Viljami Järvinen

Automaattinen ohjelmointi käyttäen geneettistä ohjelmointia

Tietotekniikan kandidaatintutkielma

26. toukokuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Viljami Järvinen

Yhteystiedot: viaielja@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Automaattinen ohjelmointi käyttäen geneettistä ohjelmointia

Title in English: Automatic programming using genetic programming

Työ: Kandidaatintutkielma

Sivumäärä: 26+0

Tiivistelmä: Kirjallisuuskatsauksessa perehdytään geneettiseen ohjelmointiin automaattisen ohjelmoinnin työkaluna, sen perusteisiin ja sen käyttötarkoituksiin, millaisten ongelmien ratkaisuun se kelpaa, ja mitkä ovat haasteita sen kehittämisessä. Geneettisessä ohjelmoinnissa havaittiin olevan pohja kirjallisuudessa, mutta geneettisen ohjelmoinnin toteutuksien havaittiin olevan tutkimustyötä runsaampia. Toisin sanoen teknologian sovellukset menevät tutkimuksen edellä. Yleisin haaste alalla ovat geneettisen ohjelmoinnin rajoittuneisuus pienten ongelmien ratkaisuun, sillä suuressa skaalassa geneettisen ohjelmoinnin suoritusaika kärsii huomattavasti, mikä on yleinen ongelma automaattisessa ohjelmoinnissa.

Avainsanat: automaattinen ohjelmointi, geneettinen ohjelmointi, ohjelmistokehitys, ohjelmointi

Abstract: In this literature review we take a look at genetic programming as a tool, it's definitions, current use and what it is capable of solving, and what kind of challenges are faced in its development. Genetic programming is found to be well-based in theory in literature However, it has to be noted that the solutions and implementations are more numerous than research papers. In other words, technological advancements go before definitions and scientific research. The most common problem to be found with genetic programming is that of scalability. Large blocks of code and complex problems increase run-time significantly, which is a common problem in the field of automatic programming.

Keywords: automatic programming, AP, Genetic Programming, GP, software development

Jyväskylässä 26. toukokuuta 2021

Kuviot

Kuvio 1. Kaavio geneettisen ohjelmoinnin pääsilmukasta 5
Kuvio 2. Kuva geneettisen ohjelmoinnin yksilöiden esitystavoista 7
Kuvio 3. Kaavio geneettisen ohjelman valmistelevista vaiheista 8
Kuvio 4. Kaavio risteytysoperaatiosta 12
Kuvio 5. Kaavio mutaation toiminnasta. 13
Kuvio 6. Esimerkki ohjelman luomisesta 14

Sisällys

1 JOHDANTO 1

2 GENEETTINEN OHJELMOINTI OSANA AUTOMAATTISTA OHJELMOINTIA 3

3 GENEETTINEN OHJELMOINTI 5
3.1 Erilaisia tapoja kuvata yksilöitä 6

4 GENEETTISEN OHJELMOINNIN VAIHEET 8
4.1 Alkuperäisen populaation luonti 9
4.2 Kelpoisuuden tarkistaminen 9
4.2.1 Turnausvalinta 10
4.2.2 Lexicase-valinta 11
4.3 Uudelleenkasvatus 11
4.4 Yksinkertaisen ohjelman etsintä geneettisellä ohjelmoinnilla 14

5 VIIMEAIKAINEN KEHITYS GENEETTISEN OHJELMOINNIN ALALLA 16

6 YHTEENVETO 18

LÄHTEET 19

1 Johdanto

Perinteisesti automaattisella ohjelmoinnilla tarkoitetaan niitä tapoja, joilla parannetaan koodin kirjoittajan tuottavuutta tuottamalla lähdekoodia automaattisesti ilman, että käyttäjän tarvitsee itse sitä kirjoittaa (Barr ja Feigenbaum 1982, s. 295). Uudemman ja hieman suppeamman kuvailun automaattiselle ohjelmoinnille ovat luoneet O'Neill ja Spector (2020), jotka määrittelevät automaattisen ohjelmoinnin tietokoneohjelman automaattiseksi luomiseksi korkean tason tiedonannon perusteella. Tämän määritelmän mukaan käyttäjän ei siis välttämättä tarvitse tuntea ohjelmointikielen syntaksia täysin saavuttaakseen haluamansa lopputuloksen.

Automaattisesta ohjelmoinnista on tullut yhä tärkeämpi osa ohjelmistotuotantoa, sillä ohjelmistojen merkitys ihmisten jokapäiväisessä elämässä on jatkuvassa kasvussa. Tästä syystä ohjelmoinnin automatisointi on kunnianhimoinen tavoite ja sen saavuttamisella on kauaskantoiset vaikutukset (Sekanina ja Hu 2019). Vuonna 2011 järjestetyssä tietotekniikan konferenssitapahtumassa International Conference on Internet Computing and Information Services todettiin, että automaattinen ohjelmointi on yksi suurimmista tietojenkäsittelytieteiden tavoitteista, ja geneettisen ohjelmoinnin teoriat ja menetelmät ovat osoittautuneet lupaaviksi tavoitteen saavuttamiseksi (Wang ja Wang 2011). Geneettinen ohjelmointi on automaattisen ohjelmoinnin alahaara, joka on saanut vaikutteita Darwinin evoluutioteoriasta (Sekanina ja Hu 2019). Geneettinen ohjelmointi nykyisellään kykenee ratkaisemaan ainakin aloittelevan ohjelmoijan tasoa vastaavia ongelmia tyydyttävästi (Igwe ja Pillay 2013).

Geneettinen ohjelmointi pyrkii luomaan valmiita ohjelmia vajaista tai keskeneräisistä ohjelmien osista niin, että ne täyttävät määrätyt kriteerit. Geneettinen ohjelmointi perustuu havaintoon siitä, että Darwinin evoluutioteoriaa voidaan soveltaa automaattisessa ohjelmoinnissa (Poli ja Koza 2014, s. 150). Periaatteena on, että populaation elinkelpoisimmat yksilöt lisääntyvät muita useammin, ja näin risteytyksen ja mutaatioiden kautta elinympäristöönsä syntyy entistä paremmin sopeutuneita yksilöitä. Tätä periaatetta voidaan soveltaa myös tietokoneohjelmia kehittäessä. Geneettinen ohjelmointi pyrkii vastaamaan siihen, miten tietokoneen voisi saada ratkaisemaan ongelma niin, että sitä ei tarvitsisi ohjelmoida eksplisiittisesti tekemään ratkaisu (Poli ja Koza 2014, s. 142). Fang ja Li (2010) toteavat, että geneettisen oh-

jelmoinnin ominaisuudet käyvät hyvin datan louhintaan, finanssien ennustamiseen, kuvien hahmontunnistukseen sekä symboliseen regressioon (engl. symbolic regression).

Tämä tutkimus on kirjallisuuskatsaus, jossa selvitetään geneettisen ohjelmoinnin osaa automaattisen ohjelmoinnin haarana. Lisäksi perehdytäänn geneettisen ohjelmoinnin perusteisiin ja nykyisiin käyttötarkoituksiin ja millaisten ongelmien ratkaisuun se kelpaa, sekä mitkä ovat haasteita sen kehittämisessä.

Luvussa 2 määritellään geneettiseen ohjelmointiin automaattisen ohjelmoinnin haarana, sekä lyhyesti selitetään mitä automaattinen ohjelmointi on. Luvussa 3 käydään läpi geneettisen ohjelmoinnin perustaa. Luvussa 4 tarkastellaan joitakin geneettisessä ohjelmoinnissa käytettyjä rakenteita ja selitetään geneettisen ohjelmoinnin vaiheita syntaksipuun kautta. Luvussa 5 tarkastellaan lähivuosina julkaistuja tutkimuksia, joilla on ollut suuri merkitys geneettisen ohjelmoinnin alan edistämiseen. Luvussa 6 esitetään kirjallisuuskatsauksen yhteenveto.

2 Geneettinen ohjelmointi osana automaattista ohjelmointia

Automaattisella ohjelmoinnilla tarkoitettiin käsitteen syntyaikoina kaikkia niitä tapoja, joilla parannetaan koodin kirjoittajan tuottavuutta (Barr ja Feigenbaum 1982, s. 295). 1950- ja 1960-luvulla käsitettä käytettiin kuvaamaan jonkin järjestelmän kykyä luoda automaattisesti konekieltä (O'Neill ja Spector 2020). Käsitteen varsinainen merkitys on muuttunut suurilta osin vuosikymmenten aikana (Parnas 1985). Automaattisena ohjelmointina voidaan pitää esimerkiksi tavallisen kääntäjän toimintaa, tai lähes itsenäisesti ohjelman kirjoittamiseen kykenevää keinoälyllä luotua kehitysympäristöä. Nykyisellään tunnettua ohjelmointia voidaan jopa rinnastaa siihen, mitä ennen kutsuttiin automaattiseksi ohjelmoinniksi (Hui ja Chun 2004).

Geneettinen ohjelmointi on geneettisistä algoritmeistä (engl. genetic algorithms) kehitetty ohjelmoinnin haara. Geneettiset algoritmit ja geneettinen ohjelmointi ovat automaattisen ohjelmoinnin alahaaroja, jotka ovat saaneet vaikutteita Darwinin biologisen evoluution teorioista ratkaistakseen ongelmia ohjelmoinnin synteesissä, parantamisessa ja korjaamisessa (McDermott ja Castelli 2017, s. 6). Geneettinen ohjelmointi (engl. genetic programming, lyh. GP) on tietotekniikan alalla verrattain uusi automaattisen ohjelmoinnin haara (Eiben ja Smith 2015, s. 15).

Automaattisen ohjelmoinnin tavoitteet ovat muuttuneet yhä enemmän siihen, että ohjelmistokoodia tai ohjelmistoja voitaisiin tuottaa korkeamman abstraktion kielillä. Esimerkiksi O'Neill ja Spector (2020) määrittelee automaattisen ohjelmoinnin tietokoneohjelman automaattiseksi luomiseksi korkean tason tiedonannon perusteella. Tavoitteena on saada tietokone seuraamaan enemmän semantiikkaa sisältävää ohjeistusta syntaktisesti oikeellisen ohjeistuksen sijaan (O'Neill ja Spector 2020).

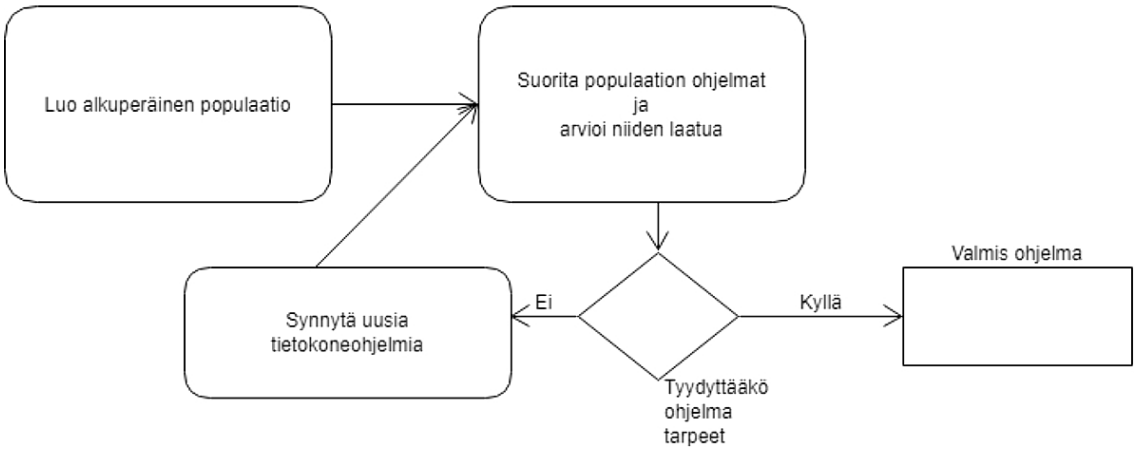
Tietokoneen matalan tason kielistä ja rajapinnoista on siirrytty yhä korkeamman tason kieliin ja abstraktioihin. Voidaan sanoa, että tavoitteena ei enää ole kertoa tietokoneelle miten sen pitää tehdä asioita, vaan mitä sen pitää tehdä (Poli ja Koza 2014, s. 171). Geneettisellä ohjelmoinnilla voidaan tietokoneelle kertoa, millainen ohjelman tuloksen pitää olla ilman,

että sille kerrotaan ohjelman rakenteesta mitään (Poli ja Koza 2014, s. 171).

3 Geneettinen ohjelmointi

Geneettinen ohjelmointi on evolutiivinen algoritmi, joka tarkastelee ohjelman tilaa, toisin kuin muut yleiset evolutiiviset algoritmit, jotka tarkastelevat ratkaisujen tilaa (Schmid 2003, s. 124). Geneettinen ohjelmointi tuottaa ratkaisun ongelmaan kehittämällä sille annettua ohjelmaa tai sen osia ja kehityksen tuloksena saadaan ratkaisu asetettuun ongelmaan (Igwe ja Pillay 2013).

Geneettisessä ohjelmoinissa pyritään luomaan ohjelma, joka palauttaa käyttäjän määrittelemän ongelman ratkaisuksi tietokoneohjelman. Menetelmänä on käyttää sellaisenaan epäkelvollisia koodeja tai sen osia, joista yhdistelemällä, muuttamalla ja parantelemalla saadaan ohjelma, joka täyttää vaatimukset (Poli ja Koza 2014, s. 142). Kuviossa 1 nähdään geneettisen ohjelmoinnin pääsilmukka.



Kuvio 1. Geneettisen ohjelmoinnin pääsilmukka

Vaikka geneettinen ohjelmointi on verratain uusi paradigma, on se yksi suurimpia automaattisen ohjelmoinnin paradigmoja (O'Neill ja Spector 2020). O'neill'n ja Spectorin (2020) mukaan Geneettisen ohjelmoinnin suosio johtuu siitä, että se, että yleisesti tietokoneella laskennallisen ongelman ratkaisemiseksi kirjoitetaan tietokoneohjelma. Geneettinen ohjelmointi kykenee tähän hyvin tehokkaasti esimerkiksi symbolisen regression keinoin (O'Neill ja Spector 2020). Geneettisiin algoritmeihin verrattunna geneettinen ohjelmointi on paljon voimakkaampi työkalu erilaisten ongelmien raktaisemiseen, koska sen tuotteena on uusi oh-

jelma, kun taas geneettinen algoritmi tuottaa tulosteena vain arvon (Sivanandam ja Deepa 2008). Toisin sanoen, geneettinen ohjelmointi antaa vastauksena ongelman ratkaisevan ohjelmakoodin, eikä pelkästään ratkaisua. Geneettistä ohjemointia on kuvailtu myös eräänlaiseksi keksintökoneeksi (O'Neill ja Spector 2020), jonka keksimät ratkaisut on näytetty olevan kilpailukykyisiä ihmisten kirjoittamien ratkaisujen kanssa (Hyde, Burke ja Kendall 2013).

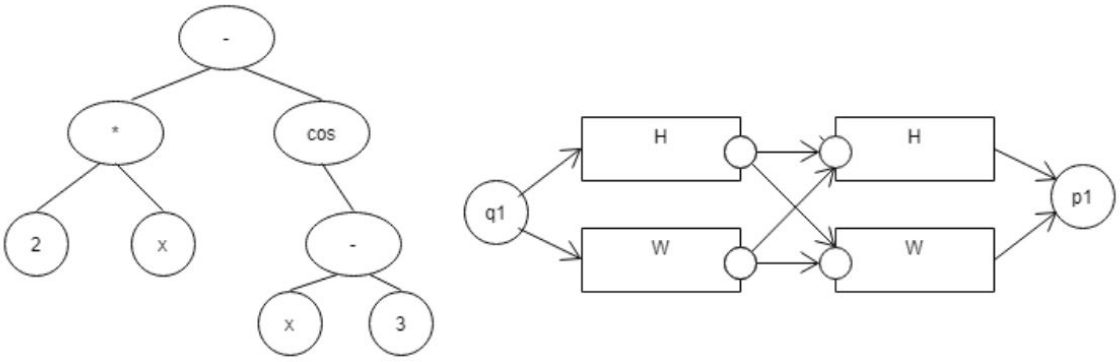
Geneettisen ohjelmoinnin ensimmäisenä varsinaisena määrittelijänä pidetään John R. Kozaa, joka yhdessä Ricardo Polin kanssa kirjoitti aiheesta kirjassa Search Methodologies vuonna 1992. Kirjassa Koza ehdottaa automaattisen ohjelmointiin tarvittavia askeleita. Sivanandam (2008) on lisäksi ehdottanut geneettiselle ohjelmoinnin järjestelmille 16 ominaisuutta, joista mahdollisimman moni tulisi täyttää. Erityisen tärkeänä hän pitää viimeistä ominaisuutta, eli geneettisen ohjelmoinnin ratkaisun kykyä kilpailla ihmisten luomien ratkaisujen kanssa. Poli ja Koza (2014, s. 162) ehdottavat, että ohjelma on kilpailullinen ihmisten luomien ratkaisujen rinnalla, jos se täyttää yhden tai useamman 8:sta eri kriteeristä. Ehdotetuissa kriteereissä arvioidaan sitä, tuottaako ohjelma yhtä hyvän tai paremman tuloksen kun aiemmat ratkaisut, tai ratkaiseeko ohjelma jonkin vielä ratkaisemattoman ongelman.

3.1 Erilaisia tapoja kuvata yksilöitä

Geneettisessä ohjelmoinnissa jokainen populaation yksilö on tietokoneohjelma. Geneettinen ohjelmointi parantelee iteroimalla jotain alkuperäistä ehdotettujen ohjelmien joukkoa arvioimalla, valikoimalla ja uudistamalla niiden rakennetta. Jokaista joukon osaa kutsutaan yksilöksi tai kromosomiksi (Poli ja Koza 2014, s. 143).

Yleinen tapa kuvata yksilöä on syntaksipuu (Sivanandam ja Deepa 2008), joka luodaan satunnaisesti yhdistämällä osia elementtejä funktiosta ja päätteiden joukon, joita kutsutaan primitiiveiksi (Poli ja Koza 2014, s. 145). Muita tapoja kuvata geneettisen ohjelmoinnin tuottamia ohjelmia ovat lineeaarinen kuvaaja tai verkko (Fang ja Li 2010).

Yksilön kuvaustapaa kutsutaan esitysmuodoksi (engl. representation). Kuviossa 2 on kuvattuna kaksi yleistä tapaa kuvata geneettisen ohjelman yksilöä. Yleisimmin yksilöitä kuvataan puun solmuina, jotka sisältävät operaatioita tai funktioita, joita ohjelman pitää suorittaa. Tä-



Kuvio 2. Vasemmalla syntaksipuun rakenne ohjelmalle (+(2*X)(Cos(+ X 3))). Oikealla kuva verkkoesityksestä, jossa ei ole semantiikkaa

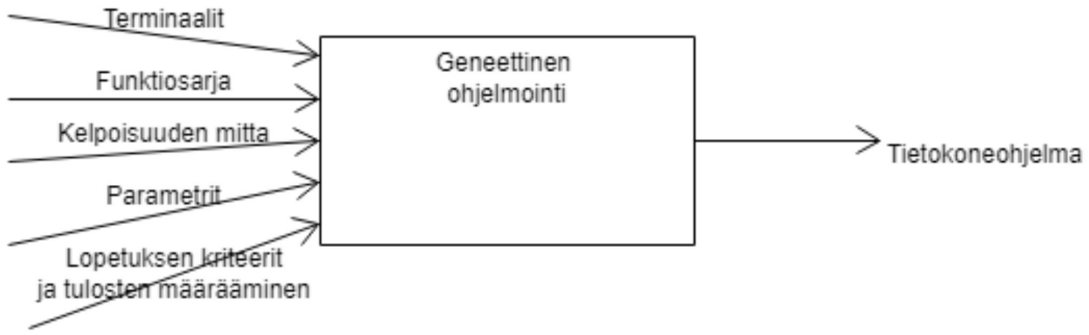
mänkaltaisia operaatioita voivat olla esimerkiksi perinteiset matemaattiset operaatiot kuten yhteenlasku, erotus, kertolasku ja jakolasku (Poli ja Koza 2014, s. 152). Päätteet sisältävät muuttujia, jotka kuvaavat ongelman alkuperäistä syötettä, sekä vakiotermejä. Funktiosolmulla on aina tietty muuttujamäärä, mikä kuvaa funktion argumenttien määrää. Päätteillä ei ole yhtään argumenttia, ja ne ovat puun lehtisolmuja, eli solmuja, joihin puun haara päättyy (Schmid 2003, s. 134).

Lineaarinen geneettinen ohjelmointi (engl. linear genetic programming) tuottaa lineaarisia esityksiä yksilöstä. Lineaarisessa esitysmuodossa yksilöt ovat ohjelmakoodin osia, joihin on liitetty syöte ja tuloste, ja ne muistuttavat läheisesti alkuperäistä ohjelmakoodia. Jokaiselle ohjelmalle annetaan syöte ja se tuottaa tulosteen, jota voidaan arvioida ja vertailla. Jos syötteitä on ohjelmia vähemmän, käytetään samoja syötteitä uudelleen samassa järjestyksessä täyttämään seuraavat ohjelmat, kunnes kaikilla ohjelmilla on syöte (Sotto ym. 2020).

Atkinsonin (2019) mukaan verkkoesityksessä geneettinen ohjelma kuvataan syötteenä, solmuina ja tulosteena, joita yhdistävät polut. Ohjelman rakenne voidaan kuvata sen kulkemana reittinä solmujen läpi, mikä tuottaa yhdessä syötteen kanssa tulosteen. Verkkoesityksen hyvänä puolena pidetään, että se voi tuottaa yhdellä verkolla useita ohjelmia ja ohjelmien sisältämää koodia voidaan käyttää uudelleen (Atkinson, Plump ja Stepney 2019, s. 968).

4 Geneettisen ohjelmoinnin vaiheet

Geneettinen ohjelmointi käy läpi useamman vaiheen suorituksen (engl. run) aikana ennen ratkaisuun saapumista. Yksi suorituskerta käsittää kaikki vaiheet geneettisen ohjelmoinnin valmisteluvaiheista siihen, että tuloksena saadaan asetetun ongelman ratkaiseva ohjelma. Suorituksen vaiheet jaetaan kahteen suurempaan osioon, joilla on omat pienemmät askeleensa. Ensimmäinen suuri osio on valmistelevat vaiheet, ja toinen on suoritettavat vaiheet (Sivanandam ja Deepa 2008). Valmistelevassa vaiheessa määritellään ja valitaan suoritukseen halutut osat ja määritetään sen lopettavat vaatimukset, ja suoritettavassa vaiheessa itse ohjelmakoodi luodaan geneettisellä ohjelmoinnilla (Poli ja Koza 2014, s. 144).



Kuvio 3. Geneettisen ohjelman valmistelevat vaiheet (Mukaillen Poli ja Koza (2014))

Geneettiseen ohjelmaan on ennen ohjelman ajoa Polin (2014, s 145) mukaan määriteltävä viisi asiaa. Nämä asiat on esitetty kuviossa 3. Ensimmäisenä pitää määritellä terminaalit, eli päätteet. Päätteiden joukko sisältää muuttujia, jotka useimmissa tapauksissa kuvaavat ongelman alkuperäisiä syötteitä ja vakiotermejä (Schmid 2003, s. 134). Toinen askel sisältää primitiivisten funktioiden määrittelyn. Primitiiviset funktiot sisältävät ohjelmakoodia. Primitiivisten funktioiden tulee olla sellaisia, että ne kykenevät käsittelemään minkä tahansa toisen primitiivin antaman tulosteen. Kolmannessa askeleessa valitaan kelpoisuuden mittari, eli millä tavalla yksilöitä arvioidaan. Neljäs askel sisältää geneettisen ohjelmoinnin säätelemiseen liittyvien parametrien määrittämisen. Parametreihin kuuluvat esimerkiksi todennäköisyydet joilla valitut yksilöt mutatoituvat tai risteytyvät keskenään. Viimeiseksi määritellään sellaiset ehdot, jotka täytyy tyydyttää, jotta suorituksen voi lopettaa (Poli ja Koza 2014, s.

145).

4.1 Alkuperäisen populaation luonti

Kun valmisteluvaiheen määritykset ovat valmiit, voidaan siirtyä suorittamaan geneettistä ohjelmointia, jonka toiminta tapahtuu kolmessa osassa. Jotta geneettisellä ohjelmoinnilla voidaan saavuttaa hyviä tuloksia, on sitä varten ensin kasvatettava suuri populaatio yksilöitä (Poli ja Koza 2014, s. 146). Populaation pieni koko saattaa johtaa huonompiin tuloksiin verrattuna suurempaan populaatioon (Sastry, Goldberg ja Kendall 2014, s. 94). Alkuperäisen, satunnaisesti luodun populaation voidaan katsoa olevan satunnainen otanta (Schmid 2003, s. 135). Jokaista samaan aikaan käsiteltävää yksilöiden joukkoa kutsutaan sukupolveksi (engl. generation).

Jokaista populaation jäsentä kuvaa syntaksipuu, joka on kasvatettu käyttämällä jotain menetelmää, joista yleisimmät ovat Kozan alunperin ehdottamat full- ja grow-menetelmät (Poli ja Koza 2014, s. 147). Menetelmän käyttö päätetään satunnaisesti valitsemalla toinen, ja niillä tuotetaan geneettisen ohjelmoinnin syntaksipuita. Full-menetelmässä algoritmi tuottaa satunnaisia puita alkuperäisisät ohjelmakoodin osista, kunnes se on saavuttanut alkuperäisen syvyytensä, kun taas grow-menetelmä tuottaa kaiken pituisia puita, kunhan niiden syvyys ei ylitä alkuperäistä syvyyttä (Burke, Gustafson ja Kendall 2003). Lopulta jokaisella solmulla täytyy olla lehtisolmu, jossa on pääte (Schmid 2003, s. 134). Nykyään alkuperäisten grow- ja full-menetelmien lisäksi suosittu menetelmä on Ramped Half-and-half -menetelmä, joka mukailee Kozan alunperin ehdottamia menetelmiä (Burke, Gustafson ja Kendall 2003). Lopullisessa alkuperäisessä populaatiossa voi olla tuhansia yksilöitä, jotka syötetään valintavaiheeseen (Schmid 2003, s. 134).

4.2 Kelpoisuuden tarkistaminen

Yksilön kelpoisuus, tai elinkelpoisuus, tarkoittaa yksilön kykyä ratkaista määritetty ongelma. Fitness checking, eli kelpoisuuden tarkistaminen tarkoittaa jokaisen yksilön kelpoisuuden arvioimista ja valintaa populaatiosta (Poli ja Koza 2014, s. 147). Tarkistuksen tavoitteena on löytää sellaisia yksilöitä, joiden kelpoisuus on erityisen korkea (Poli ja Koza 2014, s. 149).

Jos valinta kohdistuu aina maksimaalisen kelpoisiin yksilöihin, jotka kopioidaan sellaisenaan seuraavaan sukupolveen, puhutaan elitismistä (Kelly, Hemberg ja OReilly 2019, s. 154).

Kun uusi sukupolvi on on saatu luotua, valitaan siitä kelpoisimmat ohjelmat. Mikäli valittu ohjelma on jo tarpeeksi kelpoinen, voidaan prosessin katsoa olevan valmis, ja kyseinen ohjelma palautetaan. Kelpoisuuden ollessa liian matala, esimerkiksi liian suurien virheiden takia, valitaan populaatiosta yksi tai kaksi yksilöä jalostusta varten (Poli ja Koza 2014, s. 148).

Valintakriteerit vaihtelevat luonnollisesti ratkaistavan ongelman luonteen mukaan. Valintatapoja on useita, mutta suosituimmiksi ovat nousseet turnausvalinta (engl. tournament selection) ja lexicase-valinta. Turnausvalinnan suoritusaika on lineaarinen populaation kokoon nähden, mikä tekee siitä muihin valintatapoihin nähden nopean (Fang ja Li 2010). Lexicase-valinta taas tuottaa turnausvalintaa monipuolisemman populaation (Helmuth ym. 2016, s. 150). Lisäksi muita valintatyyppejä ovat roulette selection ja rank selection (Pantridge 2018, s. 1914).

4.2.1 Turnausvalinta

Perinteinen tapa valita kelpoisimmat yksilöt sukupolvesta on käyttää Kozan ehdottamaa turnausvalintaa (Poli ja Koza 2014, s. 148). Turnaukseen valitaan testattavaksi satunnaisesti joukko ohjelmia, ja niiden välillä tehdään vertailua sopivuudesta (Poli ja Koza 2014, s. 148). Pääasiassa geneettinen ohjelmointi tuottaa seuraavan sukupolven risteytyksellä, joka tuottaa kaksi jälkeläistä kahdesta vanhemmasta ja mutaatioilla, joka tuottaa yhden jälkeläisen yhdestä vanhemmasta (Fang ja Li 2010). Jos seuraavan sukupolven koko halutaan pitää yhtä suurena kuin vanhempien sukupolvi, on turnauksia tehtävä yhtä monta kertaa, kuin on vanhempia. Yleinen yksilöiden määrä yhdelle turnaukselle on 2, 4 tai 7 yksilöä (Fang ja Li 2010).

Vertailua tehdään sopivuusfunktion perusteella, joka vertailee alkuperäistä syötteille haluttua tulostetta arvioitavan ohjelman tulosteeseen (Kelly, Hemberg ja OReilly 2019, s. 155). Sopivuuden mittaaminen voi tapahtua useamman kuin yhden mitattavan ominaisuuden perusteella, tai lopulta ennen vertailua mitattavat ominaisuudet voidaan tiivistää yhdeksi muuttujaksi,

esimerkiksi luvuksi (Poli ja Koza 2014, s. 147).

Turnausvalinnassa on yleinen tapa mitata sopivuutta on yksilön oikeiden vastausten osuus kaikista vastauksista. Yleensä arvot käännetään virheluvuksi, joka kertoo paljonko tietty yksilö tuottaa virheitä, kuinka suuri on sen tuottamien virheiden yhteenlaskettu summa, tai joskus sen arvon neliöjuuri (Poli ja Koza 2014, s. 148). Parhaan arvion saanut ohjelma palautetaan turnauksen tuloksena, ja siitä tehdään vanhempi kasvatusvaihetta varten (Fang ja Li 2010).

4.2.2 Lexicase-valinta

Toinen yleinen tapa mitata sopivuutta on lexicase-valinta, jonka on todettu parantavan valittavien ohjelmien kelpoisuutta verrattuna perinteiseen turnausmenetelmään ja muihin lexicasen kaltaisiin semantiikkaan pohjautuviin menetelmiin (Helmuth ja Spector 2015, s. 1045). Lexicase-valinnan kehittämisen taustalla on ollut halu vähentää yksilöiden tuottamien tulosten välistä vertailua keskenään (Helmuth ym. 2016, s. 152).

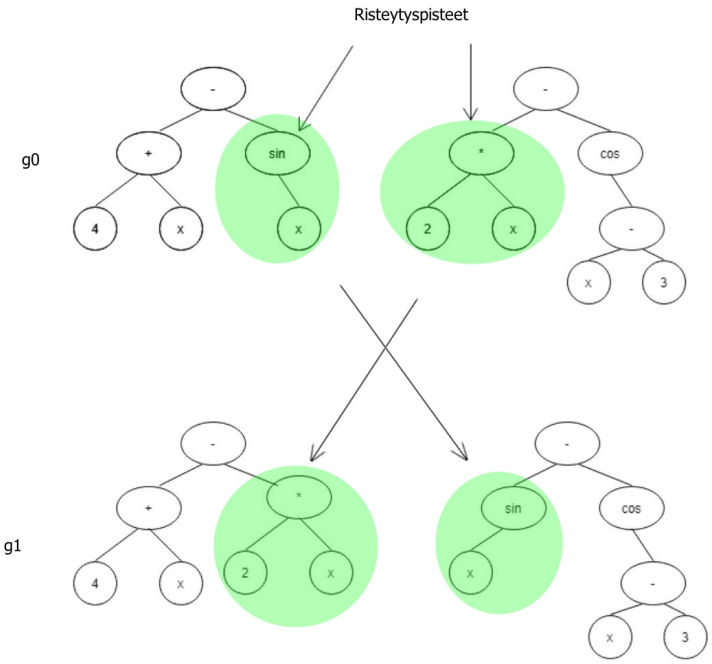
Lexicase-valinnassa populaation yksilöistä testataan yksi kerrallaan, ja jokaiselle lasketaan sen virhettä kuvaava moniulotteinen vektori, jota käytetään valinnan tekemiseen. Vain yksilöt, jotka toimivat parhaalla tavalla yhdellä tai usealla mitattavista alueista, valitaan jatkoon (Helmuth ym. 2016, s. 152). Viimeiseksi valinnasta selvinnyt yksilö palautetaan. Jos useampi yksilö omaa täysin saman vektorin, valitaan niistä yksi satunnaisesti. Lexicase-valinnan yksi tärkeäksi katsotuista ominaisuuksista on se, että sen jokaisen palauttaman vanhemman valitsemiseen käytetyt testit sekoitetaan satunnaisesti. Jos sekoitusta ei tehtäisi, jäisi aina jälkimmäisenä tulevien testien tulosten vaikutus hyvin pieneksi, mikä painottaisi aiempien testien tärkeyttä virhevektorin laskennassa (Helmuth ym. 2016, s. 152).

4.3 Uudelleenkasvatus

Uudelleenkasvatusvaiheessa valintavaiheen aikana valitut kelpoisimmat yksilöt kasvatetaan uudeksi sukupolveksi vanhemman sukupolven tilalle. Uudet yksilöt luodaan käyttäen uudelleenkasvatusmenetelmiä, joita esimerkiksi syntaksipuulle ovat lisääntyminen, risteytys, mutaatio ja ryhmä operaatioita, joita kutsutaan nimellä rakennetta muuttavat operaatiot. Eri

menetelmille voidaan määrittää eri mahdollisuudet tapahtua, mikä voi olla tarpeellista ongelman laadusta riippuen (Poli ja Koza 2014, s. 177).

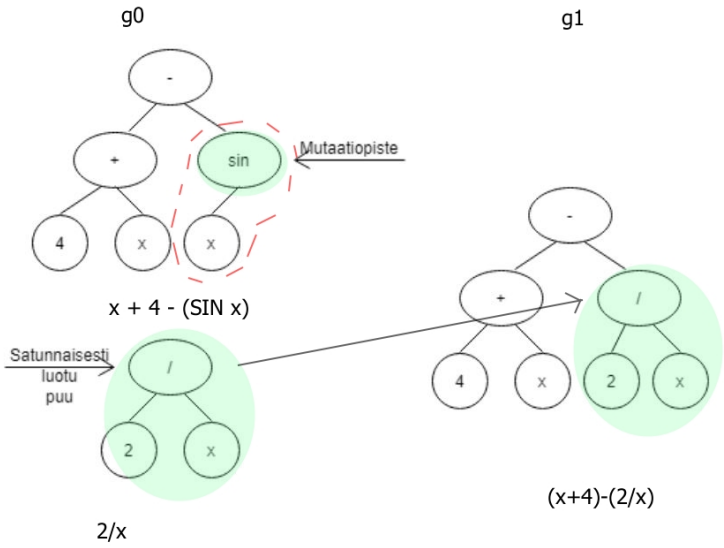
Kasvatusmenetelmistä lisääntyminen on yksinkertainen operaatio, sillä siinä alkuperäinen vanhempi kopioidaan seuraavaan sukupolveen täysin muuttumattomana (Poli ja Koza 2014, s. 154). Mutatoinnissa valitaan ensimmäiseksi syntaksipuusta piste mutaatiolle, joka voi olla esimerkiksi puun solmu. Solmu katkaistaan tästä pisteestä, ja siihen käytetään ensimmäisessä vaiheessakin käytettyä grow-menetelmää uudenlaisen haaran luomiseen (Poli ja Koza 2014, s. 154).



Kuvio 4. Kahden puun risteytys, jossa kahden puun valituista solmuista tehdään alipuut, jotka vaihdetaan keskenään, tuottaen kaksi jälkeläistä.

Risteytykseen (engl. crossover) tarvitaan kaksi vanhempaa, joiden mahdollisuus tulla valituksi satunnaisesti populaatiosta riippuu niiden kelpoisuudesta (Poli ja Koza 2014, s. 153). Molemmista vanhemmista valitaan piste risteytykselle, ja molempien pisteiden haarat vaihdetaan toisen puun pisteen kanssa, tuottaen kaksi uutta puuta seuraavaan sukupolveen (Kelly, Hemberg ja OReilly 2019, s. 154). Kuviossa 4 esitetään risteytysoperaatio syntaksipuulle.

Geneettisessä ohjelmoinnissa vanhemmat ovat yleensä hyvin erikokoisia ja muotoisia toisiinsa nähden, ja niiden risteytyksistä saadut lapset ovat myös usein erilaisia vanhempiinsa nähden (Poli ja Koza 2014, s. 153).



Kuvio 5. Puun mutaatio, jossa alkuperäiseen puuhun on kasvatettu grow-menetelmällä satunnainen osa, joka korvaa alkuperäisen puun solmun mutaatiopisteessä

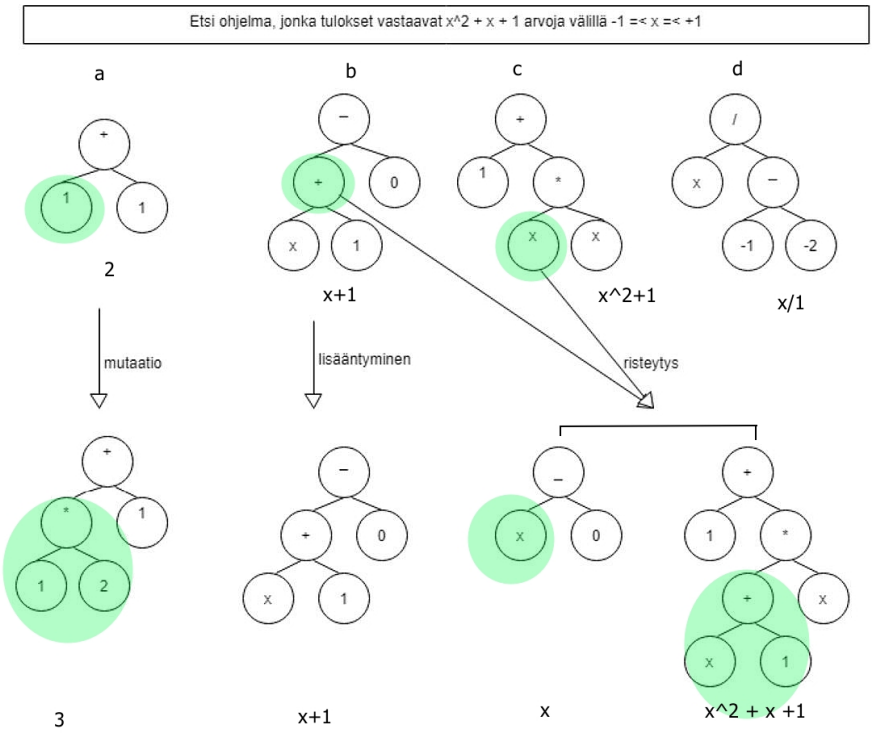
Kuviossa 5 esitetään syntaksipuun mutaatio. Mutaatiossa algoritmi muuttaa ohjelman yhtä osaa täysin satunnaisesti jollakin tavalla. Mutaatio voi olla pistemutaatio, jossa vain esimerkiksi yksi solmu mutatoidaan (Poli ja Koza 2014, s. 147), tai se voi muuttaa koko puun haaraa tai sen osaa (Schmid 2003, s. 135). Rakennetta muuttavat operaatiot ovat joukko operaatioita, jotka muuttavat ohjelman arkkitehtuuria (Poli ja Koza 2014, s. 157). Näitä operaatioita on lukuisia, ja niiden variaatioita vielä enemmän. Tämän tutkimuksen puitteissa ei ole mielekästä tarkastella jokaista erikseen.

Kozan (2014, s. 147) mukaan sukupolvia luodaan useita niin, että jokaisen sukupolven luonnin jälkeen kyseisen sukupolven yksilöiden kelpoisuus tarkistetaan, ja päätetään, onko jokin tuotetuista ratkaisuista tarpeeksi hyvä alkuperäisen ongelman ratkaisemiseksi. Jos yksikään algoritmi ei ole tarpeeksi hyvä, valitaan niistä jälleen uudelleen kelpoisimmat yksilöt vanhemmiksi, joista kasvatetaan seuraava sukupolvi, joka korvaa alkuperäisen sukupolven.

Uusia sukupolvia kasvatetaan ja niistä valitaan uudet vanhemmat niin pitkään, että niistä löytyy tarpeeksi kelpoinen yksilö ratkaisemaan ongelman, mikä lopettaa geneettisen ohjelman suorituksen (Poli ja Koza 2014, s. 147).

4.4 Yksinkertaisen ohjelman etsintä geneettisellä ohjelmoinnilla

Kuviossa 6 esitetään, miten geneettinen ohjelmointi luo alkuperäisestä sukupolvesta g0 uuden sukupolven valitsemalla kelpoisimmat yksilöt jalostettavaksi seuraavaan sukupolveen g1. Tässä esimerkissä parhaiksi turnausvalinnalla nousevat vanhemmat b ja c, koska ohjelmien tuottamat vastaukset ovat lähempänä määritellyn ongelman haluttuja ratkaisuja, kuin a:n ja d:n. Esimerkissä ilmenevät pistemutaatio, suora lisääntyminen, eli suoraan kopioiminen ja risteytys. Huomattavaa on, että vaikka b ja c ovat kelpoisimmat yksilöt, voidaan seuraavaan sukupolveen mukaan myös vanhempi a, koska satunnaisuuden säilyttämisen on todettu parantavan geneettisen ohjelmoinnin tuloksia (Pantridge 2018, s. 1917).



Kuvio 6. Geneettisellä ohjelmoinnilla luodut ohjelmat ensimmäisen sukupolven jälkeen.

Esimerkissä risteytyksen tuloksena saadaan jo toisessa sukupolvessa kysytyn ongelman par-

haiten, tässä tapauksessa täydellisesti, ratkaisema ohjelma. Koska toisen sukupolven antama ohjelma ratkaisee ongelman täydellisesti, voidaan suoritus lopettaa tähän ja ohjelma palauttaa kyseisen ohjelman (Poli ja Koza 2014, s. 147). Yleensä tällaiseen tulokseen ei päästä luonnollisissa oloissa suoritetuissa ajoissa kovin usein mielekkäässä ajassa, ellei ennen ajoa tarkoin valita solmujen sisältöä tai ongelman koko on hyvin pieni. Tällaisen ongelman ratkaisemisen arvoa voidaan pohtia, mutta toisaalta voidaan nähdä, että suuremmasta populaatiosta ja useammasta sukupolvesta olisi saattanut selvitä toinenkin ratkaisu, joka ratkaisee täyttämämme ongelman.

5 Viimeaikainen kehitys geneettisen ohjelmoinnin alalla

Tällä hetkellä geneettisen ohjelmoinnin suurimpia ongelmia ovat skaalautuvuus suurien ohjelmien kanssa, sekä skaalautuvuus suurempien yksilöiden esitysmuotojen kanssa (Kocnova ja Vasocek 2019, s. 81). Ratkaisuksi ongelmiin on kehitetty formaaleja ratkaisuja, jotka ovat mahdollistaneet esimerkiksi loogisten porttien ohjelmoinnin suuremmalla skaalalla. Kocnova ja Vasicek (2019, s. 81) julkaisivat artikkelissaan Towards a Scalable EA-Based Optimization of Digital Circuits tavan uudelleensyntetisoida paikallisista piirien verkoista tehdyistä aliverkoista uusia verkkoja, mikä lisää geneettisen ohjelmoinnin nopeutta verkkojen käsittelyn osalta. Tutkimus mahdollisti laajalti ylimääräisten piirien porttien poistamisen ja nopeamman suorittamisen perinteisiin menetelmiin verrattuna. (Kocnova ja Vasocek 2019, s. 81).

Uusia geneettisen ohjelmoinnin mittareita on ehdottanut Helmuth (2015, s. 1039), jonka julkaisema testisarja pyrkii parantamaan geneettisen ohjelman toiminnan mittaamista. Testisarjassa mitataan geneettisen ohjelman kykyä ratkaista yleisiä, aloittelevien ohjelmoijien kouluttamiseen käytettyjä tehtäviä. Testisarjan tavoitteena on parantaa geneettisen ohjelmoinnin testausta ja siten edistää tutkimustyötä. (Helmuth ja Spector 2015, s. 1039)

Rekombinaatio tarkoittaa vanhan koodin yhdistelemistä uuden koodin tuottamiseksi (Sastry, Goldberg ja Kendall 2014, s. 93). Uusi rekombinaation menetelmä keksittiin geneettiselle ohjelmoinnin työkaluksi parantamaan kaaviopohjaista geneettistä ohjelmointia. Horizontal gene transfer (HGT) saa nimensä biologian käsitteestä, ja sillä voidaan luoda kaavioita, joista poistetaan käyttämättömät osat. Toteutuksessa tunnistetaan neutraalit ja aktiiviset osiot yksilön kuvaelmasta, ja neutraalit osiot korvataan toisen yksilön aktiivisilla osioilla. Tämä toteutus suoritettuna elitistisesti tuottaa perinteistä kaaviopohjaista geneettistä ohjelmointia parempia tuloksia, kun sitä käytetään symbolista regressiota vaativissa ratkaisuissa (Atkinson, Plump ja Stepney 2019, s. 974).

Lexicase-valinnan osalta saatiin uutta tietoa sen ylivoimaisuudesta useiden ongelmien ratkaisussa, kun Pantridge et. al (2018, s. 1917) havaitsivat, että lexicase-valinta ei paranna tuloksia pelkästään populaation monipuolisuuden takia, vaan koska se valitsee nimenomaan erikois-

tuneita yksilöitä (engl. specialist). Tuloksien mukaan usein on kannattavaa valita uuteen sukupolveen yksilöitä, joiden kokonaiskelpoisuus on huono verrattuna toisiin yksilöihin, koska pitkälle erikoistuneet ohjelmat yhdistettynä seuraavassa sukupolvessa saattavat tuottaa parempia ohjelmia kuin pelkästään keskimäärin hyvien yksilöiden jalostaminen (Pantridge 2018, s. 1917).

Yksi alue, jolla geneettistä ohjelmointia on hyödynnetty laajasti jo vuosia, on säköpiirien suunnittelu elektroniikan alalla (Poli ja Koza 2014, s. 146). Yksinkertaiset sähköpiirit ovat ihmiselle tai perinteiselle verkkoihin perustuville ohjelmille mahdollista suunnitella, mutta monimutkaisempien ja laajempien sähköpiirien suunnitteluun voidaan soveltaa geneettistä ohjelmointia, koska se kykenee yksinkertaistamaan ja nopeuttamaan piirien suunnittelua huomattavasti (Eftekhar, Habib ja Hashem 2013).

6 Yhteenveto

Tutkielmassa kerrottiin geneettisestä ohjelmoinnista automaattisen ohjelmoinnin paradigmana, kuinka sen perusteet toimivat ja miten geneettinen ohjelmointi teoriassa toimii. Tutkielmassa havaittiin, että suuri osa kirjalähteistä, jotka tutkivat geneettistä ohjelmointia, nojaavat vahvasti John Kozan alkuperäisesti ehdottamaan malliin. Uusia sen pohjalta tehtyjä ratkaisuja geneettiseen ohjelmointiin on julkaistu vuosien varrella.

Suurimmiksi haasteiksi geneettiselle ohjelmoinnille havaittiin useissa lähteissä mainittu skaalautuvuus suurempien ongelmien ratkaisuun: Mitä suurempi ongelman ratkaisu, sitä suuremmaksi ongelman ratkaisuun tarvittava populaation koko paisuu samalla. Toisaalta havaittiin myös, että geneettinen ohjelmointi kykenee ratkaisemaan suurempiakin ongelmia, kuten monimutkaisten sähköpiirien suunnittelua mielekkäässä ajassa.

Suuria ongelmia, kuten satoja tai tuhansia rivejä koodia vaativat ratkaisut vaativat geneettisellä ohjelmoinnilla paljon laskutehoa saavuttaakseen tyydyttävän ratkaisun mielekkäässä ajassa. Pohjimmillaan geneettinen ohjelmointi on ennen muuta etsintäongelma, jossa etsitään sopivaa ohjelmaa ratkaisemaan ongelma. Tällaisessa tehtävässä etsittävien ratkaisujen joukko ja suoritusaika voivat helposti paisua todella suuriksi.

Tutkitun kirjallisuuden persteella nähdään, että geneettinen ohjelmointi ei vielä nykyisellään kykene ratkaisemaan automaattisen ohjelmoinnin päätavoitetta, eli tietokoneen kykyä luoda tietokoneohjelma automaattisesti korkean tason tiedonannon perusteella.

Tulevaisuudessa tulisi tutkia, miten geneettistä ohjelmointia voidaan hyödyntää yhä laajemmin automaattisen ohjelmoinnin alalla. Geneettisen ohjelmoinnin on havaittu ratkaisevan tiettyjä ongelmia tehokkaasti verrattuna muihin automaattisen ohjelmoinnin paradigmoihin verrattuna, ja sen pohjalta voidaan luoda erilaisia laajeneettavia ratkaisuja, jotka kykenevät tuottamaan monipuolisia ratkaisuja.

Lähteet

Atkinson, Timothy, Detlef Plump ja Susan Stepney. 2019. “Evolving Graphs with Horizontal Gene Transfer”. Teoksessa *Proceedings of the Genetic and Evolutionary Computation Conference,* 968–976. GECCO ’19. Prague, Czech Republic: Association for Computing Machinery. ISBN: 9781450361118. https://doi.org/10.1145/3321707.3321788.

Barr, Avron, ja E. Feigenbaum. 1982. “Chapter X - Automatic Programming”. Teoksessa *The Handbook of Artificial Intelligence,* toimittanut E. Feigenbaum A. Barr, 295–379. Butterworth-Heinemann. ISBN: 978-0-86576-090-5. https://doi.org/10.1016/B978-0-86576-090-5.50010-0.

Burke, Edmund, Steven Gustafson ja Graham Kendall. 2003. “Ramped Half-n-Half Initialisation Bias in GP”, 1800–1801. Springer, Berlin, Heidelberg. ISBN: 978-3-540-40603-7. https://doi.org/10.1007/3-540-45110-2_71.

Eftekhar, S., S. Habib ja A. Hashem. 2013. “Evolutionary Design of Digital Circuits Using Genetic Programming” (huhtikuu).

Eiben, AE., ja James E. Smith. 2015. “Chapter 6 - Popular Evolutionary Algorithm Variants”. Teoksessa *Introduction to Evolutionary Computing,* 2. painos, 13–15. Springer-Verlag Berlin Heidelberg. ISBN: 978-3-662-44873-1. https://doi.org/10.1007/978-3-662-44874-8.

Fang, Yongsheng, ja Jun Li. 2010. “A Review of Tournament Selection in Genetic Programming”. Teoksessa *Advances in Computation and Intelligence,* toimittanut Zhihua Cai, Chengyu Hu, Zhuo Kang ja Yong Liu, 5:182–185. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-16493-4.

Helmuth, T., N. McPhee, N. Freitag ja L. Spector. 2016. “Lexicase Selection for Program Synthesis: A Diversity Analysis”. Teoksessa *Genetic Programming Theory and Practice XIII,* toimittanut R. Riolo, W.P. Worzel, Mark Kotanchek ja Arthur Kordon, 151–167. Cham: Springer International Publishing. ISBN: 978-3-319-34223-8. https://doi.org/10.1007/978-3-319-34223-8_9.

Helmuth, Thomas, ja Lee Spector. 2015. “General Program Synthesis Benchmark Suite”. Teoksessa *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation,* 1039–1046. GECCO ’15. Madrid, Spain: Association for Computing Machinery. ISBN: 9781450334723. https://doi.org/10.1145/2739480.2754769.

Hui, Wendy, ja Kyong Chun. 2004. “On Software, or the Persistence of Visual Knowledge”, 29.

Hyde, M R, E K Burke ja G Kendall. 2013. “Automated code generation by local search”. *Journal of the Operational Research Society* 64 (12): 1725–1741. https://doi.org/10.1057/jors.2012.149.

Igwe, K., ja N. Pillay. 2013. “Automatic programming using genetic programming”. Teoksessa *2013 Third World Congress on Information and Communication Technologies (WICT 2013),* 1, 337–342. https://doi.org/10.1109/WICT.2013.7113158.

Kelly, Jonathan, Erik Hemberg ja Una-May OReilly. 2019. “Solution and Fitness Evolution (SAFE): Coevolution Solutions and Their Objective Functions”. Teoksessa *Genetic Programming,* toimittanut Lukas Sekanina ja Ting Hu, 1:65–66. Springer Cham, marraskuu. ISBN: 978-3-030-16670-0. https://doi.org/10.1007/978-3-030-16670-0.

Kocnova, J., ja Z. Vasocek. 2019. “Towards a scalable EA-Based Optimization of Digital Circuits”. Teoksessa *Genetic Programming,* toimittanut Lukas Sekanina ja Ting Hu, 1:81–95. Springer Cham, marraskuu. ISBN: 978-3-030-16670-0. https://doi.org/10.1007/978-3-030-16670-0.

McDermott, J., ja M. Castelli. 2017. “Genetic Programming”, 6. Springer, Cham. https://doi.org/10.1007/978-3-319-55696-3.

O’Neill, Michael, ja Lee Spector. 2020. “Automatic programming: The open issue?”, 251–262. https://doi.org/10.1007/s10710-019-09364-2.

Pantridge, E. 2018. “Specialization and Elitism in Lexicase and Tournament Selection”, 1914–1917. GECCO ’18. Kyoto, Japan: Association for Computing Machinery. ISBN: 9781450357647. https://doi.org/10.1145/3205651.3208220.

Parnas, David Lorge. 1985. “Software aspects of strategic defense systems”, 1326. http://web.stanford.edu/class/cs99r/readings/parnas1.pdf.

Poli, Riccardo, ja John R. Koza. 2014. “Chapter 6 Genetic Programming”. Teoksessa *Search Methodologies,* 144–182. Springer. ISBN: 978-1-4614-6939-1,978-1-4614-6940-7. https://doi.org/10.1007/978-1-4614-6940-7.

Sastry, Kumara, David E. Goldberg ja Graham Kendall. 2014. “Chapter 4 Genetic Algorithms”. Teoksessa *Search Methodologies,* 93–117. Springer. ISBN: 978-1-4614-6939-1,978-1-4614-6940-7. https://doi.org/10.1007/978-1-4614-6940-7.

Schmid, Ute. 2003. “Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning”. Teoksessa *Inductive Synthesis of Functional Programs,* 1:134–144. Springer-Verlag Berlin Heidelberg. ISBN: 978-3-540-44846-4. https://doi.org/10.1007/b12055.

Sekanina, Lukas, ja Ting Hu. 2019. “Preface”. Teoksessa *Genetic Programming,* 1:i–vi. Springer Cham, marraskuu. ISBN: 978-3-030-16670-0. https://doi.org/10.1007/978-3-030-16670-0.

Sivanandam, S.N., ja S.N. Deepa. 2008. “Chapter 6 - Genetic Programming”. Teoksessa *Introduction to Genetic Algorithms,* 1. painos, 452 Pages. Springer-Verlag Berlin Heidelberg. ISBN: 978-3-540-73190-0. https://doi.org/10.1007/978-3-540-73190-0.

Sotto, Léo Françoso D. P., Paul Kaufmann, Timothy Atkinson, Roman Kalkreuth ja Márcio Porto Basgalupp. 2020. “A Study on Graph Representations for Genetic Programming”. Teoksessa *Proceedings of the 2020 Genetic and Evolutionary Computation Conference,* 931–939. GECCO ’20. Cancún, Mexico: Association for Computing Machinery. ISBN: 9781450371285. https://doi.org/10.1145/3377930.3390234.

Wang, Qingchun, ja Aishu Wang. 2011. “Evolving Computing and Automatic Programming”. Teoksessa *2011 International Conference on Internet Computing and Information Services,* 213–214. https://doi.org/10.1109/ICICIS.2011.61.