TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

LEEVI KULJU

GENEETTINEN ALGORITMI MATOPELIN OHJAIMEN RAKENTAMISEEN

Kandidaatintyö

Tarkastaja: Maarit Harsu
Jätetty tarkastettavaksi
8. toukokuuta 2018

TIIVISTELMÄ

Leevi Kulju: Geneettinen algoritmi matopelin ohjaimen rakentamiseen,
Genetic algorithm learns to play Snake
Tampereen teknillinen yliopisto
Kandidaatintyö, 18 sivua, 0 liitesivua
Toukokuu 2018
Tietotekniikan kandidaatin tutkinto-ohjelma
Pääaine: Ohjelmistotekniikka
Tarkastaja: Maarit Harsu

Avainsanat: Geneettinen algoritmi, optimointi

Tässä kandidaatintyössä tutustutaan geneettisiin algoritmeihin, niihin liittyviin funktioihin ja toteutetaan geneettinen algoritmi ohjelmallisesti C++-ohjelmointikielellä Qt Creator-kehitysympäristössä.

Työssä käsitellään geneettisen algoritmin ja ysinkertaisen neuroverkon toimintaperiaatteet ja miten ne on toteutettu ohjelmallisesti. Työn toteutusvaiheessa ohjelmoitu geneettinen algoritmi etsii neuroverkon painoja siten, että neuroverkko suoriutuisi matopelin ohjaamisesta mahdollisimman hyvin. Työn toteutusvaiheen jälkeen arvioidaan, onko toteutettu algoritmi onnistunut tehtävässään ja mitkä syyt vaikuttavat hyvään tai huonoon suoriutumiseen.

Matopelille onnistuttiin geneettisen algoritmin avulla luomaan hyvin eritasoisia ohjaimia. Yhteistä parhaille löydetyille ratkaisuille oli, että ne osasivat kääntyä vain vasemmalle tai oikealle hedelmän ilmestyessä niiden näkökenttään. Työn aikana kirjoitettu ohjelma tuottaisi todennäköisesti huomattavasti paljon parempia ratkaisuja jo pienillä muutoksilla.

SISÄLLYSLUETTELO

LYHENTEET, MERKINNÄT JA TERMISTÖ

Käsite, Lyhenne	**Alkuperäinen**	**Selite**
GA	*Genetic Algorithm*	Geneettinen Algoritmi
Koodaaminen	*Encoding*	Ongelman parametrin tai parametrien esitystavan muuttaminen GA:n käyttöön sopivaksi genotyypiksi.
Genotyyppi	*Genotype*	Ratkaisuehdotuksen koodattu esitysmuoto, jota GA voi käsitellä.
Dekoodaaminen	*Decoding*	Genotyypin purkaminen ongelman parametreiksi tai ratkaisuksi, eli fenotyypiksi.
Fenotyyppi	*Phenotype*	Genotyypin edustama, dekoodattu ratkaisu. Ratkaisu itsessään.
Kromosomi	*Chromosome*	Genotyyppi
Yksilö	*Individual*	Fenotyyppi
Populaatio	*Population*	Yksilöiden muodostama joukko. Tässä työssä se joukko yksilöitä, jotka kilpailevat kelpoisuudessa, ja joista muodostetaan seuraava sukupolvi.
Sukupolvi	*Generation*	GA:n käsittelemä populaatio tietyllä ajanhetkellä.
Kantapopulaatio	*Original population, Initial population*	Geneettisen algoritmin ensimmäinen sukupolvi, josta tulevien sukupolvien yksilöt periytetään.
Kelpoisuus	*Fitness*	Käsite lainattu evoluutioteoriasta. Kelpoisuudella mitataan yksilön kykyä selviytyä, ja päästä jatkamaan perimäänsä.
Risteytys	*Crossover*	Tapahtuma, jossa kahden yksilön genomista muodostetaan uusi, tai uusia yksilöitä.

Mutaatio	*Mutation*	Tapahtuma, jossa yksilön kromosomiin tuotetaan muutos tai muutoksia.

1. JOHDANTO

Ajatus geneettisistä algoritmeista on johdettu luonnon tavasta kehittää lajin selviytymismahdollisuuksia, evoluutioteoriasta. Evoluutio on prosessi, jonka edetessä eliöiden perimä muuttuu sukupolvien välillä mutaatioiden tai pariutumisen seurauksena. Luonnonvalinta taas ohjaa evoluutiota suuntaan, jossa laji selviytyy ja sopeutuu paremmin ympäristöönsä ja vallitseviin olosuhteisiin. Vahvimmat yksilöt selviytyvät parhaiten, ja pääsevät heikkoja yksilöitä todennäköisemmin lisääntymään ja jatkamaan perimäänsä.

Geneettisiä algoritmeja sovelletaan optimointiongelmissa ja erilaisissa simulaatioissa. Geneettiset algoritmit ovat luonteeltaan helposti rinnakkaistuvia. Monille muille koneoppimisjärjestelmille ja optimointimenetelmille on tyypillistä, että ne saattavat päätyä globaalisti parhaan ratkaisun sijaan paikallisiin maksimeihin. Vaikka geneettiset algoritmit suosivat tietyllä ajanhetkellä parhaita löydettyjä ratkaisuja, myös muilla ratkaisuilla on mahdollisuus olla vaikuttamassa lopulliseen ratkaisuun. Näin paikalliset maksimit voidaan välttää. Huomionarvoista on myös se, että geneettinen algoritmi ei välttämättä löydä koskaan hyvää ratkaisua, ja voi olla perinteisiä optimointimenetelmiä hitaampi.

Tässä kandidaatintyössä tutustutaan geneettisten algoritmien rakenteeseen, ja niiden vaiheiden toteuttamiseen ohjelmallisesti. Työn tavoite on saada algoritmi pelaamaan yksinkertaista matopeliä. Madon on syötävä selvitäkseen pelissä, eikä se saa törmätä pelialueen seiniin tai häntäänsä. Matopelin ohjaaminen tapahtuu neuroverkon avulla. Neuroverkon painot etsitään geneettisen algoritmin avulla.

Toisessa luvussa esitellään geneettisen algoritmin toimintaperiaatteet ja muutamia esimerkkejä tavoista, joilla algoritmiin liittyvät operaatiot voidaan toteuttaa. Kolmas luku käsittelee neuroverkkoja siinä laajuudessa, kuin tämän työn ymmärtämiseksi on tarpeellista. Neljännessä luvussa tutustutaan työn toteuttamiseksi kirjoitetun ohjelman rakenteeseen. Viidenteen lukuun kerätään työn toteutuksen aikana kerätyt havainnot ja lopputulokset.

2. GENEETTISEN ALGORITMIN TOIMINTA

Alan kirjallisuudessa [1, s. 29] geneettinen algoritmi kuvataan ongelmanratkaisu- tai hakumenetelmäksi, jonka mallina on genetiikka ja joka etsii riittävän hyviä ratkaisuja käsiteltäviin ongelmiin. Tässä luvussa esitellään geneettisten algoritmien toimintaperiaatteet ja geneettisiin algoritmeihin yleisimmin liitettävät funktiot.

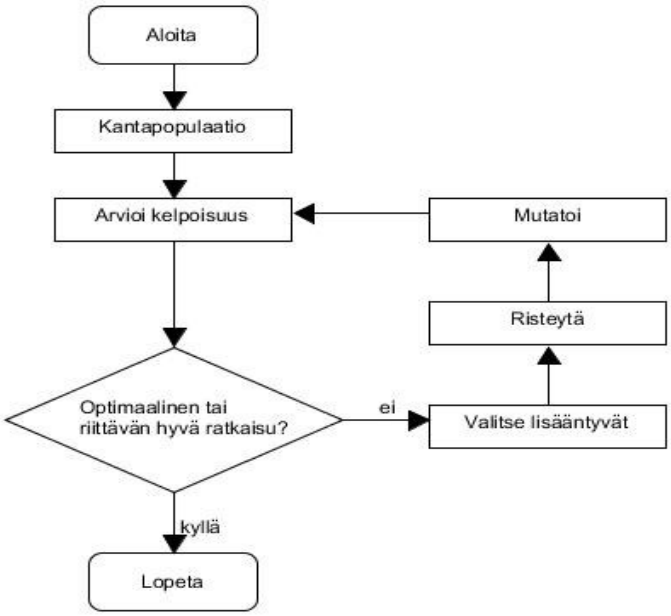
2.1 Toimintaperiaatteet

Geneettinen algoritmi sopii monenlaisiin optimointiongelmiin, kun etsintäavaruus (*search space*) tunnetaan. Perinteisesti optimointi- ja hakumenetelmät jaotellaan jatkuviin, diskreetteihin, rajoitettuihin (*constrained*), rajoittamattomiin (*unconstrained*), sarjallistuviin ja rinnakkaistuviin menetelmiin ja niiden yhdistelmiin. Geneettinen algoritmi voidaan sovittaa mihin tahansa edellisistä luokista myöhemmin esiteltävän kelpoisuusfunktionsa avulla. Geneettinen algoritmi ei välttämättä löydä optimaalisinta ratkaisua yksinkertaisiinkaan ongelmiin, mutta onnistuneen suunnittelun avulla on mahdollista löytää hyviä ratkaisuja.

Alla on listattu muita eroja perinteisiin optimointimenetelmiin:

- Geneettinen algoritmi ei tyypillisesti käytä ongelman parametreja vaan niiden koodattuja versioita.
- Geneettinen algoritmi arvioi samanaikaisesti ratkaisujen joukkoa eikä vain yksittäistä ratkaisua. Tästä syystä algoritmin on mahdollista löytää hyvin erilaisia ratkaisuja.
- Onnistumisen arviointiin käytetään kelpoisuusfunktiota. Geneettisiä algoritmeja voidaan soveltaa sekä jatkuviin että diskreetteihin ongelmiin. Kelpoisuusfunktion ansiosta geneettisen algoritmin käyttö ei rajoitu matemaattisiin ongelmiin.
- Geneettiset algoritmit eivät käytä deterministisiä sääntöjä, vaan niiden toiminta perustuu todennäköisyyksiin.

Geneettisen algoritmin toiminta ja tehokkuus perustuvat luonnonvalinnan kaltaiseen ratkaisuehdotusten mutatoimiseen ja risteyttämiseen. Risteytettäviksi valitaan yksilöitä satunnaisesti, mutta sitä todennäköisemmin, mitä paremmin ne tehtävässään suoriutuvat. Geneettisen algoritmin käyttäjä olettaa, että hyvien ratkaisujen ominaisuuksia yhdistelemällä saadaan todennäköisesti entistä parempia ratkaisuja. Geneettisen algoritmin kulku voidaan jakaa karkeasti kuvan 1 vaiheisiin.



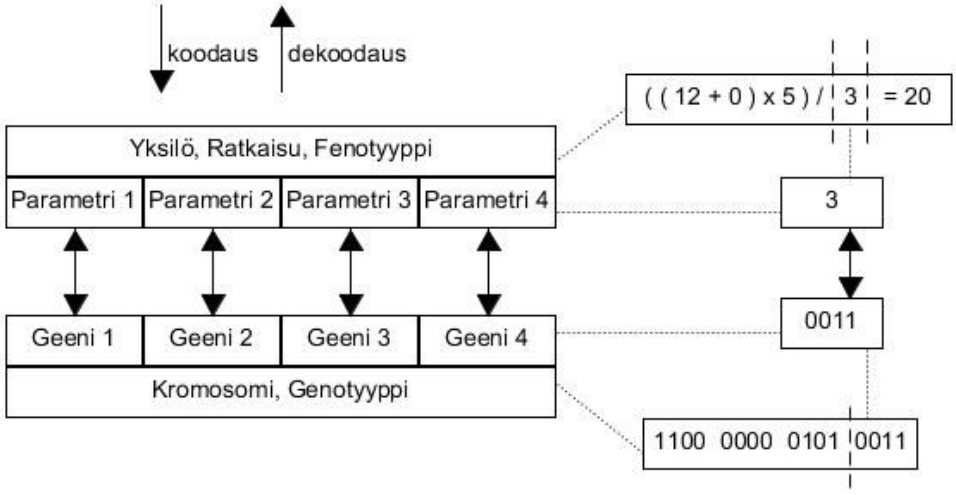
Kuva 1. *Erään geneettisen algoritmin vuokaavio.*

2.2 Ongelman koodaaminen

Geneettisen algoritmin käytön kannalta on välttämätöntä esittää käsiteltävän ongelman parametrit muodossa, jossa niille voidaan soveltaa algoritmin risteytys- ja mutaatiofunktioita. Parametrien koodaamisessa joudutaan tekemään kompromisseja muistinkulutuksen, tehokkuuden ja algoritmin muiden funktioiden toteutustavan välillä.

Tapaan, jolla parametrit halutaan koodata, vaikuttaa vahvasti ratkaistavan ongelman luonne. Sen lisäksi, että geneettisiä algoritmeja voidaan käyttää ongelman ratkaisevien numeeristen arvojen etsintään, ne soveltuvat luonteeltaan toisenlaisiin optimointiongelmiin. Oletetaan, että algoritmin käyttäjällä on kahdeksan operaatiota tai tunnettua algoritmia, joita hän aikoo käyttää ongelmansa ratkaisuun. Kromosomi voidaan koodata oktaaliluvuilla, joissa jokainen merkki edustaa operaatiota, joka tietyssä vaiheessa ratkaisua tulee suorittaa.

Kuva 2 havainnollistaa ongelman koodaamista, ja yksilön perimään liittyviä käsitteitä. Kuvan yksilö edustaa tässä tapauksessa yksinkertaista funktiota. Funktion parametrit on koodattu nelibittisiksi merkkijonoiksi, ja niistä on koostettu yksilön genotyyppi. Parametrien arvoalue on rajattu kokonaislukuvälille [0, 15]. Seuraavissa aliluvuissa esitellään kolme usein käytettyä tapaa koodata parametrit.



Kuva 2. *Ongelman koodaaminen ja dekoodaaminen.*

Mukaillen [1, s. 40, kuva 3.1].

2.2.1 Bittitason esitys

Perinteisin tapa koodata parametrit on niiden esittäminen bittitasolla. Käytännössä bittitason esitys tarkoittaa lukujen muuttamista binääriluvuiksi. Parametrin merkitsevät numerot muutetaan biteiksi, ja sen suuruusluokka ja mahdollinen etumerkki esitetään irrallisilla biteillä. Lukuarvojen lisäksi biteillä on helppo esittää esimerkiksi totuusarvoja.

Biteiksi koodaaminen on operaationa yksinkertainen ja moneen ohjelmointikieleen on kehitetty mekanismit kokonais- ja liukulukujen muuntamiseksi bittijonoiksi. Menetelmä voi vaikeuttaa lukualueen rajaamista, jos ohjelmointikielen tietotyyppien ylä- ja alarajat eivät jostain syystä sovi. 32-bittinen liukuluku eli float on toteutettu useimmiten C++ -kieleen siten, että etumerkki esitetään yhdellä bitillä, merkitsevät numerot 23 bitillä ja suuruusluokan kertova eksponentti kahdeksalla. Jos tällainen liukuluku halutaan ottaa käyttöön sovelluksessa, jossa lukualue halutaan rajata lukuväille [0, 1], mutaatiota ei enää voida kohdistaa kaikkiin merkkijonon bitteihin. Tämän ongelman ratkaisee osittain seuraavaksi kuvattu Gray-koodaaminen.

2.2.2 Gray-koodaaminen

Gray-koodaamisessa kahden perättäisen binääriluvun välimatka määritellään, ja haluttu lukualue esitetään tämän välimatkan monikertoina. Tämä on usein ratkaisu tilanteissa, joissa lukualue halutaan rajata tarkasti, ja kromosomin pituutta halutaan lyhentää.

Otetaan esimerkiksi tilanne, jossa lukuarvot halutaan rajata lukuvälille [0, 1] ja esittää kahdeksalla bitillä. Bittijono 0000 0000 saa lukuarvon 0,0 ja bittijono 1111 1111 lukuarvon 1,0. Koska esitystarkkuus on kahdeksan bittiä, yhden bitin lisääminen bittijonoon kasvattaa dekoodattua lukua $1/(2^8 - 1) = 1/255 \approx 0{,}0039$.

Bittijono tulkitaan Gray-koodaamisessa kokonaisluvuksi, jolle annetaan kerroin lukualueen rajaamiseksi. Lukualuetta voidaan myös siirtää summaamalla siihen haluttu luku. Joissain sovelluksissa kertoimen sijaan kokonaisluvuksi muunnetulle bittijonolle voidaan käyttää logaritmeja tai funktioita lukualueen muokkaamiseksi.

2.2.3 Floating point -koodaaminen

Floating point -koodaamisessa kromosomit esitetään liukulukuina tai taulukkona liukulukuja. Tapa tukeutuu käytetyn ohjelmointikielen lukujen esitystarkkuuteen ja sopii hyvin ongelmiin, joissa parametrien määrä on suuri. Floating point -koodaamisella saavutetaan etuja tehokkuudessa, koska kromosomia ei tarvitse koodata ja dekoodata jatkuvasti.

Floating point -koodaamisen haittapuolena on se, että yksittäistä geeniä ei voi katkaista risteytysfunktiossa, eikä mutaatiossa voida enää kääntää yksittäistä bittiä. Risteytyksessä vanhemmilta periytetään uusille ratkaisuille kokonaisia geenejä tai parametreja. Mutaation kohteena olevan geenin tilalle arvotaan uusi luku.

2.3 Kelpoisuusfunktiot

Kelpoisuusfunktion on tarkoitus laskea yksilölle kelpoisuus. Kelpoisuus itsessään on esimerkiksi kokonaisluku, jolla mitataan yksilön suoriutumista ratkaistavassa ongelmassa. Kelpoisuusfunktion luomiselle ei ole olemassa suoria ohjeita, koska tarvittava funktio riippuu voimakkaasti ratkaistavasta ongelmasta. Kelpoisuusfunktiota suunniteltaessa tulee pyrkiä siihen, että yksilöiden kelpoisuudet ovat vertailukelpoisia ja että kelpoisuudesta voidaan päätellä, miten lähellä optimaalista ratkaisua sen tuottanut yksilö on.

Kelpoisuusfunktio on tärkein geneettisen algoritmin toimintaan vaikuttava funktio. Kelpoisuus on tekijä, jota käytetään valinnan perusteena geneettisessä algoritmissa. Voidaan sanoa, että suurin osa algoritmin oppimisesta tapahtuu kelpoisuusfunktiossa. Jos kelpoisuusfunktio pisteyttää ratkaisuja huonosti valituin perustein, algoritmi hidastuu merkittävästi, eikä optimaalista ratkaisua välttämättä löydy koskaan.

2.4 Risteytysfunktiot ja valinta

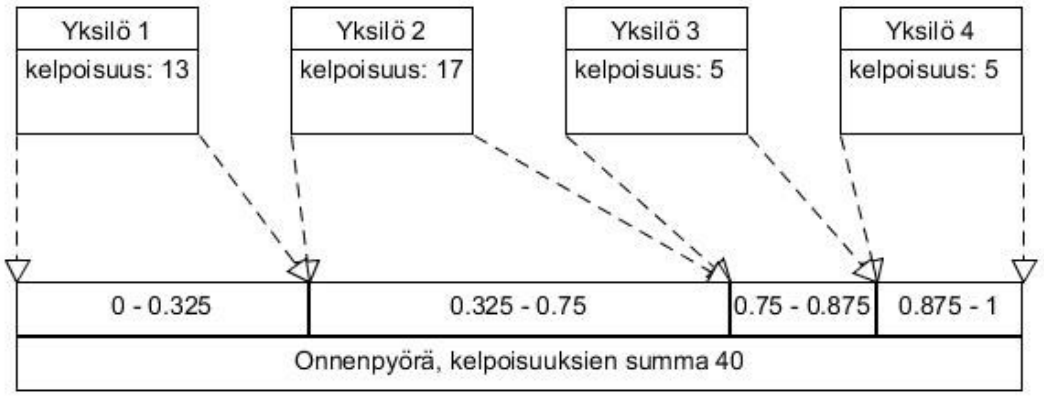
Risteytysfunktioiden ja valinnan tarkoitus on valita populaatiosta kelpoisimpia yksilöitä välittämään perimäänsä seuraavalle sukupolvelle edelleen parempien ratkaisujen toi-

vossa. Pelkän kelpoisuuden tarkastelu ei riitä, koska tällöin päädytään helposti populaatioihin, joissa yksilöt ovat liian samanlaisia. Risteytettävien yksilöiden valintaan tulee lisätä satunnaisuutta, jotta populaation yksilöt säilyvät erilaisina, eikä mahdollisesti hyviä geenejä heitetä hukkaan. Liika erilaisuus populaatiossa hidastaa algoritmin toimintaa, mutta liika samanlaisuus estää tai hidastaa parhaiden ratkaisujen löytämistä. Seuraaviin alilukuihin on koottu kolme esimerkkiä valintafunktioista.

2.4.1 Onnenpyörävalinta

Onnenpyörävalinta (*Roulette wheel selection)* on eräs perinteinen valintatekniikka. Menetelmässä valinnan todennäköisyys on suoraan verrannollinen yksilön kelpoisuuteen. Onnenpyörä voidaan alustaa esimerkiksi summaamalla populaation kelpoisuudet ja jakamalla jokaisen yksilön kelpoisuus summalla. Näin saadaan jokaiselle yksilölle todennäköisyys tulla valituksi. Todennäköisyydet jaetaan lukuvälille [0, 1], minkä jälkeen satunnaislukuja arpomalla voidaan valita lisääntymään pääsevä yksilö. Kuvassa 3 havainnollistetaan onnenpyörän rakentaminen neljälle yksilölle, joiden kelpoisuudet on jo laskettu. Onnenpyörä voidaan toteuttaa myös lisäämällä yksilön tunnisteita säiliöön kelpoisuudesta laskettavan määrän kertoja, jonka jälkeen säiliöstä voidaan arpoa valittava yksilö.

Onnenpyörävalinnan huonona puolena on se, että jos populaatioon syntyy yksilö, jonka kelpoisuus on huomattavasti muuta populaatiota korkeampi, se alkaa dominoida valintaa. Dominoiva yksilö saattaa ajautua onnenpyörävalinnassa useamman seuraavan sukupolven yksilön vanhemmaksi. Tämä vähentää erilaisuutta ja saattaa johtaa ratkaisun paikalliseen maksimiin optimiratkaisun sijaan.



Kuva 3. *Onnenpyörä valintamenetelmänä.*

2.4.2 Turnausvalinta

Menetelmässä populaatiosta valitaan satunnaisesti tietty määrä yksilöitä kilpailemaan oikeudesta lisääntyä. Turnauksen voittajan valinta tehdään suoraan kelpoisuutta tarkastelemalla. Turnauksia järjestetään niin monta, että seuraava populaatio saadaan täysilukuiseksi.

Koska yksittäisen turnauksen osanottajat valitaan satunnaisesti, menetelmä ehkäisee vahvojen yksilöiden dominointia valinnassa ja lisää erilaisuutta. Toisaalta hyvät ratkaisut saattavat karsiutua pois, jos niitä ei valita yhteenkään turnaukseen.

2.4.3 Lineaarinen sijoitusvalinta

Lineaarisessa sijoitusvalinnassa (*Linear rank selection*) yksilöt valitaan kuten onnenpyörävalinnassa. Menetelmässä yksilön todennäköisyys tulla valituksi ei kuitenkaan ole suoraan verrannollinen yksilön kelpoisuuteen. Menetelmälle on useita mahdollisia toteutustapoja ja kelpoisuuden merkitystä voidaan korostaa tai vähentää.

Yksilöt järjestetään kelpoisuuden mukaan, ja jokaiselle sijoitukselle annetaan todennäköisyys, joka on sitä suurempi, mitä parempi sijoitus on. Paras sijoitus on populaation koko, ja huonoin sijoitus on 1. Jos populaation koko on n, yksilön i valinnan todennäköisyys p voidaan laskea esimerkiksi $p(i) = \frac{\text{sijoitus(i)}}{n \cdot (n+1)}$. Menetelmä suosii kelpoisimpia yksilöitä, mutta ehkäisee vahvimpien yksilöiden dominointia valinnassa.

2.4.4 Risteyttäminen

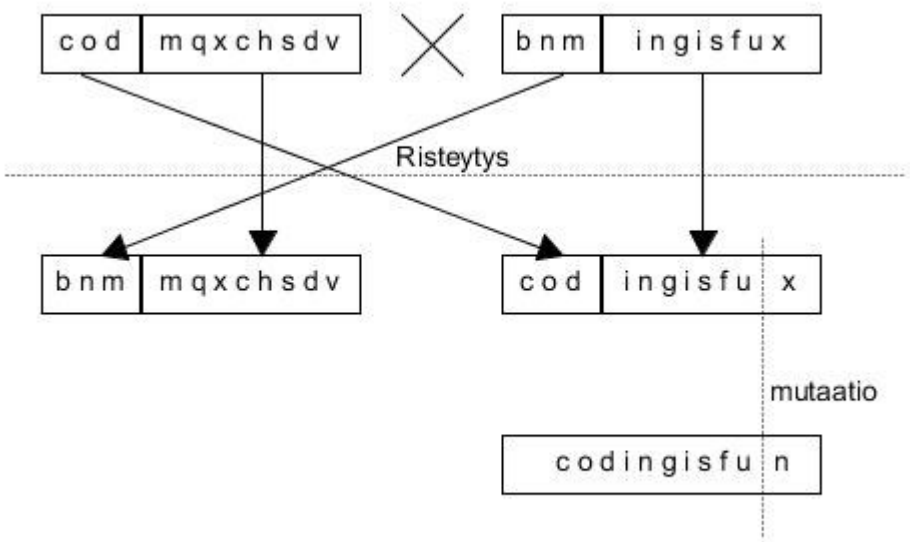
Risteytys toteutetaan pilkkomalla kahden risteytykseen valitun yksilön kromosomit yhdestä tai useammasta pisteestä. Kuvassa 4 risteytys on toteutettu katkaisemalla kromosomi yhdestä pisteestä. Pisteiden määrän kasvattaminen on puhtaasti toteutustekninen seikka. Katkaisupisteiden määrän lisäämisen ei voida olettaa parantavan tai huonontavan algoritmia.

Biteiksi koodattu kromosomi voidaan katkaista keskeltä parametria, joten katkaisupisteet voidaan arpoa mihin tahansa kohtaan kromosomia. Floating point -koodauksessa katkaisupisteiden sijaan valitaan kummaltakin vanhemmalta ne parametrit, jotka halutaan siirtää seuraavan sukupolven edustajille.

2.5 Mutaatiot

Tilanteessa, jossa kantapopulaation kromosomeissa ei esiinny tiettyä geeniä, sitä ei voi periytyä jälkeläisille risteytysoperaatioissa. Ongelma ratkaistaan arpomalla kromosomeihin satunnaisia mutaatioita. Kuvassa 4 esitetään tilanne, jossa yksilöt risteytetään, ja toiselle seuraavan sukupolven yksilöistä aiheutetaan mutaatio.

Mutaatio on yksinkertaisin toteuttaa valitsemalla yksilön kromosomista yksi tai useampi kohta, ja vaihtaa näiden bittien arvot. Floating point -koodauksessa tämä ei ole mahdollista. Floating point -koodauksessa voidaan valita jokin kromosomin sisältämistä geeneistä, ja arpoa sille uusi arvo suurimman ja pienimmän sallitun arvon väliltä.



Kuva 4. *Risteytys ja mutaatio.*

3. NEUROVERKOISTA

Työn toteutusvaiheessa on tavoitteena rakentaa matopelin ohjaamiseen soveltuva neuroverkko siten, että verkon painot etsitään geneettisen algoritmin avulla. Seuraavissa kappaleissa esitellään neuroverkot yleisesti, jotta lukijalla olisi paremmat edellytykset ymmärtää työn toteutusta.

Neuroverkko tai lyhyemmin ANN (*Artificial Neural Network*) on ihmisaivojen rakennetta pienessä mittakaavassa jäljittelevä malli tekoälyjen (*Artificial Intelligence, AI*) kehittämisessä ja koneoppimisen sovelluksissa [2]. Neuroverkkotyyppejä on useammanlaisia, mutta tämä luku esittelee ja käsittelee vain *Feed Forward* -tyyppisen verkon ominaisuuksia.

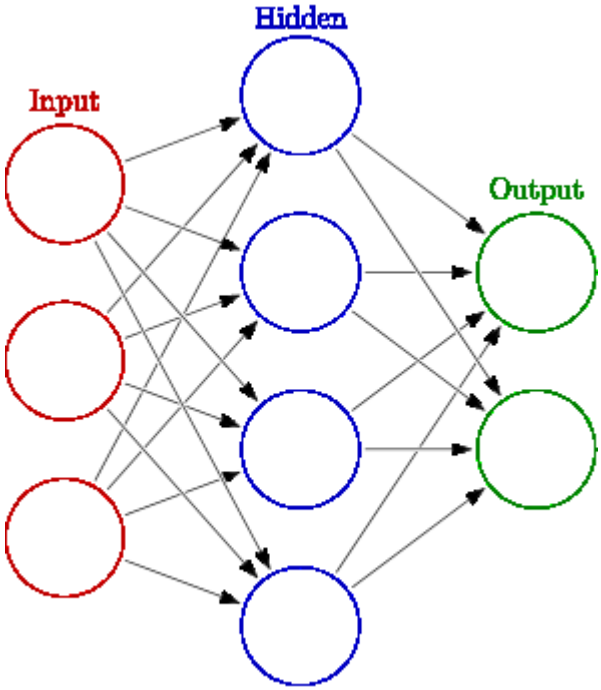
Yksinkertaisimmillaan neuroverkko on sarja toisiinsa ketjutettuja painotettuja summia, joka tuottaa tietyn ulostulon tietylle syötteelle. Neuroverkko koostuu tyypillisesti sisäänmenokerroksesta (*input layer*), piilotetuista kerroksista (*hidden layer*) ja ulostulokerroksesta (*output layer*). Kerros koostuu neuroneista. Neuroverkon muoto esitetään topologiana, joka kertoo kuinka monta neuronia missäkin kerroksessa on. Kuvan 5 mukaisen neuroverkon topologia on 3-4-2.

Jokainen neuroni sisältää aktivointifunktion (*activation function*) ja on yhteydessä jokaiseen seuraavan kerroksen neuroniin. Jokaista yhteyttä edustaa paino (*weight*). Sisäänmenokerroksen neuroneja lukuunottamatta neuroniin liitetään *N* kappaletta painotettuja sisäänmenoja, missä *N* on edellisen kerroksen neuronien määrä. Näiden sisäänmenojen summa syötetään neuronin sisältämään aktivointifunktioon, jonka tulos edustaa neuronin ulostuloa ja joka edelleen syötetään seuraavalle kerrokselle.

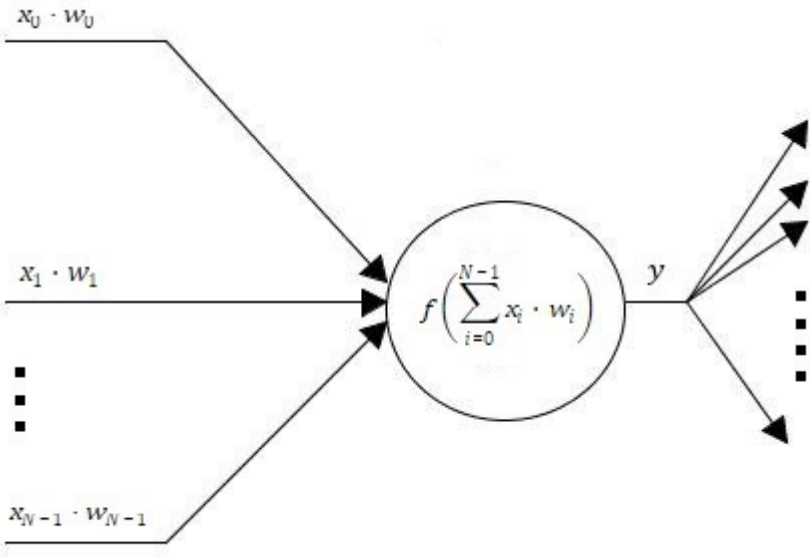
Kuva 6 havainnollistaa yksinkertaisen neuronin rakennetta. Kuvan neuroniin johdetaan N painotettua sisäänmenoa, joiden summasta saadaan ulostulo y aktivointifunktion f(x) avulla. Ulostulo y johdetaan omilla painoillaan kerrottuina edelleen kaikkiin seuraavan kerroksen neuroneihin, kuten kuvassa 5.

Neuroverkkoa opetetaan ohjaamalla painoja suuntaan, jolla verkko tuottaa tietyille sisäänmenon arvoille halutun vasteen. Painoja ohjataan laskemalla gradientti halutun ulostulon, todellisen ulostulon ja aktivointifunktion derivaatan avulla. Painoja ohjataan askel gradientin suuntaan, jolloin verkko tuottaa paremman ulostulon. Koska työn toteutusvaiheessa edellä kuvattua painojen ohjaamista (*back-propagation*) ei tehdä, vaan optimaalisia painoja etsitään geneettisen algoritmin avulla, kaavoja ei esitellä tässä.

Aktivointifunktion tehtävänä on sekä muuttaa sisäänmenosignaalit ulostuloksi, että rajata ulostulojen arvot tietylle lukuvälille, tyypillisesti välille [0, 1] tai [-1, 1]. Aktivointifunktiona toimii käytännössä mikä tahansa funktio, mutta tyypillisesti käytetään joko askelfunktioita, sigmoideja tai hyperbolisia funktioita.



Kuva 5. *Neuroverkon rakenne*



Kuva 6. *Neuronin rakenne.*

4. TOTEUTUKSEN VAIHEET

Työn toteutusvaihe sisältää testidatan esikäsittelyn, neuroverkon ohjelmoinnin, ja geneettisen algoritmin ohjelmoinnin. Vaiheiden sisältö esitellään tarkemmin seuraavissa aliluvuissa. Ohjelmointi suoritetaan C++-ohjelmointikielellä Qt-Creator -kehitysympäristössä.

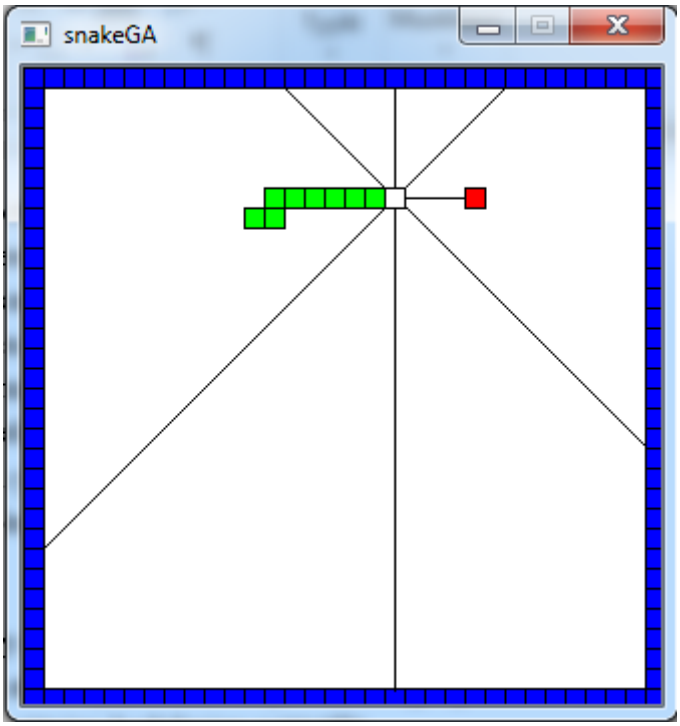
4.1 Matopelin ohjelmointi

Matopeli on yksinkertainen peli, jossa matoa voidaan ohjata ylös, alas, oikealle ja vasemmalle. Pelin tavoitteena on kerätä pisteitä syömällä pelialueelle ilmestyviä hedelmiä. Joka kerta, kun mato syö hedelmän, sen häntä kasvaa. Pelin häviää, jos mato törmää pelialueen reunaan tai häntäänsä. Matopeli ohjelmoidaan 32 x 32 ruudukkoon. Reunimmaiset ruudut ovat seiniä, ja mato asetetaan pelin aluksi laudalle satunnaiseen pisteeseen. Madolle alustetaan kolmen ruudun mittainen häntä heti pelin aluksi, jotta algoritmi oppisi nopeammin väistämään sitä. Pelilaudalle asetetaan satunnaiseen ruutuun yksi hedelmä kerrallaan. Hedelmän sijainti arvotaan, mutta sitä ei laiteta kahta ruutua lähemmäksi madon päätä.

Madolle ohjelmoidaan konenäkö siten, että se näkee kulkusuuntaansa nähden takavasemmalle, vasemmalle, etuvasemmalle, eteen, etuoikealle, oikealle ja takaoikealle. Kuva 7 on kuvankaappaus matopelistä. Madon päästä on piirretty seitsemän viivaa, jotka edustavat silmiä. Yksittäinen silmä näkee vain sitä lähimpänä sijaitsevan kohteen. Silmän näkökenttä esitetään kahtena liukulukuna. Liukuluvut ovat sitä suurempia, mitä lähempänä matoa kohde on. Jos ruudussa suoraan madon edessä on hedelmä, sen keskimmäinen silmä antaa vasteen [1.0, 0.0]. Kuvassa 7 madon ja hedelmän välissä on kolme ruutua, joten keskimmäisen silmän vaste on [0.1, 0.0]. Madon koko näkökenttä esitetään yhdistämällä silmien luvut 14 liukuluvun vektoriin siten, että silmien vaste lisätään vektoriin aina oikeassa järjestyksessä.

Matoa ohjataan neuroverkon seitsemän ulostulon avulla. Ulostuloista suurin valitaan madon seuraavaksi kulkusuunnaksi. Ulostulot edustavat suuntia, joihin madon on mahdollista kääntyä. Esimerkiksi jos ensimmäinen ulostulo antaa suurimman vasteen, mato kääntyy kulkusuuntaansa nähden takavasemmalle. Jos kuudes ulostulo antaa suurimman vasteen, mato kääntyy kulkusuuntaansa nähden oikealle.

Madolle asetetaan pelin aluksi aina tietty määrä energiaa. Kun mato liikkuu yhden ruudun pelialueella, se menettää energiaa. Kun mato syö, se kasvaa ja sen energia palautuu alkuperäiseksi energiamääräksi kerrottuna syötyjen hedelmien määrällä.



Kuva 7. *Madon näkökenttä*

4.2 Geneettisen algoritmin ohjelmointi

Geneettiselle algoritmille ohjelmoidaan tässä aliluvussa kuvatut funktiot. Funktioita ja niiden parametreja voidaan muuttaa vapaasti algoritmia testattaessa.

- Alustusfunktio ottaa parametrikseen populaation koon *P* ja käytettävän neuroverkon topologian *T*. Topologia annetaan vektorina, joka sisältää jokaisen verkon kerroksen neuronien määrät. Alustusfunktio alustaa *P* neuroverkkoa, jotka muodostavat algoritmin kantapopulaation. Alustusfunktio osaa tarvittaessa lukea matojen kromosomit tiedostoista.
- Ajofunktio huolehtii geneettisen algoritmin toiminnasta. Ajofunktio peluuttaa yksittäisen populaation kaikki yksilöt läpi. Ajon jälkeen yksilölle lasketaan kelpoisuus. Kun koko populaatio on pelannut ja pisteytetty, ajofunktio kutsuu lisääntymisfunktiota. Lisääntymisfunktion tuottama uusi populaatio korvaa nykyisen populaation ja ajofunktio käynnistetään uudelleen.
- Lisääntymisfunktio kutsuu valintafunktiota, joka palauttaa kaksi vektoria. Kumpikin vektori sisältää populaation *P/2* yksilöä. Ensimmäisen vektorin yksilöt risteytetään toisen vektorin yksilöiden kanssa. Vektoreista valitaan saman indeksin yksilöt, jotka lähetetään risteytysfunktiolle. Risteytysfunktion palauttamat kaksi yksilöä lisätään seuraavaan populaatioon. Lisääntymisfunktion lopuksi kutsutaan mutaatiofunktiota.

- Valintafunktioiksi toteutetaan luvussa 2.4 esitellyt funktiot. Valintafunktio käsittelee koko populaation, ja palauttaa seuraavan sukupolven vanhemmat kahdessa vektorissa. Valintafunktiot varmistavat, ettei yksilö pääse lisääntymään itsensä kanssa. Toisin sanoen sama yksilö ei esiinny samalla indeksillä valintafunktion palauttamissa vektoreissa.
- Risteytysfunktio ottaa parametrikseen katkaisupisteiden määrän B ja risteytettäväksi valittujen kahden yksilön kromosomit, ja palauttaa kahden jälkeläisen kromosomit. Funktio arpoo vektoriin V B kappaletta katkaisupisteitä lukuväliltä [1, k-1], missä k on kromosomin pituus. Vektoriin lisätään luvut 0 ja k, minkä jälkeen se järjestetään nousevaan järjestykseen. Vektori indeksoidaan läpi, ja jokaisella kierroksella kummankin vanhemman kromosomista poimitaan merkit *V[i-1] – V[i]*, missä i on vektorista V poimittu luku. Toinen merkkijono periytyy ensimmäiselle jälkeläiselle, ja toinen toiselle.
- Mutaatiofunktio lisää jokaiselle populaation yksilölle mutaation niin kauan, kuin ennen mutaatiota arvottava satunnaisluku lukuväliltä [0, 1] on pienempi kuin funktiossa laskettu kynnysarvo. Kynnysarvo on kääntäen verrannollinen algoritmin ajon aikana löytyneen parhaan yksilön kelpoisuuteen.
- Kelpoisuusfunktio ohjelmoidaan siten, että syötyjen hedelmien lukumäärälle annetaan kerroin x. Kelpoisuuteen lisätään yksilön kulkema matka pelialueella, toisin sanoen sen kuluttama energia. Kerrointa x kasvattamalla algoritmilla on korkeampi paine etsiä pelialueelta hedelmiä. Kuljetusta matkasta annetaan pisteitä, jotta algoritmi suosisi yksilöitä, jotka kulkevat pelialueella törmäämättä itseensä tai reunoihin.
- Algoritmin arvioinnin tueksi ohjelmoidaan tallennusfunktio. Tallennusfunktio tallentaa eniten hedelmiä syöneen yksilön kromosomin tekstitiedostoon. Tiedostoon liitetään yksilöllinen indeksi ratkaisujen tunnistamiseksi. Erilliseen raporttitiedostoon kirjoitetaan uusi rivi, johon liitetään ratkaisun indeksi, sen sukupolven numero, josta yksilö löydettiin, ja yksilön syömien hedelmien määrä. Raporttia voidaan käyttää tulosten arviointiin.

4.3 Neuroverkon käyttö työssä

Työhön ohjelmoidaan luvussa 3 kuvattu Feed Forward -verkko. Työssä käytettävän neuroverkon topologiaa voidaan muuttaa mielivaltaisesti lisäämällä piilotettuja kerroksia, mutta sisäänmenokerroksen koko on aina 14 ja ulostulokerroksen koko seitsemän neuronia.

Neuroverkko toteutetaan luokkana, joka sisältää tiedon verkon topologiasta ja painoista. Neuroverkon painot alustetaan satunnaisesti. Neuroverkkoon koodataan funktio, jonka avulla sen painot voidaan dekoodata suoraan funktiolle annetusta kromosomista, jos kro-

mosomin pituus täsmää verkolle määrättyyn topologiaan. Lisäksi neuroverkkoon toteutetaan funktio, jolla se koodaa painonsa kromosomiksi. Neuroverkon sisäänmenoina käytetään edellisessä aliluvussa kuvattua madon näkökenttää, josta se tuottaa ulostuloiksi seitsemän seuraavan kulkusuunnan määräämiseksi tarvittavaa lukua.

4.4 Genomin koodaaminen, dekoodaaminen ja istutus neuroverkkoon

Neuroverkon käyttö työssä kasvattaa tarvittavien parametrien määrän suureksi, jos topologia on monimutkainen. Neuroverkon painot ovat tietotyypiltään 32-bittisiä liukulukuja, mutta niiden arvot on rajattava lukuvälille [0, 1]. C++:n float -tyyppi takaa sovellukseen tarpeettoman suuren lukuvälin, ja lukujen suora muuntaminen 32-merkkisiksi bittijonoiksi johtaisi seuraavaan ongelmaan. Geneettisen algoritmin mutaatio-operaatio saattaisi muuttaa 32-bittisen geenin bittejä, jotka määräävät sen edustaman liukuluvun suuruusluokan. Koska painot halutaan rajata lukuvälille [0, 1], lukuja ei voida muuntaa suoraan biteiksi.

Bittitason koodaus toteutetaan Gray-koodauksena. Tässä työssä ei yksinkertaisuuden vuoksi käytetä muita koodaustapoja. Koodattava liukuluku kerrotaan kertoimella $(2^n - 1)$, missä *n* on valittu bittijonon pituus. Tulo muunnetaan 16-bittiseksi kokonaisluvuksi (*unsigned short)*. C++-kielen bitset-tietotyyppi voidaan alustaa saadulla tuloksella ja bittien määrällä *n*, ja muuntaa edelleen merkkijonoksi. Dekoodaaminen tapahtuu alustamalla bitset *n*-merkkisellä merkkijonolla, muuntamalla se kokonaisluvuksi ja kertomalla se koodauksen kertoimen käänteisluvulla.

Neuroverkon oppiminen tapahtuu painoja muokkaamalla, joten genomi koostetaan niistä. Neuroverkko-olio ohjelmoidaan siten, että sen voi rakentaa joko tyhjästä, jolloin painot ovat satunnaisia, tai genomin sisältävän merkkijonon avulla. Koska yksittäisen geenin pituus *n* tiedetään jokaisella algoritmin ajokerralla, on painojen arvot helppo määritellä indeksoimalla kromosomia *n* merkin jaksoissa. Kromosomin istutukseen käytetään edellisessä kappaleessa kuvattua dekoodausfunktiota.

5. TULOKSET

Tässä luvussa esitellään työn tulokset. Luvussa arvioidaan sitä, miten lukuun 2 kerätyt seikat näkyivät toteutuksessa, ja miten hyvään lopputulokseen milläkin funktioilla päästiin.

5.1 Kriteerit geneettisen algoritmin arvioimiseen

Työn tarkoitus on saada matopeli pelaamaan itseään geneettisen algoritmin tuottaman neuroverkon ohjaamana. Pistettä, jossa verkko on riittävän taitava pelaamaan, on vaikea määritellä, joten sitä ei määritellä ennalta.

Geneettisen algoritmin arvioiminen on suoraviivaisempaa. Algoritmin arviointikriteerinä on riittävän hyvään ratkaisuun vaadittujen sukupolvien määrä. Algoritmit eritellään sen mukaan, mitä luvussa 2 esiteltyjä funktioita ja menetelmiä niiden ohjelmointiin käytettiin, mitä bittimäärää Gray-koodaukseen käytettiin, ja minkälainen neuroverkon topologia on käytössä.

Testattavien algoritmien kantapopulaatioiden yksilöt luodaan satunnaislukuja arpomalla, joten on epätodennäköistä, että yhdellä algoritmin ajokerralla saadaan hyviä tuloksia. Tästä syystä algoritmia ajetaan siten, että 100 sukupolven jälkeen eniten hedelmiä syöneen yksilön kromosomi tallennetaan tekstitiedostoon, minkä jälkeen algoritmin suoritus aloitetaan uudelleen. Ehtona tallentamiselle on, että paras yksilö on saanut kelpoisuudekseen yli 100 pistettä. Algoritmin nopeuttamiseksi kelpoisuudekseen yli 500 pistettä saanut yksilö tallennetaan heti, minkä jälkeen seuraava 100 sukupolven kierros aloitetaan.

Kun kromosomeja on tallennettu kokonaisen sukupolven verran, nämä yksilöt valitaan viimeisen ajokerran kantapopulaatioksi. Tällä kantapopulaatiolla alustettavaa algoritmia ajetaan 20 sukupolvea. Tämän algoritmin tuottamaa parasta yksilöä käytetään verkon taitavuuden arviointiin algoritmin tuottaman kantapopulaation ohella.

5.2 Arviointi

Tähän alilukuun taulukoidaan suoritetut kriteerien mukaiset testiajot. Lisäksi arvioidaan, miten käytettävät funktiot ja menetelmät ovat vaikuttaneet lopputulokseen. Taulukkoon 1 on kerätty tiedot algoritmien käyttämistä funktioista ja suoriutumisesta. Taulukkoon 2 on kerätty aliluvun 5.1 mukaisten algoritmien viimeisten ajokertojen tulokset.

	1. ajokerta	2. ajokerta	3. ajokerta	4. ajokerta	5. ajokerta
Koodauksen bittien määrä *n*	4	8	2	4	8
Populaation koko *P*	30	30	20	20	100
Verkon topologia *T*	14-18-7	14-12-7	14-12-7	14-7-7-7	14-12-7
Hedelmän paino *x* kelpoisuusfunktiossa	10	20	10	20	20
Valintafunktio	onnen-pyörävalinta	onnen-pyörävalinta	lineaarinen sijoitusvalinta	turnausvalinta, turnauksen koko 5	onnen-pyörävalinta
Katkaisupisteet risteytysfunktiossa	4	2	4	2	2
Parhaan yksilön syömät hedelmät	78	62	46	-	34
Tallennettujen verkkojen keskimäärin syömät hedelmät	31	41	20	-	13
Sukupolvi parhaan verkon löytyessä keskimäärin	51	61	56	-	70

Taulukko 1. *Algoritmin ajokierrokset.*

	1. ajokerta	2. ajokerta	3. ajokerta	5. ajokerta
Parhaan yksilön syömät hedelmät	68	77	25	37
Parhaan yksilön sukupolvi	0	12	0	18

Taulukko 2. *Taulukon tuottamien kantapopulaatioiden suoriutuminen*

Yksittäinen algoritmin ajokierros kesti arvioitua kauemmin, joten niitä on vähän. Koska ajokierroksia ei ole montaa, käytettävien funktioiden arviointi vaikeutui, eikä varmoja johtopäätöksiä ole mahdollista tehdä. Käytettyjä funktioita tärkeämmäksi ominaisuudeksi osoittautui tässä sovelluksessa neuroverkon topologia. Topologian monimutkaistaminen kasvattaa voimakkaasti kromosomin pituutta. Yksinkertaisella topologialla päästiin kohtuullisen hyviin tuloksiin, mutta yksilöillä oli taipumus liikkua kiertäen pelialuetta joko myötä - tai vastapäivään. Monimutkaisemmilla topologioilla voidaan olettaa, että yksilö kykenisi monimutkaisempaan käyttäytymiseen, mutta tällaisia yksilöitä ei onnistuttu luomaan.

Kelpoisuusfunktio ohjaa suuntaa, johon geneettisellä algoritmilla on paine kehittyä. Tässä työssä kelpoisuusfunktio painotti liiaksi syötyjä hedelmiä, joten populaatiolla ei ollut painetta esimerkiksi oppia kääntymään useampaan suuntaan. Suurin osa löydetyistä toimivista ratkaisuista osasi kulkea suoraan ja kääntyä yhteen suuntaan, jos niiden näkökenttään ilmestyi hedelmä. Menetelmä on toimiva, mutta nykyisellään algoritmi tuottaa havaittua käyttäytymistä monimutkaisempaa ja siten ehkä parempaa käyttäytymistä hyvin epätodennäköisesti.

Tuloksia olisi todennäköisesti voinut parannella ja ratkaisujen käyttäytymistä muokata kelpoisuusfunktiota muuttamalla. Kelpoisuusfunktion olisi voinut ohjelmoida muuttumaan tiettyjen ehtojen täyttyessä ajon aikana. Algoritmin edetessä hedelmien syömisen lisäksi pisteitä olisi voinut antaa esimerkiksi jokaisesta suunnasta, johon yksilö ajon aikana kääntyy. Toinen vaihtoehto toiminnan parantelemiseksi olisi voinut olla tehokkuuden huomioiminen. Mitä nopeammin ratkaisu löytää hedelmän luo, sitä paremmat pisteet se syömisestä saisi. Toteutetuissa algoritmeissa ratkaisu sai pisteitä kulkemastaan matkasta, joten painetta lyhimmän reitin löytämiselle ei ollut. Toisaalta tämä auttaa algoritmin alkuvaiheessa niiden ratkaisujen selviytymistä, jotka eivät törmää seinään tai itseensä.

Algoritmin testaamissuunnitelmasta poikettiin taulukon 1 testiajoissa 4 ja 5. Testiajossa 4 käytettiin turnausvalintaa, mikä pienentää hyvän ratkaisun valituksi tulemisen todennäköisyyttä. Arviointikriteerien mukaisia ratkaisuja ei löydetty kuusi tuntia kestäneessä testiajossa, joten menetelmä hylättiin tässä sovelluksessa. Testiajossa 5 populaation koko on huomattavasti suurempi kuin muissa neljässä. Tästä syystä viimeisen ajon kantapopulaatio alustettiin jo, kun tallennettuja ratkaisuja oli 25 kappaletta.

Työssä löydetyt ratkaisut onnistuivat tehtävässään kohtuullisesti. Keskimäärin parhaisiin tuloksiin päästiin taulukoiden 1 ja 2 mukaisesti 2. ajokerralla. Matopelin ohjaimen valitsemisen valintatekniikaksi soveltui parhaiten onnenpyörävalinta. Muut valintatekniikat eivät toimineet yhtä hyvin todennäköisesti ongelman monimutkaisuuden takia. Onnenpyörävalinta antaa kelpoisimmille yksilöille mahdollisuuden dominoida valintaa, eivätkä löydetyt hyvät ratkaisut poistuneet populaatiosta yhtä helposti, kuin muita valintatekniikoita käytettäessä.

6. YHTEENVETO

Työssä käsiteltiin geneettisten algoritmien keskeisimpiä toimintaperiaatteita ja kirjoitettiin ohjelma, joka opetti matopeliä pelaamaan itse itseään geneettisen algoritmin avulla. Madon ohjaaminen pelilaudalla tapahtui syöttämällä sen näkökenttää esittävät luvut neuroverkolle, jonka ulostulot kertoiva mihin suuntaan madon tulisi seuraavaksi liikkua. Neuroverkon painot etsittiin geneettisen algoritmin avulla.

Työn toteutusvaiheessa löydettiin useita toimivia ratkaisuja valittuun sovellukseen, mutta yhdenkään löydetyn ratkaisun ei voida sanoa olevan ihmistä taitavampi pelaaja. Löydetyt ratkaisut olivat yksinkertaisia, mutta geneettinen algoritmi toimi sovelluksessa hyvin.

Työn tuloksia arvioitaessa konkretisoitui kelpoisuusfunktion tärkeys geneettisen algoritmin toiminnan kannalta. Jos kirjoitettua ohjelmaa haluaisi jatkokehittää, pelkkää kelpoisuusfunktiota muokkaamalla tulokset saattaisivat parantua huomattavasti.

LÄHTEET

[1] S.N.Sivanandam & S.N. Deepa*, Introduction to Genetic Algorithms*, Intia, 2008

[2] S. Chakraverty & S. Mall*, Artificial Neural Networks for Engineers and Scientists*, 2017