekijät, jotka liittyvät aikaisempiin hakutapahtumiin ja niiden onnistumisiin kunkin naapuriehdokkaan osalta. Geneettisen kehityksen onnistumista mitattiin tutkimuksessa hyvyysfunktiolla $fitness = searches_{successful} + 0.5 \times searches_{timed-out}$, missä $searches_{successful}$ on onnistuneiden hakujen lukumäärä ja $searches_{timed-out}$ on niiden onnistuneiden hakujen lukumäärä, joiden kuluttama aika on ylittänyt annetun raja-arvon. Tutkimuksessa havaittiin, että Gnutellan käyttämä protokolla oli useissa tapauksissa optimaalinen. Ver-

taisverkot, joissa solmujen kaistanleveydet olivat pieniä, klusteroituivat pienempiin saarekkeisiin pysyen kuitenkin yhtenäisenä. Lisäksi havaittiin, että Gnutella-protokollassa optimi naapurimäärä solmuille oli kuusi yleisesti käytetyn seitsemän sijaan. Tutkimuksessa havaittiin geneettisen ohjelmoinnin olevan soveltuva keino sekä räätälöimään vertaisverkkoprotokollia että luomaan mukautuvia ja ympäristöstään tietoisia solmuja. Tutkimuksen ongelmana on pieni, 30 vertaisen verkko.

Useimpien tutkimuksien perustuessa pelkästään loogisen päällysverkon tehostamiseen ovat Liu et al. kehittäneet LTM-algoritmin (engl. *Location Aware Topology Matching Techique*) [39]. Järjestämättömissä vertaisverkoissa solmut valitsevat satunnaisesti loogisia naapureita tietämättä mitään niiden sijainnista fyysisesti. Niin sanottu topologinen yhteensopimattomuusongelma (engl. *topology mismatch problem*) yhdessä tulvimishakualgoritmin kanssa aiheuttaa tilanteita, joissa sama viesti voi kulkea edestakaisin samaa fyysistä linkkiä pitkin ja rasittaa Internetiä. LTM-algoritmi perustuu jokaisessa solmussa sijaitsevaan havaitsijaan, joka tallentaa vastausaikojen viiveitä loogisesti lähellä olevista solmuista. Havaitsijaa kutsutaan TTL2-havaitsijaksi eli se näkee solmut kahden hypyn säteellä. Lisäksi havaitsijan avulla voidaan havaita, jos sama viesti kulkee jollakin välillä edestakaisin. Yhteydet, joilla on suuri viive tai paljon viestin pomputtelua edestakaisin, katkaistaan tai lisätään ehkäpoistettavien listalle ja yhteyksiä, joilla on pieni viive, pyritään tuomaan loogiseksi naapuriksi. Kirjoittajat ilmoittivat LTM-algoritmia käytettäessä yhdessä indeksivälimuistin (engl. *index cache*) kanssa jopa 75% parannuksista liikennekustannuksissa ja 65% parannuksista kyselyiden vastausnopeuksissa. Indeksivälimuistia käytettäessä tallenetaan löydetyn resurssin vastaus niiden solmujen muistiin, jotka ovat resurssivastauksen paluureitin varrella.

Tässä tutkimuksessa ei määrätä ennalta mitään ominaisuuksien arvoja, joiden mukaan solmujen tulisi olla yhteydessä toisiinsa, vaan tutkitaan, onnistuuko evoluutioon perustuva neuroverkko muodostamaan solmuille tehokkaan alustan, kun solmut suorittavat resurssikyselyitään. Ne vertaisverkon ominaisuudet, jotka halutaan mukaan topologian muodostukseen, määrätään neuroverkkojen sisääntuloissa eli esimerkiksi voidaan valita, halutaanko simulointiin ottaa mukaan solmujen kaistanleveydet tai aiempien resurssihakujen historiaa. Ominaisuuksien painoarvoja voidaan haluttaessa säätää neuroverkon sisääntuloarvoissa ja hyvyysfunktiossa.

3 Neuroverkot

Neuroverkkotutkimuksen tavoitteena on luoda yksinkertaistettuja matemaattisia malleja jäljittelemään aivojen oppimisprosessia ja tutkia, kuinka niillä ratkaistaan erilaisia laskennallisia ongelmia. Osa neuroverkoista perustuu biologiseen esikuvaansa ja osa ei, mutta yhteistä niille on tavoite menetelmästä, joka oppii tuottamaan halutunlaisia tuloksia. Neuroverkko soveltuu hyvin tilanteisiin, joissa ongelmaan vaikuttavien tekijöiden suhde on monimutkainen eikä ongelmaa voida ratkaista perinteisin matemaattisin keinoin. Hyviä esimerkkejä neuroverkkojen sovellusalueista ovat hahmontunnistus ja talouden kehityksen ennustaminen. Neuroverkkojen ongelmana voidaan pitää niiden opettamiseen tarvittavaa aikaa. Usein neuroverkko ei ole paras keino ongelmanratkaisuun, mutta joskus se voi olla ainoa. Neuroverkkojen ominaispiirteisiin kuuluvat yksinkertaiset laskentayksiköt eli neuronit (engl. *artificial neuron, AN*), neuroneista muodostettu verkkomainen rakenne, jossa on synaptiset kytkennät, ja rinnakkainen laskenta. [57]

Tässä luvussa esitellään neuroverkon perusrakennusyksikkö eli neuroni sekä yleisimmin käytetty neuroverkon malli eli MLP-monikerrosperceptronverkko (*engl. multi layer perceptron*). MLP-verkosta selvitetään ensin kerrosten määrän vaikutus verkon kykyyn kuvata eri funktioita ja toiseksi neuronien määrän vaikutusta verkon kykyyn kuvata funktio halutulla tarkkuudella. Luvun toinen osa muodostuu neuroverkkojen ohjatusta opetuksesta sekä mukautuvasta opetuksesta.

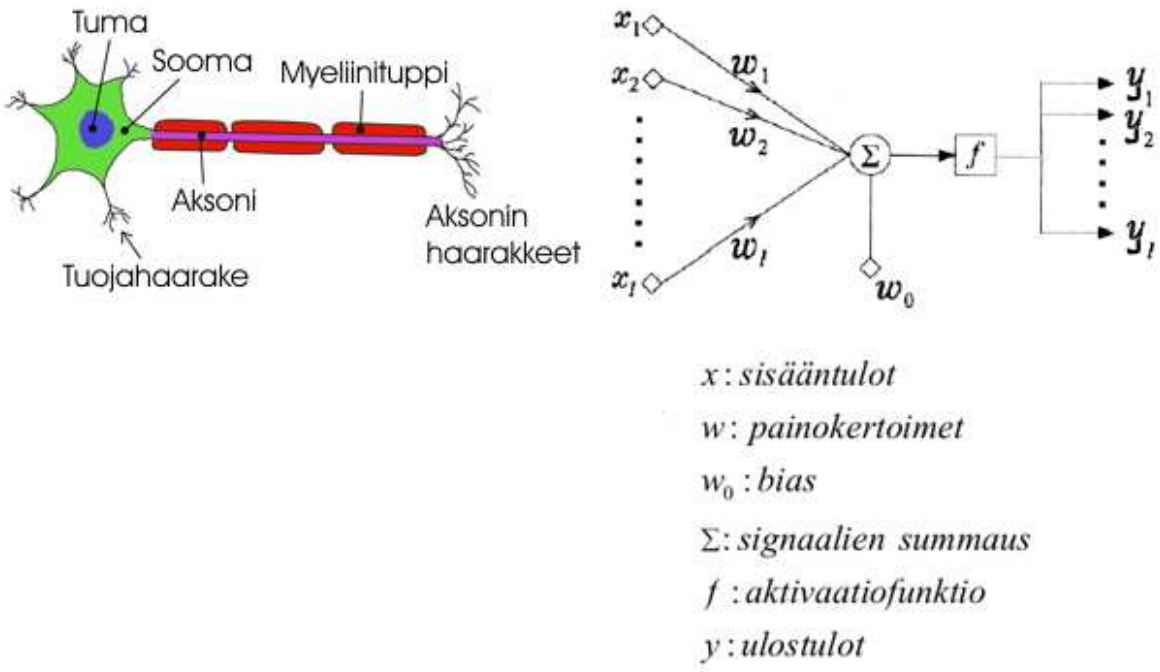
3.1 Neuroni

Neuroverkkojen toteutuksien yksityiskohdissa on eroja, mutta yleisessä mallissa neuroverkon perusyksikkönä on neuroni. Kuvassa 3.1 on esitetty neuroverkon neuronin yhtäläisyydet sen biologiseen esikuvaansa.

Neuronin peruslaskutoimitus on kaavan 3.1 mukaisesti summata sen painotetut sisääntuloarvot ja muuntaa summa aktivaatiofunktion avulla neuronin ulostuloksi sopivaan muotoon:

$$y = f(\sum_{i=1}^{l} w_i x_i + w_0) = f(\mathrm{w}^{\mathrm{T}}\mathrm{x} + w_0), \tag{3.1}$$

missä w_i on saapuvien yhteyksien painokertoimet neuronista i, w_0 on bias, l on



Kuva 3.1: Neuronin ja sitä jäljittelevän keinotekoisen neuronin (engl. *artificial neuron, AN*) yhtäläisyydet. Neuronilla on tietty perussähkövaraus (*AN: bias*). Neuronin keskuksen eli sooman vaihtaessa ioneja ympäristönsä kanssa sen sähkövaraus kasvaa ennen pitkää riittävän suureksi ja se lähettää hermoimpulssin kohti viejähaarakkeen eli aksonin kärkeä (*AN: ulostulo*). Neuronin tuojahaarakkeiden eli dendriittien (*AN: sisääntulo*) impulssit summataan, millä on sähkövarausta kasvattava tai hidastava vaikutus. Neuroneissa on yleisesti useita tuojahaarakkeita ja yksi aksoni, joka jakautuu loppupäästä haarakkeisiin vieden saman signaalin useille muille neuroneille. Signaalien lähetystaajuus vaihtelee neuronissa lepotilan noin kymmenen kertaa sekunnista tuhanteen kertaa sekunnissa (*AN: ulostulon arvo*). Aksonin ympärillä on myeliinituppeja, jotka vahvistavat aksonissa kulkevaa signaalia (*AN: yhteyden painokerroin*). Neuroneilla on myös sisäisiä ominaisuuksia, jotka vaikuttavat ulostulon arvoon (*AN: aktivaatiofunktio*). [22]

neuronien lukumäärä, $f(\cdot)$ on aktivaatiofunktio ja y on neuronin ulostulo.

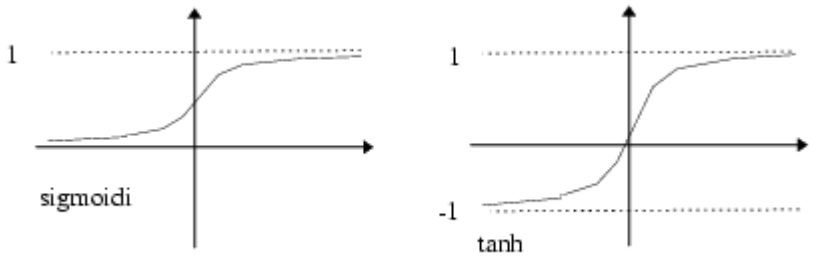
Ensimmäinen esitetty aktivaatiofunktio oli aikanaan kynnys- eli thresholdfunktio (kaava 3.2), joka antaa sisääntulojen summan merkistä riippuen neuronin ulostulon arvoksi joko 0 tai 1.

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0. \end{cases} \tag{3.2}$$

Nykyisin yleisimmät aktivaatiofunktiot ovat kuvassa 3.2 esitetyt sigmoidi (kaava 3.3) ja hyperbolinen tangentti ($tanh$, kaava 3.4) tai näiden S-muotoa mukailevat funktiot.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.4}$$



Kuva 3.2: Sigmoidi muistuttaa hyperbolista tangenttia, mutta se saa arvoja avoimelta väliltä]0,1[, kun hyperbolinen tangentti saa arvoja avoimelta väliltä]-1,1[.

Aktivaatiofunktiot ovat yleensä monotonisesti kasvavia, jatkuvasti derivoituvia funktioita, jolloin ne soveltuvat yleisesti käytettyyn ohjattuun opetustapaan eli vastavirta-algoritmiin, joka on gradienttioptimointia ja joka esitellään tarkemmin luvussa 3.3.1. Toinen yleinen piirre on, että ne skaalaavat ulostulon arvon sopivalle välille, jolloin mikään yksittäinen neuroni ei pääse ylivoimaiseen asemaan verkossa.

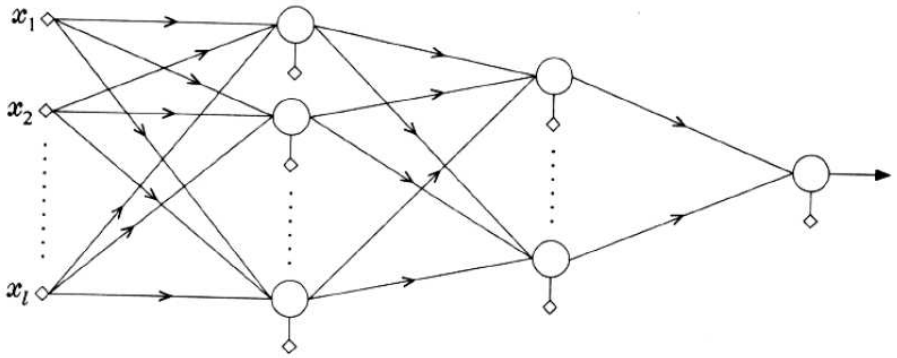
Aktivaatiofunktioiden pääsiallinen tehtävä on kuitenkin tuoda neuroverkkoon epälineaarisuutta. Peräkkäisten lineaaristen aktivaatiofunktioiden muunnosten kokonaisuus on itsessään lineaarikuvaus. Ilman kykyä kuvata epälineaarisuutta neuroverkko ei olisi sen taidokkaampi kuin yksittäinen neuronikaan ja sen soveltaminen riittäisi pelkästään lineaariseen luokitteluun.

Aktivaatiofunktio valitaan tapauskohtaisesti. Kynnysfunktiolla neuroverkon kuvauksen analysointi on tehokasta, mutta jatkuvilla funktioilla neuroverkon opettaminen on helpompaa kuin muilla funktioilla. Jatkuvista funktioista hyperbolisen

tangentin on todettu joskus olevan opetuksessa nopeampi kuin pelkästään positiivisia arvoja saavan sigmoidin [48]. Yleistä sääntöä aktivaatiofunktion valinnalle ei kuitenkaan ole olemassa.

3.2 Monikerrosperceptronverkot

Neuroni ei yksinään pysty kovinkaan monimutkaisiin laskutoimituksiin, mutta yhdistämällä niitä oikealla tavalla voidaan kuvata mitä tahansa jatkuvaa funktiota. Tällaista neuroverkkoa kutsutaan universaaliksi approksimaattoriksi. Neuroverkkojen toiminta perustuu erilaisiin neuroverkon topologioihin ja yhteyksien painokertoimiin. Neuroverkkojen rakenteille on useita eri malleja, mutta yleisin niistä on monikerrosperceptronverkko eli MLP-verkko. Kuvassa 3.3 on kolmikerrosperceptroni, jollaista käytetään myös tässä tutkimuksessa. Monikerrosperceptronverkolla



Kuva 3.3: Kolmikerrosperceptroni. Aktiivisia kerroksia on kolme: kaksi piilokerrosta ja ulostulokerros, jolla on yksi ulostuloneuroni. Sisääntulokerroksen solmuja kuvataan symbolilla $\diamond$ ja aktiivisten kerrosten solmuja symbolilla $\bigcirc$. Aktiivisten kerrosten neuroneihin alapuolelta liitetyt $\diamond$:t ovat $bias$-arvoja.

tarkoitetaan useampikerroksista neuroverkkoa, jonka neuroneja kutsutaan myös perceptroneiksi (engl. *perceptron*). Alunperin MLP-verkon neuronit antoivat ulostulona vain nollia tai ykkösiä, mutta nykyisin ne antavat jatkuvia arvoja [54]. MLP-verkossa neuronit kytketään niin, että siinä ei synny silmukoita (engl. *feedforward network*). Neuronit järjestetään useiksi kerroksiksi, joiden sisällä ei ole kytkentöjä ja jokainen kerros on kytketty täydellisesti edeltäjäänsä ja seuraajaansa. MLP-verkon ensimmäinen kerros on sisääntulokerros, jossa jokaiselle piirteelle eli ongelman parametrille on oma neuroninsa. Sisääntulokerroksen neuronit eivät suorita laskentaa. Viimeinen kerros on ulostulokerros, jonka neuronien aktivaatiofunktiot ovat usein

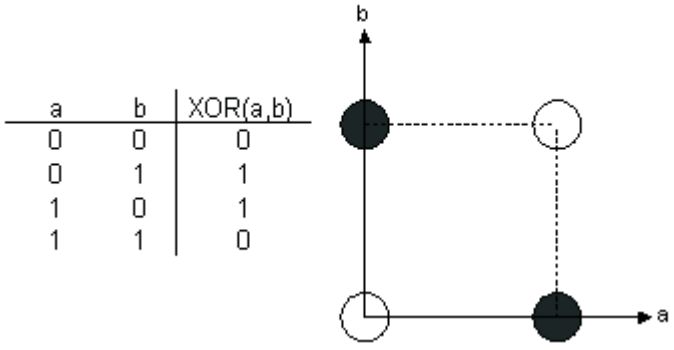
lineaarisia. Näiden kahden mainitun kerroksen välissä olevat kerrokset ovat piilokerroksia. Kerroksen l neuronin j sisääntuloarvojen summa $s_j^{(l)}$ on nyt kaavan 3.5 mukainen:

$$s_j^{(l)} = \sum_{i=1}^{N_l-1} w_{ij}^l x_i^{(l-1)} + w_{(N_{l-1}+1)j}^l, \tag{3.5}$$

missä w_{ij}^l on kerroksella $l-1$ olevan neuronin i ja kerroksella l olevan neuronin j välisen yhteyden painokerroin. Termi $w_{(N_{l-1}+1)j}^l$ on bias-arvo. Neuroverkossa on L kerrosta eli $L-1$ piilokerrosta. Sisääntulokerroksella on N_0 neuronia, piilokerroksella l on N_l neuronia ja ulostulokerroksella L on N_L neuronia. Neuronin ulostulon arvon laskemiseksi tulee summa vielä skaalata kyseisen aktiivisen kerroksen aktivaatiofunktiolla.

3.2.1 Kerrosten ja neuronien lukumäärän vaikutus

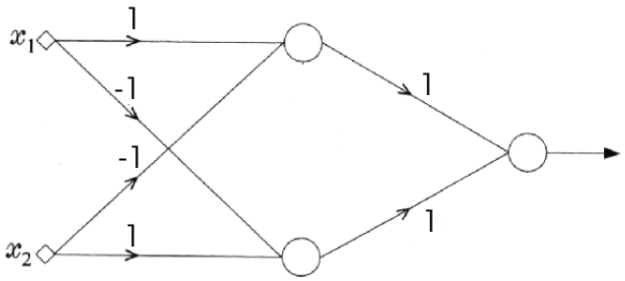
Yksikerrosneuroverkko, jossa ei ole piilokerroksia eikä siis ole MLP-verkko, rajoittuu kuvaamaan vain lineaarisesti separoituvia tehtäviä. Lineaarisesti separoituvassa tehtävässä joukot A ja B ovat eroteltavissa lineaarisella päätöspinnalla siten, että luokkien pisteet erottuvat toisistaan täydellisesti. Kuuluisin esimerkki ei-lineaarisesti separoituvasta ongelmasta on kuvassa 3.4 esitetty XOR-ongelma (engl. *eXclusive OR*). Rajoittuminen pelkästään lineaarisesti separoituviin ongelmiin on siinä mielessä merkittävä, että monet luokitteluongelmat eivät ole lineaarisesti separoituvia. Neuroverkkojen kehitys lähes lamaantui 1960-luvun loppupuolella yli kymmeneksi vuodeksi, kun todettiin, että nämä aivojen toimintaa mallintavat menetelmät eivät pysty ratkaisemaan edes niin yksinkertaista ongelmaa kuin XOR-ongelma [57].



Kuva 3.4: XOR-ongelma. Pisteiden ○ ja ● väliin on mahdotonta sijoittaa suoraa viivaa niin, että ne erottuisivat täydellisesti.

Neuroverkkojen tutkimus elpyi 1980-luvulla, kun koneet tehoistuivat ja MLP-verkkojen opetusmenetelmät tulivat yleiseen tietoon luvussa 3.3.1 esiteltävän vas-

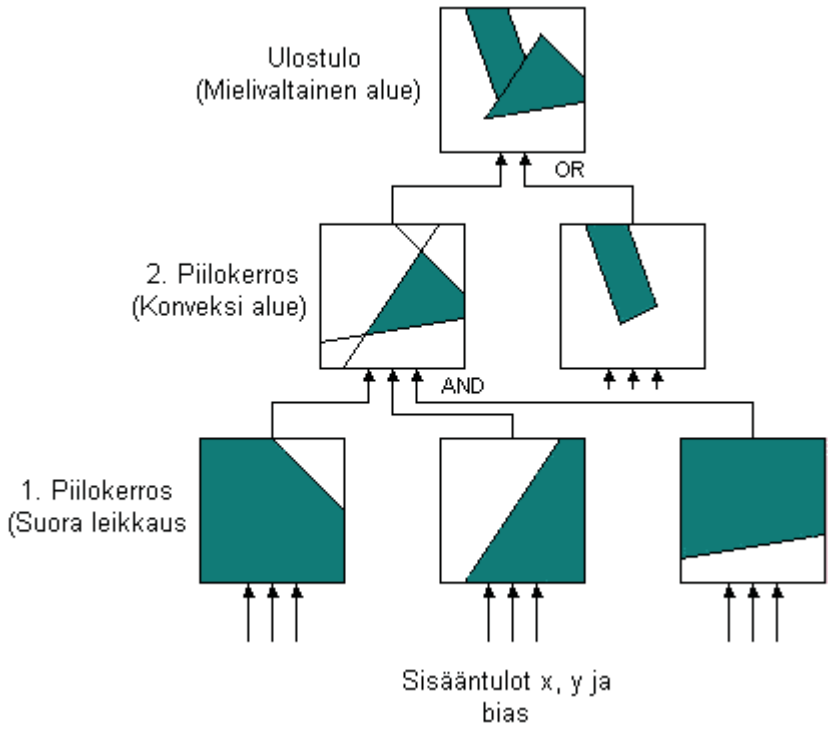
tavirta-algoritmin julkaisun myötä. Lisäämällä yksikerrosverkkoon piilokerros, saadaan XOR-ongelmakin ratkaistuksi kuvan 3.5 mukaisesti. Neuroverkon kerrosten sekä niissä olevien neuronien määrä valitaan tehtäväkohtaisesti. Hyvänä periaatteena voidaan pitää Occamin partaveitseä eli tehtävä kannattaa ratkaista mahdollisimman yksinkertaista, mutta riittävää, mallia käyttäen. Valinta ei ole yksinkertainen, mutta on olemassa joitain periaatteita, mitä voi käyttää hyväksi. [57]



Kuva 3.5: XOR-ongelman ratkaiseva 2-kerrosneuroverkko.

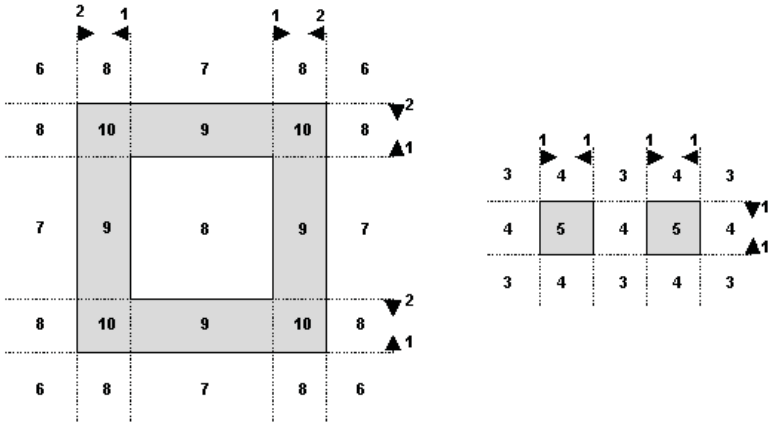
Seuraavana esiteltävän esimerkin ideat esiintyvät myös todistuksissa, joissa monikerrosperceptronien osoitetaan olevan universaaleja approksimaattoreita. Toisin sanoen ne pystyvät kuvaamaan mielivaltaisen tarkasti jatkuvat kuvaukset hyperkuutiolta $[-1,1]^n$ välille (-1,1) ja näin ollen myös minkä tahansa funktion. Täytyy muistaa, että kyky kuvata mikä tahansa funktio ei ole mikään erikoinen ominaisuus. Siihen pystyvät esimerkiksi polynomit tai Fourierin sarja. Mutta jos neuroverkoilla ei olisi universaalia approksimointikykyä, se olisi niiden kehitykselle huono uutinen. [54]

Kahden piilokerroksen verkko eli kolmikerrosverkko on riittävä kuvaamaan minkä tahansa funktion. Kuvan 3.6 esimerkissä ensimmäinen piilokerros jakaa sisääntuloavaruuden kahteen osaan lineaarisella kynnysfunktiolla, toinen piilokerros muodostaa AND-operaation avulla näistä puoliavaruuksista konvekseja päätöspintoja ja ulostulokerros muodostaa OR-operaatiolla näistä pinnoista mielivaltaisia, mahdollisesti erillään olevia päätöspintoja. Päätöspintojen reunat ovat paloittain lineaarisia, mutta lisäämällä neuroverkkoon tarpeeksi neuroneja, voidaan reunat kuvata mielivaltaisen tarkasti. Jatkuvien funktioiden kuvaamisessa voidaan loogisen OR-operaation sijaan lisätä tai vähentää konvekseja pintoja sopivien painokertoimien avulla, jolloin kaksi piilokerrosta on tarpeeksi suuri määrä kuvaamaan minkä tahansa rajatun ja jatkuvan funktion eli se on universaali approksimaattori. [54]



Kuva 3.6: Kolme kerrosta kynnysfunktioneuroneja on riittävä määrä kuvaamaan minkä tahansa funktion. Ensimmäinen kerros jakaa sisääntuloavaruuden hypertasoilla, toinen kerros muodostaa konvekseja pintoja ja ulostulokerros yhdistää nämä mielivaltaisiksi, mahdollisesti irrallaan oleviksi pinnoiksi. Esimerkissä rajat ovat paloittain jatkuvia, mutta lisäämällä tarpeeksi neuroneja, voidaan reunat kuvata mielivaltaisen tarkasti. [54]

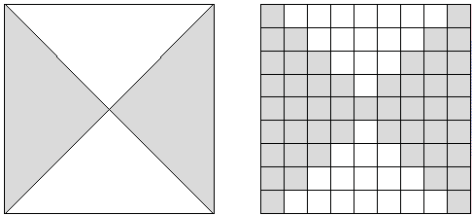
Kuvan 3.6 verkon toisen piilokerroksen neuronit saavat sisääntulonaan konvekseja pintoja. Usein luullaankin, että kaksikerrosverkoilla voidaan kuvata vain konvekseja päätöspintoja, mutta aina näin ei ole. Kaksikerrosverkko on oikeanlaisissa tehtävissä tehokas työkalu ja huomattavasti yksinkertaisempi opettaa kuin kahden piilokerroksen neuroverkot. Kuvassa 3.7 on esimerkkejä ei-konvekseista ja irrallisista päätöspinnoista, joita kaksikerrosverkoilla voidaan kuvata. [54]



Kuva 3.7: Neuroverkolla, jossa on yksi piilokerros voidaan esimerkiksi hahmontunnistuksessa kuvata ei-konvekseja sekä irrallisia päätöspintoja. Vasemmalla: Kuvio, jonka voi muodostaa neuroverkolla, jossa on kahdeksan neuronia yhdellä piilokerroksella ja kaksi ulostuloneuronia. Jokainen viiva muodostuu yhden piilokerrosneuronin muodostamalla päätöksellä. Neuroni on aktiivinen viivan sillä puolella, jota kuvataan nuolella ja on yhdistetty nuolen vieressä olevan numeron osoittamaan ulostuloneuroniin. Muut numerot osoittavat summatut sisääntuloarvot eri alueilla. Kynnysfunktion raja-arvolla 8.5 saadaan neliön muotoinen donitsi. Oikealla: Irrallisia alueita kuvattuna kaksikerrosverkolla samalla periaatteella. Neuroverkolla on kuusi piilokerrosneuronia ja yksi ulostuloneuroni. [54]

On esitetty joukko todistuksia, joissa kaksikerrosverkon todetaan olevan riittävä minkä tahansa jatkuvan funktion esittämiseen. Vaikka tulos onkin mielenkiintoinen, sen käytännön arvo on rajoitettu, sillä siinä ei oteta kantaa tarvittavien neuronien määrään. Kuvassa 3.8 on esimerkki funktiosta, jonka tarkkaan kuvaamiseen tarvittaisiin kaksikerrosverkossa ääretön määrä neuroneja, mutta kolmikerrosverkolla siihen riittää pienikin verkko. On olemassa epäjatkuvia funktioita, joita ei voida esittää kaksikerrosverkolla, vaikka sen neuronimäärä olisi kuinka suuri tahansa, mutta kolmikerrosverkolla näiden kuvaus onnistuu. Joka tapauksessa kaksikerrosverkot riittävät approksimaattoriksi suurimmalle osalle funktioista ja kannattaakin arvioida tapauskohtaisesti, käyttääkö kaksikerrosverkkoa vai kolmikerrosverkkoa. Kol-

mikerrosverkot pystyvät monimutkaisiin kuvauksiin pienemmillä neuronimäärillä kuin kaksikerrosverkot, mutta käytännössä kaksikerrosverkoilla on monesti pienemmät opetusajat sekä virheet. Kerrosten määrään on hyvä soveltaa jo aiemmin mainittua Occamin partaveitsiperiaatetta: on oltava varma, että kerrokset riittävät, mutta ylimääräiset kerrokset ovat todennäköisesti vain haitallisia. [54]



Kuva 3.8: Joitakin funktioita voidaan kuvata pienellä kolmikerrosverkolla, mutta kaksikerrosverkko tarvitsisi samaan kuvaukseen äärettömän määrän neuroneja. Kuvassa on vasemmalla yksinkertainen funktio, joka on kuvattu kolmikerrosverkolla, jossa on kahdella piilokerroksella kaksi neuronia. Oikealla on kuvattu sama funktio pelkästään yhdellä piilokerroksella. [54]

Toinen tärkeä valinta neuroverkoille on kerrosten neuronien määrä. Todistettaessa universaalia approksimointikykyä on todettu, että lisäämällä riittävä määrä neuroneja voidaan kuvata mikä tahansa haluttu funktio. Tällöin ei kuitenkaan oteta kantaa tapauksiin, joissa halutaan approksimoida funktiota, jonka muodostaa rajattu määrä diskreettejä datapisteitä. Vaikka olisikin mahdollista käyttää mielivaltaisen suurta neuroverkkoa, ei se välttämättä toimisi tehokkaasti, jos opetusdatasta ei saa tarpeeksi informaatiota. Selvästikään neuroneja ei voida sijoittaa neuroverkkoon ääretöntä määrää, joten niille on määrättävä jokin yläraja. Tietyn neuronimäärän jälkeen neuroverkko ylittää kriittisen pisteen, jolloin se sopii opetusdataan täydellisesti, mutta ei enää yleisty siihen funktioon, joka tuotti opetusdatan alunperin. Yleisesti ottaen ongelman suhteen minimaalisella verkolla on todettu olevan huomattavasti parempi yleistyskyky kuin suurilla ja monimutkaisilla verkoilla [57]. Jos piilokerroksen solmujen lukumäärä on pienempi kuin syötteiden lukumäärä, niin neuroverkko hukkaa tietoa eli toisin sanoen pudottaa ongelman dimensiota. Tällaisessa tapauksessa neuroverkko valitsee syötevektorista ne, jotka korreloivat eniten tavoitearvojen kanssa. Jones ja Barron ovat osoittaneet, että jos yritetään approksimoida funktiota $f(x)$ neuroverkolla, jossa solmujen lukumäärä on rajallinen, niin seurauksena on residuaalivirhe. Virhe vähenee funktion $O(1/M)$ mukaisesti, kun piilokerroksen solmujen määrää M lisätään [58].

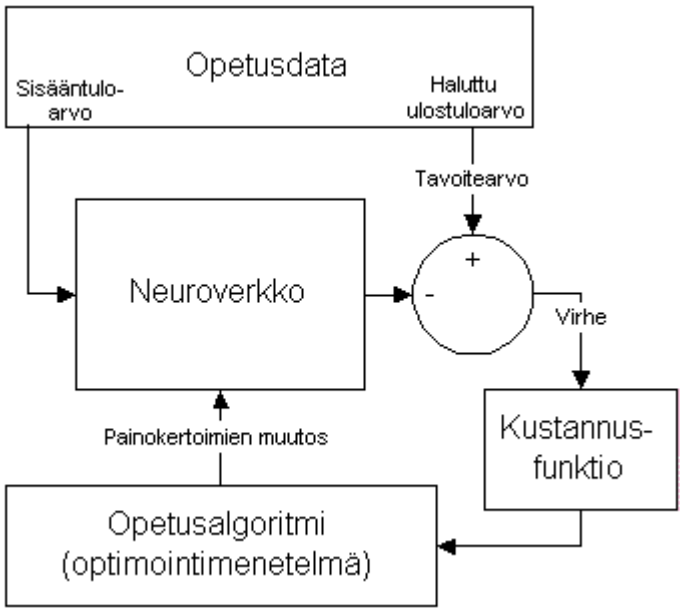
3.3 Neuroverkon oppiminen

Kun neuroverkon arkkitehtuuri on päätetty ongelmaan sopivaksi, voidaan aloittaa sen opettaminen. Opettamisessa on kaksi lähestymistapaa. Ohjatussa oppimisessa tiedetään, mitä neuroverkolta halutaan ulostulona ja optimoidaan verkon painot sen mukaisesti. Mukautuvassa oppimisessa haluttuja arvoja ei etukäteen tiedetä ja oppimisen tehtävänä on löytää annetuista syötteistä lainalaisuuksia, joita voidaan hyödyntää informaation tulkinnassa.

3.3.1 Ohjattu oppiminen

Ohjatussa oppimisessa (engl. *supervised learning*) neuroverkon yhteyksien painokertoimia säädetään niin, että opetuksen jälkeen se tuottaa sisääntuloarvoista haluttuja tuloksia. Sisääntuloarvojen ja haluttujen ulostuloarvojen keskinäisiä funktionaalisia riippuvuuksia ei yleensä tiedetä. Muulloin käytettäisiin todennäköisesti jotain toista menetelmää, koska funktio, jota halutaan approksimoida, tiedetään etukäteen. Kuvassa 3.9 on ohjatun oppimisen malli. Menetelmää, jossa saatuja arvoja verrataan haluttuun arvoon ja tämän jälkeen arvioidaan, mitkä yhteydet ovat syypäitä virheelliseen tulokseen, kutsutaan virheenjako-ongelmaksi (kirjoittajan suomennos, engl. *blame assignment problem*) [57]. Neuroverkoille, joissa ei ole piilokerroksia, tämän tyyppisiä tekniikoita on ollut tiedossa suhteellisen pitkän aikaa. Kuitenkaan niillä ei saatu tyydyttäviä tuloksia yleisessä tapauksessa ja se osaltaan johti kiinnostuksen hiipumiseen neuroverkkoja kohtaan.

1980-luvulla kehitettiin menetelmä, joka osoittautui yksinkertaiseksi ja tehokkaaksi. Menetelmän perusideana on mitata koko järjestelmän suoriutuminen ja sen jälkeen optimoida sitä [57]. Vastavirta-algoritmi (engl. *back-propagation algorithm*) on yleisin MLP-verkkojen opetukseen käytetty algoritmi. Opetusdatasta tuotetaan neuroverkon avulla ulostuloarvot, joita verrataan haluttuihin arvoihin. Neuroverkon antaman tuloksen ja tavoitearvon erosta lasketaan tuotettu virhe. Virhesignaali lähetetään ulostulokerrokselta sitä edeltävään piilokerrokseen. Kukin piilokerroksen solmu saa virheestä osan suhteutettuna sen osuuteen ulostuloarvosta. Tätä prosessia jatketaan, kunnes jokainen verkon solmu on saanut tietoonsa oman osuutensa kokonaisvirheestä. Yhteyksien painokertoimia päivitetään tämän tiedon perusteella siten, että ulostuloarvo lähenee tavoitearvoa. Vastavirta-algoritmi on seuraavanlainen:



Kuva 3.9: Ohjatun oppimisen periaate [54].

1. Määritelmät

virhefunktio:	E
solmun j sisääntuloarvojen summa:	net_j
solmun j ulostulo:	y_j
solmun j ulostulon tavoitearvo:	t_j
virhesignaali solmulle j:	$\delta_j = -\frac{\partial E}{\partial net_j}$
gradientti painolle w_{ij}:	$\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}}$
solmujen joukko ennen solmua i (engl. *anterior*):	$A_i = \{j : \exists w_{ij}\}$
solmujen joukko jälkeen solmun j (engl. *posterior*):	$P_j = \{i : \exists w_{ij}\}$

2. Gradientti. Gradientin saa jaettua kahteen osaan käyttämällä derivoinnin ketjusääntöä:

$$\Delta w_{ij} = -\frac{\partial E}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}} \tag{3.6}$$

Ensimmäinen tekijä on solmun i virhe ja toinen tekijä on:

$$\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\sum_{k \in A_i} w_{ik} y_k = y_j \tag{3.7}$$

Yhdistämällä nämä, saadaan painokertoimen muutos solmujen i ja j väliselle yhteydelle:

$$\Delta w_{ij} = \delta_i y_j \tag{3.8}$$

Tämän gradientin laskemiseen tarvitaan tieto kaikkien solmujen ulostuloista ja virheistä neuroverkossa.

3. Solmujen ulostulot. Sisääntulokerroksen ulostulot määräytyvät sisään tulevan ulkoisen signaalin x arvosta. Muiden solmujen ulostulot määräytyvät myötävirtaan aiem-

pien solmujen (joukko A_i) ulostuloista.

$$y_i = f_i(\sum_{j \in A_i} w_{ij} y_j) \tag{3.9}$$

4. Ulostulon virhe. Käytettäessä neliösummaa

$$E = \frac{1}{2} \sum_o (t_o - y_o)^2 \tag{3.10}$$

ulostulosolmun virhe on:

$$\delta_o = t_o - y_o \tag{3.11}$$

5. Virheen levittäminen takaisin päin. Piilokerroksen solmuilla virhe on levitettävä taaksepäin. Käyttämällä jälleen derivoinnin ketjusääntöä voidaan piilokerroksen solmun virhe ilmaista sen aiempien solmujen avulla:

$$\delta_j = -\sum_{i \in P_j} \frac{\partial E}{\partial net_i} \frac{\partial net_i}{\partial y_j} \frac{\partial y_j}{\partial net_j} \tag{3.12}$$

Ensimmäinen tekijä on virhe solmulle i. Toinen tekijä on:

$$\frac{\partial net_i}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_{k \in A_i} w_{ik} y_k = w_{ij} \tag{3.13}$$

Kolmas tekijä on derivaatta solmun j aktivaatiofunktiosta:

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f_j(net_j)}{\partial net_j} = f_j'(net_j) \tag{3.14}$$

Piilokerroksen solmuille, jotka käyttävät aktivaatiofunktiona $tanh$-funktiota, voidaan käyttää hyväksi tietoa $tanh(u)' = 1 - tanh(u)^2$, jolloin saadaan:

$$f_h'(net_h) = 1 - y_h^2 \tag{3.15}$$

Kokoamalla nämä yhteen saadaan:

$$\delta_j = f_j'(net_j) \sum_{i \in P_j} \delta_i w_{ij} \tag{3.16}$$

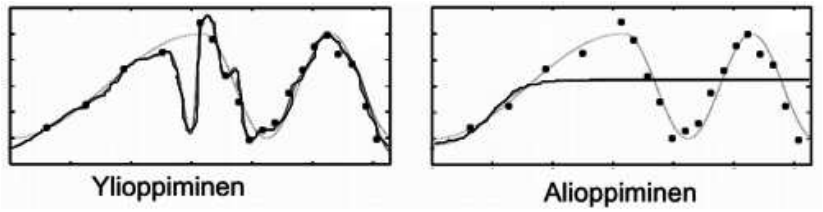
Painokertoimien muutoksissa käytetään usein oppimisnopeuteen vaikuttavaa kerrointa tai niin sanottua askelpituutta α, jolloin saadaan:

$$\Delta w_{ij} = -\alpha \delta_j y_i \tag{3.17}$$

Pienellä askelpituudella α saadaan taattua tarkempi tulos, mutta suurella askelpituudella opetus on nopeampaa. Liian suurella askelpituudella on mahdollista, että opetus konvergoituu liian aikaisin. Joissain algoritmeissa oppimisnopeuskerrointa säädetään automaattisesti. Esimerkiksi Rprop-algoritmissa kerroin α ja muutos painoon riippuvat pelkästään derivaatan etumerkistä [7].

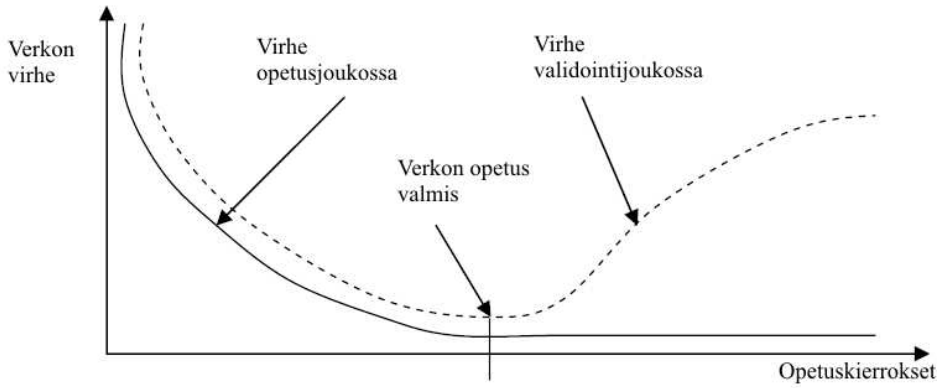
On huomattava, että tietääkseen solmun j ulostulon, on tiedettävä kaikkien sitä edeltävien solmujen (joukko P_j) ulostulot. MLP-verkossa ei ole silmukoita, jolloin vastavirta-algoritmin käyttö on yksinkertaista ja virhe voidaan levittää suoraan samaa reittiä, kuin ulostulot ovat muodostuneet.

Neuroverkon käytettävyyden kannalta oleellinen päätös on, milloin sen opettaminen lopetetaan. Erityisesti universaalin approksimaattorin tapauksessa ongelmaksi saattaa muodostua liian tarkka havaintopisteiden oppiminen, jolloin neuroverkko ei enää kuvaa haluttua funktiota. Jos data on opittu liian hyvin, puhutaan ylioppimisesta (engl. *overfitting*) ja toisaalta liian vähäinen opetus johtaa alioppimiseen (engl. *underfitting*). Yli- ja alioppiminen on esitelty kuvassa 3.10. Yleinen syy ylioppimiseen on liian suuri neuroverkko. Minimaalisen neuroverkon on todettu yleistyvän paremmin opetusdataan kuin monimutkaisen verkon [57]. Toinen tärkeä syy ylioppimiseen on kohinainen opetusdata, jolloin neuroverkon saattaa olla vaikea löytää sitä funktiota, jolla opetusdata on tuotettu. Yksi keino estää ylioppimista on lisätä virhefunktioon ylimääräinen virhetermi, joka sakottaa mallin liiasta monimutkaisuudesta ja esimerkiksi pakottaa painojen itseisarvoja pieniksi. Suuret painot verkossa tuottavat yleensä monimutkaisen ja voimakkaasti heilahtelevan ulostulon [22].



Kuva 3.10: Ylioppimisessa neuroverkon muodostama funktio (paksu käyrä) seuraa opetuksessa käytettyjä datapisteitä liian tarkasti eikä enää yleisty alkuperäiseen funktioon (ohut käyrä). Ylioppimiseen on usein syynä liian pitkään kestävä opetus tai liian monimutkainen verkko. Jos opetus puolestaan lopetetaan liian aikaisin tai käytetään liian yksinkertaista verkkoa, on seurauksena yleensä alioppiminen, jossa opetusdataa seurataan liian jäykästi.

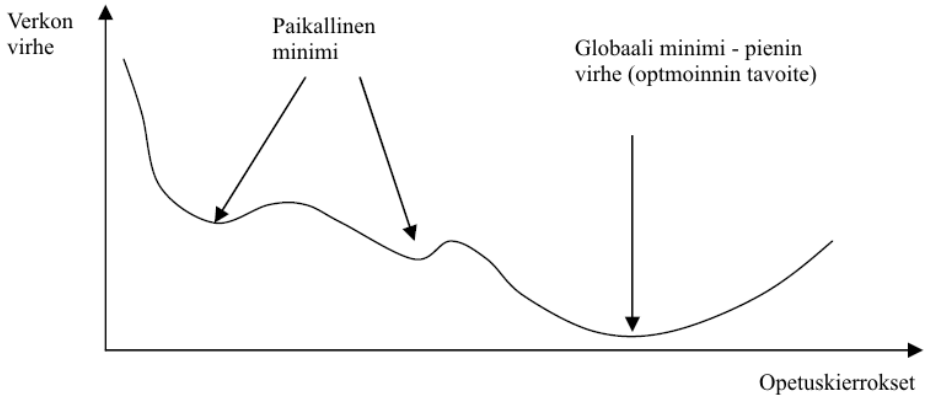
Yleisin keino ylioppimisen estämiseen on ristiinvalidointi (engl. *cross-validation*) , jonka avulla päätetään sopiva hetki opetuksen lopettamiseen. Ristiinvalidoinnissa data jaetaan kolmeen osaan: yhtä osaa käytetään opetukseen, toista osaa käytetään yleistyksen mittaamiseen ja se laitetaan sivuun lopputestiä varten ja kolmatta osaa käytetään ristiinvalidointiin. Validointijoukon käyttöä on havainnollistettu kuvassa 3.11. Jokaisen epookin (aikajakso, jonka aikana koko opetusaineisto on käyty läpi, engl. *epoch*) jälkeen neuroverkon suorituskyky arvioidaan validointijoukossa. Niin kauan kun neuroverkon suorituskyky paranee validointijoukossa, opetusta jatketaan. Jos ylioppimista tapahtuu, se ilmenee huonompana suorituksena validointijoukossa ja opetus lopetetaan. Tämän jälkeen parhaan suorituksen antanutta neuroverkkoa mitataan vielä yleistysjoukossa, jolloin saadaan selville myös sen kyky halutun funktion approksimoinnissa. [57]



Kuva 3.11: Opetuksen keskeyttäminen validointijoukon avulla.

Toinen tärkeä ongelma neuroverkkojen opetuksessa on paikalliset minimit. Tämä ongelma esiintyy kaikissa paikallisissa hakualgoritmeissa, ei pelkästään neuroverkkojen optimoinnissa. Paikallisen minimin ongelma johtuu siitä, että useat opetusalgoritmit muuttavat painoja aina virhettä pienentävään suuntaan. Sellaisissa tapauksissa, joissa virhe paikallisesti kasvaa, opetusalgoritmi ei pääse ylämäkeen ja jää jumiin (kuva 3.12). Tähän ongelmaan on useita ratkaisumalleja, joista ehkä yksinkertaisin on aloittaa opetus useasta eri lähtötilanteesta. Tämä voidaan tehdä aloittamalla opetus aina uudestaan tai aloittamalla opetus usealla eri neuroverkolla. Opetusajat voivat olla pitkiä, jolloin kannattaa opettaa useita neuroverkkoja tietyn aikaa ja jatkaa opetusta pitempään niillä verkoilla, jotka ovat pärjänneet hyvin. Vaikka takuita globaalin minimin löytämisestä ei tällä keinolla voida taata, sen on todettu karsivan huonoja lähtötilanteita tehokkaasti [54]. Toinen yleinen keino paikallisille minimeille on momenttitermin lisääminen. Momenttitermiä käytettäessä painojen muuttaminen ei perustu pelkästään sen hetkiseen virheeseen, vaan myös aiempien painomuutosten virheisiin. Tällöin saadaan aikaan vaikutus, jossa opetuksella on omaa sisäistä painoa ja se jatkaa tiettyyn suuntaan, vaikka hetkellinen gradientti olisikin vastakkaissuuntainen. Momenttitermiä käytetään tehokkaasti myös vastavirta-algoritmin nopeutukseen [42].

Ohjatun oppimisen etuna on, että se on hyvin määritelty. Se on tarpeeksi yksityiskohtainen ollakseen käyttökelpoinen, mutta samaan aikaan tarpeeksi yksinkertainen ollakseen helposti analysoitavissa. Mallia on kritisoitu sen keinotekoisesta yhteydestä biologiseen oppimiseen, mutta sen yleisin käyttötarkoitus on funktioapproksimointi ja siihen se sopii hyvin.



Kuva 3.12: Paikallisen minimin ongelma. Opetusalgoritmi, joka osaa vain pienentää virhettä saattaa joutua jumiin paikallisen minimin kohdalle.

3.3.2 Mukautuva oppiminen

Ohjatussa oppimisessa tarvitaan tavoitearvot, joita neuroverkko opetetaan tuottamaan. Aina tällaisia tavoitearvoja ei voida esittää. Mukautuvassa oppimisessa opetusdata on usein luokittelematonta (engl. *unlabeled data*) eikä etukäteen voida sanoa, mitä arvoja niistä halutaan. Mukautuvassa oppimisessa neuroverkko löytää sisääntuloarvoista säännöllisyyksiä riippuen sen rakenteesta eli sen tavoitearvot muotoutuvat sisäisen toiminnan perusteella. Mukautuva oppiminen on käyttökelpoinen, sillä usein käytettävä opetusdata on luokittelematonta. Mukautuvaa oppimista käytetään joskus myös ohjatun oppimisen osana esimerkiksi vastavirta-algoritmin askelpituuden määräämiseen. [54]

Hebbin algoritmi [30] oli ensimmäisiä neuroverkkojen opetusalgoritmeja. Se perustuu ajatukseen, että kahden neuronin yhteyttä tulee vahvistaa, jos niiden aktivaatiot (ulostulot) ovat samoja. Matemaattisessa muodossa Hebbin algoritmi on kaavassa 3.18.

$$\Delta W_{ij} = \gamma x_i x_j, \tag{3.18}$$

missä γ on opetuksen nopeustermi, x_i ja x_j ovat neuronien i ja j aktivaatiot, W_{ij} niiden välisen yhteyden paino ja ΔW_{ij} painon muutos. Vaikkakin Hebbin algoritmin on todettu olevan rajoittunut joissain tapauksissa [44], niin sen on todettu mallintavan aivojen toimintaa tietyissä hippokampuksen muistitoimintojen synapseissa [56].

Myös tässä tutkimuksessa käytettävät evoluutioalgoritmeihin perustuvat opetusalgoritmit ovat mukautuvaa oppimista, mistä kerrotaan tarkemmin seuraavassa luvussa.

4 Evoluutioalgoritmit

Evoluutio on populaatioon perustuva optimointiprosessi. Simuloimalla tätä prosessia tietokoneella saadaan aikaan stokastinen optimointimenetelmä, joka on joissakin tehtävissä huomattavasti käyttökelpoisempi kuin monet perinteiset optimointimenetelmät. Evoluutioalgoritmit (engl. *evolutionary algorithms*, EA) jaetaan usein kolmeen osaan, jossa kussakin painotetaan eri näkökulmaa evoluutioprosessiin. Geneettiset algoritmit (engl. *genetic algorithms*, GA) painottavat kromosomien muuntelua. Evoluutiostrategiat (engl. *evolutionary strategies*, ES) korostavat yksilöiden käyttäytymisen eroja. Evoluutio-ohjelmointi (engl. *evolutionary programming*, EP) puolestaan painottaa käyttäytymistä lajitasolla. [20]

Evoluutioalgoritmit soveltuvat hyvin neuroverkkojen opetukseen, sillä ne tarvitsevat ongelmasta vain vähän yksityiskohtaista tietoa. Menetelmän heikkouksiin kuuluu sen suuri laskentatehon tarve, mistä johtuen algoritmi voi olla joskus hyvinkin hidas. [54]

4.1 Evoluutioalgoritmien taustaa

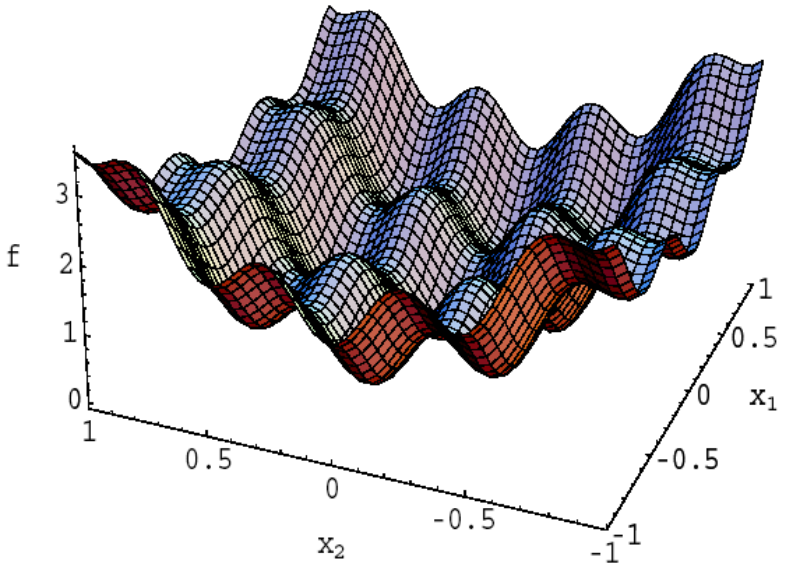
Optimoinnin periaatteena on muodostaa yksittäinen mitta eli kustannusfunktio, jolla ilmaistaan ratkaisun suorituskyky tai hyvyys, ja iteratiivisesti parantaa tämän arvoa valitsemalla ratkaisuja saatavilla olevista vaihtoehdoista. Nämä vaihtoehdot muodostetaan perinteisesti laskemalla deterministinen sarja ratkaisuja, jotka perustuvat tietoihin kustannusfunktion gradienteista tai korkeamman asteen funktioista. Kustannusfunktioon vaikuttavien tekijöiden ollessa säännönmukaisia näiden menetelmien on todettu lähenevän lokaaleja minimejä asymptoottisesti ja joissain tapauksissa jopa eksponentiaalisen nopeasti. Yksi esimerkki tällaisen menetelmän toteutuksesta on luvussa 3.3.1 esitelty vastavirta-algoritmi. Nämä menetelmät eivät kuitenkaan suoriudu riittävän hyvin, jos optimointiin liittyy satunnaisia häiriötekijöitä, kuten esimerkiksi muuttuva ympäristö tai heikosti määritellyt muuttujat. Lisäksi paikallisen minimin ongelma saattaa muodostua ratkaisevaksi esteeksi tällaisen menetelmän käyttöönotossa. Evolutionääristä prosessia voidaan soveltaa ongelmiin, joissa perinteiset matemaattiset optimointimenetelmät ovat riittämättömiä tai ongelman tarkka määrittely on puutteellisista lähtötiedoista johtuen hankala tai mahdoton tehtävä. [20]

Darwinin evoluutio-oppi on itsessään monipuolinen haku- ja optimointimenetelmä. Optimoituvaa käyttäytymistä esiintyy sen kaikissa osa-alueissa: soluissa, yksilöissä ja populaatiossa. Biologisten eliöiden ratkaisemien ongelmien piirteisiin kuuluu kaaosta, todennäköisyyksiä, jatkuvaa kehitystä ja epälineaarisuutta. Evoluutio perustuu monipuoliseen populaatioon. Yksilöiden jälkeläisten on perittävä riittävästi ominaisuuksia vanhemmiltaan, mutta myös tuotettava eroavaisuuksia sopeutuakseen muuttuvaan ympäristöön. Osa jälkeläisistä ei sopeudu ympäristöönsä ja kuolee, mutta osa selviää ja saa mahdollisuuden välittää ominaisuuksiaan eteenpäin. Yksilön elinkyky riippuu suurelta osin sen geneettisestä koodista, mutta sattuma saattaa joskus hävittää jopa parhaan yksilön ja jättää heikompia eloon. Useiden generaatioiden jälkeen populaation sopeutuvuus ympäristöönsä kuitenkin paranee yksilöiden epäedullisten ominaisuuksien tullessa harvinaisemmiksi. Uusdarwinistisen teorian argumenteissa esitetään, että elämän historia voidaan esittää täysin fyysisillä prosesseilla, joita esiintyy populaatiossa ja eliölajeissa. Nämä prosessit ovat lisääntyminen, mutaatio, kilpailu ja valinta. Lisääntymisessä välitetään yksilön ominaisuuksia geenikartan avulla seuraaville yksilöille. Ympäristön vaikutuksesta informaation välityksessä voi sattua virheitä eli mutaatiota. Kilpailu on seurausta rajoitetusta resurssimäärästä ja estää populaatiota lisääntymästä rajattomasti ja valinta vaikuttaa siihen, mitkä yksilöt saavat oikeuden käyttää näitä resursseja. Evoluutio on seurausta näiden yksinkertaisten prosessien vaikutuksesta populaatioon generaatiosta toiseen. [31]

Yksilöt voidaan kuvata niiden genotyypin (engl. *genotype*) ja fenotyypin (engl. *phenotype*) avulla. Genotyyppiin on tallennettu ohjeet, joiden mukaan geenien ilmentymä eli fenotyyppi muodostuu. Genotyypin ja fenotyypin suhdetta voidaan kuvata yhteydellä *genotyyppi* + *ympäristö* + *satunnaistekijät* → *fenotyyppi*. Pleiotropia (engl. *pleiotropy*) on ilmiö, jossa yksi geeni vaikuttaa useaan ominaisuuteen yksilössä ja geenin muuttuessa on mahdollista, että yksi ominaisuus yksilössä paranee ja toinen huononee. Polygeenisessä periytymisessä (engl. *polygenic inheritance*) puolestaan jokin ominaisuus on useamman geenin yhteistulos. Luonnollisesti kehittyneissä systeemeissä ei ole yksi-yhteen -suhdetta geenistä johonkin ominaisuuteen, vaan fenotyyppi toimii monimutkaisena, epälineaarisena funktiona vuorovaikutuksessa genotyypin ja vallitsevan ympäristön kanssa. Hyvinkin erilaiset geneettiset rakenteet voivat muodostaa samanlaisia ilmentymiä aivan kuten erilaiset tietokoneohjelmat voivat tuottaa samanlaisia tuloksia. [20]

Valinta vaikuttaa pelkästään yksilöiden ilmentymiin eli fenotyyppiin. Yleinen tapa kuvata yksilöiden hyvyys (engl. *fitness*) on muodostaa topografinen kuvaaja, jossa genotyyppien joukko kuvataan muodostuneiden yksilöiden hyvyysarvoksi. Ku-

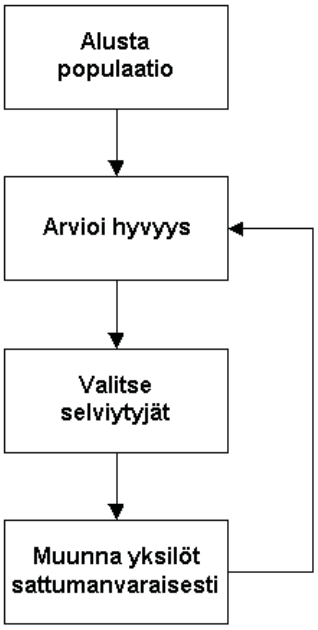
vaajan jokainen minimi vastaa joukkoa optimeja fenotyyppejä ja näin ollen myös yhtä tai useampaa joukkoa optimoituja genotyyppejä. Kuvassa 4.1 on esimerkki yksinkertaisesta topografisesta kuvauksesta. Evoluutio etenee graafin pintaa pitkin kohti laakson pohjia valinnan karsiessa huonoja yksilöitä pois populaatiosta.



Kuva 4.1: Topografinen kartta on määritelty kuvaamaan kaikkien fenotyyppien hyvyysarvo. Valintaprosessin edetessä evoluutio etenee ratkaisupinnalla kohti kuvaajan minimejä. Esimerkin funktio on $f(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$. [26]

4.2 Evoluutioalgoritmien toteutuksista

Kuva 4.2 esittää evoluutioalgoritmin vaiheet, jotka ovat pääpiirteiltään samanlaisia kaikissa evoluutioalgoritmien toteutuksissa. Jokainen populaation yksilö on ratkaisu annettuun ongelmaan ja tämän ratkaisun hyvyyttä mitataan hyvyysfunktiolla (engl. *fitness function*). EA:ssa yksilöille on valittava sopiva esitystapa, jotta niitä voidaan käsitellä matemaattisesti ja esimerkiksi GA:ssa yksilöt kuvataan bittijonoina. EA:t ovat stokastista optimointia eli etukäteen ei voida sanoa, kuinka monta iteraatiokierrosta eli generaatiota tarvitaan optimiratkaisun löytämiseksi. Lisäksi ympäristö saattaa olla myös jatkuvasti muuttuva, jolloin yksilöt jäävät saavutetusta optimiratkaisusta jälkeen ja ovat jatkuvassa muutoksessa kohti uutta optimia. Yleensä EA:n suorittamisen lopetusehtona on jokin ennalta sovittu generaatiomäärä tai konvergenssin pieneneminen tietyn raja-arvon alapuolelle, jolloin uudet generaatiot ei-



Kuva 4.2: Evoluutioalgoritmin vaiheet [26].

vät enää paranna hyvyysarvoa merkittävästi. Aluksi yksilöt alustetaan sattumanvaraisesti, jonka jälkeen niitä operoidaan generaatioittain operaatioilla hyvyyden arviointi, valinta, rekombinaatio ja mutaatio. [5]

4.2.1 Geneettiset operaatiot

Valinnan tarkoituksena on valita sopivat vanhemmat seuraavalle generaatiolle. Yleisimpiä valintatapoja ovat stokastinen valinta (engl. *stochastic universal sampling*), rulettivalinta (engl. *roulette selection*), turnajaisvalinta (engl. *tournament selection*), sijoitukseen perustuva valinta (engl. *fitness ranking selection*) ja elitistinen valinta (engl. *elitist selection*). [8] [28]

- Stokastinen valinta on yksinkertaisin valintamenetelmä, jossa jokainen yksilö saa saman todennäköisyyden tulla valituksi hyvyysarvostaan riippumatta.
- Rulettivalinnassa yksilön todennäköisyys tulla valituksi on $F_i = \frac{f_i}{\sum_j f_j}$, missä f_i on yksilön i hyvyysarvo. Yksilöt saavat rulettipyörästä sektorin, jonka koko on suhteutettu sen hyvyysarvoon. Tämän jälkeen rulettia pyöräytetään m kertaa, jolloin m yksilöä tulee valituksi.
- Turnajaisvalinnassa k:n yksilön joukko valitaan satunnaisesti. Tämän jälkeen nämä k yksilöä osallistuvat turnajaisiin ja parhaan hyvyysarvon omaava yk-

silö pääsee seuraavaan generaatioon. Rekombinaatiossa pidetään kaksi turnajaista, jossa kummassakin valitaan jälkeläiselle toinen vanhempi. On mahdollista, että sama yksilö tulee valituksi useammin kuin kerran. Turnajaisvalinnan etu on siinä, että huonoimpia yksilöitä ei tule koskaan valituksi ja parhaimmilla yksilöillä on selvä etulyöntiasema.

- Sijoitukseen perustuvassa valinnassa yksilöt järjestetään hyvyysarvon mukaisesti paremmuusjärjestykseen. Yksilön todennäköisyys tulla valituksi on suoraan verrannollinen sen normalisoituun sijoitukseen. Tässä valinnassa suuren hyvyysarvon omaavat yksilöt eivät pääse dominoimaan valintaprosessia, kuten esimerkiksi rulettivalinnassa helposti käy.

- Elitistisessä valinnassa voidaan valita myös nykyisen generaation yksilöitä suoraan seuraavaan generaatioon. Suhteelliseen hyvyysarvoon perustuvissa valintatavoissa ei ole takuita, että globaalia optimia saavutetaan asymptoottisesti, sillä jos parhaat yksilöt katoavat, on mahdollista, että optimointi ei saavuta seuraavassa generaatiossa jo aiemmin saavutettua hyvyysarvoa [20]. Elitistisessä valinnassa paras yksilö ei katoa evoluution edetessä ja näin ollen taataan asymptoottinen konvergenssi. Takuun hintana on, että osa populaation koosta toimii säilönä ja optimointi hidastuu.

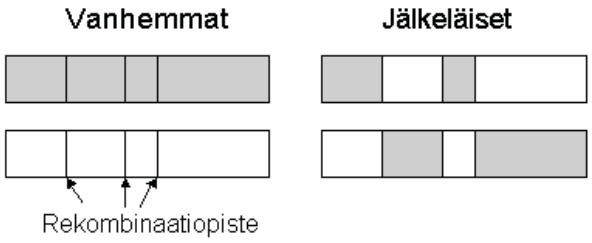
GA:ssa ja EP:ssa suositaan edellä mainituista menetelmistä neljää ensimmäistä, jotka ovat todennäköisyyksiin perustuvia valintamenettelyjä, kun taas ES:ssa käytetään usein elitististä valintaa. [5]

Rekombinaatiossa eli risteytyksessä vanhempien kromosomeja sekoitetaan keskenään jälkeläisen tuottamiseksi. Yksipisteristeytyksessä jälkeläinen saa toiselta vanhemmalta kromosomista katkaisupistettä edeltävät geenit ja toiselta vanhemmalta katkaisupisteen jälkeiset geenit. Lisäämällä katkaisupisteitä saadaan kaksipisteristeytys, joka ilmenee myös luonnossa, sekä monipisteristeytys. Kuvassa 4.3 on monipisteristeytys, jossa on kolme katkaisupistettä. Katkaisupisteet arvotaan yleensä satunnaisesti. Tasaisessa risteytyksessä geenejä arvotaan yksi kerrallaan:

$$x'_i = \begin{cases} x_{S,i}, & \chi > \frac{1}{2} \\ x_{T,i}, & \chi \leq \frac{1}{2}, \end{cases} \tag{4.1}$$

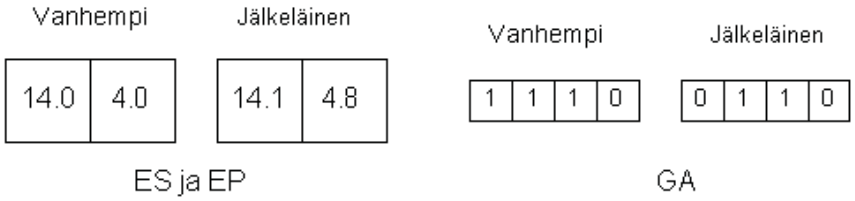
missä S ja T ovat uudelle yksilölle x'_i satunnaisesti valitut vanhemmat, i on yksilön geeni ja $\chi \in [0, 1]$ on satunnaisluku. [5]

Mutaatiolla tuotetaan uusia jälkeläisiä muuttamalla vanhempien geenejä satunnaisesti. Mutaatio on yksilöiden esitystavasta johtuen ES:ssa ja EP:ssa erilainen kuin



Kuva 4.3: Kolmepisteristeytys.

GA:ssa, kuten myös kuvasta 4.4 käy ilmi. GA:ssa yksilöt esitetään bittijonoina, jolloin geenin mutatoiminen tarkoittaa käytännössä kyseisen bitin vaihtamista jollain tietyllä todennäköisyydellä. Yleisesti GA:ssa tämä todennäköisyys vaihtelee välillä [0.001,0.01]. GA:ssa mutatoidaan geeneittäin, kun taas ES:ssa ja EP:ssa mutatoidaan kerralla koko yksilö. [20]



Kuva 4.4: Mutaatio. GA:ssa mutaation vaikutus saattaa joskus olla lopputuloksen kannalta suuri, jos geenejä mutatoidaan suoraan bittijonossa. Esimerkiksi kuvassa bittijonon 1110 arvo muuttuu arvoon 0110 eli kymmenjärjestelmässä arvosta 14 arvoon 6. Ongelman saa ratkaistua järjestelemällä bittijono Grey coding -tekniikalla [27]. ES:ssa ja EP:ssa mutaatiolla hienosäädetään yksilöiden geenien arvoja suoraan.

ES:ssa ja EP:ssa yleisin mutaatio on Gaussin mutaatio, jossa Gaussin normaalijakauman avulla tuotettua satunnaislukua käytetään yksilön muuntamiseen. Jos yksilö x kuvataan n-ulotteisena vektorina. Jälkeläiset x_i' tuotetaan tällöin kaavalla

$$x_i' = x_i + N(0,1), \tag{4.2}$$

missä $N(0,1)$ on normaalijakautunut satunnaisluku ja $i = 1, \ldots, n$.

Mutaatioon lisätään usein strategiaparametri, jonka tarkoituksena on määrittää, kuinka suuria muutoksia kullakin iteraatiokierroksella tehdään. Parametrivektori σ on myös n-ulotteinen ja se mutatoidaan samalla tavoin kuin yksilövektorikin. Schwefel ja Rechenberg esittivät, että σ voidaan asettaa myös itsemukautuvaksi (engl. *self-adaptation*):

$$x_i' = x_i + N(0, \sigma_i') \tag{4.3}$$

$$\sigma_i' = \sigma_i \exp(\tau' N(0,1) + \tau N_i(0,1)), \tag{4.4}$$

missä $\exp(\tau' N(0,1))$ toimii globaalina tekijänä, joka sallii mutaation kokonaisvaikutuksen ja $\exp(\tau N_i(0,1))$ toimii paikallisena tekijänä, joka sallii yksilölliset muutokset askelkoolla σ_i. Askelkokojen määrittämiseen käytetään muuttujia $\tau' = \frac{1}{\sqrt{2n}}$ ja $\tau = \frac{1}{\sqrt{2\sqrt{n}}}$. Nyt mutatoinnin askelkoko määräytyy aina sen hetkisen tilanteen mukaan sen sijaan, että sitä säädeltäisiin jonkin aikataulun avulla eli toisin sanoen se on itsemukautuva. [20]

Muita yleisiä mutaatiomenetelmiä ovat esimerkiksi Cauchy-mutaatio ja Lévy-mutaatio. Yhteistä näille menetelmille on, että kyseisen menetelmän jakaumalla tuotetaan satunnaisluku, jolla yksilöiden geenejä kerrotaan. [6]

4.2.2 Geneettiset algoritmit

Geneettiset algoritmit esitteli ensimmäisen kerran J. H. Holland vuonna 1962. Ne oli alunperin suunniteltu yleiseksi malliksi mukautuvasta prosessista, mutta myöhemmin niiden yleiseksi käyttötarkoitukseksi on muodostunut optimointi. GA:t mallintavat luonnossa esiintyviä kromosomeja ja niiden DNA-rakenteita. Evoluution simulointi perustuu geneettisen koodin muuttamiseen. Yksilön hyvyysarvo perustuu kromosomin koodin purkuun ja DNA:sta muodostuvan ilmentymän suorituskykyyn. [65]

GA:ssa yksilöiden geenit kuvataan binäärilukuina, jotka voivat olla ongelmasta riippuen numeroita tai symboleja. Yksilöiden esitystapa asettaa menetelmälle rajoitteen, jossa esitystarkkuus riippuu bittijonon pituudesta.

GA:ssa rekombinaatio on pääoperaattori ja mutaatiolla on vain pieni merkitys evoluution etenemiseen. Rekombinaatio suoritetaan kohdan 4.2.1 mukaisesti ja mutaatio tehdään geeni kerrallaan vaihtaen geenin bittiä tietyllä todennäköisyydellä. Yleisimmät valintamenetelmät ovat todennäköisyyksiin perustuvat rulettivalinta ja turnajaisvalinta. [5]

Geneettinen algoritmi on seuraavanlainen:

1. Generaatio $g = 0$.
2. Alusta populaatio C_g
3. kunnes $g \geq max\ generaatiot$ tai $ei\ konvergenssia$:
   - 3.1 Laske kaikkien yksilöiden hyvyys
   - 3.2 $g = g + 1$

3.3 Valitse vanhemmat joukosta C_{g-1}

3.4 Suorita rekombinaatio (risteytys) vanhempien joukossa, jolloin muodostuu jälkeläisten joukko O_g

3.5 Suorita mutaatio jälkeläisille O_g

3.6 Valitse uusi generaatio edellisen generaation C_{g-1} ja jälkeläisten joukosta O_g

4.2.3 Evoluutiostrategiat

Evoluutiostrategioiden kehittäjinä pidetään Rechenbergiä ja Schwefeliä, jotka esittelivät 1960-luvulla geneettisille algoritmeille vaihtoehtoisen tavan mallintaa evoluutiota. ES:aa kehitettiin aluksi kokeellisesti ja tavoitteena oli ratkaista optimointiongelmia, jotka olivat diskreettejä ja mahdottomia ratkaista perinteisillä menetelmillä. Mallissa käytettiin aluksi vain yhden yksilön kokoista populaatiota, jossa vanhempi kilpaili selviytymisestä jälkeläisensä kanssa ($(1+1)-ES$-algoritmi). Yksilönä käytettiin n-ulotteista reaaliarvoista vektoria. Menetelmä todettiin vajavaiseksi optimoinnin kannalta, sillä se konvergoitui hitaasti ja se oli altis seisahtumaan lokaaleihin minimeihin. Tämän jälkeen ES:ssa on lisätty yksilöiden määrää sekä otettu käyttöön uusia valintamenettelyjä. [5]

Yksilöt eli kohdemuuttujat kuvataan reaaliarvoisina n-ulotteisina vektoreina $\mathrm{x} \in \mathbb{R}^n$. Kohdemuuttujaan lisätään usein strategiamuuttuja, jolla voidaan parantaa optimoinnin mukautumista johonkin tiettyyn optimointitehtävään (katso luku 4.2.1). Tällöin yksilö $\mathrm{a}_i = (\mathrm{x}_i, \sigma_i)$ koostuu kahdesta osasta: kohdemuuttujista $\mathrm{x}_i = [x_1, \ldots, x_n]$ ja strategiaparametreistä $\sigma_i = [\sigma_1, \ldots, \sigma_n]$. [20]

Rekombinaatio voi olla ES:ssa lokaalia, missä jälkeläinen tuotetaan kahdella vanhemmalla, tai globaalia, missä jälkeläinen tuotetaan koko populaatiosta (panmiktinen rekombinaatio). Luvussa 4.2.1 esitellystä rekombinaatiosta poiketen ES:n rekombinaatio-operaatiot ovat diskreetti rekombinaatio, väli- ja keskiarvorekombinaatio, joista kaikista on lokaalit ja globaalit muodot. Rekombinaatiot yksilölle x'_i ja strategiaparametrille σ'_i on esitelty kaavoissa 4.5 ja 4.6. Käytetyimpiä näistä ovat diskreetti rekombinaatio, jossa satunnaisesti valitaan, miltä vanhemmalta geeni peritään, ja välirekombinaatio, jossa jälkeläisen geeni on vanhempien vastaavien geenien keskiarvo. [5]

$$x'_i = \begin{cases} x_{S,i}, & \text{ei rekombinaatiota} \\ x_{S,i} \; tai \; x_{T,i}, & \text{diskreetti rekombinaatio} \\ (x_{S,i} + x_{T,i})/2, & \text{välirekombinaatio} \\ \sum_{k=1}^{\mu} x_{k,i}/\mu, & \text{globaali keskiarvo} \end{cases} \tag{4.5}$$

$$\sigma_i' = \begin{cases} \sigma_{S,i}, & \text{ei rekombinaatiota} \\ \sigma_{S,i} \; tai \; \sigma_{T,i}, & \text{diskreetti rekombinaatio} \\ (\sigma_{S,i} + \sigma_{T,i})/2, & \text{välirekombinaatio} \\ \sum_{k=1}^{\mu} \sigma_{k,i}/\mu, & \text{globaali keskiarvo} \end{cases} \tag{4.6}$$

missä S ja T ovat vanhempia ja μ on vanhempien populaatio.

ES:ssa mutaatio etenee luvun 4.2.1 strategiaparametrin sisältävän version mukaisesti. Mutaatiolla on ES:ssa tärkeä rooli päinvastoin kuin GA:ssa.

Valinta on ES:ssa elitistinen. Valintatapana ES:ssa käytetään yleensä joko $(\mu+\lambda)-ES$- tai $(\mu,\lambda)-ES$-algoritmia, missä μ on vanhempien populaatio ja λ on jälkeläisten populaatio. $(\mu,\lambda)-ES$-algoritmissa vain jälkeläiset μ voivat tulla valituksi. $(\mu+\lambda)-ES$-algoritmissa myös vanhemmat λ kuuluvat valittavien joukkoon.

ES eroaa muista EA:eista siis risteytyksen ja valinnan osalta. Evoluutiostrategioiden algoritmi on seuraavanlainen:

1. Generaatio $g = 0$.
2. Alusta populaatio C_g
3. Laske kaikkien yksilöiden hyvyys
4. Kunnes $g \geq max\ generaatiot$ tai $ei\ konvergenssia$:
   - 4.1 kaikille $k = 1, 2, \ldots, \lambda$, missä λ on jälkeläisten lukumäärä
     - 4.1.1 Valitse $\rho \geq 2$ vanhempaa satunnaisesti
     - 4.1.2 Suorita rekombinaatio kohdemuuttujille ja strategiaparametreille
     - 4.1.3 Suorita mutaatio jälkeläisten kohdemuuttujille ja strategiaparametreille
     - 4.1.4 Laske jälkeläisten hyvyysarvo
   - 4.2 Valitse p parasta yksilöä jälkeläisten tai jälkeläisten ja vanhempien joukosta seuraavaan generaatioon
   - 4.3 $g = g + 1$

4.2.4 Evoluutio-ohjelmointi

Evoluutio-ohjelmointi kehitettiin erillään evoluutiostrategiosta L. J. Fogelin [21] toimesta. Fogel esitti näkemyksen, jossa älykäs käyttäytyminen sisältää myös ennakointia tulevasta ympäristöstä. Lisäksi yksilön tulisi myös reagoida ennustukseen tehtävässä annetun tavoitteen perusteella. Luodakseen mahdollisimman yleistyskelpoisen menetelmän, monien testien jälkeen esiteltiin simulointiympäristö, jossa luetaan symbolijonoa symboli kerrallaan. Tehtävänä oli tämän jälkeen tuottaa algoritmin ulostulona symbolijonoon seuraava symboli, joka todennäköisesti maksimoi

algoritmin tehokkuuden annetulla kustannusfunktiolla eli toisin sanoen ennustaa merkkijonon seuraava symboli. Kuvan 4.5 kaltaiset äärelliset tilakoneet (engl. *finite state machine, FSM*) tarjosivat sopivan esityksen tämäntyypiselle käytökselle. [20]

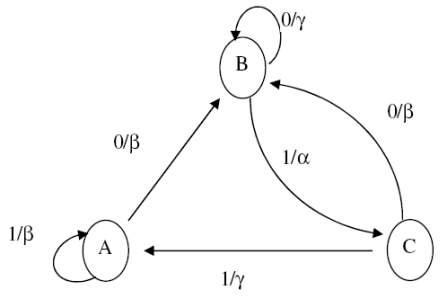
EP toimii FSM:n tapauksessa seuraavasti:

1. Luodaan alkupopulaatio FSM:a sattumanvaraisesti
2. Vanhemmat altistetaan ympäristölle eli toisin sanoen etukäteen tunnetulle jonolle symboleja. FSM:n hyvyysarvo määritellään vertaamalla kullakin sisääntulosymbolilla tuotettua ulostulosymbolia jonon seuraavaan sisääntulosymboliin. Jos ulostulosymboli on sama kuin seuraava sisääntulosymboli, on ennustus onnistunut. Koneen käytyä koko symbolijono läpi lasketaan sille hyvyysarvo sen tekemien ennustuksien perusteella.
3. Jälkeläiskoneet luodaan mutatoimalla vanhempia. Koneista voidaan muuttaa viittä osaa: ulostulosymbolia, tilan siirroksia, tilan lisäystä, tilan poistoa ja alkutilan vaihtamista. Mutaatiossa käytetään todennäköisyyksiin perustuvaa laskentaa ja jakaumana on yleensä normaalijakauma.
4. Jälkeläiset arvioidaan samalla tavalla kuin vanhempansakin aiemmin ja valintamenettelyn mukaisesti osa koneista valitaan seuraavan generaation vanhemmiksi. Yleensä populaation koko pidetään evoluution aikana samana.
5. Askeleita 3 ja 4 toistetaan, kunnes osataan ennakoida seuraava merkki ympäristönsä perusteella. Symbolijonoon lisätään uusi symboli ja paras kone valitaan mukaan askeleeseen 2.

Edellä esitetty ennustusongelma on sarja staattisia optimointitehtäviä, joissa adaptiivinen topografia eli hyvyysfunktio on ajasta riippuvainen. Edellä kuvattu prosessi voidaan helposti yleistää tilanteisiin, joissa yksilöiden käyttäytymisen kannattavuus ei riipu pelkästään sen omasta kustannusfunktiosta, vaan myös muiden yksilöiden käyttäytymisestä.

Lawrence J. Fogelin poika David B. Fogel on lisännyt EP:iin reaaliarvoisten, jatkuvien parametrien optimoinnin. EP on tänä päivänä käytännössä lähes samanlainen evoluutioalgoritmi kuin ES. Suurin ero ES:aan on siinä, että EP:ssa valinta perustuu todennäköisyyksiin, kun ES:ssa valinta on elitistinen. EP:ssa yksilön esitystavassa ei ole mitään rajoitteita. [5]

EP:ssa on se periaatteellinen ero muihin evoluutioalgoritmeihin, että siinä evoluutio perustuu yksilön käyttäytymiseen, kun ES:ssa ja GA:ssa se perustuu yksilön koodaukseen. Näin ollen EP:ssa ei ole rekombinaatiota lainkaan, vaan kaikki yksilöiden muunnokset perustuvat mutaatiolle. Pelkän mutaation käyttöä perustellaan myös sillä, että Fogel ja Atmar ovat todistaneet Gaussin jakaumaan perustuvan mutaation olevan tehokkaampi hakuoperaatio kuin yksipisterekombinaatio tai rekombinaation ja mutaation yhdistelmä, kun tarkastellaan optimointiongelmaa, johon liittyy pleiotropiaa. Geneettisten algoritmien puolella Eshelman ja Schaffer esittivät on-



Kuva 4.5: Yksinkertainen esimerkki äärellisestä tilakoneesta. Evoluutio-ohjelmointi perustuu ajatukselle, jossa tällainen tilakone on populaation yksilö, jota mutatoidaan sen toiminnan tehostamiseksi [20]. Taulukossa 4.1 on tilakoneen sisääntulo- ja ulostulosymbolit sekä tilan siirrokset.

Nykytila	C	B	C	A	A	B
Sisääntulosymboli	0	1	1	1	0	0
Seuraava tila	B	C	A	A	B	B
Ulostulosymboli	β	α	γ	β	β	γ

Taulukko 4.1: Kuvan 4.5 tilakoneen tuottamat ulostulosymbolit koneen tilan ja sisääntulosymbolin perusteella.

gelman, jossa puolestaan risteytys oli mutaatioon nähden selvästi parempi operaatio. D. B. Fogel esitti tähän biologiseen evoluutioon pohjautuvan näkemyksen, jossa evoluutiota tutkittaessa painotetaan liikaa yksilöiden geneettisten muutosten mekanismeja eikä kiinnitetä huomiota fenotyyppien tai yksilöiden käytöksen vaikutuksesta kehitykseen. [5]

EP:ssa ei siis käytetä rekombinaatiota. Mutaatiossa voidaan käyttää kaikkia kohdassa 4.2.1 mainittua menetelmiä, mutta yleisin käytetty mutaatio on Gaussin mutaatio. Jokaisessa generaatiossa on μ yksilöä. Jokainen yksilö mutatoidaan ja sen hyvyysarvo lasketaan. Tämän jälkeen valitaan μ yksilöä mutatoidusta joukosta käyttäen todennäköisyyksiin perustuvia menetelmiä.

Evoluutio-ohjelmoinnissa käytetään seuraavaa algoritmia:

1. Generaatio $g = 0$.
2. Alusta populaatio C_g
3. Laske kaikkien yksilöiden hyvyysarvo
4. Kunnes $g \geq max\ generaatiot$ tai $ei\ konvergenssia$:
   - 4.1 Suorita mutaatio kaikille yksilöille
   - 4.2 Laske jälkeläisten hyvyysarvo
   - 4.3 Valitse uusi generaatio joukosta C_g
   - 4.4 $g = g + 1$

4.2.5 Yhteenveto evoluutioalgoritmien toteutuksista

Taulukossa 4.2 on tiivistettynä tärkeimmät evoluutioalgoritmien väliset erot. GA:ssa mutatoidaan bittijonoja geeneittäin eikä siinä käytetä strategiaparametrejä. Valinta perustuu GA:ssa todennäköisyyksiin. ES:ssa käytetään yleisimmin diskreettiä rekombinaatiota ja elitististä valintaa. EP:ssa ei käytetä rekombinaatiota, vaan kaikki muutokset perustuvat mutaatioon. ES:n ja EP:n mutaatiot ovat yleensä samanlaiset lukuun ottamatta ES:ssa käytettävää strategiaparametriä, jota EP:ssa käytetään harvemmin.

	GA	**EP**	**ES**
Esitys	Binaariarvo	Reaaliarvo	Reaaliarvo
Mutaatio	Vain vähän merkitystä, bittikohtainen	Pääoperaattori, yleensä Gaussin jakauma	Pääoperaattori, yleensä Gaussin jakauma
Rekombinaatio	Pääoperaattori, monipisteristeytys tai tasainen risteytys	Ei ole	Yleensä diskreetti, voi olla paikallinen tai panmiktinen
Valinta	Todennäköisyyksiin perustuva	Todennäköisyyksiin perustuva	Deterministrinen, elitisitinen
Strategia-parametri	Ei ole	Harvoin käytössä	Keskihajonta
Rajoite	Esitysmekanismi	Ei ole	Usein jokin sakkofunktio

Taulukko 4.2: Kolmen eri evoluutioalgoritmin tärkeimmät ominaisuudet. [5]

4.3 Evoluutioalgoritmit ja neuroverkot

Sekä evoluutioalgoritmit että neuroverkot perustuvat biologisiin järjestelmiin. Pääosa biologisista hermorakenteista muodostuu geneettisesti ja evoluutioalgoritmeja käytetäänkin paljon neuroverkkojen opettamiseen ja muodostamiseen [73]. Tärkeä syy evoluutioalgoritmien suosioon on niiden kyky skaalautua epälineaarisissa ja monimutkaisissa optimointiongelmissa, joissa ilmenee useita paikallisia minimejä. Evoluution kyky selviytyä paikallisista minimeistä perustuu populaation käyttöön, jolloin hakuavaruudessa voidaan etsiä ratkaisuja useammasta paikasta samanaikaisesti [10]. Lisäksi ehkä vielä tärkeämpää on se, että EA:t eivät tarvitse paljoa täsmällistä tietoa ratkaistavasta ongelmasta. Tarvitaan vain hyvyysfunktio, joka ilmaisee

tietyn ratkaisun hyvyyden. Evoluutioalgoritmien heikkous on niiden suuri laskentatehon tarve, ja muut menetelmät ovatkin huomattavasti EA:ja tehokkaampia silloin, kun niitä on mahdollista käyttää [54].

EA:ja käytetään neuroverkkojen yhteydessä kolmella tavalla. Ensinnäkin niillä voidaan määrätä neuroverkon yhteyksien painokertoimet. Tämä käsittää mukautuvan oppimisen lisäksi ohjatun oppimisen, sillä EA:ja on yhdistetty gradientteihin perustuviin opetusmenetelmiin ja evoluution avulla voidaan säätää esimerkiksi vastavirta-algoritmin käyttämää askelpituutta. Toiseksi EA:ja käytetään neuroverkon topologian rakentamiseen eli neuronien ja piilokerrosten määrän sekä näiden välisten yhteyksien määrittämiseen. Tähän ongelmaan EA:t ovat antaneet lupaavia tuloksia. Neuroverkon topologian määrittäminen on erittäin monimutkainen ongelma. Ei ole pelkästään mahdotonta muodostaa optimaalista rakennetta annettuun ongelmaan, vaan lisäksi on mahdotonta todistaa, että annettu neuroverkon rakenne olisi optimaalinen. Kolmas neuroverkkojen sovellusalue, jossa EA:ja käytetään, on opetusdatan valinta ja neuroverkkojen ulostulojen tulkinta. [73]

Evoluutioalgoritmeja käytettäessä on määriteltävä ongelmasta seuraavat asiat:

- *esitys:* EA:ssa yksilöt esitetään jossakin geneettisessä muodossa. Yleensä esitysmuotona on jono reaaliarvoja tai kokonaislukuja. Tämän lisäksi on muodostettava funktio, jolla yksilöiden geneettisestä muodosta saadaan muodostettua ratkaisuyritteet eli fenotyypit.

- *suorituskyky:* on muodostettava funktio, jolla mitataan kunkin yksilön suorituskyky annettuun ongelmaan.

- *jälkeläisten tuottaminen:* on määritettävä operaatiot, joita käytetään (esimerkiksi risteytys tai mutaatio) uusien yksilöiden muodostamiseen yhdestä tai useammasta vanhemmasta. Usein näihin funktioihin on myös lisättävä apufunktioita, joilla varmistetaan tuotetun jälkeläisen järkevyys.

Neuroverkkojen tapauksessa suoraviivaisin esitysmuoto yhteyksien painokertoimille saadaan muodostamalla niistä merkkijono. Tietojen sijoittelussa merkkijonoon on huomioitava, että risteytys todennäköisesti erottaa toisistaan ne tiedot, jotka ovat merkkijonossa kaukana toisistaan. Yleensä neuronin sisääntulo- ja ulostulokertoimet sijoitetaan vierekkäin. On myös mahdollista muodostaa merkkijonoon lohkoja, jotka muodostuvat aina yhdestä toiminnallisesta yksiköstä, ja niitä käsitellään geneettisillä operaatioilla, niin kuin ne olisivat yksi geeni. Tärkeämpi valinta tehdään kuitenkin reaaliarvojen ja kokonaislukujen (käytännössä binäärilukujen) välillä. Tänä päivänä reaaliarvoesitykset ovat dominoivassa asemassa johtuen

binääriesityksen muodostamista erittäin pitkistä merkkijonoista. Jos evoluutioalgoritmeja käytetään lisäksi neuroverkon rakenteen muodostamiseen, korostuu esitystavan valinta entisestään. Karkeasti yleistettynä voidaan esitystapa valita kahdesta vaihtoehdosta: tarkasta matalan tason esityksestä (engl. *low-level encoding*) tai yleistetymmästä korkean tason esityksestä (engl. *high-level encoding*). Matalan tason esityksissä ilmenee tarkasti kunkin yhteyden painokertoimet, kun taas korkean tason esitys on enemmänkin suuntaa antava ja siinä on niputettu useita yhteyksiä ja neuroneja ryhmiin. Esitystavan valinnassa on tehtävä kompromisseja esitystarkkuuden ja laskennan nopeuden välillä. Tarkempi esitys tuottaa myös tarkempia tuloksia mutta sen vaatima suuri muistinkulutus hidastaa laskentaa ja huonontaa skaalautuvuutta. Matalan tason esityksiä käytetään vain pienissä neuroverkoissa. Korkean tason esityksiä, joissa käsitellään neuroverkon toiminnallisia kokonaisuuksia yhtenä yksikkönä, perustellaan skaalautuvuuden lisäksi usein myös niiden yhdenmukaisella käyttäytymisellä sekä samankaltaisuudella biologisiin esikuviinsa. [10]

Suorituskyvyn mittaaminen on luonnollisesti helpompaa, jos käytössä on tavoitearvot, joihin saatuja tuloksia verrataan. Tällöin luonnollinen valinta on muodostaa virhefunktiot, joita minimoidaan. Toinen vaihtoehto on laskea oikeiden vastausten prosentuaalista määrää eli neuroverkon luokitustarkkuutta. Tällainen kustannusfunktio ei ole derivoituva, mutta koska evoluutioalgoritmit eivät vaadi kustannusfunktiolta gradienttilaskentaa, tällaiset luokitteluun perustuvat menetelmät ovat mahdollisia. Yksilöiden suorituskykyjen eroja on kuitenkin helpompi käsitellä, jos käytetään jatkuvia funktioita. Mukautuvan oppimisen tapauksessa on käytännössä tehtävä päätös, milloin valittu kustannusfunktio on saavuttanut halutun käyttäytymisen. Yleensä valitaan etukäteen tietty määrä generaatioita, mihin asti evoluutio etenee tai vaihtoehtoisesti tutkitaan, milloin kustannusfunktion konvergenssi on pienentynyt tietyn raja-arvon alapuolelle eikä evoluutio enää muuta neuroverkkoa ratkaisevasti. [10]

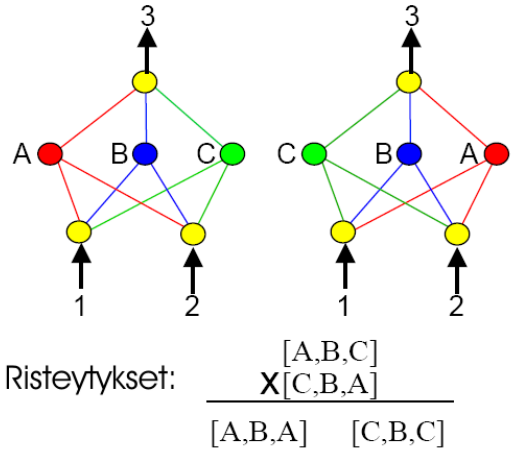
Jälkeläisten tuottamisessa on valittava, millä tasolla evoluutio-operaatiot toimivat neuroverkossa. Neuroverkon yhteyksien painokertoimien tapauksessa valinta on käytännössä tehtävä sen välillä, että operoidaanko yhteyksiä yksi kerrallaan vai niputetaanko esimerkiksi kaikki sisääntuloyhteydet tai ulostuloyhteydet yhdeksi operoitavaksi yksiköksi. Aiheesta on tehty monia tutkimuksia, mutta yleiskäyttöistä ohjetta ei ole esitetty. [10]

Tärkeämpi valinta jälkeläisten tuottamisessa vaikuttaisi olevankin se, käytetäänkö operaattoreina rekombinaatiota, mutaatiota vai molempia. Rekombinaatio muodostaa neuroverkkojen tapauksessa vakavan ongelman. Tämä ongelma on permutaatio-ongelma (engl. *permutation problem, competing conventions problem*), joka on

vastaava ilmiö luvussa 4.1 mainitun pleiotropian kanssa. Ongelmaa on syytä pitää esillä sen takia, että geneettiset algoritmit ovat selvästi suosituin evoluutioalgoritmi ja näin ollen ne ovat vahvasti mukana myös neuroverkkojen tutkimisessa. Geneettisten algoritmien toiminta perustuu vahvasti rekombinaatiolle. Permutaatio-ongelma tarkoittaa sitä, että neuroverkon painokertoimien määrittelyssä on mahdollista tuottaa sama tulos useammalla kuin yhdellä neuroverkolla. Tällaisissa tapauksissa on todennäköisempää, että jälkeläisten tuottamisessa rekombinaatio tekee enemmän vahinkoa kuin edistystä. Esimerkiksi kuvan 4.6 kaltaisilla neuroverkoilla, joissa on kolme neuronia (A, B ja C) yhdellä piilokerroksella, voidaan tuottaa samanlainen vastaus $3! = 6$ eri permutaatiolla. Jos tällaisia permutaatioita risteytetään keskenään, on mahdollista, että informaatiota menetetään. Esimerkiksi risteyttämällä $[A, B, C]$ ja $[C, B, A]$, saatetaan jälkeläiseksi saada $[C, B, C]$ ja menettää yksi kolmasosan siitä informaatiosta, joka oli molemmilla vanhemmilla. Yleisessä tapauksessa n:lle piilokerroksen solmulle on olemassa $n!$ tapausta, jotka tuottavat keskenään saman tuloksen [68]. Jos tähän vielä lisätään se mahdollisuus, että aktivaatiofunktio on pariton ja muutetaan piiloneuronien välisiä yhteyksiä vaihtamalla vain niiden painokertoimien etumerkit, ei muutoksella ole vaikutusta neuroverkon kokonaistoimintaan. Piilokerroksen neuroneja ollessa n kappaletta on mahdollista tuottaa $\sum_{i=0}^{n} \binom{n}{i} = 2^n$ erilaista verkkoa, jotka tuottavat keskenään saman tuloksen [10].

Ei ilmeisesti ole vielä tarkkaan määritelty, kuinka paljon permutaatio-ongelma heikentää GA:n toimintaa. Teoriassa on mahdollista, että hakuavaruus kasvaa eksponentiaalisesti ($n!2^n$) piilokerrosten neuronien lukumäärän funktiona. Todennäköisesti vaikutus riippuu lähinnä neuroverkon koosta ja populaation koosta. Ongelmaa on kuitenkin syytä välttää aina, kun se on mahdollista. [10]

Yksinkertaisin keino välttää permutaatio-ongelma on olla käyttämättä rekombinaatiota, kuten esimerkiksi EP:ssa tehdään. Myös muita keinoja on tutkittu laajalti. Pienempien populaatioiden ja aggressiivisten valinta- ja mutaatiomenetelmien käyttö pienentää riskiä laskea kilpailevia samanlaisia tapauksia yhtäaikaa. Useassa tutkimuksessa pyritään jaottelemaan piilokerrosten neuroneja ryhmiin jollain perusteella ja ohjata tai muuttaa risteytysoperaatiota tämän tiedon avulla. Radcliffe jaotteli neuroneja niiden yhteyksien muodostaman kuvion perusteella. Montana ja Davis suorittivat neuroneille testisyötteitä, joiden perusteella samankaltaiset neuronit oli mahdollista havaita. Lisäksi on esitelty paljon muunlaisiakin jaotteluperusteita neuronien toiminnan tai piilokerroksen rakenteen perusteella, kuten esimerkiksi Garcia-Pedrajas et al. tutkimuksessa [24] on tehty. Mitään yleispätevää ratkaisua risteytyksen aiheuttamiin ongelmiin ei ole kuitenkaan esitetty. [10]



Kuva 4.6: Permutaatio-ongelma: Kaksi neuroverkkoa, jotka laskevat samaa funktiota vaikka niiden piilokerroksen neuronit ovat eri järjestyksessä. Tällaisten neuroverkkojen risteyttäminen aiheuttaa helposti vain vahinkoa. Esimerkissä on kaksi risteytystä, jotka ovat menettäneet yhden kolmesta pääkomponentista vanhempiinsa nähden. [68]

5 P2PRealm-simulaattori ja NeuroTopologia

Tässä luvussa luodaan lyhyt katsaus olemassa oleviin vertaisverkkosimulaattoreihin ja kerrotaan, miksi ollaan päädytty käyttämään Cheese Factory -projektin [13] omaa P2PRealm-simulaattoria [37]. Lisäksi esitellään tämän tutkimuksen yhteydessä simulaattoriin lisätty topologian hallinta -algoritmi NeuroTopologia. Luvun lopuksi kerrotaan tarkemmin NeuroTopologian opetuksesta P2PRealm-simulaattorissa.

5.1 Vertaisverkkosimulaattoreista yleisesti

Vertaisverkkoalgoritmien tutkimiseen on ainakin kolme eri lähestymistapaa: ryömijät (engl. *crawlers*), emulaattorit (engl. *emulators*) ja simulaattorit (engl. *simulators*). Ryömijä on vertaisverkon solmun ilmentymä, joka kerää sen läpi kulkevaa dataa ja tutkii näin ollen vertaisverkkoa paikallisesti. Lisäämällä ryömijöiden lukumäärää voidaan luoda yleinen kuva vertaisverkosta. Ryömijät eivät kuitenkaan muodosta globaalia kuvausta vertaisverkosta, sillä niitä solmuja, jotka eivät ole yhteydessä ryömijöihin, ei oteta huomioon. Emulaattoreilla voidaan luoda globaali kuva vertaisverkosta, sillä niissä vertaisverkon solmun ilmentymillä rakennetaan kokonainen vertaisverkko. Käyttämällä useampaa emulaattoria voidaan tutkia suuriakin vertaisverkkoja. Emulaattoreiden rajoitteena on kuitenkin niiden hitaus, sillä solmujen välisen liikenteen kontrollointiin käytetään standardeja verkkoprotokollia kuten esimerkiksi TCP:tä. Simulaattoreissa käytetään abstraktoitua toteutusta, joka toimii muuten samalla tavoin kuin emulaattorit, mutta viestien välitykseen käytetään paikallisia tietorakenteita. Simulaattorit nopeuttavat vertaisverkkojen tutkimista huomattavasti ja sopivat laskennallisesti vaativiin tehtäviin. Niiden huono puoli on se, että abstraktiosta johtuen niiden tulokset saattavat poiketa niistä tuloksista, joita saataisiin oikeassa vertaisverkossa. [37]

Vertaisverkkoalgoritmien tuottaminen neuroverkkojen avulla on erittäin vaativa tehtävä laskennallisesti. Esimerkiksi hakualgoritmin tuottaminen melko pieneen vertaisverkkoon vie yhdeltä koneelta noin viikon [72]. Yksi tässä tutkimuksessa käytettävän simulaattorin tärkeimmistä vaatimuksia onkin sen nopeus laskennallisesti vaativissa tehtävissä.

Vertaisverkkojen tutkimiseen on saatavilla lukuisia simulaattoreja. Suurin osa

niistä on kuitenkin laskennallisesti tehottomia eikä yhdessäkään ole toiminnallisuuksia neuroverkoille. Koska nopeus on ratkaiseva tekijä tässä tutkimuksessa, selvää on että vertaisverkon mallia on yksinkertaistettava eli abstraktoitava. Simulaattorit voidaan jakaa kahteen pääryhmään niiden viestinvälitysmekanismin perusteella. Pakettitason simulaattoreissa (engl. *packet-level simulators*) käytetään tarkkaa kuvausta pakettien otsikkotiedoista ja data-osiosta, kun taas viestitason simulaattoreissa (engl. *message-level simulators*) tutkitaan vain pakettien kokoa ja määrää. Pakettitason simulointi on yleensä toivottavaa tarkempien tulosten saamiseksi, mutta käytännössä se on usein liian vaativaa käytettävissä olevien laskentaresurssien kannalta. Nopeuden lisäksi tämän tutkimuksen simuloinnilta vaaditaan tulosten tilastointia sekä vertaisverkkojen visualisointia. Näiden vaatimusten valossa luodaan seuraavassa yleiskuva tärkeimpiin olemassa oleviin vertaisverkkosimulaattoreihin.

NS-2 [49] on yleisimmin käytetty verkkosimulaattori. Se on olio-pohjainen, OSI-mallia mukaileva ja erillisiin tapahtumiin perustuva simulaattori (engl. *discrete-event simulator*). NS-2 sopii hyvin pakettikytkentäisten verkkojen ja pienten verkkojen simulointiin. Erillinen tutkimusryhmä The Parallel and Distributed Simulation Research Group (PADS) [51] on lisännyt NS-2 -simulaattoriin lisäosan, joka mahdollistaa rinnakkaisen laskennan useammilla koneilla. NS-2 käyttää erittäin tarkkaa kuvausta verkkoliikenteestä, joten se ei skaalaudu kovin hyvin ja on liian hidas tätä tutkimusta varten. Myöskään NS-2 -simulaattoria ei ole kovin helppo laajentaa sen monimutkaisen rakenteen vuoksi [46].

PLP2P (Packet-Level Peer-to-Peer Simulator) [29] on kehys pakettitason vertaisverkkosimulaattoreille, kuten esimerkiksi NS-2. PLP2P tarkentaa simuloinnin erittäin tarkaksi matalan tason simuloinniksi. Tämä tehdään käyttämällä välittäjiä (engl. *wrappers*), jotka muuttavat P2P-tason tapahtumat tarkemmiksi alemman kerroksen paketeiksi. Simulaattorin kehittäjät vakuuttavat, että liika abstraktointi väärentää simulointituloksia huomattavasti ja alemman kerroksen tapahtumat on otettava huomioon. PLP2P-simulaattorissa skaalautuvuusongelmat ratkaistaan rinnakkaislaskennan avulla. PLP2P-simulaattorissa ei kuitenkaan ole valmiuksia neuroverkkojen käyttöön, joten se ei sovellu tähän tutkimukseen.

QueryCycle [63] on erikoistunut simuloimaan tiedostojen jakamista. Siinä on tarkat mallit muun muassa sisältöjen jakelusta, kyselyaktiivisuudesta ja lataamiskäyttäytymisistä. Sisältöjen jakelu perustuu malliin, jossa tiedostot kuuluvat johonkin kategoriaan ja tämä kategoria määräytyy sen tiedostojen suosion perusteella. Simulointi tapahtuu kierroksittain eli se on aikajaoteltu, jolloin samaa tietoa ei samaanaikaan käytetä ja lasketa. Kyselyt käsitellään jonossa FIFO-periaattella.

3LS [69] on avoimen lähdekoodin simulaattori, jonka suunnittelussa on paino-

tettu sen helppoa laajennettavuutta ja käytettävyyttä. Systeemi on jaettu kolmeen eri tasoon: verkkotaso käyttää kaksiulotteista matriisia tallentamaan solmujen etäisyyksiä, protokollataso määrittelee kulloinkin simuloitavan verkkoprotokollan ja käyttäjätaso on käyttäjän syötteille tarkoitettu rajapinta. 3LS käyttää suuren osan muistiresurssista verkon tapahtumien tallentamiseen graafisessa käyttöliittymässä suoritettavaa visualisointia varten. Tämä rajoittaa käytettävän vertaisverkon koon enintään tuhanteen solmuun.

PeerSimin [52] kehityksessä on painotettu skaalautuvuutta ja dynaamisen verkon tukea. PeerSim on Java-pohjainen simulaattori, jossa voidaan käyttää kahta simulointitapaa: ensimmäinen on aikajaoteltu (engl. *cycle-based engine*) ja toinen tapahtumapohjainen (engl. *event-driven engine*). Aikajaoteltu simulaatio on skaalautuva, muttei kovin tarkka. Suurten verkkojen käsittelyssä on esimerkiksi luovuttu viestinvälitysprotokollan yksityiskohtaisimmista ominaisuuksista. Tapahtumapohjainen simulaatio on tarkka, muttei skaalautuva. Aikapohjaisen simulaation abstraktion tuomat hyödyt käytetään PeerSimissä skaalautuvuuden parantamiseen. Tässä tutkimuksessa käytettävän simulaattorin on ohjattava kaikki abstraktiosta saatava hyöty laskentatehon parantamiseen.

NeuroGrid [34] kehitettiin alunperin tuottamaan vertailukelpoisia tuloksia kolmesta eri verkosta: FreeNetistä [14], Gnutellasta [50] ja NeuroGrid-verkosta [34]. Simulaattori on yksisäikeinen, Java-pohjainen ja tapahtumiin perustuva eli diskreetti. NeuroGridiin on lisätty paljon uusia protokollia, kuten esimerkiksi nimipalvelu DNS (engl. *Domain Name Service*) ja hajautettu sähköpostiprotokolla. NeuroGridissä käytetään konfiguraatiotiedostoja, joilla voidaan määritellä ajettavan simulaation parametrit. Tämä on haluttava ominaisuus myös tässä tutkimuksessa käytettävältä simulaattorilta. NeuroGrid olisi lupaava simulaattori myös tähän tutkimukseen, mutta se on yksisäikeinen ja siinä ei ole mahdollisuutta rinnakkaiseen laskentaan.

GPS [76] (General Peer-to-Peer Simulator) pyrkii vastaamaan yleiseen pyyntöön yleiskäytettävästä ja helposti laajennettavasta vertaisverkkosimulaattorikehyksestä, jolla voitaisiin tutkia P2P-verkkoja tehokkaasti ja tarkasti. Tehokkuus saavutetaan viestitason tarkkuudella toimivalla simulaatiolla. Tarkkuus puolestaan saavutetaan seuraamalla vertaisverkon rakennetta ja käyttämällä makroskooppisia matemaattisia malleja tarkkaan arviointiin viestien käyttäytymisestä (esimerkiksi TCP). GPS:ssä käytetään myös malleja tiedostojen lataamiseen, mikä useimmista vertaisverkkosimulaattoreista puuttuu. GPS:ää voidaan laajentaa simuloimaan mitä tahansa P2P-protokollaa ja siinä on rajapinnat graafiselle käyttöliittymälle, verkon visualisoinnille sekä tuki topologioiden luomiseen. GPS on vielä kehityksensä alkuvaiheissa ja tiedot sen skaalautuvuudesta, käytettävyydestä ja suorituskyvystä ovat niuk-

koja. Simulaattori on vielä yksisäikeinen, mutta tekijöidensä mukaan siihen ollaan lisäämässä monisäikeisyys myöhemmin.

Yhteenveto yleisimmistä vertaisverkkosimulaattoreista on taulukossa 5.1. Päällysverkko reitittimillä -sarake kertoo, käytetäänkö simulaattorissa päällysverkon lisäksi sen alla olevan fyysisen reititinverkon mallintamista. Tämän lyhyen vertaisverkkosimulaattoreiden kirjallisuuskatsauksen jälkeen on melko selvää, että kyseinen tieteenala on erittäin pirstaloitunut ja olisi selvä tarve yleiskäyttöiselle simulaattorille [33]. Useimmat tutkimusryhmät luovat oman simulaattorinsa, joka on räätälöity juuri heidän tarvitsemien ominaisuuksien perusteella.

	Viestinvälityksen mallinnustarkkuus	**Rinnakkaislaskenta**	**Skaalautuvuus**	**Päällysverkko reitittimillä**	**Dynaaminen verkko**	**Ohjelmointikieli**
NS-2	Paketti	Kyllä	Erittäin alhainen	Kyllä	Ei	C++
PLP2P	Paketti	Kyllä	Alhainen	-	-	C++
QueryCycle	Viesti	Ei	?	Kyllä	Kyllä	Java
3LS	Viesti?	Ei	Erittäin alhainen (<1000 solmua)	Kyllä	?	Java
PeerSim	Viesti	Ei	Erittäin korkea (10^6 solmua)	Kyllä	Kyllä	Java
NeuroGrid	Viesti	Ei	Korkea (300 000)	Ei	Kyllä	Java
GPS	Viesti	Ei	?	Ei	Kyllä	Java
P2PRealm	Viesti	Kyllä	Keskikoko (100 000)	Ei	Kyllä	Java

Taulukko 5.1: Puhtaille järjestämättömille vertaisverkoille sopivien simulaattoreiden ominaisuuksia [37]

Vaikka yleiskäyttöinen simulaattori olisikin ihannetilanne, niin neuroverkkojen ja vertaisverkkojen yhdistäminen on vielä sen verran spesifinen ongelma, että päädyttiin luomaan uusi simulaattori. Neuroverkkojen opettaminen on erittäin vaativaa laskennallisesti ja vaatii simulaattorilta yksinkertaisia vertaisverkkomalleja ja rinnakkaista laskentaa. Topologian hallinnan tutkimukseen simulaattorin rinnakkaislaskentaa ei vielä ole toteutettu, mutta se kuuluu ohjelman kehitykseen tulevaisuudessa. Oman simulaattorin kehittämiseen päädyttiin myös sillä perusteella, että

olemassa olevissa vertaisverkkosimulaattoreissa ei ole valmiita toteutuksia neuroverkkojen opetukseen [37].

5.2 P2PRealm

P2PRealm on Java-pohjainen vertaisverkkosimulaattori, joka on suunniteltu optimoimaan vertaisverkoissa käytettäviä neuroverkkoja. Simulaattorilla voidaan luoda erilaisia vertaisverkkoskenaarioita ja vaatimuksia, jotka määrittelevät resurssien hakualgoritmien ja topologian hallinta -algoritmin toimintaa. Skenaario voi olla esimerkiksi Gnutella-verkon topologia, resurssijakauma ja kysely-ympäristö. Simulaattorille annetaan vaatimus tuottaa algoritmi, jolla löydetään esimerkiksi 150 resurssia niin pienellä pakettimäärällä kuin mahdollista. Lopputuloksena saadaan resurssin hakualgoritmi, joka toimii juuri tässä skenaariossa. Resurssin hakualgoritmien kehittämisestä P2PRealm-simulaattorilla löytyy enemmän Vapan et al. tutkimuksesta [72]. Simulaattoriin toteutettuja hakualgoritmeja, kuten esimerkiksi luvussa 2.4 esiteltyjä BFS-, satunnaiskävelijä- ja HDS-algoritmeja, voidaan käyttää mittareina, kun vertaillaan eri skenaarioita ja vaatimuksia.

Simulaattori on jaettu neljään osaan: P2P-verkkoon, P2P-algoritmeihin, neuroverkon optimointiin sekä syötön ja tulostuksen rajapintaan. P2P-verkko pitää sisällään vertaisverkon topologian sekä resurssien jakautumisen. P2P-algoritmit on kokoelma resurssien hakuun ja topologian hallintaan tarkoitettuja algoritmeja. Neuroverkkojen optimoinnin tehtävänä on huolehtia neuroverkon rakenteesta ja kulloinkin käytettävästä optimointimenetelmästä kuten esimerkiksi Gaussin mutaatiosta. Syötön ja tulostuksen rajapinnan avulla simulaattorille saadaan annettua kulloisenkin simuloinnin konfiguraatiotiedot sekä saadaan ulostulona tilastot neuroverkon opetuksen kulusta ja lopulliset tulokset kuten esimerkiksi parhaan neuroverkon tuottama topologia. P2PRealmin syötteet ja tulosteet on eritelty tarkemmin luvussa 5.4.1.

5.3 NeuroTopologia

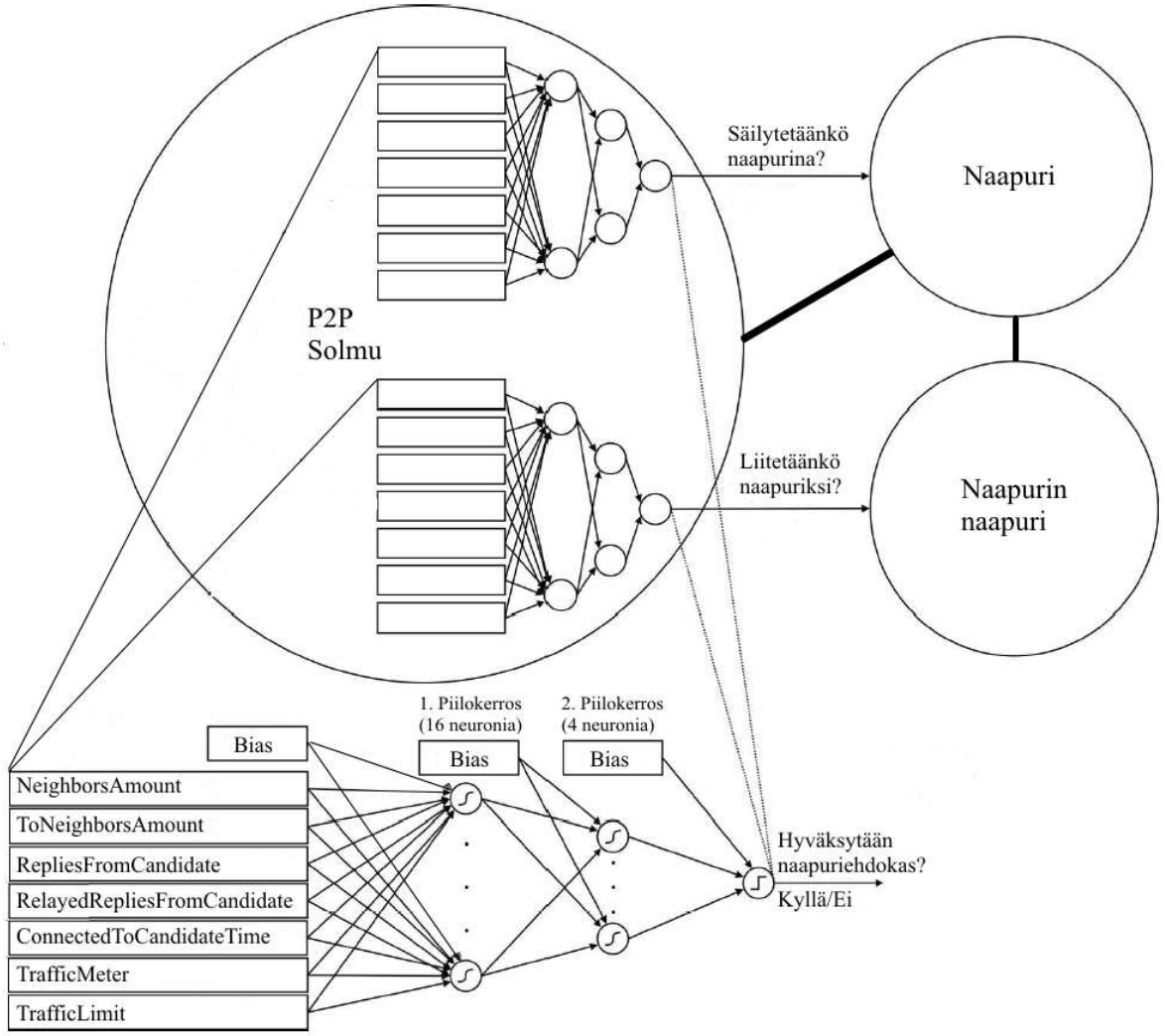
Tässä osassa esitellään P2PRealm-simulaattoriin lisätty topologian hallinta- algoritmi NeuroTopologia. Ensin käydään läpi itse algoritmi ja sen jälkeen esitellään tarkemmin algoritmin käyttämä neuroverkko.

5.3.1 NeuroTopologia-algoritmi

NeuroTopologian tehtävänä on resurssihakujen lomassa muuttaa vertaisverkon topologiaa sen tiedon avulla, mitä solmut ovat resurssihakujen yhteydessä keränneet, ja näin ollen muuttaa hakualgoritmin toimintaympäristöä. NeuroTopologian käytöstä on hyötyä, jos se muuttaa topologiaa niin, että solmut löytävät haluamiaan resursseja pienemmillä liikennemäärillä kuin staattisessa verkossa. Myös NeuroTopologian tuottama lisäliikenne on otettava huomioon. NeuroTopologia tuottaa liikennettä, kun se arvioi naapuriksi sopivia solmuja sekä muodostaa uusia yhteyksiä.

NeuroTopologia käyttää päätöksentekoon paikallista tietoa tietämistään solmuista. NeuroTopologia-algoritmi suoritetaan kaikille solmuille aina, kun tietty määrä resurssihakuja on tehty. Algoritmin toiminta on esitetty kuvassa 5.1.
Algoritmin parametrit ja muuttujat ovat:

- Yhtenäinen graafi $G = (V_g, E_g)$, joka sisältää vertaisverkon topologian. $V_g = \{v_1, v_2, \ldots, v_n\}$ on G:n solmut ja E_g on joukko suuntamattomia yhteyksiä solmujen välillä $(v_i, v_j), v_i, v_j \in V_g, i \neq j$.
- Solmun u naapurisolmu on $s \iff \exists(s,u) \in E_g \vee \exists(u,s) \in E_g$.
- Solmun s naapurisolmu w eli solmun u naapurin naapuri $\iff (\exists(s,w) \in E_g \wedge (\exists(s,u) \in E_g \vee \exists(u,s) \in E_g)) \vee (\exists(w,s) \in E_g \wedge (\exists(s,u) \in E_g \vee \exists(u,s) \in E_g))$.
- $L \subseteq V_g$ ovat solmut, joilta on saatu vastauksia.
- $t =$ topologiapakettien määrä eli topologiakyselyiden tuottama liikennemäärä.
- $t_a =$ topologiavastausten määrä eli uusien yhteyksien muodostamisessa syntyvä liikennemäärä.



Kuva 5.1: Vertaisverkon solmun naapurit määrätään neuroverkolla, joka saa tietoa naapuriehdokkaista sisääntuloparametreissaan.

Vertaisverkon solmu u valitsee naapurinsa seuraavan algoritmin mukaisesti:

1. Kokoa naapuriehdokkaiden joukko N naapureista, naapureiden naapureista sekä niistä solmuista, joilta on saatu resurssikyselyjen yhteydessä vastauksia. $N = \{s_1, s_2, \ldots, s_k, w_1, w_2, \ldots, w_h, L\}$, missä $k \leq n - 1$ ja $h \leq n - 2$.
2. Kaikille $b_i \in N$:
   - 2.1 Aseta neuroverkon sisääntuloparametrit naapuriehdokkaan b_i tietojen mukaiseksi.
   - 2.2 Laske neuroverkon ulostulo ehdokkaalle b_i.
   - 2.3 Jos ulostulo on 1:
     - 2.3.1 Solmu u pyytää solmua b_i naapuriksi. Lisää topologiapakettien määrää yhdellä $t = t + 1$.
     - 2.3.2 Aseta neuroverkon sisääntuloparametrit solmun u tietojen mukaiseksi.
     - 2.3.3 Laske neuroverkon ulostulo solmulle u.
     - 2.3.4 Jos ulostulo on 1, merkitse yhteys solmujen u ja b_i välillä vahvistetuksi ja lisää topologiavastausten määrää yhdellä $t_a = t_a + 1$.
     - 2.3.5 Jos ulostulo on 0, solmu b_i vastaa solmun u kyselyyn kieltävästi. Lisää topologiapakettien määrää yhdellä $t = t + 1$. Lisäksi, jos b_i on solmun u naapuri s, poista yhteys solmuun u.
   - 2.4 Jos ulostulo on 0 ja b_i on nykyinen naapuri s, poista yhteys solmuun b_i.
3. Mene kohtaan 2.

Algoritmin suorittaminen loppuu, kun kaikki naapuriehdokkaat on käyty läpi. Uutta naapuriyhteyttä ei siis muodosteta yksipuolisesti, vaan myös haluttu naapuriehdokas arvioi samalla neuroverkolla, haluaako se muodostaa yhteyden alkuperäiseen kyselijään. On myös mahdollista, että naapuriehdokas, joka on myös nykyinen naapuri, tulkitsee kyselijän huonoksi naapuriksi ja katkaisee yhteyden.

5.3.2 Neuroverkon sisääntuloparametrit

Seuraavassa luetellaan NeuroTopologiassa käytettävät neuroverkon sisääntuloparametrit. Kandidaattisolmusta puhuttaessa tarkoitetaan sitä solmua, jota ollaan harkitsemassa solmun naapuriksi.

- *Bias* = 1 on bias termi. Bias on kynnysparametri, jolla säädellään neuronin aktivoitumistasoa.
- *CurrentNeighborsAmount* on solmun naapurien lukumäärä.
- *ToNeighborsAmount* on kandidaattisolmun naapurien lukumäärä.
- *RepliesFromCandidate* on resurssikyselyihin kandidaattisolmulta saatujen vastausten lukumäärä.

- *RelayedRepliesFromCandidate* on resurssikyselyihin kandidaattisolmun välittämien vastausten lukumäärä.
- *ConnectedToCandidateTime* on laskuri, joka simuloi, kuinka kauan kandidaattisolmu on ollut solmun naapurina. Laskuri kasvaa aina, kun suoritetaan topologian muutoskierros. Esimerkiksi jos tämä parametri on kaksi, niin kandidaatti on säilynyt nykyisen solmun naapurina kahdesta topologian muutoskierroksesta huolimatta.
- *TrafficMeter*-parametrin avulla mitataan kandidaattisolmun kaistanleveyden käyttöä. *TrafficMeter* on käytännössä laskuri, jolla pidetään kirjaa solmun läpi kulkeneiden resurssikyselypakettien lukumäärästä.
- *TrafficLimit*-parametrilla simuloidaan kandidaattisolmun kaistanleveyttä. *TrafficMeter*-arvon ylittäessä *TrafficLimit*-arvon solmu ei enää vastaa resurssikyselyihin.

Sisääntuloparametrit on skaalattava välille [0,1], jotta mikään tietty parametri ei pääsisi vallitsevaan asemaan. Esimerkiksi kandidaattisolmun naapurien lukumäärä ja siltä saatujen vastausten määrä ovat yleensä eri kokoluokkaa.

RepliesFromCandidate, RelayedRepliesFromCandidate, ConnectedToCandidateTime ja *TrafficMeter* voivat olla arvoiltaan nolla ja ne skaalataan funktiolla 5.1.

$$f(x) = \frac{1}{x+1} \tag{5.1}$$

ToNeighborsAmount, CurrentNeighborsAmount ja *TrafficLimit* skaalataan funktiolla 5.2.

$$f(x) = \frac{1}{x} \tag{5.2}$$

5.3.3 Neuroverkon rakenne

NeuroTopologian käyttämä neuroverkko on kolmikerrosneuroverkko eli siinä on kaksi piilokerrosta. Bias-solmut mukaan luettuna ensimmäisessä piilokerroksessa on 16 solmua ja toisessa 4. Piilokerrosten aktivaatiofunktiona käytetään kaavan 3.4 hyperbolista tangenttia ja ulostulokerroksen aktivaatiofunktiona käytetään kaavan 3.2 kynnysfunktiota.

Neuroverkon ulostulo saadaan yhdistämällä edellä olevat funktiot ja neuronien ulostuloarvot seuraavalla kaavalla:

$$O = s(1 + \sum_{k=1}^{3} w_{3k} t(1 + \sum_{j=1}^{15} w_{2j} t(1 + \sum_{i=1}^{7} w_{1i} f(I_i)))), \tag{5.3}$$

missä I_i on sisääntulon i arvo, w_{xy} on neuroverkon solmujen x ja y välisen yhteyden painokerroin, s on ulostulokerroksen aktivaatiofunktio ja t on piilokerrosten aktivaatiofunktio.

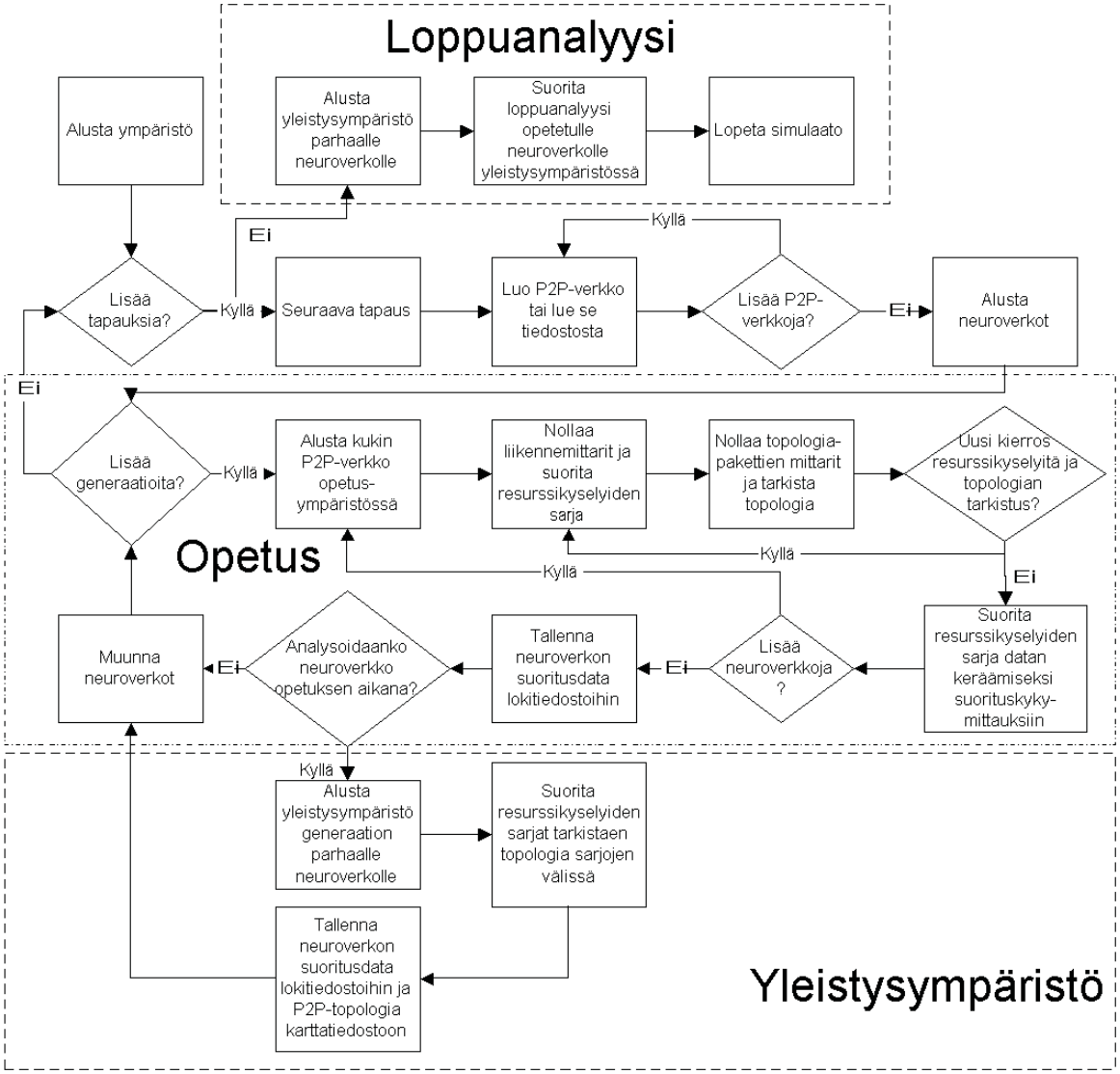
5.4 NeuroTopologian opettaminen P2PRealmissa

Ennen kuin NeuroTopologia-algoritmia voidaan käyttää vertaisverkon topologian hallintaan, neuroverkon painokertoimet on optimoitava. Seuraavassa kuvataan neuroverkon painokertoimien määrääminen evoluutioalgoritmeilla P2PRealm-simulaattorissa, mikä on myös kuvattu vuokaaviona kuvassa 5.2. Yksi simulaattorin ajo voi sisältää useita tapauksia. Jokaisella tapauksella on omat, luvussa 5.4.1 esitellyt muuttujat. Tuloksena saadaan kunkin tapauksen ympäristöön optimoitu neuroverkko. Usean tapauksen käytöstä saavutettava hyöty ilmenee silloin, kun suoritetaan niin sanottua parametrien lakaisua (engl. *parameter sweep*), jossa yhdelle parametrille annetaan useita arvoja. Tällöin simulaattoria ei tarvitse käynnistää manuaalisesti joka kerta, kun muutetaan jotain parametria. Esimerkiksi voidaan asettaa neuroverkkojen määrä juoksemaan arvosta 18 arvoon 30 kuuden välein, jolloin saadaan kolme tapausta (neuroverkkojen määrillä 18, 24 ja 30). Lisäksi tätä voidaan myöhemmin hyödyntää käytettäessä hajautettua laskentaa [38], jolloin tapaukset hajautetaan usean tietokoneen laskettavaksi.

Yhden tapauksen laskenta voidaan jakaa kolmeen eri osa-alueeseen:

- Neuroverkkojen opetus
- Opetuksen etenemisen analysointi testaamalla opetusjoukossa kulloinkin parhaiten suoriutunutta neuroverkkoa yleistysympäristössä
- Opetuksen jälkeen parhaan neuroverkon suorituskyvyn analysointi yleistysjoukossa ja resurssikyselyiden reittien tallennus.

Tapauksen alussa vertaisverkot, neuroverkot ja muut parametrit alustetaan, jonka jälkeen edetään opetusvaiheeseen. Jokaisen generaation yhteydessä opetetaan useampaa neuroverkkoa. Alussa neuroverkkojen painokertoimet on määrätty satunnaisesti välille [-0.2, 0.2]. Kunkin neuroverkon opetus aloitetaan suorittamalla ensin resurssikyselyjen sarja ja käyttämällä näistä kyselyistä saatua tietoa neuroverkkojen sisääntuloarvoissa. Tämän jälkeen neuroverkon ohjauksella jokainen vertaisverkon solmu suorittaa topologiakyselyn luvussa 5.3.1 esitetyn algoritmin mukaisesti ja suoritetaan uusi resurssikyselyjen sarja. Tätä toimenpiteiden ketjua jatkamalla vertaisverkon topologia muuttuu joka kierroksella hiukan. Kun toimenpitei-



Kuva 5.2: Vuokaavio P2PRealm-simulaattorin toiminnasta NeuroTopologian opetuksessa.

den ketju on edennyt ennalta asetetun määrän kierroksia, suoritetaan vielä viimeinen resurssikyselyiden sarja, joka määrää neuroverkon hyvyysarvon resurssikyselyiden osalta. Opetusosuuden lopuksi tallennetaan neuroverkkojen suoriutumisista kertovat tiedot talteen. Neuroverkoista valitaan parhaat suoraan jatkoon niin, että puolet populaatiosta saadaan suoraan ja toinen puolisko muodostetaan mutaation avulla.

Generaatioiden välissä voidaan suorittaa tästä vaiheesta parhaiten suoriutuneen neuroverkon tarkempi analyysi yleistysympäristössä. Simulaattorissa ei käytetä luvussa 3.3.1 esiteltyä ristiinvalidointia, mutta yleistysympäristöä kylläkin. Yleistysympäristön avulla voidaan tarkkailla sitä, milloin neuroverkko erikoistuu liikaa opetusympäristöön eikä ole enää käyttökelpoinen sille aiemmin tuntemattomassa, mutta samankaltaisessa ympäristössä kuin opetusympäristö. Yleistysympäristön tulokset eivät vaikuta opetuksen lopettamiseen, vaan simulaattori suorittaa aina sen generaatiomäärän, joka konfiguraatiotiedostossa annetaan.

Ennalta asetetun generaatioiden määrän suoritettuaan simulaattori etenee tapauksen viimeiseen osaan: parhaan neuroverkon loppuanalysointiin. Tämän analyysin yhteydessä tallennetaan myös resurssikyselyiden kulkemat reitit vertaisverkossa.

Opetusvaiheessa neuroverkon painokertoimia muutetaan hieman alkuperäisestä käyttämällä normaalijakaumaan perustuvaa mutaatiota. Mutaatio etenee luvun 4.2.1 strategiaparametrin sisältävän version mukaisesti, jolloin mutaatioon tarvittavat kaavat ovat 5.4, 5.5 ja 5.6:

$$w_i'(j) = w_i(j) + \sigma_i'(j)N_j(0,1), \quad j = 1, \ldots, N_w \tag{5.4}$$

$$\sigma_i'(j) = \sigma_i(j)\exp(\tau N_j(0,1)), \quad j = 1, \ldots, N_w \tag{5.5}$$

$$\tau = \frac{1}{\sqrt{2\sqrt{N_w}}} \tag{5.6}$$

missä N_w on painokertoimien ja bias-termien lukumäärä neuroverkossa, $N_j(0,1)$ on Gaussin satunnainen muuttuja jokaiselle j, σ on itsemukautuva parametrivektori, joka määrittelee askelpituuden uuden painokertoimen löytämiseksi, τ:ta käytetään askelkokojen määrittämiseen ja $w_i'(j)$ on uusi painokerroin.

Käytetty evoluutioalgoritmi on evoluutio-ohjelmointi (EP), jossa jälkeläiset tuotetaan pelkästään mutaatiolla eikä risteytystä käytetä lainkaan. Mukana oleva itsemukautuva parametri on evoluutiostrategian (ES) piirre, joka on otettu mukaan

NeuroTopologiaan. Mutaatiossa käytetään siis Gaussin mutaatiota ja valinta suoritetaan elitistisenä.

Neuroverkon hyvyys määräytyy vertaisverkossa tapahtuvan liikenteen perusteella. Kyselyt (sekä resurssikyselyt että topologian muutokseen liittyvät kyselyt) j pisteytetään neuroverkolle h ja hyvyys saadaan laskemalla nämä pisteet F_j yhteen:

$$fitness_h = \sum_{j=1}^{n} F_j \tag{5.7}$$

Kyselyn pisteet puolestaan lasketaan seuraavilla kaavoilla:

$$F_j = \begin{cases} 0, & \text{jos } p > 300 \\ \text{Rr-p}, & \text{jos } p \leq 300 \text{ ja } Rr > p \\ \frac{1}{p-Rr}, & \text{jos } p \leq 300 \text{ ja } Rr - p < 1 \end{cases} \tag{5.8}$$

$$F_j = \begin{cases} 0, & \text{jos } p > 300 \\ \text{2Rr-(p+Tt)}, & \text{jos } p \leq 300 \text{ ja } Rr - (p + Tt) \geq 1 \text{ ja } r \geq G \\ \text{Rr+(t-p)}, & \text{jos } p \leq 300 \text{ ja } Rr - (p + Tt) \geq 1 \text{ ja } r < G \\ \frac{1}{p+Tt-Rr}, & \text{muulloin,} \end{cases} \tag{5.9}$$

missä p on resurssikyselyn pakettimäärä, r on resurssikyselyn vastausten määrä, R on vakio, jolla voidaan säätää vastausten ja lähetettyjen pakettien vaikutusta pisteytykseen, t on topologiakyselyn käyttämä pakettimäärä ja T on vakio, jolla voidaan säätää topologian muutospakettien vaikutusta pisteytyksessä. G on tavoite resurssien määrästä, joka resurssikyselylle on asetettu.

Jos vertaisverkossa ei ole suoritettu topologian muutoksia käytetään, kaavaa 5.8. Kaavassa oleva vakio 300 toimii kriteerinä sille, että yli kolmesataa pakettia kuluttava kysely luokitellaan ikuisesti jatkuvaksi kyselyksi. Jos vastauksia katsotaan olevan riittävästi suhteessa lähetettyjen pakettien määrään, niin neuroverkko saa parempia arvoja vähentämällä pakettien määrää: $F_j = Rr - p$. Jos löydettyjä resursseja ei ole riittävästi suhteessa lähetettyjen pakettien määrään, niin neuroverkko saa parempia arvoja kasvattamalla pakettien määrää: $F_j = \frac{1}{p-Rr}$.

Jos vertaisverkossa on suoritettu topologian muutoksia käytetään kaavaa 5.9. Jos päästään tavoitteeseen löydettyjen resurssien määrästä, niin palkitaan neuroverkko tuplaamalla vastauksista saatavat pisteet. Tällöin neuroverkko voi saada vielä parempia arvoja vähentämällä pakettien ja topologian muutospakettien määrää. Erityisesti topologian muutospakettien määrää kannustetaan pienentämään $F_j = 2Rr - (p + Tt)$, koska nykyisessä topologiassa on jo haluttuja ominaisuuksia. Jos

vastauksia katsotaan olevan riittävästi suhteessa lähetettyjen pakettien määrään, mukaan luettuna topologian muutospaketit, mutta kuitenkaan ei ole päästy tavoitteeseen löydettyjen resurssien suhteen, niin neuroverkko saa parempia arvoja lisäämällä topologian muutospakettien määrää: $F_j = Rr + (t - p)$. Jos löydettyjä resursseja ei ole riittävästi suhteessa lähetettyjen pakettien määrään, niin neuroverkko saa parempia arvoja kasvattamalla pakettien ja topologian muutospakettien määrää: $F_j = \frac{1}{p+Tt-Rr}$.

Simuloinnin viimeisessä osassa eli parhaan neuroverkon loppuanalyysissä hyvyysarvo lasketaan eri kaavalla kuin opetusvaiheessa. Tämä johtuu siitä, että säätelemällä kaavojen 5.8 ja 5.9 arvoja R ja T, hyvyysarvoista tulee tapauskohtaisia eivätkä ne ole vertailukelpoisia keskenään. Käyttämällä loppuanalyysissä kaavaa

$$F_j = \begin{cases} 0, & \text{jos } p > 300 \\ \text{Rr-(Tt+p)}, & \text{jos } p \leq 300 \text{ ja } Rr > p \\ \frac{1}{(Tt+p)-Rr}, & \text{jos } p \leq 300 \text{ ja } Rr - p < 1 \end{cases} \tag{5.10}$$

saadaan vertailukelpoinen hyvyysarvo eri tapausten tuottamille neuroverkoille. Kaava on muodostettu samoilla periaatteilla kuin kaava 5.8, mutta nyt paketteihin lisätään topologiamuutosten pakettimäärä sopivalla kertoimella kerrottuna.

5.4.1 P2PRealmin syötteet ja tulosteet

P2PRealm-simulaattorille annetaan seuraavat muuttujat konfiguraatiotiedostossa:

- Vertaisverkkojen topologiat lähtötilanteessa, resurssien jakautuminen solmuilla sekä solmujen kaistanleveydet
- Resurssien hakualgoritmi
- Kuinka suuri osuus kaikista mahdollisista resursseista halutaan löytää
- Resurssikyselyiden määrä kussakin generaatiossa
- Generaatioiden määrä opetuksessa, neuroverkkojen määrä ja neuroverkkojen rakenne
- Neuroverkkojen sisääntuloparametrit
- Neuroverkkojen optimointimenetelmä.

Simulaattorin ulostulona saadaan seuraavat tiedot:

- Vertaisverkon topologia ja naapurijakauma, kun parhaan neuroverkon hyvyys on parantunut sekä määritellyin väliajoin (esimerkiksi joka 100. generaatio)
- Tietoja resurssikyselyiden suorituskyvyistä ja topologian muutoksista sekä opetuksessa että yleistysympäristössä generaatioiden aikana
- Neuroverkot jokaisen generaation jälkeen
- Resurssikyselyiden reitit jokaisesta kyselijästä parhaan neuroverkon muuttamassa vertaisverkossa.

6 Tutkimustapaukset ja tulosten arviointi

Luvussa esitellään ensin neuroverkkojen opetuksessa käytetyt parametrit. Tämän jälkeen käydään läpi neuroverkkojen opetuksen eteneminen kolmessa vertaisverkkoskenaariossa, jossa kussakin käytetään eri hakualgoritmia. Korkeimman asteen haku -algoritmilla tutkitaan viittä eri tapausta muuttaen hyvyysfuntion parametrejä. Satunnaiskävelijä- ja leveyshakualgoritmien tapauksia esitellään kaksi kummastakin. Leveyshaulla käytetään testeissä kahta eri TTL-arvoa.

Luvun lopussa eri tapauksilla tuotettuja neuroverkkoja analysoidaan yleistysympäristössä eli tutkitaan, kuinka hyvin eri hakualgoritmien avulla opetetut neuroverkot soveltuivat yleistysympäristöön, sekä vertaillaan tuotettuja topologioita Grid-verkkoon, satunnaiseen vertaisverkkoon ja potenssijakautuneeseen vertaisverkkoon.

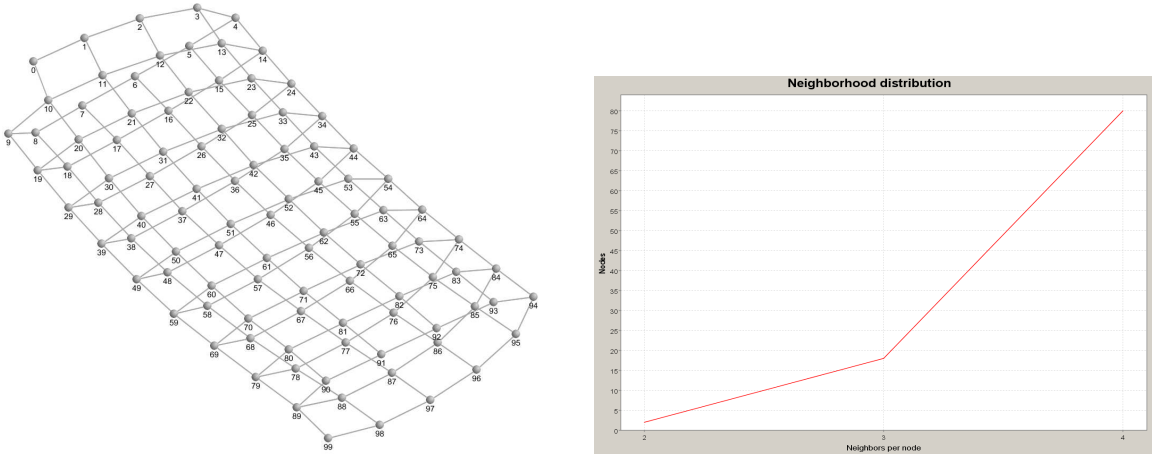
6.1 Tutkimuksessa käytettävät parametrit

Neuroverkkoja opetettiin kolmella eri resurssienhakualgoritmilla korkeimman asteen haulla, satunnaiskävelijällä ja leveyshaulla.

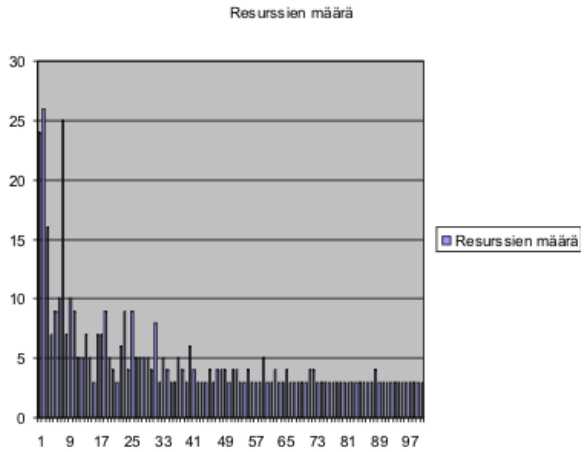
Simuloinnissa on paljon liikkuvia osia. Suurin osa tässä esiteltävistä parametrien arvoista on valittu testaamalla eri tapauksia parametrien lakaisuperiaatteella eli parametrejä on juoksutettu tietyllä arvoalueella ja tietyin välein. Näin ollaan saatu parametrien arvoille eri yhdistelmiä, joilla kaikilla on omat piirteensä neuroverkon opetuksessa. Mielenkiintoinen vaihtoehto olisi käyttää parametrien arvojen valintaan evoluutioalgoritmeja. Esimerkiksi neuroverkon rakenteen muodostamisessa EA:lla on saatu hyviä tuloksia, mutta tämän tutkielman yhteydessä sitä ei ole toteutettu.

Seuraavaksi esitellään tässä tutkimuksessa käytettävät parametrit. Vertaisverkon topologia on joka generaation alussa kuvan 6.1 kaltainen Grid-verkko, jossa on 100 solmua ja resurssit on sijoitettu potenssijakautuneesti kuvan 6.2 mukaisesti. Pieninumeroisilla solmuilla on enemmän resursseja kuin suurinumeroisilla. Vertaisverkon solmuilla on liikennerajoittimet, jotka simuloivat kaistanleveyttä. Kuvassa 6.3 on esitelty simuloinneissa käytettyjä kaistanleveyksiä. Solmut on jaettu neljään yhtä suureen osaan: parhaat solmut pystyvät käsittelemään 30 pakettia 10 resurssikyselyä kohden, toiseksi parhaat tästä puolet eli 15 pakettia, kolmanneksi parhaat kol-

masosan parhaasta eli 10 ja huonoimmat yhden viidesosan parhaasta eli 6 pakettia. Pieninumeroiset solmut ovat parempia kuin suurinumeroiset.

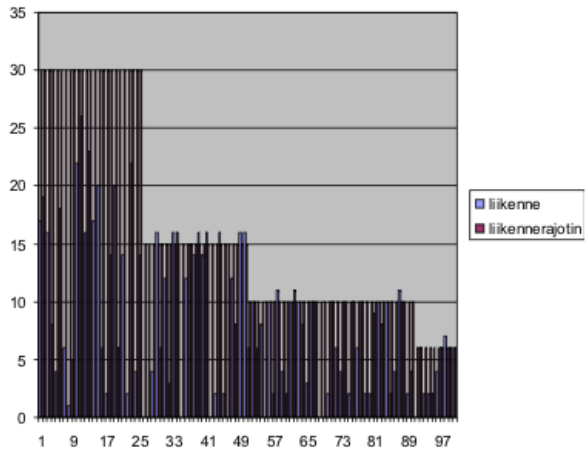


Kuva 6.1: Grid-verkon topologia ja naapurijakauma.



Kuva 6.2:
Kuvaaja solmujen resurssimääristä. Kolmella solmulla resurssien määrä x on $20 < x < 24$, kolmella $10 < x < 20$ ja lopuilla $3 < x < 10$. Resurssien yhteismäärä on 491 ja erilaisia resursseja on 25 kappaletta.

Valinta suoritetaan niin, että 24 neuroverkon joukosta 12 parasta pääsee suoraan seuraavaan generaatioon ja toiset 12 tuotetaan Gaussin mutaatiolla. Neuroverkkona käytetään luvuissa 5.3.2 ja 5.3.3 esiteltyä kolmikerrosneuroverkkoa. Resurssihakujen tavoitteena on löytää 50% vertaisverkon resursseista. Yhdessä generaatiossa kullakin neuroverkolla suoritetaan opetusvaiheessa sarja, jossa sadasta vertaisesta jokainen suorittaa 20 kertaa tarkistuksen naapurisuhteistaan (NeuroTopologia-



Kuva 6.3:
Kuvaaja solmujen liikennerajoittimista. Kuvaajassa on mukana liikennemäärät eräästä HDS-algoritmilla suoritetusta kyselyiden sarjasta potenssijakautuneessa P2P-verkossa.

algoritmi luvussa 5.3.1) ja näiden tarkistusten välissä tehdään 10 resurssikyselyä. Generaatioita suoritetaan 10000 kappaletta. Generaatioiden välissä opetusjoukon parasta neuroverkkoa testataan yleistysjoukossa, jotta nähtäisiin, kuinka se menestyy opetusympäristöstä poikkeavassa ympäristössä. Myös yleistysjoukossa suoritetaan 20 kertaa topologian tarkistuskierros, jossa jokainen solmu tarkistaa naapurisuhteensa. Resurssikyselyt tehdään niin, että jokainen solmu suorittaa resurssikyselynsä ja hyvyysarvot määritellään näiden tapahtumien perusteella. Toisin sanoen sadan solmun vertaisverkon tapauksessa resurssikyselyjä suoritetaan 100 joka kierroksella. Liikennerajoittimet eivät toimi oikein, jos ne pidetään samoina 10 resurssikyselyn ja 100 resurssikyselyn tapauksissa. Näin ollen liikennemittarit nollataan aina 10 resurssikyselyn jälkeen.
Opetuksessa käytettävät parametrien arvot ovat:

- Lähtötopologia: Grid-verkko, 100 solmua.
- Resurssit jakautuneet potenssijakautuman mukaisesti niin, että pieninumeroisilla solmuilla on enemmän resursseja.
- Suurin solmujen liikennerajoitin: 30.
- Käytetty mutaatio: Gaussin mutaatio.
- Neuroverkkojen lukumäärä: 24.

- Tavoitteena löytää 50% resursseista.
- Resurssikyselyjä topologiatarkistusten välillä: 10.
- Topologiatarkistusten määrä/vertainen/generaatio: 20.
- Generaatioiden lukumäärä: 10000.
- Neuroverkon rakenne: 1. piilokerroksella 16 neuronia ja toisella 4. Tarkemmin kerrottu luvussa 5.3.3.
- Neuroverkon sisääntuloparametrit: 7 + $bias$. Tarkemmin yksilöity luvussa 5.3.2.
- Hyvyysfunktion parametri $T = 5$. Topologian muutosten määrä vaikuttaa hyvyysfunktioon. Topologian muutospakettien määrää kerrotaan kertoimella 5. Tämä simuloi uuden TCP-yhteyden muodostuksessa syntyvää liikennettä.
- Hyvyysfunktion parametri R. Löydettyjen resurssien määrä vaikuttaa hyvyysfunktioon. Löydettyjen resurssien määrä kerrotaan kertoimella R. Tämän parametrin arvon vaikutusta neuroverkon opetukseen tutkitaan tarkemmin HDS-algoritmin yhteydessä.

6.2 Neuroverkkojen opetus

Neuroverkkojen opetuksen eteneminen käydään läpi suurin osin kuvaajien avulla. Kustakin suoritetusta yhdeksästä opetustapauksesta esitetään hyvyysarvot opetusjoukossa, hyvyysarvot yleistysjoukossa generaatioiden välissä, resurssikyselyt ja vastaukset yleistysjoukossa, topologian muutoskyselyt ja vastaukset yleistysjoukossa sekä epäonnistuneet kyselyt yleistysjoukossa.

Kuvaajassa, jossa esitetään topologiapaketit ja topologian muutokset, on esitetty pelkästään viimeisen topologiakyselyiden sarjan kyselyt ja vastaukset. Näin kuvaajasta nähdään selkeämmin, jos neuroverkon toiminta konvergoi vertaisverkon sellaiseksi, että naapureiden muuttumista ei enää tapahdu. Jos viimeisellä topologian muutoskierroksella kyselyiden ja vastauksien määrät ovat pieniä, neuroverkko tulkitsee, että vertaisverkon topologiaa ei ole syytä muuttaa.

Hyvyysarvojen kuvaajissa on huomioitava, että hyvyysfunktion parametrin R arvo vaikuttaa hyvyysarvoon paljon. Näin ollen vain tapaukset, joissa on käytetty keskenään samoja parametrien arvoja, ovat vertailukelpoisia keskenään. Yleistysjoukon hyvyysarvon kuvaajissa olisi kannattanut käyttää funktiota 5.10, sillä yleistysjoukossa suoritettavat testit eivät vaikuta neuroverkon opetukseen. Näin hyvyy-

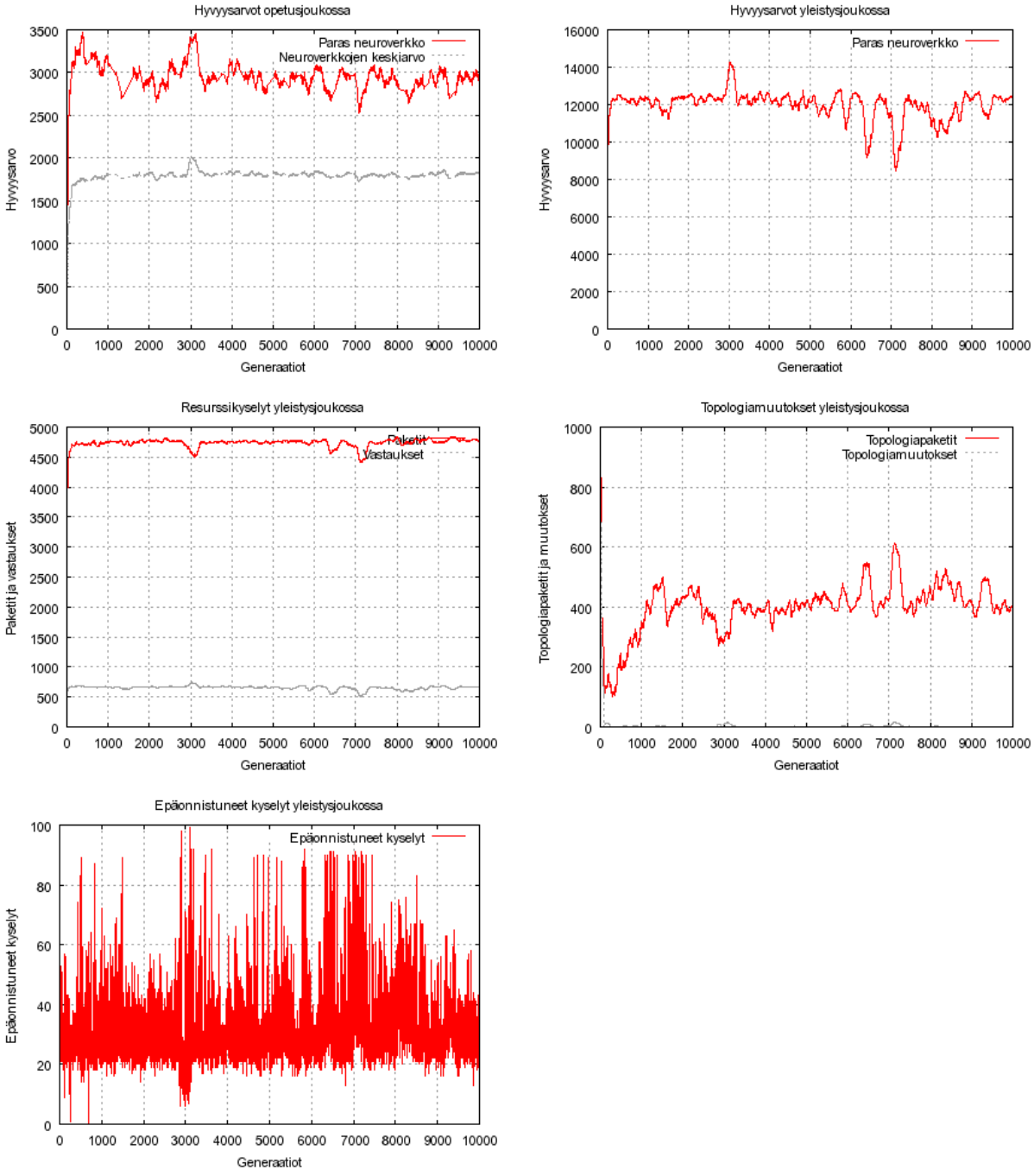
sarvot olisivat olleet vertailukelpoisia kaikissa tapauksissa. Lisäksi yleistysjoukon hyvyysarvon kuvaaja on joissain tapauksissa melko kohinainen. Todennäköinen syy tähän on se, että hyvyysfunktio 5.9 on määritelty niin, että kun kaikki tavoitellut resurssit löydetään, niin löydetyistä resursseista saa kaksinkertaisen palkinnon. Kuvaajan ollessa kohinainen, vertaisverkossa mahdollisesti löydetään kaikki tavoitellut resurssit ajoittain ja aina välillä jää joitain resursseja löytymättä. Näin ollen kuvaaja heiluu edestakaisin eikä anna arvokasta informaatiota. Palkinnon tuplaamisen ideana on ohjata neuroverkon opetusta oikeaan suuntaan, mutta neuroverkon testauksen seuraamista tällainen hyvyysfunktio pelkästään vaikeuttaa. Lisäksi mitä suurempi on R:n arvo sitä suurempi on kohinaisuus. Tämä on toinen syy, miksi tämä kuvaaja olisi pitänyt muodostaa hyvyysfunktiolla 5.10. Kohinaisuudesta huolimatta kuvaajista näkee, onko hyvyysarvon kehityksen trendi nouseva, jolloin neuroverkon suorituskyky paranee.

6.2.1 Opetustapaukset

Seuraavassa esitellään yhdeksän eri tutkimustapausta. Korkeimman asteen hakua on käytetty viidessä tapauksessa. Satunnaiskävelijän ja leveyshaun tapauksia on kaksi kumpaakin.

6.2.1.1 HDS, R=25

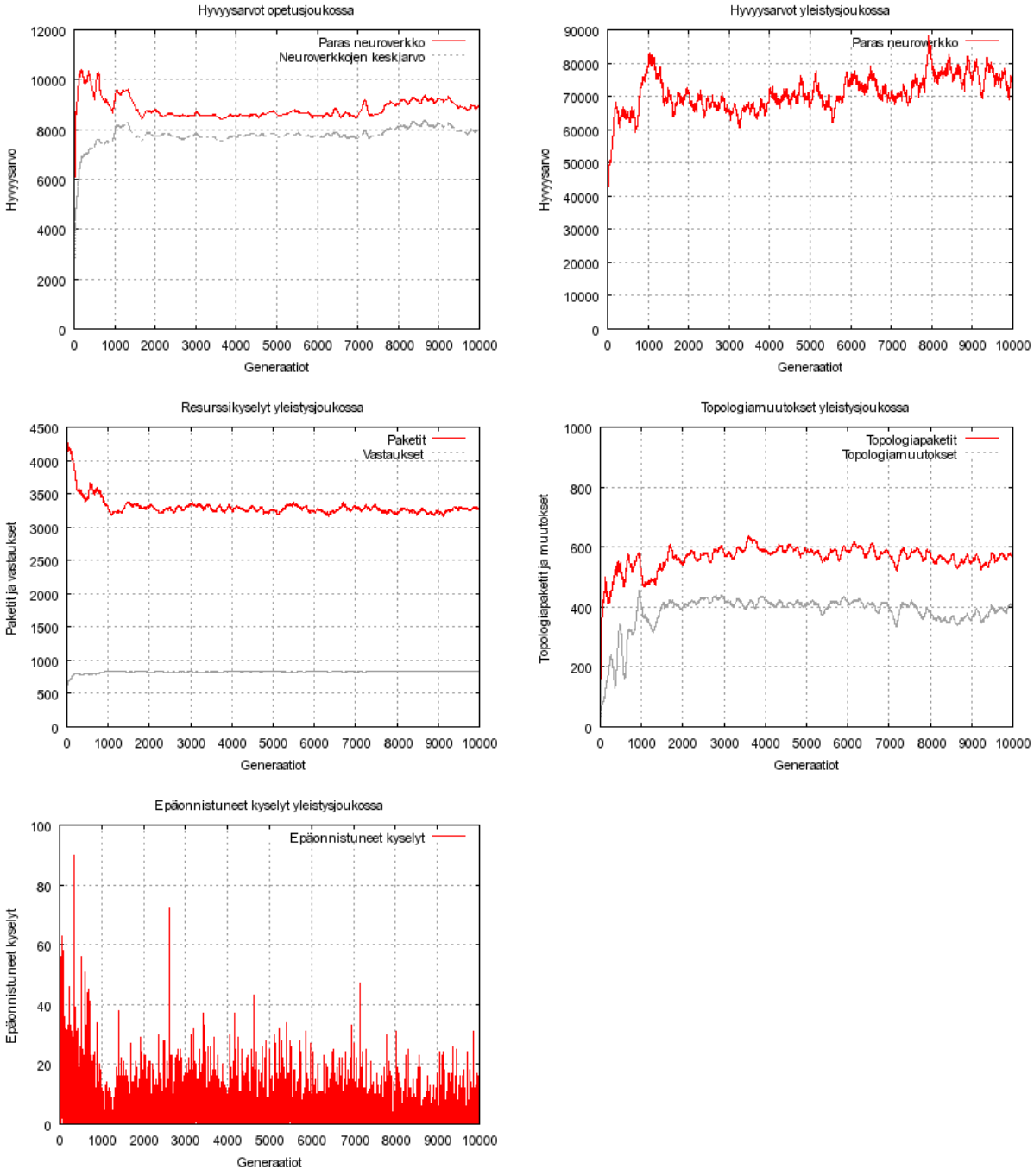
Kuvassa 6.4 on esitetty opetuksen eteneminen, kun opetukseen käytetään luvun 6.1 parametrejä. Resurssien löytämisestä palkitseva kerroin eli hyvyysfunktion 5.10 parametri R on 25. Kuvaajista nähdään, että opetus ei ole edennyt 10000 generaation aikana käytännössä yhtään. Hyvyysarvot ovat lähes samoja koko opetuksen ajan, epäonnistuneiden kyselyjen määrä ei muutu eikä topologian muutoksia tehdä juuri lainkaan. Tämä opetustapaus vaikuttaa epäonnistuneelta, mutta sen suoritus käydään vielä kuitenkin läpi loppuanalyysissa luvussa 6.3.1.1.



Kuva 6.4: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on korkeimman asteen haku ja hyvyysfunktion parametri R on 25.

6.2.1.2 HDS, R=75

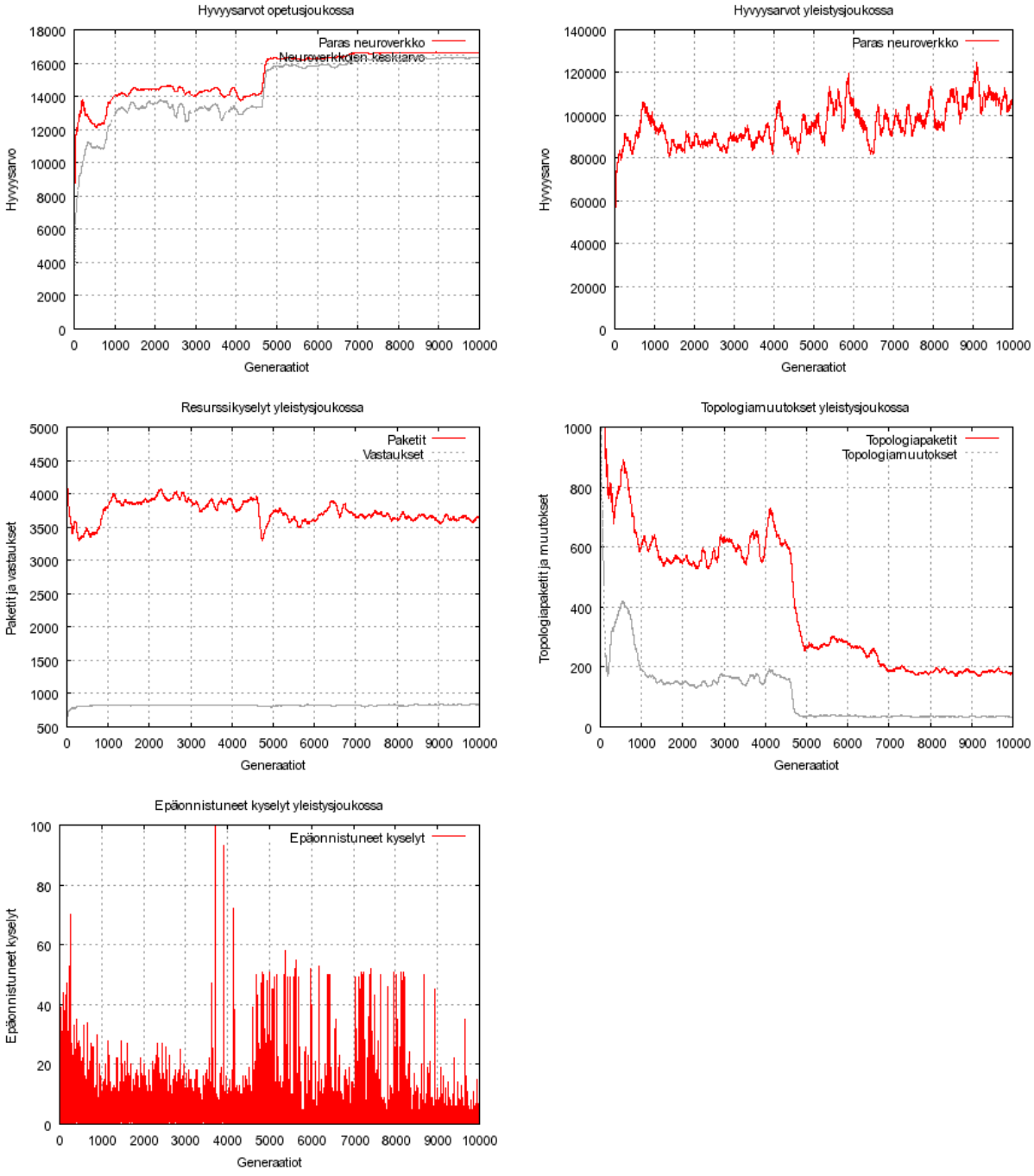
Opetustapaus, jossa parametrin R arvo on 75, ei ole kuvan 6.5 perusteella edellistä tapausta parempi. Epäonnistuneiden kyselyiden määrä on pienempi kuin $R = 25$:n tapauksessa, mutta topologian muutokset jatkuvat generaatiosta toiseen. Tämä tarkoittanee sitä, että neuroverkko ei osaa konvergoida vertaisverkkoa tehokkaaksi topologiaksi, mutta tämäkin asia saadaan varmistettua luvussa 6.3.1.2.



Kuva 6.5: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on korkeimman asteen haku ja hyvyysfunktion parametri R on 75.

6.2.1.3 HDS, R=100

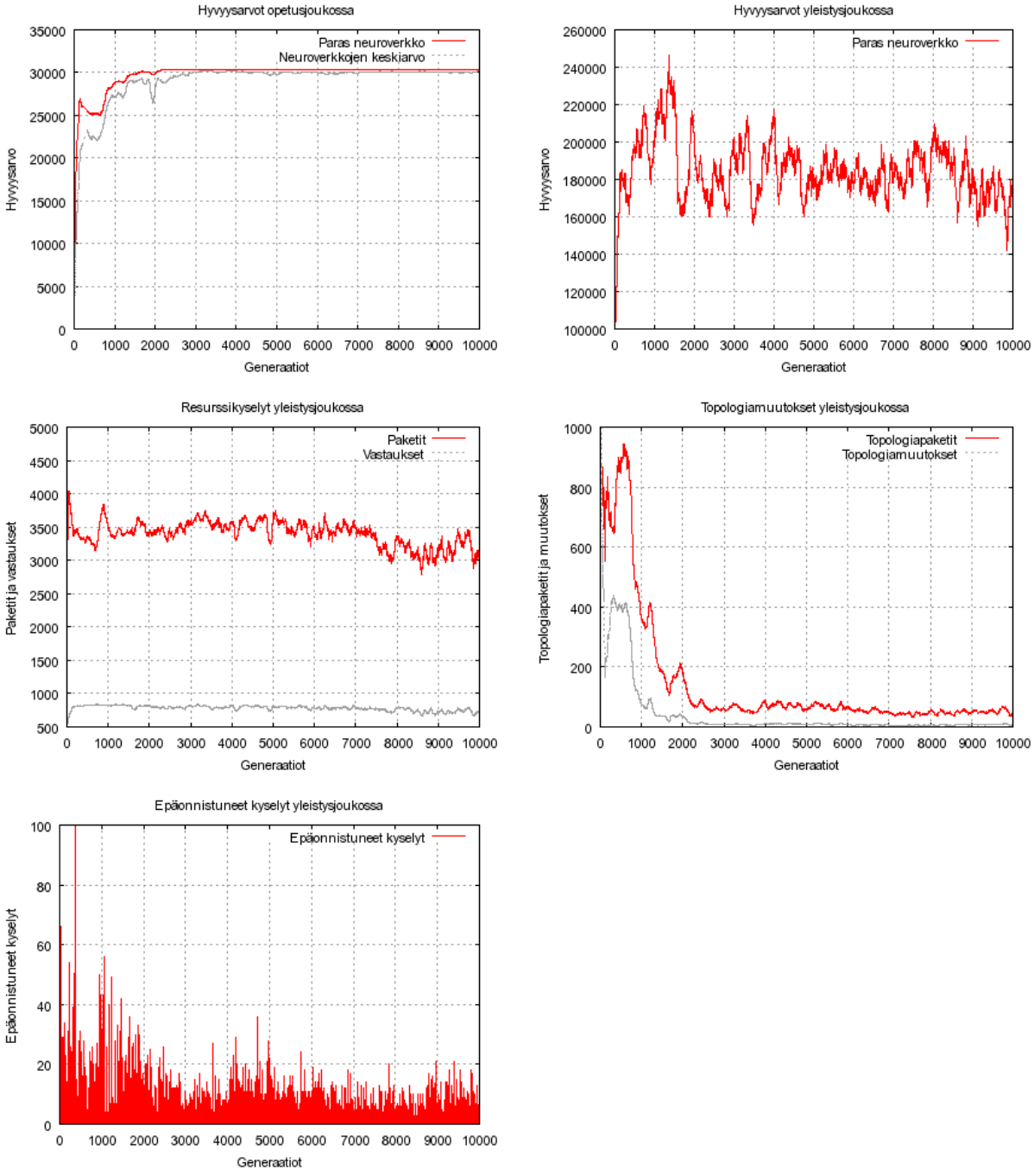
Kuvasta 6.6 nähdään, että nyt vertaisverkossa tapahtuu konvergoitumista. Generaation 4800 tietämillä topologian muutokset ovat pienentyneet huomattavasti. Opetusjoukossa hyvyysarvo paranee samaan aikaan, mutta yleistysjoukon testeissä tätä hyvyyden paranemista ei nähdä. Epäonnistuneet kyselyt yleistysjoukossa kasvavat ajoittain generaation 4000 jälkeen. Epäonnistuneiden kyselyiden perusteella voi päätellä, että opetusjoukossa mahdollisesti hyvin toimiva neuroverkko ei osaa parantaa vertaisverkkoa yleistysjoukossa, jossa kaikki solmut osallistuvat resurssienhakuun. Neuroverkko siis osoittaa oppimisen merkkejä, kun vertaisverkossa suoritettavien kyselyiden määrä on pieni (opetusjoukon 10 kyselyä topologian tarkistuskierrosten välissä).



Kuva 6.6: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on korkeimman asteen haku ja hyvyysfunktion parametri R on 100.

6.2.1.4 HDS, R=175

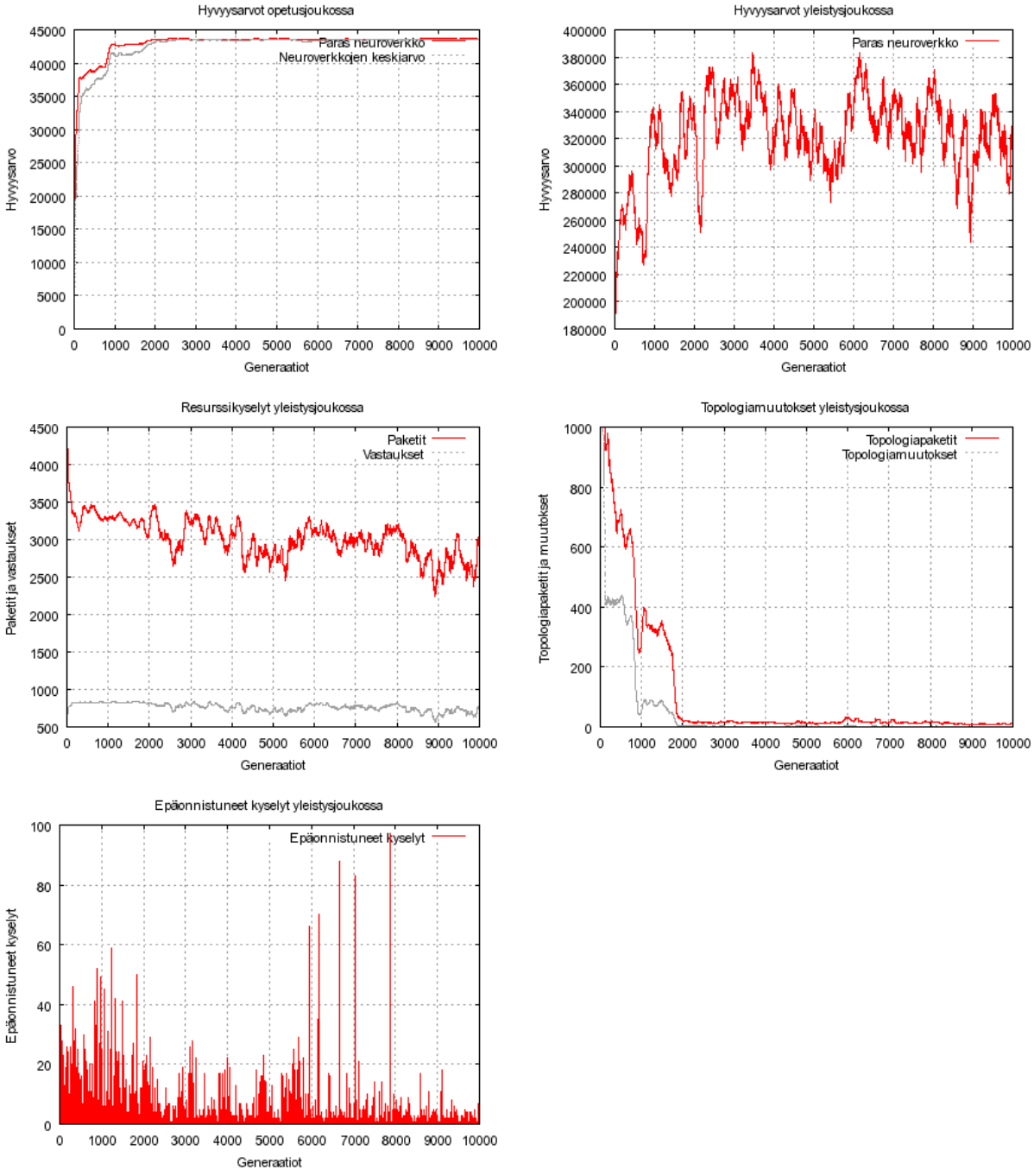
Tässä tapauksessa nähdään kuvan 6.7 perusteella, että neuroverkon mahdollinen oppiminen on tapahtunut generaatioon 2000 mennessä. Topologiamuutosten kuvaajasta nähdään, että oppimisen jälkeen neuroverkko ei ole halukas muuttamaan topologiaa viimeisellä kyselykierroksella. Epäonnistuneiden kyselyiden määrä pysyy tasaisen pienenä generaation 2000 jälkeen, eikä suuria hyppäyksiä enää tapahdu.



Kuva 6.7: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on korkeimman asteen haku ja hyvyysfunktion parametri R on 175.

6.2.1.5 HDS, R=250

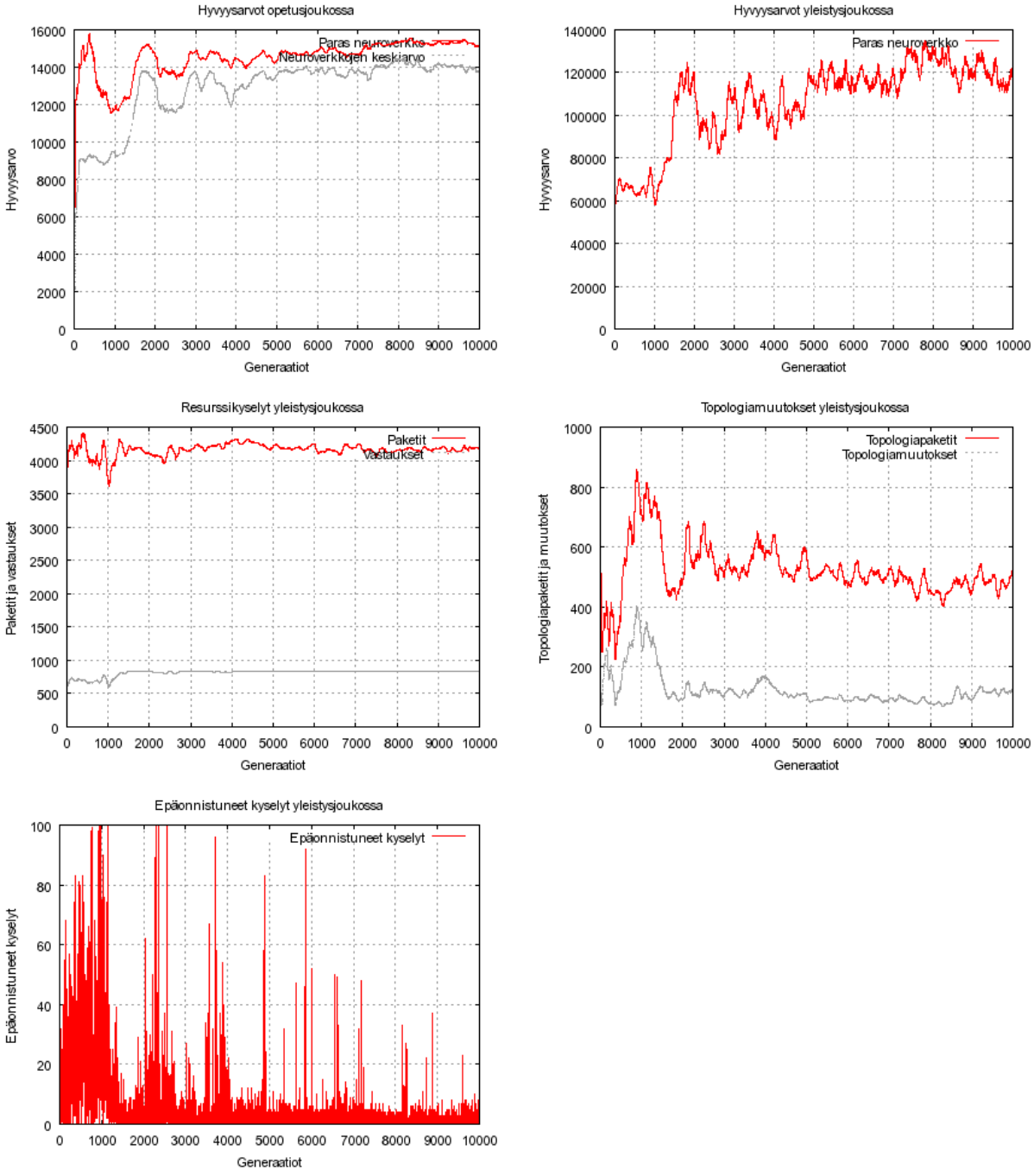
Tämän opetustavan kuva 6.8 viittaa onnistuneeseen konvergoitumiseen. Generaation 2000 jälkeen ei suoriteta topologian muutoksia viimeisellä tarkistuskierroksella käytännössä ollenkaan. Resurssien etsimiseen käytettävät pakettimäärät laskevat melko tasaisesti ja epäonnistuneiden kyselyiden määrä pysyy pääasiallisesti pienenä. Generaation 2000 jälkeen nähdään viisi kyselysarjaa, jotka epäonnistuvat kokonaan. Opetusjoukon 24:stä neuroverkosta on valittu paras yleistysjoukkoon testattavaksi ja se epäonnistuu pahoin. Tämä viittaisi siihen, että opetusjoukon ympäristö on ajoittain liian erilainen, jotta neuroverkot toimisivat myös yleistysjoukossa. Täysin epäonnistuneita kyselyitä on kuitenkin vähemmän kuin R:n arvolla 100 ja suurin osa generaatiosta saa paremman tuloksen kuin parametrin R arvolla 175.



Kuva 6.8: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on korkeimman asteen haku ja hyvyysfunktion parametri R on 250.

6.2.1.6 Random Walker, R=100

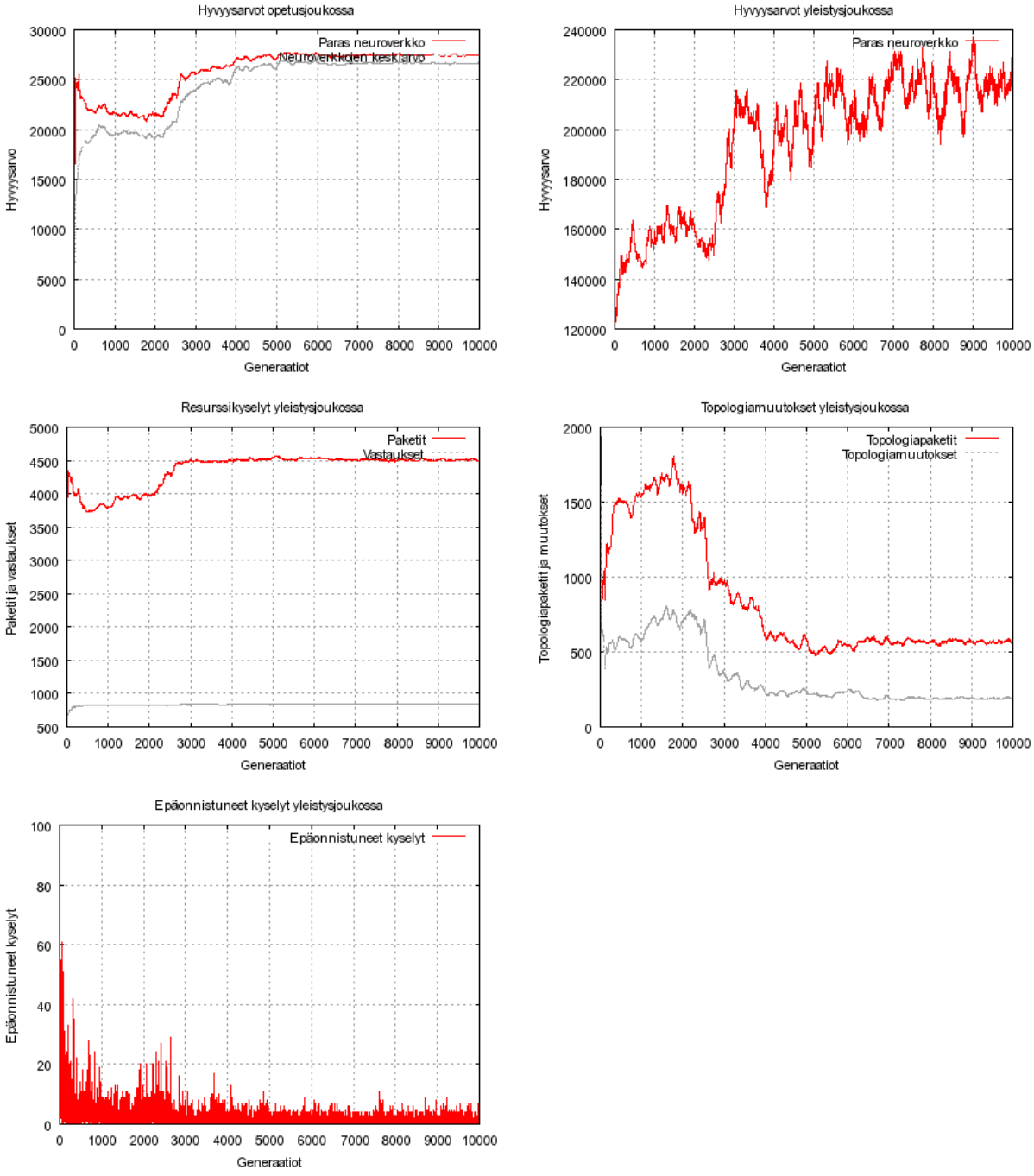
Satunnaiskävelijällä opetettu neuroverkko osoittaa kuvan 6.9 perusteella pieniä konvergoitumisen merkkejä. Topologian muutokset tasaantuvat generaation 5000 jälkeen, mutta eivät lopu kokonaan. Yleistysjoukon hyvyysarvon trendi on selvästi nouseva generaatioon 8000 asti. Epäonnistuneita kyselyitä tulee ajoittain melko paljon, jotka näkyvät viimeisessä kuvaajassa piikkeinä. Suurin osa kyselyistä kuitenkin onnistuu melko hyvin.



Kuva 6.9: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on satunnaiskävelijä ja hyvyysfunktion parametri R on 100.

6.2.1.7 Random Walker, R=175

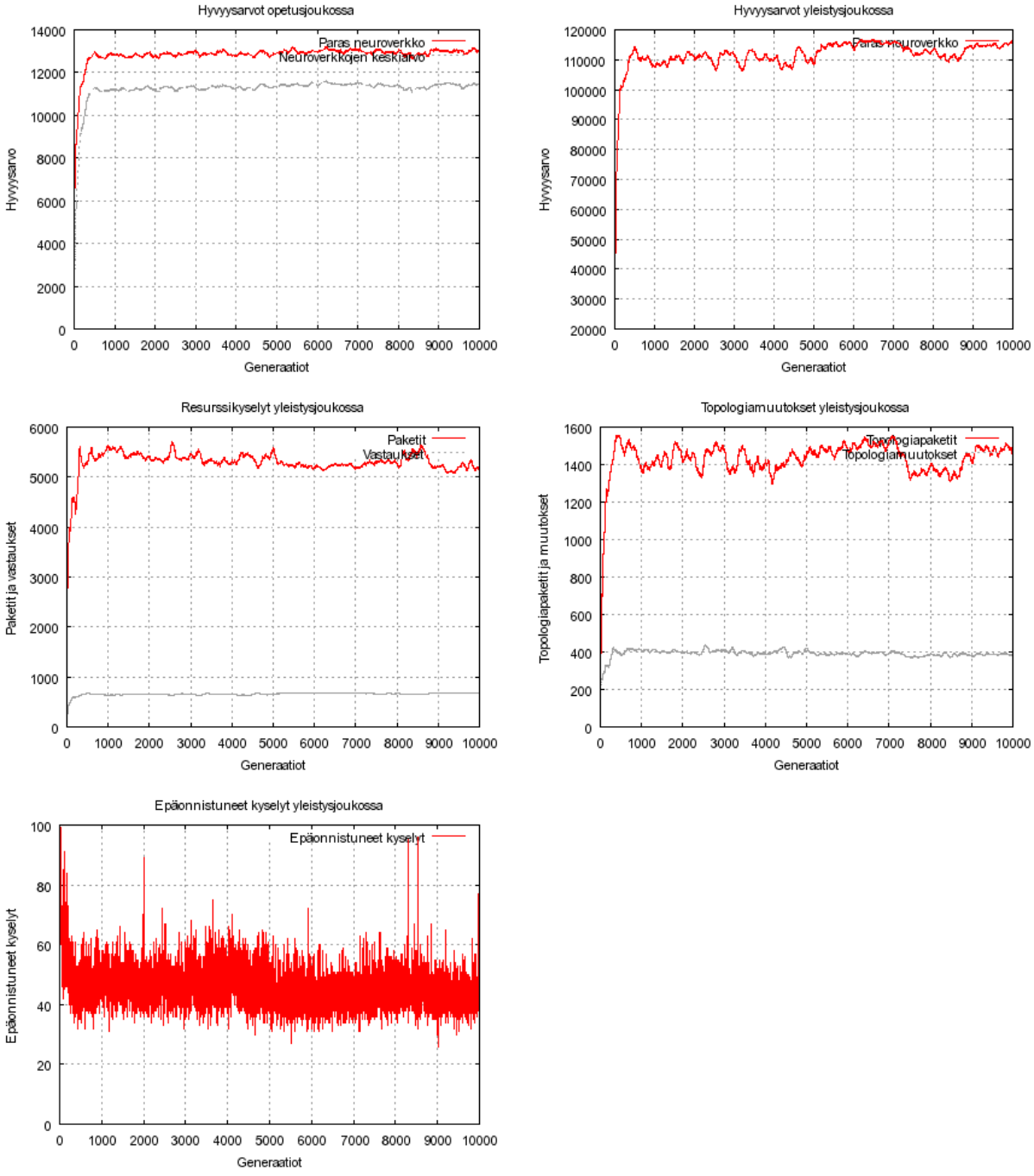
Tämän opetustavan kuvaajat 6.10 ovat melko selkeitä. Neuroverkon oppiminen on valmis generaation 5500 kohdalla. Neuroverkko ei kuitenkaan osaa kokonaan lopettaa topologian muutoksia, vaan uusia yhteyksiä muodostetaan paljon myös viimeisellä kyselykierroksella. Epäonnistuiden kyselyiden määrä pysyy pienenä generaation 5500 jälkeen.



Kuva 6.10: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on satunnaiskävelijä ja hyvyysfunktion parametri R on 175.

6.2.1.8 BFS, TTL=3, R=175

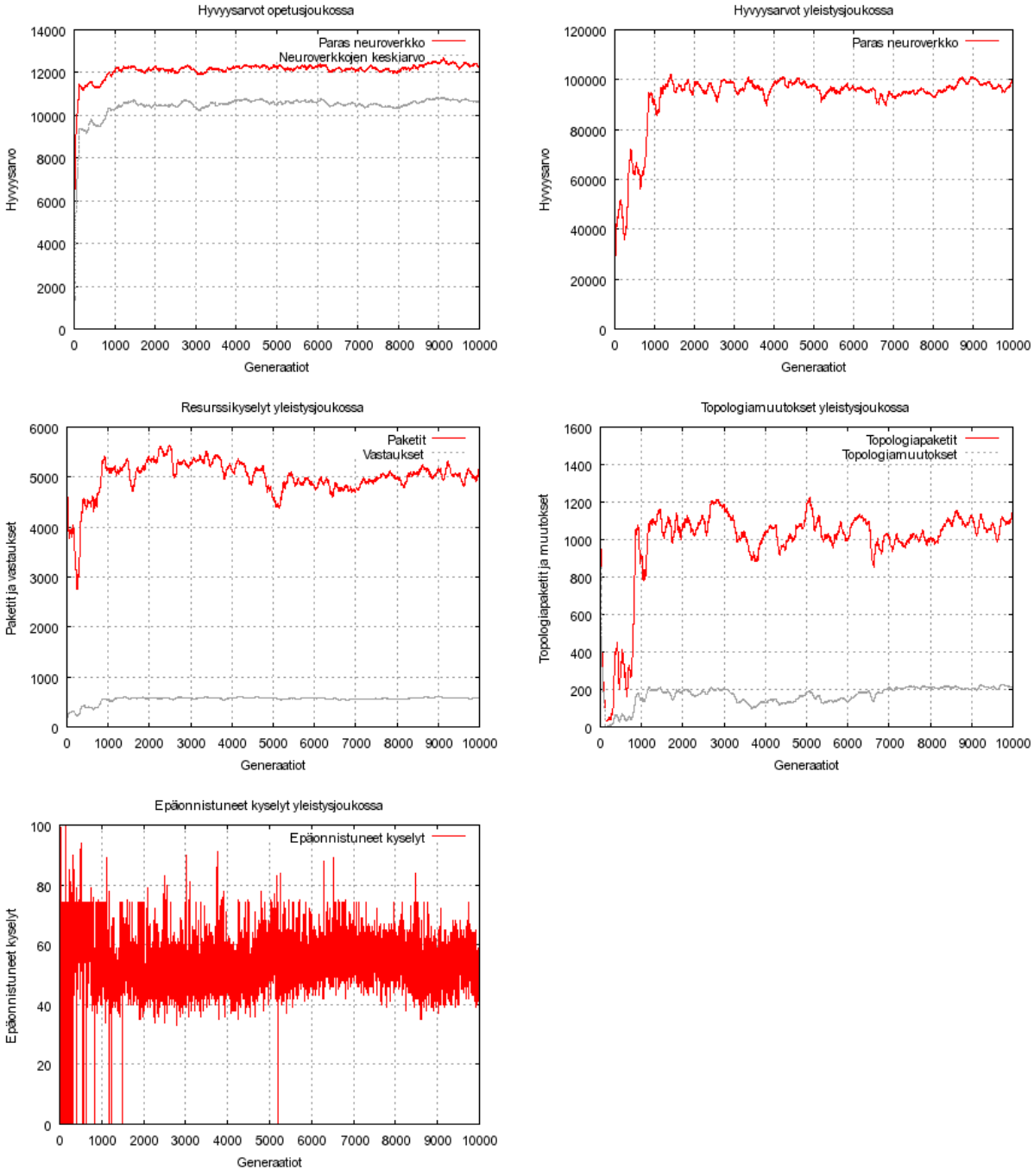
Neuroverkon opetus leveyshaun TTL-arvolla 3 näyttää loppuvan kuvan 6.11 perusteella generaation noin 500 kohdalla. Neuroverkko ei opi konvergoimaan vertaisverkkoa. Topologian muutoksia tehdään noin 400 vielä viimeiselläkin kyselykierroksella.



Kuva 6.11: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on leveyshaku TTL-arvolla 3 ja hyvyysfunktion parametri R on 175.

6.2.1.9 BFS, TTL=6, R=175

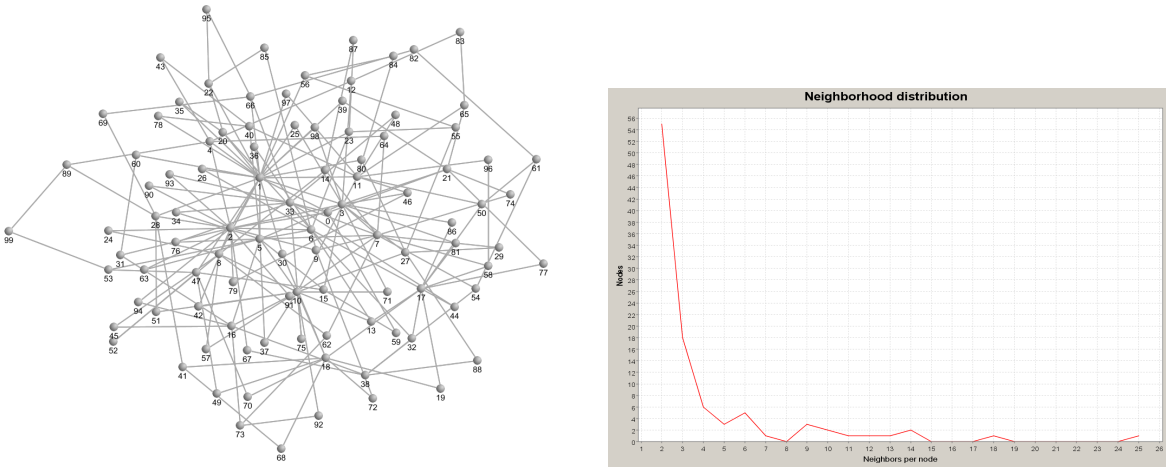
Kuvasta 6.12 nähdään, että myös leveyshakua TTL-arvolla 6 käyttäen neuroverkon opetus on valmis melko aikaisin. Generaation 1000 jälkeen ei muutoksia neuroverkon käyttäytymisessä juuri ole. Neuroverkko ei suorita yhtä paljon topologian mutoksia kuin TTL-arvon 3 tapauksessa, mutta vieläkään neuroverkko ei osaa konvergoida vertaisverkkoa. Yhteyksien muodostamista suoritetaan suhteellisen paljon vielä viimeiselläkin topologian tarkistuskierroksella.



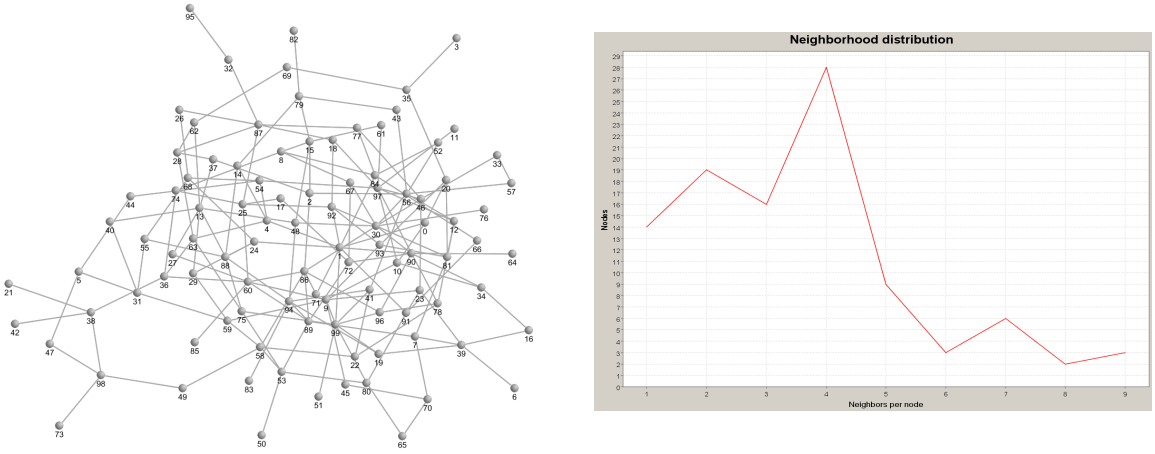
Kuva 6.12: Neuroverkon opetuksen eteneminen, kun hakualgoritmina on leveyshaku TTL-arvolla 6 ja hyvyysfunktion parametri R on 175.

6.3 Neuroverkkojen analysointi

Tutkitaan ensin, kuinka neljä eri hakualgoritmia suoriutuu kolmessa erilaisessa sadan solmun vertaisverkossa, jossa ei tehdä topologian muutoksia. Hakualgoritmit ovat korkeimman asteen haku (HDS), satunnaiskävelijä (RW), leveyshaku TTL-arvolla 3 (BFS-3) ja leveyshaku TTL-arvolla 6 (BFS-6). Topologioina ovat Grid-verkko (kuva 6.1), potenssijakautunut verkko (Power, kuva 6.13) ja satunnainen verkko (Random, kuva 6.14). Potenssijakautunut verkko ja satunnainen verkko on muodostettu Airoldin [3] artikkelissa esitettyjä periaatteita noudattaen. Satunnaisen verkon generoinnissa on käytetty samaisessa artikkelissa esiteltyä Erdösin menetelmää. Vertaisverkon jokainen solmu suorittaa resurssikyselynsä kerran, jolloin kyselyjä kertyy yhteensä 100 kappaletta. Tulokset on kirjattu taulukkoon 6.1. Kyselyjen hyvyys on laskettu kaavalla 5.10, jossa parametri $T = 5$ ja parametri $R = 50$. Kyselyiden tavoitteena on löytää yhteensä 50% resursseista, joka on lukumääräisesti 845 resurssia. Epäonnistuneet kyselyt -sarakkeessa ilmoitetaan, kuinka monta solmua epäonnistui 50% tavoitteessaan. Viestien matka -sarakkeessa ilmoitetaan, kuinka monen solmun kautta kyselyt ovat keskimäärin kulkeneet. Jos vertaisverkossa ei olisi liikennerajoitteita, niin esimerkiksi BFS-6 -haun viestit kulkisivat aina kuuden solmun läpi. Liikennerajoittimet aiheuttavat epäonnistuneita kyselyitä ja kuten taulukosta 6.1 nähdään, viestit kulkevat BFS-6 -haun tapauksessa keskimäärin neljän solmun läpi.



Kuva 6.13: Potenssijakautunut vertaisverkko ja sen naapurijakauma.



Kuva 6.14: Satunnainen vertaisverkko ja sen naapurijakauma.

Haku-algoritmi	**Topologia**	**Hyvyys-arvo**	**Pake-tit**	**Vasta-ukset**	**Epä-onnis-tuneet kyse-lyt**	**Viestien matka**	**Tehok-kuus**
HDS	Grid	30059.0	4791	697	24	47.93	0.145
RW	Grid	30449.0	4501	699	24	45.08	0.155
BFS-3	Grid	10302.0	2598	258	99	2.92	0.099
BFS-6	Grid	12955.0	4445	348	74	3.78	0.078
HDS	Power	38216.0	3634	837	6	36.34	0.23
RW	Power	36193.0	4507	814	7	45.07	0.18
BFS-3	Power	18293.0	4707	460	71	2.86	0.097
BFS-6	Power	8872.0	6178	301	77	3.52	0.048
HDS	Random	28209.0	3891	642	45	38.97	0.164
RW	Random	26047.0	3603	593	50	36.07	0.164
BFS-3	Random	19340.0	3350	470	87	2.96	0.14
BFS-6	Random	16136.0	5214	427	74	4.43	0.081

Taulukko 6.1: Hakualgoritmien suorituskyky vertaisverkoissa, joissa ei tehdä topologian muutoksia.

6.3.1 Analysointitapaukset

Seuraavaksi tutkitaan, kuinka yhdeksällä eri tavalla opetetun neuroverkon käyttäminen topologian muutoksiin vaikuttaa vertaisverkon toimintaan. Luodaan ensin yleiskuva kaikkien opetettujen neuroverkkojen toiminnasta eri skenaarioissa, jotta

nähdään, mitä tapauksia on syytä tutkia tarkemmin. Lisäksi nähdään, onko neuroverkkojen toiminnassa ylipäätään mitään opittua käyttäytymistä.

Taulukossa 6.2 on esitelty neuroverkoista johtuva hyvyysarvojen prosentuaalinen paraneminen tai huononeminen verrattuna vertaisverkkoihin ilman topologian muutoksia. Jokaista yhdeksää neuroverkkoa käytetään neljän eri hakualgoritmin tapauksessa ja kukin neuroverkko-hakualgoritmi -yhdistelmä aloitetaan kolmesta eri lähtötopologiasta. Neuroverkkojen testaus on suoritettu siten, että kussakin lähtötopologiassa jokainen solmu suorittaa resurssikyselynsä. Tämän jälkeen jokainen solmu tarkistaa naapurisuhteensa neuroverkon avulla. Resurssikysely-naapuritarkistus -toimenpidesarja tehdään 20 kertaa eli resurssikyselyjä kertyy kaikenkaikkiaan 2000 ja jokainen solmu suorittaa topologian tarkistuksen 20 kertaa.

Taulukkoa luetaan niin, että esimerkiksi jos on käytössä vertaisverkko, jonka hakualgoritmina on korkeimman asteen haku ja jonka topologia on potenssijakautunut, niin kyselyjen hyvyysarvo on keskimäärin 38216. Jos tähän vertaisverkkoon otetaan mukaan topologian muutokset neuroverkolla, joka on opetettu käyttäen satunnaiskävelijää ja hyvyysfunktion parametria $R = 100$, niin kyselyjen hyvyysarvo pienenee 4.4 %, (Rivi: *HDS - Power*; Sarake: *RW, R=100*).

Haku-algoritmi	**Lähtö-topologia**	**Ei neuro-verkkoa**	**HDS, R=25**	**HDS, R=75**	**HDS, R=100**	**HDS, R=175**	**HDS, R=250**
HDS	Grid	30059	-2,7 %	10,7 %	18,4 %	24,0 %	26,4 %
RW	Grid	30449	-0,3 %	-3,3 %	6,7 %	7,3 %	14,6 %
BFS-3	Grid	10302	-0,3 %	18,2 %	53,8 %	11,1 %	0,0 %
BFS-6	Grid	12955	1,0 %	-78,7 %	-8,7 %	-4,6 %	-0,1 %
HDS	Power	38216	-8,2 %	-12,3 %	-1,9 %	0,3 %	0,4 %
RW	Power	36193	-6,2 %	-16,0 %	-4,4 %	-4,3 %	1,3 %
BFS-3	Power	18293	-11,8 %	-28,8 %	5,9 %	21,7 %	36,8 %
BFS-6	Power	8872	-21,9 %	-69,7 %	28,5 %	2,9 %	9,9 %
HDS	Random	28209	-4,9 %	16,2 %	28,1 %	31,1 %	34,9 %
RW	Random	26047	-8,0 %	15,3 %	25,8 %	31,4 %	35,0 %
BFS-3	Random	19340	0,0 %	-31,5 %	7,8 %	25,8 %	30,0 %
BFS-6	Random	16136	-6,3 %	-82,8 %	0,1 %	-12,4 %	-21,6 %

Haku-algoritmi	**Lähtö-topologia**	**Ei neuro-verkkoa**	**RW, R=100**	**RW, R=175**	**BFS-3, R=175**	**BFS-6, R=175**
HDS	Grid	30059	20,0 %	8,8 %	-7,1 %	-7,2 %
RW	Grid	30449	10,9 %	5,7 %	-16,0 %	-23,3 %
BFS-3	Grid	10302	82,6 %	7,6 %	72,9 %	49,7 %
BFS-6	Grid	12955	-81,3 %	-92,6 %	-33,0 %	29,3 %
HDS	Power	38216	-4,4 %	-10,3 %	-24,4 %	-25,4 %
RW	Power	36193	-4,2 %	-8,3 %	-28,1 %	-35,9 %
BFS-3	Power	18293	18,5 %	-34,6 %	5,1 %	-13,0 %
BFS-6	Power	8872	-89,1 %	-74,4 %	1,4 %	86,5 %
HDS	Random	28209	23,7 %	17,6 %	-0,9 %	-5,0 %
RW	Random	26047	31,5 %	24,4 %	0,5 %	-11,7 %
BFS-3	Random	19340	11,6 %	-32,1 %	1,1 %	-18,2 %
BFS-6	Random	16136	-67,6 %	-84,1 %	-38,8 %	7,5 %

Taulukko 6.2: Yhdeksän eri neuroverkon vaikutus hyvyysarvoihin, kun kunkin neuroverkon kahtatoista tutkimustapausta verrataan staattiseen vertaisverkkoon.

Tuloksista nähdään heti, että opetustapana HDS arvolla R=25 on täysin epäonnistunut. Se on huonontanut kaikkien vertaisverkkojen toimintaa, vaikka niissä olisi käytössä mikä hakualgoritmi tahansa (lukuunottamatta Grid-verkkoa TTL-6 -leveyshaulla). Hyvyysfunktion parametri R vaikuttaa siihen, kuinka paljon hakua palkitaan resurssin löytämisestä. Todennäköisesti R:n arvo 25 on liian pieni rohkaisemaan neuroverkkoa topologian muutoksiin, jolloin luonnollinen testi on suuren-

taa R:n arvoa. HDS-haun avulla opetetut neuroverkot parantavatkin HDS-hakua Grid-verkossa sitä enemmän mitä suurempi R on. RW- ja BFS-hakujen tapauksessa R:n arvon valintaan on vaikuttanut eniten se, että tuloksia voitaisiin vertailla eri hakualgoritmien välillä. Näin ollen kaikilla neuroverkoilla on ainakin käytetty R:n arvoa 175.

Toinen yleinen huomio on, että potenssijakautunutta verkkoa on vaikein tehostaa. Ainoa neuroverkko, joka on onnistunut parantamaan potenssijakautunutta verkkoa kaikille hakualgoritmeille on HDS arvolla R=250 -skenaariossa opetettu neuroverkko. Kolmas huomio on, että kun opetuksessa parametri R on ollut 175, niin melko useissa tapauksissa neuroverkot tehostavat parhaiten tai heikentävät vähiten juuri sitä hakualgoritmia, millä neuroverkko on opetettukin. Osassa tapauksia paraneminen on ollut melko suurta. Esimerkiksi BFS TTL-6 arvolla R=175 -tapauksessa hakualgoritmi BFS TTL-6 tehostuu 29.3 %, kun lähtötopologia on Grid-verkko, ja 86.5 %, kun lähtötopologia on potenssijakautunut verkko. HDS arvolla 175 tai HDS arvolla 250 tapauksilla opetetut neuroverkot vaikuttavat tämän taulukon perusteella yleiskäyttöisimmiltä, jos halutaan ottaa kaikki tutkitut hakualgoritmit huomioon. Jos tarkastellaan vain kävelijäalgoritmeja, niin HDS arvolla R=250 -verkko on paras. Leveyshakualgoritmeja ja kävelijäalgoritmeja näyttää olevan vaikea saada toimimaan samanlaisissa ympäristöissä keskenään. Toisaalta, jos katsotaan pelkästään TTL-3 leveyshakua, niin nähdään, että esimerkiksi HDS-haulla opetetut neuroverkot parantavat myös BFS-3 -hakua etenkin, kun käytetään suurta parametrin R arvoa. Leveyshaku, jonka TTL=6, on melko raskas hakualgoritmi näin pienessä vertaisverkossa. Sen käyttämä liikennemäärä on erittäin suuri ja etenkin tässä tutkimusympäristössä asetetut liikennerajoittimet aiheuttavat sen, että verkko tukkeutuu varsin nopeasti. Syitä nyt esiteltyihin ilmiöihin saataneen selvitettyä tutkimalla kutakin opetettua neuroverkkoa tarkemmin.

Tapauskohtaisissa tarkasteluissa esiteltävissä taulukoissa, kuten esimerkiksi 6.3, esitetyt arvot ovat keskiarvoja, kun resurssikysely-topologian muutos -sarjoja on suoritettu 20 kappaletta. Esimerkiksi sarakkeen Topo-paketit ensimmäisen rivin arvo 49 on siis keskiarvo yhdellä topologian muutoskierroksella suoritetuista topologiakyselyistä. Toisin sanoen on mahdollista, että ensimmäisellä kierroksella topologiakyselyitä on ollut 980 kappaletta ja lopuilla 19 kierroksella 0 kappaletta. Sarakkeen arvo ei siis ole kovin kuvaava, jos halutaan tietää vertaisverkon konvergoitumisesta kierrosten aikana. Tästä syystä mielenkiintoisimmissa tapauksissa vertaisverkon konvergoitumista esitetään kuvaajilla, joilla nähdään topologiakyselyt ja topologiamuutokset kierroskohtaisesti. Vertaisverkon konvergoituminen on toivottava ominaisuus, sillä se kertoo, että tehokas vertaisverkkotopologia on löydetty ja si-

tä ei ole enää syytä muuttaa. Jos topologiamuutokset jatkuvat kierroksesta toiseen, ei neuroverkko selvästikään tiedä, minkälainen topologia tulisi muodostaa, vaan vertaisverkon solmut vaihtelevat naapureitaan jatkuvasti tuottaen samalla suuren määrän liikennettä. Neuroverkon vaikutus -sarakkeessa verrataan neuroverkollisen vertaisverkon hyvyysarvoa vastaavan tapauksen staattiseen vertaisverkkoon.

6.3.1.1 HDS, R=25

Haku-algoritmi	**Lähtö-topologia**	**Hyvyys**	**Neuro-verkon vaikutus**	**Pake-tit**	**Vasta-ukset**	**Epä-onnis-tuneet ky-selyt**	**Topo-Pake-tit**	**Topo-Vasta-ukset**	**Vies-tien mat-ka**	**Tehok-kuus**
HDS	Grid	29254.0	-2.7%	4801	686	26	49	0	48.05	0.141
RW	Grid	30364.0	-0.3%	4516	709	21	114	0	45.21	0.153
BFS-3	Grid	10267.0	-0.3%	2598	258	99	7	0	2.92	0.099
BFS-6	Grid	13081.0	1.0%	4334	364	73	157	0	3.87	0.081
HDS	Power	35064.0	-8.2%	3541	819	7	469	0	35.42	0.204
RW	Power	33963.0	-6.2%	4247	814	7	498	0	42.48	0.171
BFS-3	Power	16143.0	-11.8%	4707	460	71	430	0	2.86	0.089
BFS-6	Power	6932.0	-21.9%	6178	301	77	388	0	3.52	0.045
HDS	Random	26827.0	-4.9%	3793	643	44	306	0	37.99	0.156
RW	Random	23951.0	-8.0%	3599	586	49	350	0	36.02	0.148
BFS-3	Random	19340.0	0.0%	3350	470	87	162	0	2.96	0.133
BFS-6	Random	15126.0	-6.3%	5214	427	74	202	0	4.43	0.078

Taulukko 6.3: HDS arvolla R=25 -neuroverkon tulokset eri testitapauksissa.

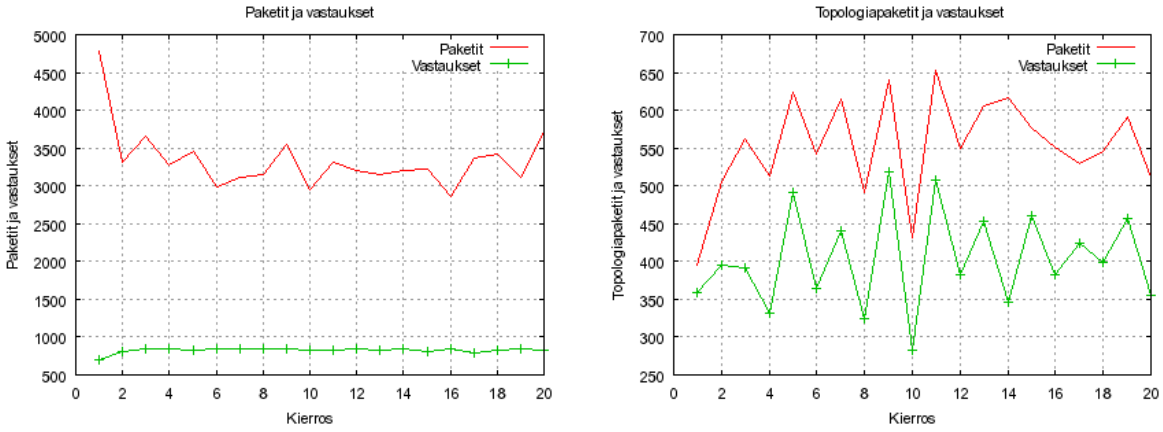
HDS arvolla R=25 -tapauksella opetettu neuroverkko ei ole onnistunut parantamaan vertaisverkon toimintaa käytännössä millään testaustavalla. Taulukosta 6.3 nähdään, että ainoa hyvyysarvo, joka on parantunut hieman staattiseen vertaisverkkoon nähden, on BFS TTL-6 -leveyshaulla suoritettu testi Grid-verkossa. Topologiakyselyitä suoritetaan keskimäärin kierrosten aikana paljon, mutta topologian muutoksia ei ollenkaan. Neuroverkko toisin sanoen tulkitsee usein, että solmujen tulisi pyytää uusia naapureita, mutta ei sitten kuitenkaan muodosta uusia yhteyksiä, kun alkuperäinen topologiakyselijä arvioidaan samalla neuroverkolla. HDS arvolla R=25 -tapauksella opetettua neuroverkkoa ei kannata käyttää minkäänlaisessa vertaisverkkoympäristössä.

6.3.1.2 HDS, R=75

Haku-algo-ritmi	Lähtö-topologia	Hyvyys	Neuro-verkon vaiku-tus	Pake-tit	Vasta-ukset	Epä-onnis-tuneet ky-selyt	Topo-Pake-tit	Topo-Vasta-ukset	Vies-tien mat-ka	Tehok-kuus
HDS	Grid	33289.0	10.7%	3306	827	3	550	401	33.06	0.194
RW	Grid	29442.0	-3.3%	4128	763	17	528	388	41.29	0.151
BFS-3	Grid	12173.0	18.2%	5202	426	74	434	351	2.81	0.071
BFS-6	Grid	2754.0	-78.7%	6126	250	74	393	331	3.49	0.036
HDS	Power	33532.0	-12.3%	3358	832	3	546	396	33.58	0.193
RW	Power	30388.0	-16.0%	4117	789	12	562	427	41.18	0.154
BFS-3	Power	13029.0	-28.8%	5546	451	71	433	362	2.84	0.071
BFS-6	Power	2687.0	-69.7%	6203	247	76	377	315	3.65	0.035
HDS	Random	32791.0	16.2%	3409	820	6	555	405	34.09	0.187
RW	Random	30040.0	15.3%	4060	776	15	538	402	40.61	0.155
BFS-3	Random	13252.0	-31.5%	5473	452	72	422	353	2.83	0.072
BFS-6	Random	2775.0	-82.8%	6165	249	75	378	324	3.68	0.036

Taulukko 6.4: HDS arvolla R=75 -neuroverkon tulokset eri testitapauksissa.

HDS arvolla R=75 -tapauksen neuroverkko ei myöskään näytä taulukon 6.4 perusteella kovin lupaavalta. Hyvyysarvot huononevat huomattavasti suurimmassa osassa tutkimusympäristöjä. Tämä johtuu suuresta määrästä topologian muutoksiin käytettävästä liikenteestä. Kävelijäalgoritmien resurssien haku on kuitenkin tehostunut huomattavasti lukuun ottamatta potenssijakautunutta verkkoa. Esimerkiksi staattisessa Grid-verkossa satunnaiskävelijä käytti 4501 pakettia 699 resurssin löytämiseen, kun nyt kuluu keskimäärin 4128 pakettia 763 resurssin löytämiseen. Myöskin kävelijäalgoritmien viestien kulkema matka on pienentynyt. Esimerkiksi HDS-haun viesti kulkee staattisessa Grid-verkossa keskimäärin 47.93 solmun läpi ja kuitenkin 24% viesteistä ei onnistu tavoitteessaan. Tämän neuroverkon muodostamassa topologiassa viestin kulkema matka on kokoluokkaa 34 ja viesteistä epäonnistuu keskimäärin 3%. Topologian muutoksista johtuva liikenne kuitenkin huonontaa itse hyvyysarvoa. Kuvassa 6.15 näkyy, kuinka neuroverkko ei osaa lopettaa topologian muutoksia kierrosten edetessä, kun hakuna käytetään HDS-hakua. Vertaisverkko ei siis konvergoidu.



Kuva 6.15: Vertaisverkon konvergoituminen, kun topologiaa hallitaan HDS arvolla R=75 -neuroverkolla ja resursseja etsitään korkeimman asteen haulla. Vertaisverkko ei konvergoidu, vaan sen topologia muuttuu kierroksesta toiseen.

6.3.1.3 HDS, R=100

Haku-algo-ritmi	Lähtö-topologia	Hyvyys	Neuro-verkon vaiku-tus	Pake-tit	Vasta-ukset	Epä-onnis-tuneet ky-selyt	Topo-Pake-tit	Topo-Vasta-ukset	Vies-tien mat-ka	Tehok-kuus
HDS	Grid	35595.0	18.4%	4180	816	5	167	38	41.81	0.186
RW	Grid	32482.0	6.7%	4258	759	15	202	40	42.6	0.168
BFS-3	Grid	15849.0	53.8%	2676	392	83	171	44	2.91	0.135
BFS-6	Grid	11834.0	-8.7%	4086	339	75	167	39	3.86	0.078
HDS	Power	37476.0	-1.9%	3419	838	2	164	37	34.19	0.231
RW	Power	34610.0	-4.4%	4135	797	10	188	33	41.36	0.182
BFS-3	Power	19381.0	5.9%	4259	493	74	174	28	2.9	0.11
BFS-6	Power	11399.0	28.5%	6546	373	69	118	23	3.99	0.055
HDS	Random	36129.0	28.1%	3586	818	6	194	43	35.87	0.213
RW	Random	32755.0	25.8%	4235	767	17	232	40	42.36	0.17
BFS-3	Random	20853.0	7.8%	3172	501	82	170	35	2.9	0.148
BFS-6	Random	16157.0	0.1%	5958	462	60	166	31	4.47	0.075

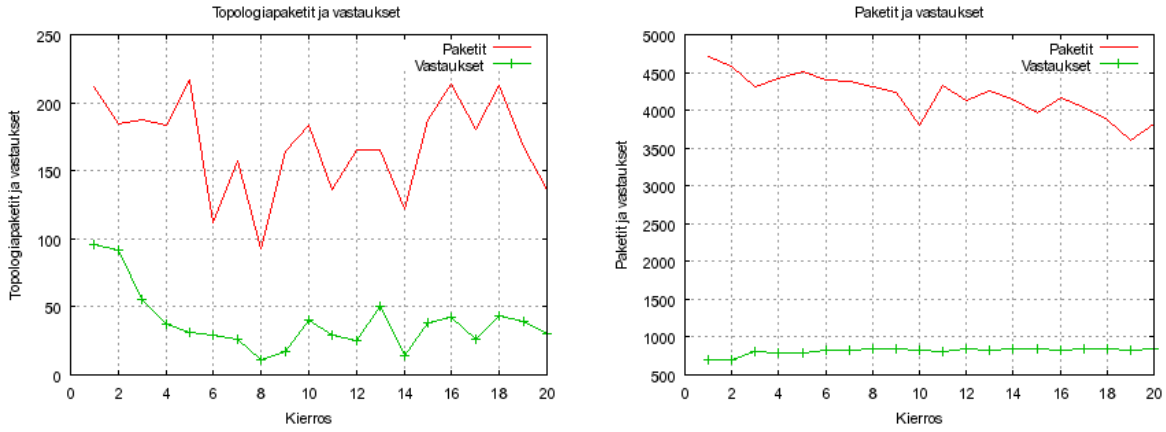
Taulukko 6.5: HDS arvolla R=100 -neuroverkon tulokset eri testitapauksissa.

HDS arvolla R=100 -tapauksella opetettu neuroverkko näyttää taulukon 6.5 perusteella melko lupaavalta. Erityisesti Grid-verkkoa sekä satunnaista verkkoa on onnistuttu tehostamaan kävelijäalgoritmeille huomattavasti. Keskimääräisesti topo-

logian muutoksiin käytettävä liikenne on pienentynyt selvästi kahteen ensimmäiseen tapaukseen verrattuna. Erityisesti topologiapakettien määrän pieneneminen viittaisi siihen, että neuroverkon mielestä solmujen ei ole syytä etsiä uusia naapureita niin paljoa. On syytä kuitenkin tarkistaa topologian muutoksiin tarvittava liikenne kierroskohtaiseti. Kuvassa 6.16 nähdään, että HDS-hakua käytettäessä vertaisverkossa tapahtuu pientä konvergoitumista, mutta topologian muutokset jatkuvat vielä 20 kierroksen jälkeenkin.

Leveyshaku TTL-arvolla 3 on parantunut prosentuaalisesti paljon, kun on käytetty Grid-verkkoa lähtötopologiana. On kuitenkin huomattava, että Grid-verkko on tälle hakualgoritmille todella huono ympäristö, joten sen parantaminen 53.8% ei ole välttämättä mikään tieteellinen läpimurto. BFS-3 on pärjännyt parhaiten, kun lähtötopologia on ollut satunnainen verkko ja silloinkin sen sadasta kyselystä on epäonnistunut keskimäärin 82%.

Leveyshaku TTL-arvolla 6 on parantunut myöskin prosentuaalisesti melko paljon, kun lähtötopologiana on ollut potenssijakautunut verkko. Kuitenkin hyvyysarvolla mitattuna hakualgoritmi saavuttaa parhaan tuloksensa, kun on lähdetty liikkeelle satunnaisesta verkosta.



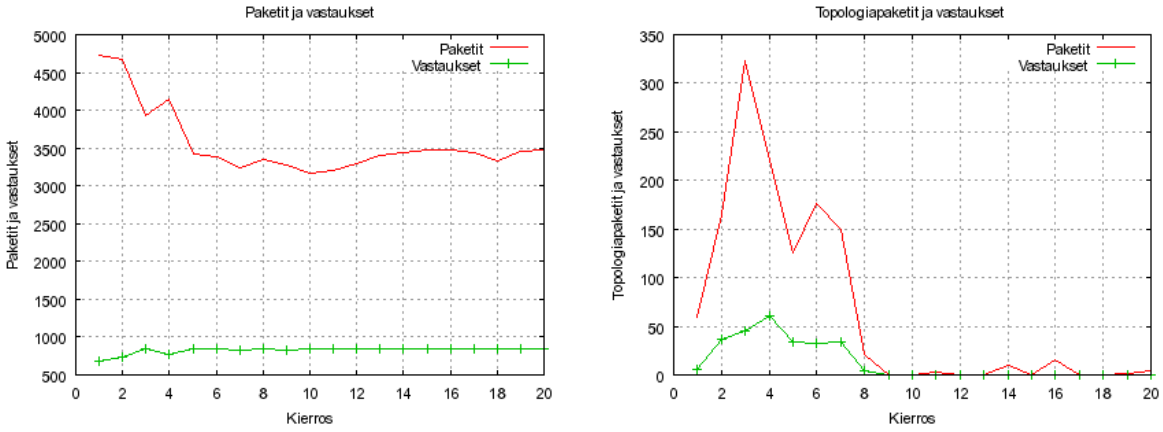
Kuva 6.16: HDS arvolla R=100 -neuroverkko osoittaa topologiavastausten osalta pientä konvergoitumista, kun resursseja etsitään korkeimman asteen haulla. Täysin tyytyväinen neuroverkko ei ole missään vaiheessa.

6.3.1.4 HDS, R=175

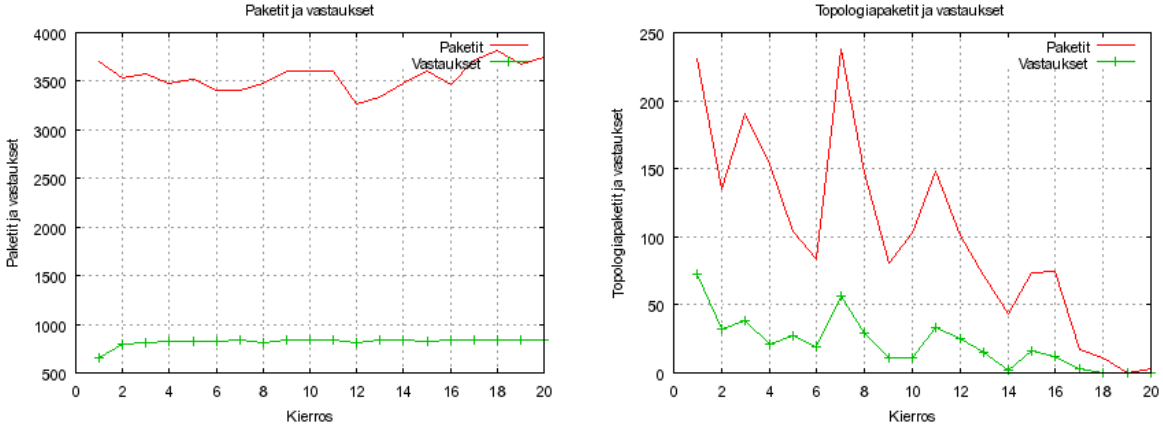
Haku-algo-ritmi	Lähtö-topologia	Hyvyys	Neuro-verkon vaiku-tus	Pake-tit	Vasta-ukset	Epä-onnis-tuneet ky-selyt	Topo-Pake-tit	Topo-Vasta-ukset	Vies-tien mat-ka	Tehok-kuus
HDS	Grid	37284.0	24.0%	3556	824	4	60	12	35.56	0.227
RW	Grid	32659.0	7.3%	4176	760	16	194	39	41.78	0.172
BFS-3	Grid	11444.0	11.1%	2686	283	96	3	1	2.92	0.105
BFS-6	Grid	12358.0	-4.6%	5037	348	67	1	0	3.6	0.069
HDS	Power	38336.0	0.3%	3179	840	1	85	12	31.79	0.256
RW	Power	34645.0	-4.3%	4020	797	9	206	31	40.21	0.187
BFS-3	Power	22268.0	21.7%	4827	561	63	161	30	2.93	0.111
BFS-6	Power	9125.0	2.9%	6685	348	71	266	52	3.89	0.049
HDS	Random	36972.0	31.1%	3553	822	5	95	20	35.53	0.224
RW	Random	34215.0	31.4%	4320	788	11	147	26	43.21	0.175
BFS-3	Random	24329.0	25.8%	3836	585	69	179	38	2.96	0.144
BFS-6	Random	14130.0	-12.4%	6365	440	66	253	48	4.51	0.066

Taulukko 6.6: HDS arvolla R=175 -neuroverkon tulokset eri testitapauksissa.

HDS arvolla R=175 -tapauksessa neuroverkon antamat tulokset näyttävät taulukon 6.6 perusteella lupaavilta. Neuroverkko on parantanut kaikkia muita hakualgoritmeja kaikista lähtötopologioista lukuun ottamatta leveyshakua TTL-arvolla 6 Grid-verkossa ja satunnaisessa verkossa sekä satunnaiskävelijää potenssijautuneessa verkossa. HDS-hakualgoritmilla on päästy jo tilanteeseen, missä lähes kaikki kyselyt ovat onnistuneet käyttäen suhteellisen vähän paketteja. Myöskin topologian muutokset ovat 20 kierroksen aikana keskimäärin vähäisiä. Tämä viittaisi vertaisverkon konvergoitumiseen. Epäilyksen todistaa oikeaksi kuva 6.17. Kahdeksan kierroksen jälkeen, kun lähtötopologia on Grid-verkko, topologian muutoksia ei tapahdu enää ja resurssivastausten määrä on saavuttanut maksimitasonsa eli 845/kierros. Kun lähtötopologiana käytetään satunnaista vertaisverkkoa, niin konvergoituminen tapahtuu kuvan 6.18 mukaisesti vasta kierroksella 19.



Kuva 6.17: HDS arvolla R=175 -neuroverkko konvergoi alkuperäisen Grid-verkon 9 kierroksen aikana mieleisekseen, kun resursseja etsitään korkeimman asteen haulla. Resurssikyselyjen pakettimäärä osoittaa, että uusi vertaisverkko on tehokkaampi kuin alkuperäinen Grid-verkko.



Kuva 6.18: Potenssijakautunut verkko tuottaa HDS arvolla R=175 -neuroverkolle haasteen, mutta topologian muutokset loppuvat kuitenkin ennen 20. kierrosta. Itse vertaisverkko ei kuitenkaan tehostu alkuperäisesta potenssijakautuneesta verkosta. Hakualgoritmina on korkeimman asteen haku.

6.3.1.5 HDS, R=250

Haku-algo-ritmi	Lähtö-topologia	Hyvyys	Neuro-verkon vaiku-tus	Pake-tit	Vasta-ukset	Epä-onnis-tuneet ky-selyt	Topo-Pake-tit	Topo-Vasta-ukset	Vies-tien mat-ka	Tehok-kuus
HDS	Grid	37999.0	26.4%	3261	831	2	46	12	32.61	0.25
RW	Grid	34884.0	14.6%	4271	796	10	113	16	42.72	0.18
BFS-3	Grid	10302.0	0.0%	2598	258	99	0	0	2.92	0.099
BFS-6	Grid	12945.0	-0.1%	4445	348	74	2	0	3.78	0.078
HDS	Power	38377.0	0.4%	3103	839	1	81	13	31.04	0.262
RW	Power	36655.0	1.3%	3980	827	3	112	31	39.8	0.2
BFS-3	Power	25027.0	36.8%	5533	627	49	131	27	2.93	0.11
BFS-6	Power	9752.0	9.9%	6883	354	66	165	48	4.01	0.049
HDS	Random	38055.0	34.9%	3090	829	3	45	16	30.9	0.263
RW	Random	35155.0	35.0%	4065	800	9	115	41	40.66	0.189
BFS-3	Random	25139.0	30.0%	4471	607	59	120	28	2.98	0.131
BFS-6	Random	12650.0	-21.6%	6465	400	66	158	19	4.3	0.06

Taulukko 6.7: HDS arvolla R=250 -neuroverkon tulokset eri testitapauksissa.

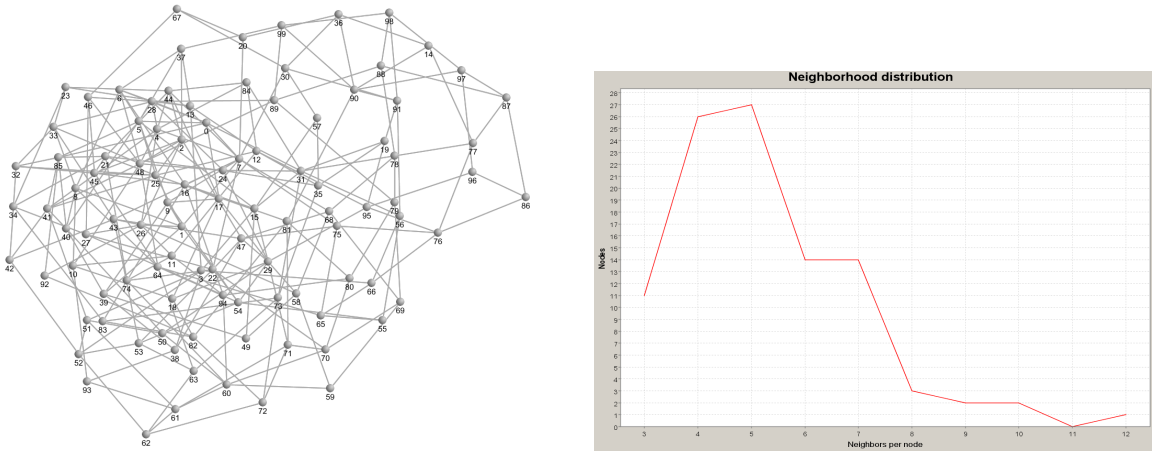
HDS arvolla R=250 -tapauksen tuottama neuroverkko on tähän mennessä paras. Taulukon 6.7 arvoista päätellen se on onnistunut konvergoimaan vertaisverkkotopologiat HDS-haulle ja satunnaiskävelijälle, oli lähtötopologiana mikä tahansa kolmesta testausalustasta. Kuvassa 6.19 on alunperin Grid-verkosta muodostunut topologia, kun hakuna on HDS. Muista lähtötopologioista tuotetut topologiat ovat suoritusarvoiltaan samanlaisia, mutta alun perin satunnaisen verkon naapurijakaumasta on jäänyt piirteitä myös uuteen kuvan 6.20 topologiaan. Kuitenkin tämä topologia on hyvyysarvoltaan yhtä hyvä kuin kahdessa muussakin lähtötopologian tapauksessa. Kuvaaja tämän verkon konvergoitumisesta HDS-haun tapauksessa on kuvassa 6.21. HDS-hakua käyttävä alunperin Grid-verkko ei muutu enää 11 kierroksen jälkeen. Myös muut tapaukset konvergoituvat kierroksilla 10-12 lukuun ottamatta satunnaiskävelijää satunnaisessa vertaisverkossa, jossa konvergoituminen tapahtuu kierroksella 16.

Satunnaiskävelijällä kaikki lähtötopologiat konvergoituvat samanlaisiksi vertaisverkoiksi. Kuvassa 6.22 on alunperin satunnaisesta verkosta muodostunut topologia, kun hakualgoritmina on satunnaiskävelijä.

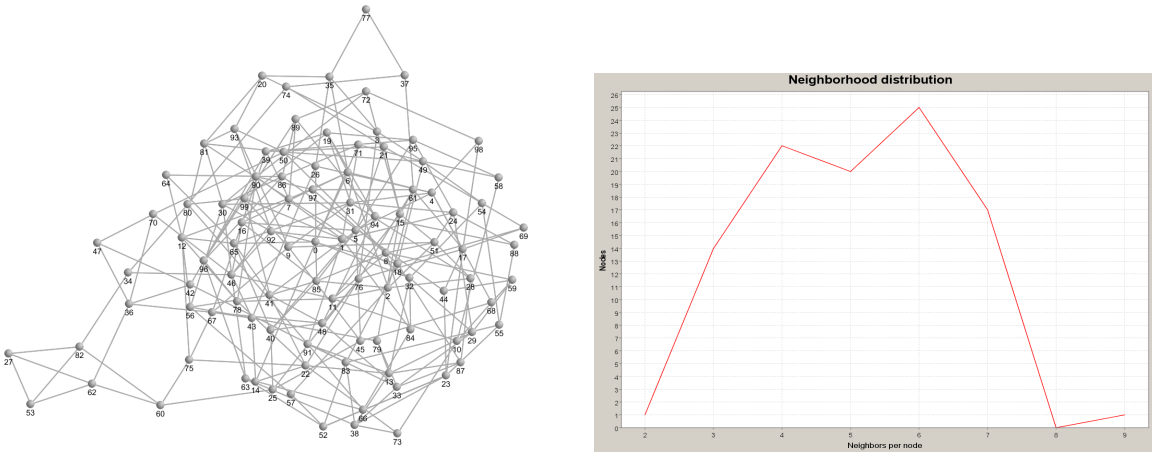
Leveyshaulla TTL-arvolla 3 ei tapahdu mitään muutoksia Grid-verkosta liikkeelle lähdettäessä. Tapausta on testattu moneen kertaan, mutta jostain syystä neuro-

verkko ei näe syytä topologian muutoksille eikä näin ollen lähde käyntiin ollenkaan. Kun lähtötopologiana käytetään potenssijakautunutta tai satunnaista verkkoa, niin vertaisverkko konvergoituu, mutta lopputulokset eivät ole samanlaisia topologioita. Muodostetut topologiat ovat kuvissa 6.23 ja 6.24. On mielenkiintoista, että satunnaisesta verkosta muodostettuihin uusiin topologioihin on jäänyt piirteitä alkuperäisestä naapurijakaumasta, vaikka ensimmäisessä (kuva 6.20) on käytetty HDS-hakua ja toisessa (kuva 6.24) on käytetty BFS-3 leveyshakua. Esimerkki konvergoitumisesta, kun lähtötopologiana käytetään potenssijakautunutta verkkoa, on kuvassa 6.25. Leveyshaku TTL-arvolla 6 ei konvergoidu millään HDS arvolla R=250 -neuroverkon testaustapauksella.

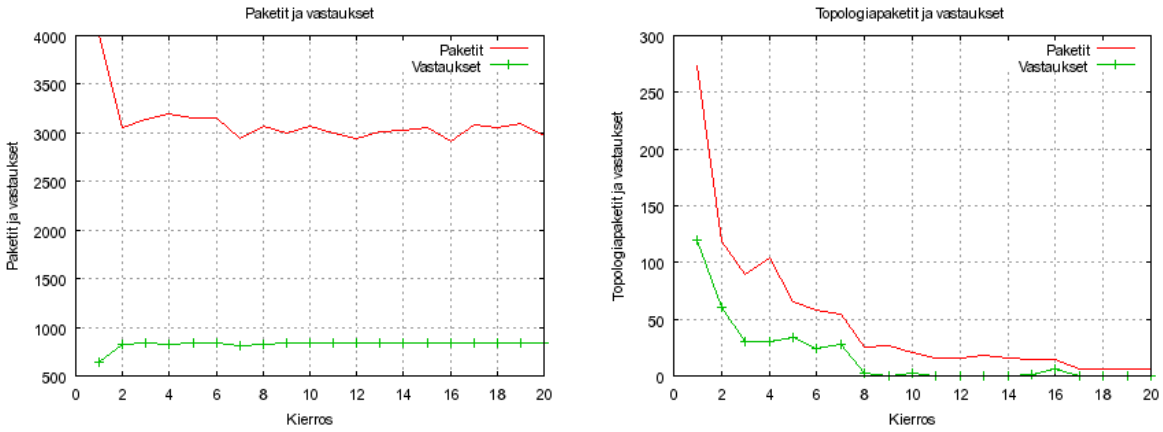
Kävelijäalgoritmeilla tuotetuissa topologioissa on melko paljon potenssijakautuneisuuden piirteitä. Pieninumeroiset solmut, joilla on eniten resursseja, ovat kulkeneet verkon keskelle ja naapurijakautuneisuudessa pienellä osalla solmuja on suurin määrä naapureita. Leveyshaulla tuotetuissa topologioissa solmut eivät ole muodostaneet verkon keskelle niin selvää ryhmittymää ja naapurijakauma muistuttaa enemmän normaalijakaumaa kuin potenssijakaumaa.



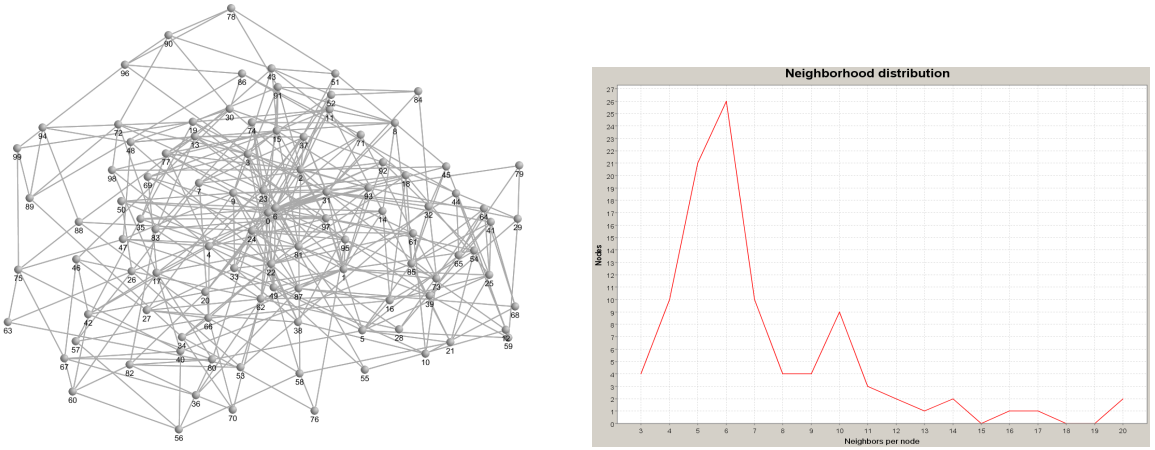
Kuva 6.19: HDS arvolla R=250 -neuroverkon konvergoima vertaisverkko, kun käytetään korkeimman asteen hakua ja lähtötopologiana on ollut Grid-verkko. Naapurijakaumassa on potenssijakautuneisuuden piirteitä.



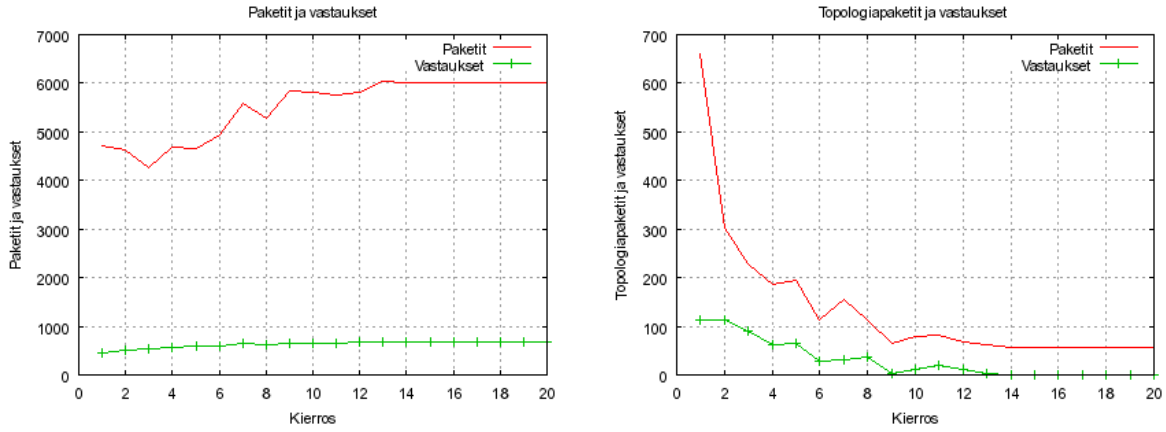
Kuva 6.20: HDS arvolla R=250 -neuroverkon konvergoima vertaisverkko, kun käytetään korkeimman asteen hakua ja lähtötopologiana on ollut satunnainen verkko. Naapurijakaumaan on jäänyt piirteitä satunnaisesta verkosta ja se on melko lailla normaalijakautunut.



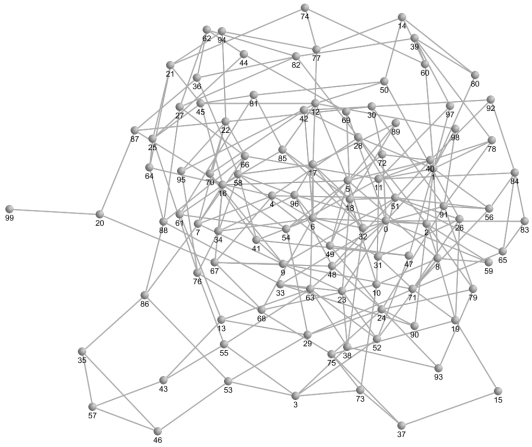
Kuva 6.21: HDS arvolla R=250 -neuroverkko muuttaa satunnaisen verkon mieleisekseen 8 kierroksen aikana. Kaikkien resurssien löytäminen 3000 paketilla on hyvä suoritus korkeimman asteen haulla.

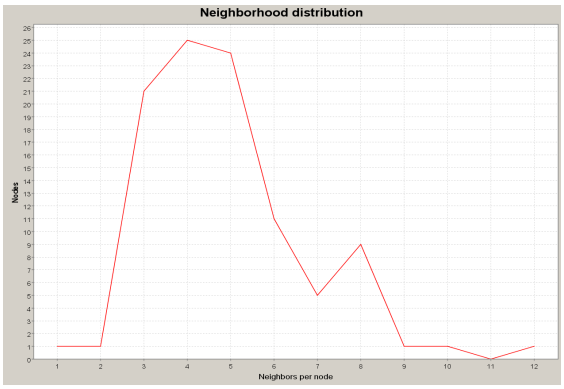


Kuva 6.22: HDS arvolla R=250 -neuroverkon konvergoima verkko satunnaiskävelijälle, kun lähtötopologiana on ollut satunnainen verkko. Potenssijakautuneen verkon piirteet ovat selvästi nähtävissä.

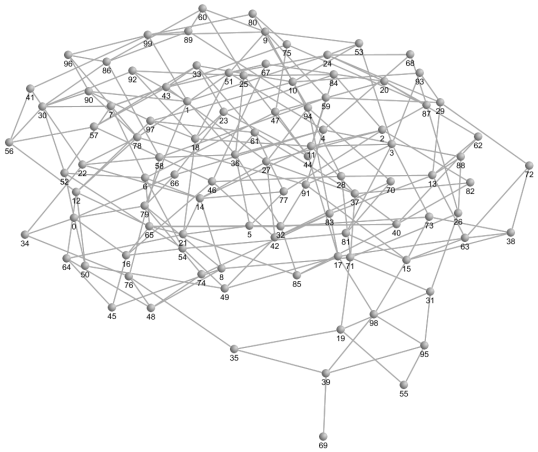


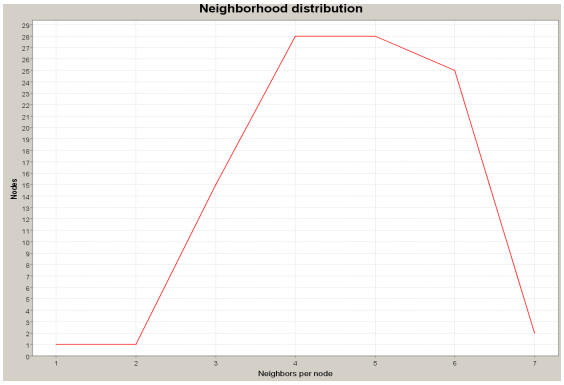
Kuva 6.25: HDS arvolla R=250 -neuroverkko leveyshaun apuna, kun lähtötopologiana on Grid-verkko. TTL-arvolla 3 toimiva hakualgoritmi leviää pidemmälle neuroverkon muodostamassa verkossa, jolloin resurssien haussa käytettävä pakettimäärä kasvaa.





Kuva 6.23: HDS arvolla R=250 -neuroverkon muodostama vertaisverkko TTL-3 leveyshaulle, kun on lähdetty liikkeelle potenssijakautuneesta verkosta. Potenssijakautuneisuuden piirteitä näkyy vielä, mutta verkko alkaa jo muistuttamaan enemmän normaalijakautunutta verkkoa.





Kuva 6.24: HDS arvolla R=250 -neuroverkon muodostama vertaisverkko TTL-3 -leveyshaulle, kun on lähdetty liikkeelle satunnaisesta verkosta. Naapurijakauma on melko lailla normaalijakautunut.

6.3.1.6 Random Walker, R=100

Haku-algo-ritmi	Lähtö-topologia	Hyvyys	Neuro-verkon vaiku-tus	Pake-tit	Vasta-ukset	Epä-onnis-tuneet ky-selyt	Topo-Pake-tit	Topo-Vasta-ukset	Vies-tien mat-ka	Tehok-kuus
HDS	Grid	36063.0	20.0%	2882	832	1	443	88	28.83	0.243
RW	Grid	33754.0	10.9%	4016	810	6	452	94	40.17	0.177
BFS-3	Grid	18809.0	82.6%	6091	561	54	504	126	2.89	0.083
BFS-6	Grid	2423.0	-81.3%	6537	235	74	371	187	3.17	0.033
HDS	Power	36547.0	-4.4%	2968	843	0	446	81	29.68	0.241
RW	Power	34673.0	-4.2%	4147	841	1	505	141	41.47	0.175
BFS-3	Power	21686.0	18.5%	6619	620	50	428	111	2.92	0.086
BFS-6	Power	964.0	-89.1%	6576	208	76	383	189	3.17	0.029
HDS	Random	34885.0	23.7%	3330	834	2	547	150	33.3	0.207
RW	Random	34263.0	31.5%	4092	824	4	481	88	40.93	0.176
BFS-3	Random	21591.0	11.6%	6544	621	48	470	113	2.93	0.087
BFS-6	Random	5227.0	-67.6%	6828	291	70	348	151	3.51	0.039

Taulukko 6.8: Random Walker arvolla R=100 -neuroverkon tulokset eri testitapauksissa.

RW arvolla R=100 -tapauksella opetettu neuroverkko tuottaa Grid-verkosta ja satunnaisesta verkosta kohtalaisen hyvät topologiat kävelijäalgoritmeille, kuten taulukosta 6.8 nähdään. HDS-haku saavuttaa topologian, jossa kaikki kyselyt onnistuvat ja satunnaiskävelijä pärjää myös hyvin. Tulokset eivät kuitenkaan yllä HDS arvolla R=250 -tapauksen neuroverkolle, sillä satunnaiskävelijän opettama neuroverkko ei konvergoidu, vaan topologian muutokset jatkuvat kierroksesta toiseen. Leveyshakujen tapauksessa TTL=3 -haku paranee kaikilla lähtötopologioilla, mutta TTL=6 -haku menettää vähäisenkin suorituskykynsä. Tämän neuroverkon ongelma on, että se ei konvegoidu ja topologian muutokset aiheuttavat paljon liikennettä.

6.3.1.7 Random Walker, R=175

Haku-algo-ritmi	Lähtö-topologia	Hyvyys	Neuro-verkon vaiku-tus	Pake-tit	Vasta-ukset	Epä-onnis-tuneet ky-selyt	Topo-Pake-tit	Topo-Vasta-ukset	Vies-tien mat-ka	Tehok-kuus
HDS	Grid	32690.0	8.8%	4145	828	2	673	240	41.46	0.163
RW	Grid	32178.0	5.7%	4482	820	4	637	231	44.82	0.153
BFS-3	Grid	11090.0	7.6%	6685	447	67	613	302	2.71	0.058
BFS-6	Grid	963.0	-92.6%	6647	242	72	590	308	3.12	0.032
HDS	Power	34276.0	-10.3%	3934	842	0	588	190	39.34	0.178
RW	Power	33189.0	-8.3%	4371	832	2	603	205	43.71	0.16
BFS-3	Power	11965.0	-34.6%	6785	462	67	603	267	2.76	0.06
BFS-6	Power	2271.0	-74.4%	6629	263	72	556	294	3.36	0.035
HDS	Random	33163.0	17.6%	4022	832	2	667	216	40.22	0.169
RW	Random	32412.0	24.4%	4423	823	4	632	231	44.24	0.155
BFS-3	Random	13131.0	-32.1%	6639	486	66	617	289	2.81	0.064
BFS-6	Random	2564.0	-84.1%	6491	265	72	540	299	3.51	0.036

Taulukko 6.9: Random Walker arvolla R=175 -neuroverkon tulokset eri testitapauksissa.

Taulukon 6.9 perusteella RW arvolla R=175 -tapauksella opetettu neuroverkko häviää kaikilla mittareilla RW arvolla R=100 -neuroverkolle. Topologian muutoksiin käytettävät kyselyt ja vastaukset ovat lukumäärältään melko suuria, mutta vastausten määrän ja pakettien määrän välinen suhde on parempi kuin RW arvolla R=100 -tapauksessa. Ilmeisesti parametrin R arvo on asetettu liian suureksi ja resurssien löytämisestä saatava palkinto on niin suuri, että neuroverkko tulkitsee topologian muutokset kannattaviksi liian usein. Tästä johtuen vertaisverkon konvergoitumista ei tapahdu.

6.3.1.8 BFS, TTL=3, R=175

Haku-algo-ritmi	Lähtö-topologia	Hyvyys	Neuro-verkon vaiku-tus	Pake-tit	Vasta-ukset	Epä-onnis-tuneet ky-selyt	Topo-Pake-tit	Topo-Vasta-ukset	Vies-tien mat-ka	Tehok-kuus
HDS	Grid	27920.0	-7.1%	3440	827	2	1592	406	34.41	0.152
RW	Grid	25578.0	-16.0%	4132	794	11	1607	391	41.33	0.129
BFS-3	Grid	17816.0	72.9%	4779	626	51	1365	376	2.94	0.096
BFS-6	Grid	8685.0	-33.0%	6285	417	64	896	280	4.05	0.055
HDS	Power	28874.0	-24.4%	3316	839	1	1564	388	33.16	0.159
RW	Power	26033.0	-28.1%	4017	796	10	1560	390	40.18	0.133
BFS-3	Power	19223.0	5.1%	4987	659	47	1382	366	2.95	0.097
BFS-6	Power	8993.0	1.4%	6357	426	66	907	283	4.17	0.056
HDS	Random	27956.0	-0.9%	3299	826	4	1605	404	33.0	0.155
RW	Random	26177.0	0.5%	4053	794	10	1507	387	40.54	0.133
BFS-3	Random	19544.0	1.1%	5081	664	46	1354	361	2.94	0.097
BFS-6	Random	9880.0	-38.8%	6225	446	64	963	276	4.2	0.059

Taulukko 6.10: BFS-3 arvolla R=175 -neuroverkon tulokset eri testitapauksissa.

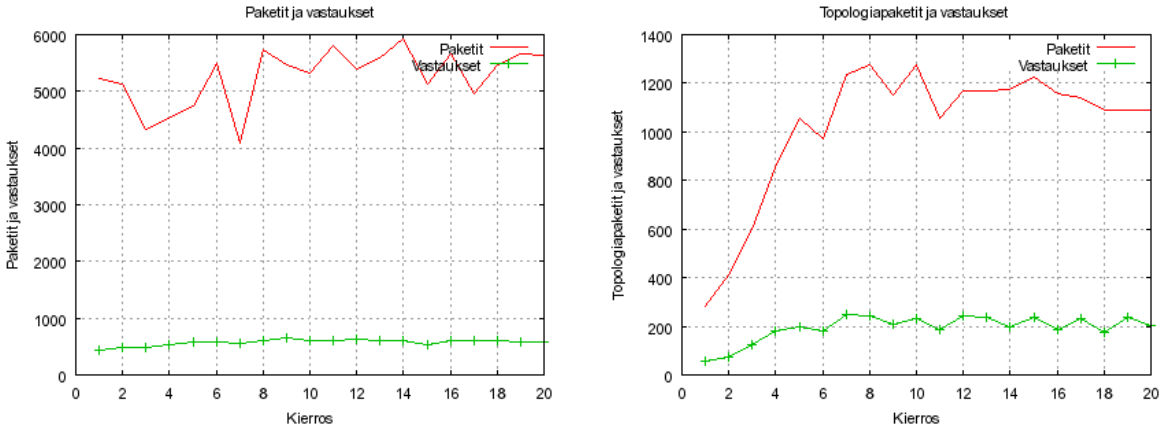
BFS-3 arvolla R=175 -tapauksessa opetettu neuroverkko parantaa vertaisverkkoympäristöjä, joissa käytetään hakualgoritmina samaa BFS-3 -hakua, kuten taulukosta 6.10 näkyy. Muissa tapauksissa suoritusarvot käytännössä huononevat. Lisäksi BFS-3 -haun paraneminen ei yllä sille tasolle, joka saavutettiin jo HDS arvolla R=250 -neuroverkon tapauksessa eikä tällä neuroverkolla vertaisverkot konvergoidu, kuten HDS arvolla R=250 -neuroverkon tapauksessa, vaan topologiapakettien määrä ja topologiavastausten pysyy suurena.

6.3.1.9 BFS, TTL=6, R=175

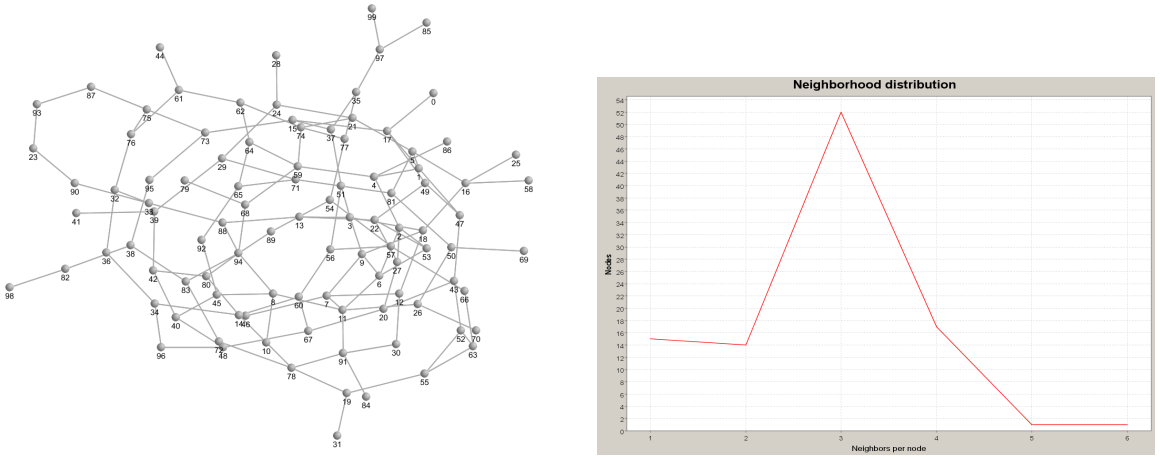
Haku-algo-ritmi	Lähtö-topologia	Hyvyys	Neuro-verkon vaiku-tus	Pake-tit	Vasta-ukset	Epä-onnis-tuneet ky-selyt	Topo-Pake-tit	Topo-Vasta-ukset	Vies-tien mat-ka	Tehok-kuus
HDS	Grid	27908.0	-7.2%	3452	793	9	1386	272	34.54	0.155
RW	Grid	23345.0	-23.3%	3865	702	28	1328	250	38.68	0.128
BFS-3	Grid	15420.0	49.7%	2420	491	73	1068	274	2.9	0.13
BFS-6	Grid	16756.0	29.3%	5119	559	51	1017	198	5.01	0.088
HDS	Power	28528.0	-25.4%	3412	801	9	1360	262	34.13	0.159
RW	Power	23189.0	-35.9%	3786	698	29	1344	241	37.88	0.129
BFS-3	Power	15924.0	-13.0%	2481	522	78	1277	262	2.97	0.129
BFS-6	Power	16550.0	86.5%	5620	566	51	1017	209	5.05	0.082
HDS	Random	26806.0	-5.0%	3589	768	16	1343	258	35.91	0.147
RW	Random	23004.0	-11.7%	3851	692	32	1301	248	38.54	0.128
BFS-3	Random	15828.0	-18.2%	2297	514	78	1249	266	2.98	0.134
BFS-6	Random	17350.0	7.5%	5250	574	50	1026	194	5.09	0.088

Taulukko 6.11: BFS-6 arvolla R=175 -neuroverkon tulokset eri testitapauksissa.

BFS-6 arvolla R=175 -tapauksella opetettu neuroverkko on siinä mielessä mielenkiintoinen, että viimein löydettiin neuroverkko, jolla leveyshakua TTL-arvolla 6 saadaan parannettua. Lisäksi taulukosta 6.11 nähdään, että Grid-verkon BFS-3 -tapausta lukuun ottamatta muut hakulgoritmit huononevat tällä neuroverkolla. Tämä tarkoittaa sitä, että hakualgoritmina suuren TTL-arvon leveyshaku on niin erilainen kuin kävelijäalgoritmit, että se ei yksinkertaisesti toimi samanlaisessa topologiassa kuin muut haut. Sitä, minkälainen topologia olisi TTL-arvon 6 -leveyshaulle paras mahdollinen, ei tämäkään neuroverkko tiedä, sillä se ei saa konvergoitua mitään lähtötopologiaa, vaan topologian muutokset jatkuvat kuvan 6.26 mukaisesti tasaisena kierroksesta toiseen. Kuvassa 6.27 on neuroverkon tuottama topologia 20 kierroksen jälkeen, kun lähtötopologiana on ollut potenssijakautunut verkko. Verkon solmuilla on keskimäärin melko vähän naapureita. Naapurijakuma on suunnilleen normaalijakautunut. Leveyshaku ei vaadi solmuilta suuria naapurimääriä, sillä se leviää verkossa tulvimisesta johtuen pienemmällä pakettimäärällä, jos naapurijakauma on tasainen verrattuna esimerkiksi potenssijakaumaan.



Kuva 6.26: BFS-6 arvolla R=175 -neuroverkko ei osaa lopettaa vertaisverkon muuttamista TTL-6 -leveyshaulle. Hakualgoritmi löytää hiukan yli puolet tavoitelluista resursseista ennenkuin vertaisverkon solmut ylikuormittuvat.



Kuva 6.27: BFS-6 arvolla R=175 -neuroverkon muodostama vertaisverkko 20 resurssikysely-topologiamuutos -kierroksen jälkeen. Suurimmalla osasta solmuja on 3 naapuria. Leveyshaku etenee pienemmällä pakettimäärällä kun naapureita on vähemmän.

6.3.1.10 Yhteenveto opetetuista neuroverkoista

Taulukossa 6.12 on tiivistettynä kaikkien yhdeksän neuroverkon kaikki hyvyysarvot eri testaustapauksissa. Kunkin testaustapauksen paras hyvyysarvo on tummennettuna. HDS arvolla R=250 -neuroverkko saavutti parhaat hyvyysarvot kahdeksassa tapauksessa kahdestatoista. Leveyshakua TTL-arvolla 6 tämä neuroverkko ei onnistunut tehostamaan, mutta siinä ei onnistunut käytännössä muutkaan neurover-

kot lukuun ottamatta BFS-6 arvolla R=175 -neuroverkkoa, joka puolestaan paransi TTL-6 -leveyshakua, mutta heikensi samaan aikaan muita hakualgoritmeja.

Haku-algoritmi	**Lähtö-topologia**	**Ei neuro-verkkoa**	**HDS, R=25**	**HDS, R=75**	**HDS, R=100**	**HDS, R=175**	**HDS, R=250**
HDS	Grid	30059	29254	33289	35595	37284	**37999**
RW	Grid	30449	30364	29442	32482	32659	**34884**
BFS-3	Grid	10302	10267	12173	15849	11444	10302
BFS-6	Grid	12955	13081	2754	11834	12358	12945
HDS	Power	38216	35064	33532	37476	38336	**38377**
RW	Power	36193	33963	30388	34610	34645	**36655**
BFS-3	Power	18293	16143	13029	19381	22268	**25027**
BFS-6	Power	8872	6932	2687	11399	9125	9752
HDS	Random	28209	26827	32791	36129	36972	**38055**
RW	Random	26047	23951	30040	32755	34215	**35155**
BFS-3	Random	19340	19340	13252	20853	24329	**25139**
BFS-6	Random	16136	15126	2775	16157	14130	12650

Haku-algoritmi	**Lähtö-topologia**	**Ei neuro-verkkoa**	**RW, R=100**	**RW, R=175**	**BFS-3, R=175**	**BFS-6, R=175**
HDS	Grid	30059	36063	32690	27920	27908
RW	Grid	30449	33754	32178	25578	23345
BFS-3	Grid	10302	**18809**	11090	17816	15420
BFS-6	Grid	12955	2423	963	8685	**16756**
HDS	Power	38216	36547	34276	28874	28528
RW	Power	36193	34673	33189	26033	23189
BFS-3	Power	18293	21686	11965	19223	15924
BFS-6	Power	8872	964	2271	8993	**16550**
HDS	Random	28209	34885	33163	27956	26806
RW	Random	26047	34263	32412	26177	23004
BFS-3	Random	19340	21591	13131	19544	15828
BFS-6	Random	16136	5227	2564	9880	**17350**

Taulukko 6.12: Yhdeksän eri neuroverkon antamat hyvyysarvot, kun niitä on testattu kolmella eri lähtötopologialla ja jossa jokaisessa on käytetty neljää eri hakualgoritmia. HDS arvolla R=250 -neuroverkko on saavuttanut parhaat hyvyysarvot kahdeksassa tapauksessa kahdestatoista. Tämän neuroverkon käytöstä on hyötyä suurimmassa osassa tapauksia, vaikka topologian hallinta tuottaakin liikennettä.

HDS arvolla R=250 -neuroverkko oli siis selvästi paras ja se onnistui konvergoimaan hyvät topologiat korkeimman asteen haulle (kuva 6.19) ja satunnaiskävelijälle (kuva 6.22), kun käytettiin luvussa 6.1 esiteltyjä simulointiparametrejä. Kaikissa tapauksissa päästiin tavoitteisiin resurssien löytämisessä ja resurssipakettien määrä pieneni huomattavasti staattiseen verkkoon verrattuna. Myöskään topologian muutoksista johtuva liikenne ei syönyt saavutettua hyötyä, vaan hyvyysarvot paranivat. Lisäksi on erittäin lupaavaa, että vaikka käytettiin kolmea hyvin erilaista lähtötopologiaa, niin silti neuroverkko muodosti kyselysarjojen aikana suorituskyvyltään samanlaiset topologiat keskimäärin 12 kyselykierroksen aikana.

Jos vertaillaan nyt tuotettuja topologioita alkuperäisiin lähtötopologioihin niin, että niissä etsitään vain resursseja eikä tehdä topologian muutoksia, saadaan taulukossa 6.13 olevat tulokset. Sekä korkeimman asteen haku että satunnaiskävelijä toimivat huomattavasti tehokkaammin neuroverkon tuottamissa topologioissa. Molemmissa tapauksissa kaikki tavoitteeksi asetetut resurssit löytyvät ja käytettyjen pakettien määrä on pienempi kuin muissa topologioissa. Parhaan testituloksen löytyminen parametrin R arvoalueen suurimmasta päästä tarkoittaa sitä, että neuroverkon opetusta on syytä tutkia myös suuremmilla R:n arvoilla. Nythän ei tiedetä paraneeko tulokset esimerkiksi arvolla 300. Tämä on kuitenkin jätettävä jatkotutkimuksiin. Parametrien arvoalueen valinta tässä tutkielmassa on siis hiukan epäonnistunut.

Leveyshakujen kohdalta tilanne ei ole näin yksioikoinen. TTL-arvolla 3 toimiva leveyshaku tehostui keskimääräisesti parhaiten HDS arvolla R=250 -neuroverkon avulla. Lähtötopologiana Grid-verkko oli tälle neuroverkolle liian haastava, eikä topologian muutoksia suoritettu testin aikana yhtään kappaletta. Muilla neuroverkoilla topologian muutoksia suoritettiin mutta konvergoitumista ei tapahtunut. HDS arvolla R=250 -neuroverkko onnistui konvergoimaan vertaisverkkoa, kun lähtötopologia oli satunnainen verkko tai potenssijakautunut verkko ja sillä saavutettiin myös parhaat hyvyysarvot. Lähtötopologiat eivät kuitenkaan konvergoituneet samanlaisiksi verkoiksi, vaan satunnaisesta vertaisverkosta muodostettu uusi topologia (kuva 6.24) on tehokkaampi kuin potenssijakautuneesta muodostettu uusi topologia (kuva 6.23). Näin ollen ei voida sanoa, että löydettiin yksiselitteisesti paras topologia leveyshaulle TTL-arvolla 3. Taulukkoon 6.13 on listattu molempien topologioiden tehokkuus, kun niissä suoritetaan resurssikyselyjä ilman topologian muutoksia. Leveyshaku hyötyy neuroverkon käytöstä, sillä nyt resursseja löytyy keskimäärin 672 kappaletta, kun sama luku on potenssijakautuneessa verkossa ja satunnaisessa verkossa keskimäärin 465. Käytetyt pakettimäärät ovat suurempia. Satunnainen verkko käyttää 3350 pakettia 470 resurssin löytämiseen, kun uudessa

topologiassa kuluu suurimmillaan noin 6000 pakettia 669 resurssin löytämiseen.

Leveyshaku TTL-arvolla 6 on liian raskas hakualgoritmi näin pieneen vertaisverkkoon tässä tutkimuksessa annetuilla liikennerajoittimilla. Oikeastaan mikään neuroverkko ei parantanut hakualgoritmin suorituskykyä. BFS-6 arvolla R=175 -neuroverkko antoi paremmat hyvyysarvot, mutta sekään ei konvergoinut vertaisverkkoa missään vaiheessa, vaan topologian muutosten määrä säilyi suurena kierroksesta toiseen. Taulukossa 6.13 on staattisena versiona topologia, jonka BFS-6 arvolla R=175 -neuroverkko oli muodostanut 20 kyselykierroksen jälkeen. Epäonnistuneiden kyselyiden määrä on tippunut noin 74:stä 48:aan. Pakettien määrä on samaa suuruusluokkaa kuin satunnaisessa verkossa ja huomattavasti suurempi kuin Grid-verkossa. Löydettyjen resurssien määrä kuitenkin ratkaisee sen, että hyvyysarvo on parantunut melko paljon.

Haku-algoritmi	**Topologia**	**Hyvyys-arvo**	**Pake-tit**	**Vasta-ukset**	**Epä-onnis-tuneet kyse-lyt**	**Viestien matka**	**Tehok-kuus**
HDS	Grid	30059.0	4791	697	24	47.93	0.145
HDS	Power	38216.0	3634	837	6	36.34	0.23
HDS	Random	28209.0	3891	642	45	38.97	0.164
HDS	HDS, R=250	39332.0	2918	845	0	30.15	0.289
RW	Grid	30449.0	4501	699	24	45.08	0.155
RW	Power	36193.0	4507	814	7	45.07	0.18
RW	Random	26047.0	3603	593	50	36.07	0.164
RW	HDS, R=250	38187.0	4063	845	0	40.63	0.207
BFS-3	Grid	10302.0	2598	258	99	2.92	0.099
BFS-3	Power	18293.0	4707	460	71	2.86	0.097
BFS-3	Random	19340.0	3350	470	87	2.96	0.14
BFS-3	HDS, R=250 power	27429.0	6021	669	43	2.92	0.11
BFS-3	HDS, R=250 random	28492.0	5308	676	47	3.0	0.126
BFS-6	Grid	12955.0	4445	348	74	3.78	0.078
BFS-6	Power	8872.0	6178	301	77	3.52	0.048
BFS-6	Random	16136.0	5214	427	74	4.43	0.081
BFS-6	BFS6, R=175	22823.0	5627	569	48	5.09	0.10

Taulukko 6.13: Hakualgoritmien suorituskyky vertaisverkoissa, joissa ei tehdä topologian muutoksia.

7 Yhteenveto

Neuroverkkojen käyttö vertaisverkkojen topologian hallinnassa vaikuttaa tutkielmassa tehtyjen ensitestien perusteella lupaavalta. Korkeimman asteen haun avulla opetettu neuroverkko (HDS arvolla R=250) tehosti vertaisverkon toimintaa niissäkin tapauksissa, joissa hakualgoritmina käytettiin satunnaiskävelijää tai leveyshakua. Tässä tutkielmassa on kuitenkin vain esitelty yksi vertaisverkkoskenaario, jossa suurin osa parametreista on koko ajan samoja. Tarvitaan lisää tutkimusta erilaisissa vertaisverkoissa, joissa esimerkiksi solmujen määrä on suurempi ja kaistanleveydet sekä resurssit ovat eri tavoin jakautuneet.

Lisäksi itse neuroverkon rakennetta on syytä tarkastella myöhemmissä tutkimuksissa tarkemmin. Yksi mahdollisuus voisi olla evoluutiolgoritmien käyttö myös piilokerrosten ja neuroverkon neuronien määrän laskemiseen. Lisäksi sisääntuloarvoja on analysoitava yksityiskohtaisemmin, jolloin tiedetään paremmin, kuinka eri vertaisverkon ominaisuudet vaikuttavat sen topologiaan ja sen kautta tehokkuuteen.

Jatkotutkimusten tavoitteena on myöhemmin yhdistää neuroverkkoa käyttävä hakualgoritmi NeuroHaku [72] toimimaan NeuroTopologian kanssa yhtäaikaa.

Viitteet

[1] K. Aberer ja M. Hauswirth. An Overview on Peer-To-Peer Information Systems. Kirjassa *Records of the 4th International Distributed Data & Structures Meeting (WDAS 2002)*, 2002.

[2] L. Adamic, R. Lukose, ja B. Huberman. Local Search in Unstructured Networks. Kirjassa S. Bornholdt ja H.G. Schuster, toim., *Handbook of Graphs and Networks:From the Genome to the Internet*. Wiley-VCH, Berlin, 2000.

[3] E.M. Airoldi ja K.M. Carley. Sampling Algorithms for Pure Network Topologies: a Study on the Stability and the Separability of Metric Embeddings. *SIGKDD Explor. Newsl.*, 7-2:13–22, 2005.

[4] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen, ja J. Vuori. Chedar: Peer-to-Peer Middleware. Kirjassa *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006)*, 2006.

[5] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[6] H-G. Beyer ja S. Meyer-Nieberg. Self-Adaptation in Evolutionary Algorithms. Kirjassa F. Lobo, C. Lima, ja Z. Michalewicz, toim., *Parameter Setting in Evolutionary Algorithm*. Springer, Berlin, 2006.

[7] C. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[8] T. Blickle ja L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4-4:361–394, 1996.

[9] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Kirjassa *Communications of the ACM archive*, osa 13, ss. 422–426, 1970.

[10] J. Branke. Evolutionary Algorithms for Neural Network Design and Training. Kirjassa J.T. Alander, toim., *Proceedings of the 1st Nordic Workshop on Genetic Algorithms and its Applications*, osa 95-1, ss. 145–163, 1995.

[11] R. Buyya, H. Stockinger, J. Giddy, ja D. Abramson. Economic Models for Management of Resources in Peer-to-Peer and Grid Computing. Kirjassa *Proceedings of the SPIE International Conference on Commercial Applications for High-Performance Computing*, 2001.

[12] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, ja S. Shenker. Making Gnutella-like P2P Systems Scalable. Kirjassa *Proceedings of ACM SIGCOMM*, Elokuu 2003.

[13] Cheese Factory. <URL: http://tisu.it.jyu.fi/cheesefactory>. viitattu 3.7.2006.

[14] I. Clarke, O. Sandberg, B. Wiley, ja T. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval Service. Kirjassa *Proceedings of Workshop on Design Issues in Anonymity and Unobservability (ICSI)*, 2000.

[15] T. Condie, S. Kamvar, ja H. Garcia-Molina. Adaptive Peer-to-Peer Topologies. Kirjassa *Proceedings of the Fourth IEEE International Conference on Peer-To-Peer Computing*, 2004.

[16] B.F. Cooper ja H. Garcia-Molina. Ad hoc, Self-Supervising Peer-to-Peer Search Networks. Tekninen raportti, 2003.

[17] G. Coulouris ja J. Dollimore. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, 1991.

[18] A. Crespo ja H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. Kirjassa *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

[19] A. Crespo ja H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Tekninen raportti, 2002.

[20] D.B. Fogel. An Introduction to Simulated Evolutionary Optimization. Kirjassa D.B. Fogel ja L.J. Fogel, toim., *IEEE Transactions on Neural Networks: Special Issue on Evolutionary Computation*, osa 5-1, ss. 3–14, 1994.

[21] L.J. Fogel, A.J. Owens, ja M.J. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley Publishing, New York, 1966.

[22] J. Freeman ja D. Skapura. *Neural Networks: Algorithms, Applications and Programming Techniques*. Addison-Wesley Publishing Company, 1991.

[23] N. Ganguly ja A. Deutsch. Developing Efficient Search Algorithms for P2P Networks Using Proliferation and Mutation. Kirjassa *Artificial Immune Systems, Third International Conference, ICARIS*, Syyskuu 2004.

[24] N. García-Pedrajas, D. Ortiz-Boyer, ja C. Hervás-Martínez. An Alternative Approach for Neural Network Evolution with a Genetic Algorithm: Crossover by Combinatorial Optimization. *Neural Networks*, 19-4:514–528, 2006.

[25] Gnutella. <URL: http://www.gnutella.com/>. viitattu 4.7.2006.

[26] G.W. Greenwood, G.B. Fogel, ja M. Ciobanu. Emphasizing Extinction in Evolutionary Programming. Kirjassa *Proceedings of 2000 IEEE Congress on Evolutionary Computation*, ss. 666–671, 2000.

[27] D. Greiner, G. Winter, J.M. Emperador, ja B. Galván. Gray Coding in Evolutionary Multicriteria Optimization: Application in Frame Structural Optimum Design. Kirjassa C.A. Coello Coello, A.H. Aguirre, ja E. Zitzler, toim., *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*, ss. 576–591, 2005.

[28] P.J.B. Hancock. An empirical comparison of selection methods in evolutionary algorithms. Kirjassa *Evolutionary Computing, AISB Workshop*, ss. 80–94, 1994.

[29] Q. He, M. Ammar, G. Riley, H. Raj, ja R. Fujimoto. Mapping Peer Behavior to Packet-level Details: A Framework for Packet-level Simulation of Peer-to-Peer Systems. Kirjassa *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS 2003)*, 2003.

[30] D.O. Hebb. *The Organization of Behavior*. John Wiley Inc, New York, 1949.

[31] A. Hoffman. *Arguments on Evolution: A Paleontologist's Perspective*. Oxford University Press, New York, 1989.

[32] M. Iles ja D. Deugo. Adaptive Resource Location in a Peer-to-Peer Network. Kirjassa *The 16th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, Heinäkuu 2003.

[33] S. Joseph. An Extendible Open Source P2P Simulator. *P2P Journal*, 65:1–15, Marraskuu 2003.

[34] S. Joseph ja T. Hoshiai. Decentralized Meta-Data Strategies: Effective Peer-to-Peer Search. *IEICE Transactions on Communications*, E86-B6:1740–1753, 2003.

[35] M. Khambatti, K. Ryu, ja P. Dasgupta. Efficient Discovery of Implicitly Formed P2P Communities. *International Journal of Parallel and Distributed Systems and Networks*, 2002.

[36] N. Kotilainen, M. Vapa, A. Auvinen, M. Weber, ja J. Vuori. Peer-to-Peer Studio - Monitoring, Controlling and Visualization Tool for Peer-to-Peer Networks Research. Kirjassa *ACM International Workshop on Performance Monitoring, Measurement and Evaluation of Heterogeneous Wireless and Wired Networks*, 2006.

[37] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, ja J. Vuori. P2PRealm - Peer-to-Peer Network Simulator. Kirjassa *Proceedings of the 11th IEEE International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*, 2006.

[38] N. Kotilainen, M. Vapa, M. Weber, J. Töyrylä, ja J. Vuori. P2PDisco - Java Distributed Computing for Workstations Using Chedar Peer-to-Peer Middleware. Kirjassa *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2005)*, 2005.

[39] Y. Liu, L. Xiao, X. Liu, L. Ni, ja X. Zhang. Location Awareness in Unstructured Peer-to-Peer Systems. Kirjassa *IEEE Transactions on Parallel and Distributed Systems*, Helmikuu 2005.

[40] Q. Lv, P. Cao, E. Cohen, K. Li, ja S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. Kirjassa *Proceedings of the 16th International Conference on Supercomputing*. ACM Press, 2002.

[41] N. Lynch. *Distributed Algorithms*. Morgan Kauffman Publishers, 1996.

[42] G.D. Magoulas, M.N. Vrahatis, ja G.S. Androulakis. Improving the Convergence of the Backpropagation Algorithm Using Learning Rate Adaptation Methods. Kirjassa *Neural Computation 11*, ss. 1769–1796, 1999.

[43] P. Makosiej, G. Sakaryan, ja H. Unger. Measurement Study of Shared Content and User Request Structure in Peer-to-Peer Gnutella Network. Kirjassa *Design, Analysis, and Simulation of Distributed Systems (DASD 2004)*, ss. 115–124, Arlington, USA, Huhtikuu 2004.

[44] R.J. McEliece, E.C. Posner, E.R. Rodemich, ja S.S. Venkatesh. The Capacity of the Hopfield Associative Memory. Kirjassa *IEEE Transactions on Information Theory*, 1987.

[45] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, ja Z. Xu. Peer-to-Peer Computing. Tekninen raportti, HP Laboratories, Palo Alto, California, USA, 2002.

[46] A. Montresor, G. Di Caro, ja P. Heegaard. Architecture of the Simulation Environment. Tekninen raportti, 2003.

[47] Napster. <URL: http://www.napster.com/>. viitattu 4.7.2006.

[48] Neural Network FAQ. <URL: ftp://ftp.sas.com/pub/neural/FAQ.html>. viitattu 6.10.2006.

[49] NS-2. <URL: http://www.isi.edu/nsnam/ns/>. viitattu 6.10.2006.

[50] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, Sevastopol, CA, 2001.

[51] PDNS - Parallel/Distributed NS. <URL: http://www.cc.gatech.edu/computing/compass/pdns/>. viitattu 6.10.2006.

[52] Peersim. <URL: http://peersim.sourceforge.net/>. viitattu 6.10.2006.

[53] M.K. Ramanathan, V. Kalogeraki, ja J. Pruyne. Finding Good Peers in Peer-to-Peer Networks. Kirjassa *Proceedings of IEEE International Parallel and Distributed and Computing Symposium*, Huhtikuu 2002.

[54] R. Reed ja R. Marks II. *Neural Smithing, Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, Cambridge Massachusetts, London, England, 1999.

[55] C. Rohrs. Query Routing for the Gnutella Network. <URL: http://rfc-gnutella.sourceforge.net/src/qrp.html>. viitattu 6.10.2006.

[56] E.T. Rolls ja A. Treves. *Neural Networks and Brain Function*. Oxford University Press, New York, 1998.

[57] D. Rumelhart, B. Widrow, ja M. Lehr. The Basic Ideas in Neural Networks. *Communications of the ACM*, 1994.

[58] S. Saalasti. *Neural Networks for Heart Rate Time Series Analysis*. Väitöskirja, University of Jyväskylä, 2003.

[59] G. Sakaryan ja H. Unger. Influence of the Decentralized Algorithms on Topology Evolution in P2P Distributed Networks. Kirjassa *Proceedings of Design, Analysis, and Simulation of Distributed Systems (DASD 2003)*, 2003.

[60] G. Sakaryan, M. Wulff, ja H. Unger. Search Methods in P2P Networks: a Survey. Kirjassa *Proceedings of I2CS-Innovative Internet Community Systems (I2CS 2004)*, Heinäkuu 2004.

[61] K. Samant ja S. Bhattacharyya. Topology, Search, and Fault Tolerance in Unstructured P2P Networks. Kirjassa *Proceedings of the 37th IEEE Conference on System Sciences*, Tammikuu 2004.

[62] S. Saroiu, P.K. Gummadi, ja S.D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. Kirjassa *Proceedings of Multimedia Computing and Networking (MMCN)*, Tammikuu 2002.

[63] M. Schlosser, T. Condie, ja S. Kamvar. Simulating a P2P File-Sharing Network. Kirjassa *First Workshop on Semantics in P2P and Grid Computing*, Joulukuu 2002.

[64] R. Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. Kirjassa *Proceedings of the First IEEE International Conference on Peer-to-Peer Computing (P2P'01)*, Elokuu 2001.

[65] M.C. Sinclair. *Evolutionary Algorithms for Optical Network Design: A Genetic-algorithm/Heuristic Hybrid Approach*. Väitöskirja, University of Essex, 2001.

[66] Skype. <URL: http://www.skype.com>. viitattu 4.7.2006.

[67] K. Sripanidkulchai, B. Maggs, ja H. Zhang. Efficient Content Location Using Interest-based Locality in Peer-to-Peer Systems. Kirjassa *Proceedings of Infocom*, 2003.

[68] K. Stanley ja R. Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation*, 10:99–127, Marraskuu 2002.

[69] N. Ting ja R. Deters. 3LS - A Peer-to-Peer Network Simulator. Kirjassa *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P 2003)*, 2003.

[70] D. Tsoumakos ja N. Roussopoulos. A Comparison of Peer-to-Peer Search Methods. Kirjassa *Proceedings of Sixth International Workshop on the Web and Databases*, 2003.

[71] D. Tsoumakos ja N. Roussopoulos. Adaptive Probabilistic Search for Peer-to-Peer Networks. Kirjassa *Proceedings of the Third IEEE International Conference on Peer-to-Peer Computing (P2P'03)*, 2003.

[72] M. Vapa, N. Kotilainen, A. Auvinen, H. Kainulainen, ja J. Vuori. Resource Discovery in P2P Networks Using Evolutionary Neural Networks. Kirjassa *International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA 2004)*, Marraskuu 2004.

[73] D. Whitley. Genetic Algorithms and Neural Networks. Kirjassa G. Winter, J. Périaux, M. Galan, ja P. Cuesta, toim., *Genetic Algorithms in Engineering and Computer Science*. John Wiley & Sons Ltd., England, 1996.

[74] B. Yang ja H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. Kirjassa *Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, 2002.

[75] B. Yang ja H. Garcia-Molina. Designing a Super-Peer Network. Kirjassa *Proceedings of the 19th IEEE International Conference on Data Engineering*, 2003.

[76] W. Yang ja N. Abu-Ghazaleh. GPS: A General Peer-to-Peer Simulator and its Use for Modeling BitTorrent. Kirjassa *Proceedings of the 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '05)*, 2005.

[77] M. Ylianttila, E. Harjula, J. Ala-Kurikka, T. Koskela, O. Kassinen, J-Z. Sun, J. Sauvola, J. Riekki, ja K. Kuusipalo. Sovellukset superverkoksi. *Prosessori*, Marraskuu 2004. In Finnish.

[78] E. Zegura, K. Calvert, ja M. Donahoo. A Quantitative Comparison of Graph-based Models for Internet Topology. *IEEE/ACM Transactions on Networking*, 1997.