Itsesäätely evolutiivisissa algoritmeissa

Marko A. Grönroos
LuK-tutkielma
Tietojenkäsittelyoppi
Matemaattisten tieteiden laitos
Turun Yliopisto, 1998

Sisältö

Luku 1

Johdanto

Evolutiiviset algoritmit (EA:t) ovat heuristisia hakumenetelmiä, jotka perustuvat biologisesta evoluutiosta johdettuihin periaatteisiin. Algoritmien osat voidaan kuitenkin tehdä usealla eri tavalla ja eri tavoillakin on lukuisia parametreja. Tavallisimpia parametreja EA:eissa ovat mutaatiotaajuus tai -jakauma, rekombinaatiotaajuus, ja valinnan voimakkuus. Parametrien optimiarvot ovat luonnollisesti riippuvia ongelmasta, johon EA:eja sovelletaan. Evolutiivisia algoritmeja on yleensä kuitenkin pidetty melko sietokykyisinä näiden parametrien suhteen ja niissä käytetään hyviksi todettuja yleisparametreja. Epäoptimaalisten parametrien käyttö saattaa kuitenkin joissain tapauksissa olla hyvin tehotonta ja johtaa suhteellisen huonojen ratkaisujen löytymiseen.

Yksinkertaisin tapa löytää sopivat parametrien arvot on suorittaa hakualgoritmi useaan kertaan eri parametrien arvoilla ja valita paras tulos. Epäsuorempi tapa on arvioida hakuavaruuden luonnetta taustatuntemuksen perusteella ja käyttää algoritmeja kyseisenlaiselle hakuavaruudelle hyviksi todetuilla parametreilla. Parametrien optimi-arvo kuitenkin muuttuu myös haun aikana algoritmin tarkentaessa hakua johonkin hakuavaruuden osaan. Tällöin parametrien muuttuminen haunkin aikana voidaan parametrisoida lisäparametreilla, joiden arvot määritellään arvioimalla kuten edellä.

Tarkastelemme tässä työssä parametrien itsesäätelyä (itseadaptaatiota, self-adaptation), jossa hakualgoritmin parametrit annetaan algoritmin itsensä optimoitaviksi. Itsesäätelyn on havaittu löytävän hyvät parametrien arvot varsin nopeasti. Arvot myös muuttuvat tarpeen mukaan haun edetessä. Täysin ei parametreilta tällöinkään säästytä, mutta uudet säätöparametrit ovat toivon mukaan sietokykyisempiä kuin alkuperäiset.

Kuvaamme itsesäätelyn toteutuksen binääriluvuilla operoivilla geneettisillä algoritmeilla (GA), sekä suoraan reaaliluvuilla operoivilla evoluutiostrategioilla (ES). Lisäksi luomme silmäyksen itsesäätelyyn rakenteellisessa oliomallissa.

Luku 2

Evolutiiviset algoritmit

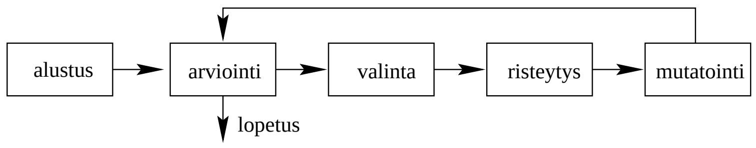
Kuvaamme tässä luvussa evolutiivisten algoritmien (EA) perusteet lyhyesti. EA:t ovat joukko laskennallisia menetelmiä, jotka ovat syntyneet biologisen evoluution periaatteiden soveltamisesta erinäisten hakuongelmien ratkaisuun. Ne ovat osoittautuneet käytännöllisiksi ja tehokkaiksi tietyntyyppisten vaikeiden ongelmien ratkaisemisessa.

2.1 Perusteet

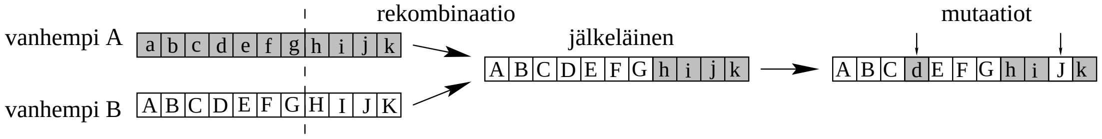
Geneettiset algoritmit (genetic algorithms, GA) ovat John Hollandin 60-luvulla kehittämä idea (Holland 1975; Bäck 1996, s. 66-91). Hän näki geneettisen koodin binääriluvuista koostuvana sarjana, »kromosomina», jota voisi evolvoida käyttämällä sopivaa algoritmia. Hänen algoritmiinsa viitataan usein *kanonisena geneettisenä algoritmina*. Toinen merkittävä haara on Rechenbergin (1973), Schwefelin (1977) ja Bienertin 60-luvulla kehittämät *evoluutiostrategiat* (Evolution Strategies, ES:t), jotka operoivat reaalilukuvektoreilla (Bäck 1996, s. 66-91). On myös EA:ien toteutuksia, jotka käyttävät muita tietorakenteita.

Evolutiivisia algoritmeja voi tietojenkäsittelyn näkökulmasta käsitellä »vain» heuristisina hakumenetelminä (-strategioina). Useimmat perinteiset hakualgoritmit käyttävät kuitenkin vain yhtä *ratkaisukandidaattia*, jota muutetaan (eli »mutatoidaan») käyttäen jotain tiettyä (usein satunnaista) sääntöä. Jos muutoksen kautta saatu ratkaisu on parempi kuin vanha, siirrytään käyttämään sitä. EA:t käyttävät yhden ratkaisukandidaatin sijasta ratkaisujen *populaatiota*. Kyseisenlaisten populaatioiden elinkaari on kuvattu kuvassa 2.1.

Yksilön (ratkaisukandidaatin) geneettinen perusta on sen *genotyyppi*. Käsitettä *genomi* käytetään viittaamaan genotyyppiin konkreettisella tasolla; biologisissa organismeissa genomi koostuu nukleiinihappojen muodostamista *kromosomeista*, kun taas GA:issa se



Kuva 2.1: Populaatioiden elinkierto EA:issa.



Kuva 2.2: Rekombinaatio ja mutaatio EA:issa.

on tyypillisesti binäärilukuvektori. Tarvitaan myös kuvaus genotyypistä *fenotyyppiin*, eli yksilön ilmiasuun. Toisin sanoen yksilön ilmiasu on »koodattu» genotyyppiin, josta se yksilön »kasvuprosessin» aikana puretaan, eli »dekoodataan».

Tämän dekoodauksen jälkeen populaation yksilöt ovat valmiita arvioitaviksi jossain *ympäristössä*. Arvioinnin päämääränä on määrittää yksilön *kelpoisuus* kyseisessä ympäristössä. Koska evolvoimme yksilöitä ratkaisuiksi johonkin ongelmaan, tuo ongelma on ympäristömme. Yksilöiden fenotyyppi saattaa olla esimerkiksi yksinkertainen reaalilukuvektori parametriksi funktiolle, jota haluamme optimoida. Biologisissa systeemeissä vastaavaa eksplisiittistä kelpoisuuden mittausta ei ymmärrettävästi tapahdu, vaan kelpoisuus ilmenee implisiittisesti organismien selviytymis- ja lisääntymiskykynä.

Arvioinnin jälkeen muodostetaan seuraava sukupolvi valitsemalla yksi tai useampi (yleensä kaksi) vanhempaa kullekin seuraavan sukupolven yksilölle. Vanhempien genomit yhdistetään risteyttämällä eli *rekombinaatiolla*. Lopulta jälkeläisten perimää *mutatoidaan* hieman kääntämällä bittejä tietyllä todennäköisyydellä. Tämä on esitetty kuvassa 2.2.

2.1.1 Evolutiivisen algoritmin toteutus

Kuvaamme seuraavaksi evolutiivisen algoritmin siten, kuin se on Cakra-ohjelmakirjastossa (Grönroos 1998) toteutettu. Määrittelemme evolutiiviset algoritmit käyttäen *oliosuuntautunutta pseudokoodia*. Aluksi tarvitsemme kuvaukset perusolioille, joihin evolutiiviset oliot perustuvat. Alla esitettävä oliohierarkia on visualisoitu kuvassa 2.3.

Olio on perusolio, jonka kaikki muut oliot perivät. Oliot ovat nimettyjä, esimerkiksi Y_3 tai p_m. Perintä ilmaistaan $A \Rightarrow B$, joka luetaan *»**A** perii **B**:n»*.

Tärkeimpiä olioluokkia ovat *Rakenne*:n ohella *atomiset olioluokat*, kuten $\Re$ ja $\{0,1\}$, sekä *operaatiot* eli funktiot eli menetelmät (method). Operaatioiden voidaan yleensä katsoa olevan abstraktin operaatioluokan $\Omega \Rightarrow Olio$ perillisiä.

Rakenne on vaihtuvanmittainen, olioita sisältävä lista, joka kuvataan seuraavasti:

$$Rakenne = (o_1, ..., o_n \in Olio) \Rightarrow Olio, \tag{2.1}$$

jossa *Olio* on yllä määritelty olio. Rakenteen sisältämien olioiden tyyppi merkitään $\in$-ilmaisulla. Samantyyppiset peräkkäiset oliot on erotettu pilkulla, erityyppiset puolipisteellä. Yläluokilta perityistä olioista ilmoitetaan vain ne, jotka määritellään perillisessä uudelleen. Määritelmä mahdollistaa rekursiiviset rakenteet. Merkitsemme annetun rakenteen r sisältämää oliota o ilmaisulla $r[o]$. Merkitsemme vastaavasti operaatioiden $\omega \in \Omega$ kutsumista $r[\omega(p_1, ..., p_n)]$, jossa p_i ovat parametreja. Operaatio voi palauttaa arvon, jolloin sitä kutsutaan funktioksi.

Rakenne on myös »luokka». Emme siis tee eroa luokan ja rakenteen, eli sisällysolion (container) välillä. Tämä yksinkertaistaa olioiden käsittelyä evolutiivisissa algoritmeissa.

Rakenteen koon tulee voida muuttua evoluutio-ajon aikana, mikäli esimerkiksi mutaatio-operaattori lisää tai poistaa geenejä. Tästä johtuen sitä ei voi määritellä normaalina matemaattisena vektorina.

Evolutiivinen rakenne on luokka:

$$Evolutiivinen = (\omega_a, \omega_m, \omega_r \in \Omega) \Rightarrow Rakenne, \tag{2.2}$$

jonka määrittelemät operaatiot ovat keskeisiä evolutiivisten algoritmien kaikille komponenteille; ω_a on alustus-, ω_m mutaatio- ja ω_r rekombinaatio-operaatio.

Alustusoperaattori on yksinkertainen rekursiivinen kutsu:

$$\omega_a() \equiv \{\forall 1 \leq i \leq n : g[o_i[\omega_a()]]\}. \tag{2.3}$$

Mutaatio-operaattori on täsmälleen samanlainen (mutta määritellään myöhemmin uudestaan). Myös rekombinaatio-operaattori on rekursiivinen ja perillisillä voi olla siitä omia versioitaan.

Tämänkaltaisia rakenteita ovat alla määritellyt genomit (»kromosomit»), geenit jotka koostuvat pienemmistä rakenteista, yksilöt, populaatiot ja korkeimmalla tasolla ympäristö. Olio-kuvaustapaa käytettäessä genomit eivät kuitenkaan ole välttämättä yhtenäisiä bittijonoja, kuten perinteisessä GA:ssa. Tämä rakenteellinen tapa mahdollistaa poikkeavien geneettisten operaattorien käytön eri geeneillä ja on yhtenevä myös *geneettisen ohjelmoinnin* (GP, genetic programming) rakenteen kanssa.

Tämän oliomallin peruste on selvä: biologisessa evolutiikassa ei ole ulkoista »algoritmia», joka evolvoisi eliöitä ulkoapäin, vaan eliöt osaavat tehdä sen itse. Eliöiden evolutiivinen algoritmi on siis koodattu niiden genomiin ja on siten myös itse altis evoluutiolle. Tämä malli mahdollistaa myös evolutiivisen algoritmin itsesäätelyn, muun muassa siten kuin itsesäätely myöhemmissä kappaleissa kuvataan (vaikkakaan tämä malli ei ole niille välttämätön).

Ympäristö määrittelee tyypillisesti vain kelpoisuusfunktion $\Phi : \phi \mapsto \Re$, jossa ϕ on yksilön fenotyyppi. Eli:

$$Ympäristö = (\Phi \in \Omega) \Rightarrow Rakenne. \tag{2.4}$$

Joissain evolutiivisten algoritmien tyypeissä, kuten tekoelämäjärjestelmissä, ympäristö voi olla tätä monimutkaisempi ja sisältää vuorovaikutusta yksilöiden kanssa ja toimia yksilöiden välisen vuorovaikutuksen välittäjänä.

Populaatio on joukko yksilöitä, jonka ominaisuutena on evolvoitua ympäristössä, jossa se elää.

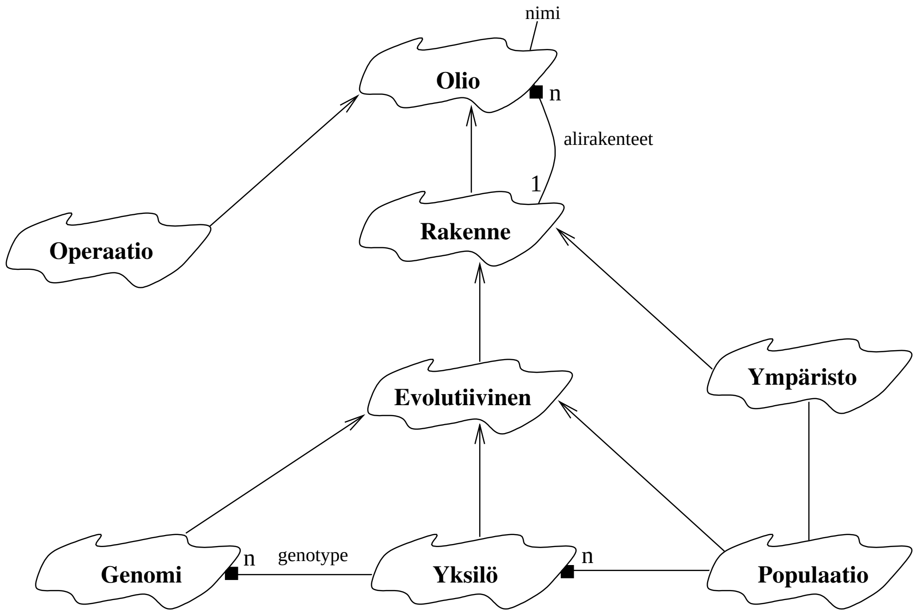
$$Populaatio = (\omega_{evolvoi}, \omega_\iota \in \Omega;\ t \in N, y_1, ..., y_\lambda \in Yksilö) \Rightarrow Evolutiivinen, \tag{2.5}$$

jossa $\omega_{evolvoi}$ on EA-strategian toteuttava korkeimman tason algoritmi, ω_ι on lopetusfunktio, joka palauttaa arvon *tosi,* jos evoluution kriteerit (tyypillisesti tietty määrä sukupolvia) ovat täyttyneet ja t on evolvoitujen sukupolvien määrä. Ilmaisemme muuttujalla λ populaation yksilöiden määrää.

Yksilö on evolutiivinen rakenne, jolla on sekä genomi $\gamma \in \Gamma$, että sille ympäristössä määritellyn kelpoisuusfunktion avulla laskettu kelpoisuus $W \in \Re$.

$$Yksilö = (\gamma \in Evolutiivinen;\ \delta \in \Omega;\ \rho \in Olio;\ W \in \Re) \Rightarrow Evolutiivinen, \tag{2.6}$$

jossa γ on yksilön genomi ja ρ on fenotyyppi, joka määritellään dekoodausfunktion avulla $\rho = \delta(\gamma)$. Kelpoisuus voidaan tällöin laskea $W = \Phi(\rho)$.



Kuva 2.3: Evolutiivisten olioiden hierarkia (esitettynä käyttäen Boochin tunnetun relaationotaation muunnelmaa).

Evolutiivinen algoritmi. Tyypillinen EA on tällöin (määriteltynä *Populaatio*-luokan operaationa $\omega_{evolvoi}$):

$Populaatio[\omega_{evolvoi}(e \in Ympäristö)] \equiv$

$\ulcorner$

$\omega_a()$; — Alustetaan

while $\omega_\iota() \neq true$ **do** — Tarkistetaan lopetusehto

$\{\forall y_i \in Yksilö : y_i[w] := e[\Phi(y_i)]\}$; — Arvioidaan kelpoisuudet

$p' \leftarrow \{y_i \in Yksilö | \forall 1 \leq i \leq \lambda : y_i \leftarrow \omega_r(\omega_s(), \omega_s())\}$; — Valitaan ja risteytetään

$\{\forall 1 \leq i \leq \lambda : y_i \leftarrow p'[y_i]\}$; — Korvataan uudella sukupolvella

$\omega_m()$; — Mutatoidaan

od;

$\lrcorner$

Tämä algoritmi sisältää useita populaation ja yksilöiden operaatioita, joita emme tässä tutkielmassa määrittele.

2.2 Koodaustapa

Vaikeimpia tehtäviä kaikessa ongelmanratkaisussa on ongelman formaali määrittely, eli sellaisen esitystavan löytäminen, jollaista käytettävä ratkaisumenetelmä osaa parhaiten käyttää. Evolutiivisia algoritmeja on käytetty binääri-, kokonais- ja reaaliluvuilla, kombinatorisilla rakenteilla, jäsennyspuilla, loogisilla lauseilla, graafeilla ja monilla muilla rakenteilla. Edellämainittua koodausta, eli kuvausta genotyypistä fenotyyppiin, ei itse asiassa välttämättä edes tarvita: voimme evolvoida fenotyyppisiä rakenteita suoraan, jos vain määrittelemme niille mutaatio- ja rekombinaatio-operaatiot. Jos haluamme kehittää algoritmin, joka on hyvin yleinen, tapahtuu algoritmin soveltaminen tällöin liittymän eli koodauksen kautta. Luonnollisesti haluamme koodauksen olevan mahdollisimman yksinkertainen.

Koodauksessa on kysymys myös skaalauksesta, sillä se voi sisältää myös tiedon pakkausta. Pakkaavat koodaukset käyttävät hyväkseen fenotyyppisissä rakenteissa olevia säännönmukaisuuksia, kuten biologisissa organismeissa on havaittavissa. Koodaus myös vääristää hakuavaruutta tavalla, joka saattaa helpottaa hakua. Liian suorassa koodauksessa haku voi juuttua paikalliseen minimiin, kun taas sopivan monimutkainen koodaus saattaa auttaa pääsemään niiden yli. Toisaalta monimutkainen koodaus saattaa vaikeuttaa tarkkojen muutosten tekemistä.

2.2.1 Reaaliluvut

Evoluutiostrategioissa reaalilukuja evolvoidaan suoraan, ilman binääristä esitysmuotoa. Tällöin niille täytyy määritellä omat mutaatio- ja rekombinaatio-operaatiot. Reaalilukujen arvoaluetta ei periaatteessa tarvitse rajata, mutta lienee selvää, että rajaaminen ongelmalle tyypilliselle arvoalueelle helpottaa hakua.

Binääristä genomia käyttävissä geneettisissä algoritmeissa tarvitaan jokin koodaus reaaliluvuille. Yleisimmin käytettyjä vaihtoehtoja ovat suora binäärikoodaus ja Gray-koodaus.

Reaalilukugeeni on rakenne:

$$Geeni_{\Re}(\delta_{\Re} \in \Omega) \Rightarrow Evolutiivinen, \tag{2.7}$$

jossa $\delta_{\Re}$ on dekoodausfunktio $\delta_{\Re} : Geeni_{\Re} \mapsto \Re$.

2.2.1.1 Evoluutiostrategiat

Evoluutiostrategiat ovat erityisesti reaalilukujen käsittelyyn erikoistuneita EA:ita, joiden geeni määritellään seuraavasti:

$$Geeni_{\Re ES} = (\delta_{\Re}, \omega_{m\Re}, \omega_{r\Re} \in \Omega;\ x \in \Re) \Rightarrow Geeni_{\Re}, \tag{2.8}$$

jossa yksinkertaisesti $\delta_{\Re} \equiv x$.

2.2.1.2 Binäärikoodaus

Binäärikoodaus on geneettisissä algoritmeissa käytetty suoraviivainen tapa reaalilukujen koodaamiseksi. Reaaliluvun binääriesitys on binäärilukuvektori $b \in \{0,1\}^l$, eli käyttämällämme oliotavalla ilmaistuna

$$Geeni_{\Re b} = (\delta_{\Re}, \omega_m, \omega_r \in \Omega;\ b_1, ..., b_n \in G_B;\ v_{min}, v_{max} \in \Re) \Rightarrow G_{\Re}, \tag{2.9}$$

jossa $b_1, ..., b_l$ on joukko binäärilukugeenejä:

$$Geeni_B = (\omega_m, \omega_r \in \Omega;\ b \in \{0,1\}) \Rightarrow G_{\Re}. \tag{2.10}$$

Binäärilukuvektori dekoodataan diskreetisti reaalilukuvälille $v \in [v_{min}, v_{max}] \Rightarrow \Re$ jakamalla arvoalue 2^l osaan siten, että

$$Geeni_{\Re b}[\delta_{\Re}] \equiv v_{min} + (v_{max} - v_{min}) \cdot \frac{\sum_{i=0}^{l-1} b_i[b] \cdot 2^i}{2^l}. \tag{2.11}$$

2.2.1.3 Gray-koodaus

Gray-koodaus on usein käytetty vaihtoehto suoralle binäärikoodaukselle. Se perustuu myös binääriluvuille, mutta mutaatiot vaikuttavat siihen jossain määrin eri tavalla. Sen uskotaan tuottavan pehmeämpiä evoluutiomaastoja. Tämä johtuu siitä, että perättäiset arvot poikkeavat toisistaan vain yhdellä bitillä. Tämä on havaittavissa taulukossa 2.1. Gray-koodattu arvo muutetaan binääriluvuksi seuraavasti:

$$b_i = \bigoplus_{j=1}^{i} g_j, \tag{2.12}$$

Kokonaisluku	**Tavallinen**	**Gray**
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Taulukko 2.1: Binäärikoodattujen ja Gray-koodattujen kokonaislukujen vertailu.

jossa g on Gray-koodattu binäärilukuvektori ja $\bigoplus$ on XOR (eli pariteetti) -operaatio; se antaa arvon 1, mikäli parametrivektorissa on pariton määrä ykkösiä, muuten arvon 0.

2.2.2 Kokonaisluvut

Kokonaislukujen koodaus vastaa reaalilukujen koodausta, *mikäli* lukualue on jaettavissa 2^n:llä (kuvaus on bijektio). Mikäli näin ei ole, kuvauksen täytyy limittyä jollain lailla. Limitystapoja on useita ja kaikissa on huonot puolensa. Yleensä limitysongelma pystytään kuitenkin välttämään valitsemalla käytettävä lukualue sopivasti.

Yleinen tapa on toteuttaa kokonaislukujen koodaukselle omat mutaatio- ja rekombinaatio-operaationsa. Arvojen ei esimerkiksi tarvitse kaikissa ongelmissa korreloida mutaatioiden suhteen, vaan ne voidaan tehdä täysin satunnaisesti (ks. alla).

2.3 Valinta

Valinta on tekijä, joka kuljettaa evolutiivista prosessia kelpoisuuden ohjaamaan suuntaan. Biologisilla organismeilla valinta muodostuu yksilöiden kyvystä selviytyä ja lisääntyä. EA:eissa valinta tehdään yksilöille määritellyn kelpoisuusarvon perusteella. Valintatavat voidaan jakaa kahteen pääluokkaan: verrannolliseen (relative) tai absoluuttiseen kelpoisuuteen perustuviin. Verrannollinen valinta perustuu kelpoisuuksien järjestykseen, kun taas absoluuttinen valinta perustuu kelpoisuusarvojen absoluuttisiin arvoihin. Valinnassa voidaan myös estää saman yksilön valitseminen molemmiksi vanhemmiksi, jos niin halutaan. Kuvaamme populaation kokoa[1] muuttujalla λ ja valinnassa huomioon otetta-

[1] Tämä määritelmä on hieman eri kuin evoluutiostrategioissa, jossa sillä tarkoitetaan jälkeläisten lukumäärää. Jälkeläisten lukumäärä on kuitenkin sama kuin populaation koko, mikäli hakustrategia ei ole elitistinen (ks. alla). Käytämme tätä määritelmää, koska pidämme sitä käsitteellisesti helpompana.

vien vanhempien lukumäärää muuttujalla μ. Useimmissa valintamenetelmissä on $\mu = \lambda$, eli koko populaatio on mahdollisia vanhempia. (μ, λ)-valinta tekee tästä (nimensä mukaisesti) poikkeuksen.

2.3.1 Suhteellinen valinta

Suhteellinen valinta (proportional selection) on suosituimpia valintamenetelmiä ja perustuu yksilöiden absoluuttisiin kelpoisuusarvoihin. Valintatodennäköisyys lasketaan yksinkertaisella kaavalla (Bäck 1996, s. 167):

$$p_i = \frac{\Phi_i}{\sum_{i=1}^{n} \Phi_i}. \tag{2.13}$$

Suhteellista valintaa kutsutaan myös nimellä *rulettivalinta* (roulette-wheel selection), joka kuvaa sen toimintaperiaatetta: jokaiselle yksilölle annetaan rulettiympyrästä oma lovi, jonka koko on suhteessa yksilön kelpoisuuteen.

2.3.2 Järjestämisvalinta

Järjestämisvalinta (ranking) ei välitä kelpoisuuksien absoluuttisista arvoista lainkaan, vaan vain yksilöiden kelpoisuusjärjestyksestä. Esitämme tässä *lineaarisen järjestämisvalinnan* (linear ranking). Populaation $P(t)$ koko ilmaistaan parametrilla λ, ja sen yksilöt järjestetään aluksi niiden kelpoisuuden mukaan. Seuraava sukupolvi $P(t+1)$ muodostetaan valitsemalla kullekin uudelle yksilölle kaksi (tai useampaa) vanhempaa järjestetyn populaation μ:n kelpoisimman yksilön joukosta todennäköisyydellä (Bäck 1996, s. 169-170)

$$p_i = \frac{1}{\mu} \cdot \left(\eta^+ - (n^+ - \eta^-) \cdot \frac{i-1}{\mu - 1} \right). \tag{2.14}$$

Rajoitukset $\sum_{i=1}^{\mu} p_i = 1$ ja $p_i \geq 0 \forall i \in \{1, ..., \mu\}$ vaativat, että $1 \leq \eta^+ \leq 2$ ja $\eta^- = 2 - \eta^+$. Valintapainetta voidaan tällöin säätää parametrilla η^+, joka on tämän lineaarisen funktion antama korkein valintatodennäköisyys.

2.3.3 (μ, λ)-valinta

Erityisesti evoluutiostrategioissa käytetään usein (μ, λ)-*valintaa* (Bäck 1996, s. 78-80, 176), joka on järjestämisvalinnan erikoistapaus; jälkeläisten (λ kappaletta) vanhemmat valitaan μ:n vanhemman joukosta *tasaisella* todennäköisyydellä:

$$p_i = \begin{cases} \frac{1}{\mu} & \text{, jos } i \leq \mu \\ 0 & \text{muutoin} \end{cases} . \tag{2.15}$$

2.3.4 Turnajaisvalinta

Turnajaisvalinta (tournament selection) viittaa valintamenetelmää, jossa jokainen vanhempi etsitään »turnajaisten» avulla. Vanhasta populaatiosta valitaan (satunnaisesti) q yksilöä, joista paras valitaan vanhemmaksi. Tämä voidaan esittää myös yhtälöllä (Bäck 1996, s. 172-174):

$$p_i = \frac{1}{\mu^q} \cdot ((\mu - i + 1)^q - (\mu - i)^q). \tag{2.16}$$

2.3.5 Elitismi

Elitismi tarkoittaa parhaimpien yksilöiden (eli »eliittien») säästämistä seuraavaan populaatioon sellaisenaan. Tämä estää hyvien ratkaisujen menettämisen vahingossa. Vahinko saattaa johtua esimerkiksi valintamenetelmän epädeterministisyydestä. Hyvät ratkaisut saatetaan myös parittaa epäsopivien ratkaisujen kanssa, jolloin tuloksena voi olla hyvin huonoja jälkeläisiä.

Elitismiä on kritisoitu monista ongelmista, kuten sen ominaisuudesta konvergoida haku mahdollisesti ennenaikaisesti (Bäck 1996, s. 79), jolloin haku helposti päätyy huonoon paikalliseen kelpoisuusmaksimiin. Monet ovat kuitenkin käyttäneet jopa äärimmäistä *steady-state*-strategiaa, jossa vain huonoin yksilö korvataan kussakin sukupolvessa. Steady-state-strategian valintavoimakkuus voidaan säätää niin alhaiseksi, että se ei välttämättä ole kovin altis ennenaikaiselle konvergenssille.

Elitismi haittaa myös itsesäätelyä, jossa evoluutio ei kohdistu vain kohdemuuttujiin, vaan myös kykyyn löytää uusia ratkaisuja helpommin. Steady-state-strategiassa itsesäätely olisi luultavasti jo mahdotonta.

2.3.6 Valinta oliomallissa

Edellä valintamenetelmät kuvattiin vain tavalla laskea valintatodennäköisyys. Valinta toteutetaan populaation ω_{select}-operaation avulla. Esimerkiksi (μ, λ)-valinnassa

$$Populaatio[\omega_{select}] \equiv Y_{\mathbf{rnd}(1\ldots\mu)}, \tag{2.17}$$

jossa Y_i on populaation (ks. sivu 5) i:s yksilö ja $\mathbf{rnd}(1\ldots n)$ palauttaa satunnaisen kokonaisluvun välillä $1\ldots n$.

2.4 Mutaatiot

Evolutiiviset algoritmit vaativat, että kullekin koodaukselle määritellään ainakin mutaatio-operaattori $\omega_m : I \mapsto I$. Rekombinaatioiden (ks. alla) käyttö ei ole EA:issa välttämätöntä, mutta mutaatioiden käyttö jotakuinkin on: ne ovat keino tuoda uusia ratkaisuehdokkaita silloinkin, kun koko populaatio on jumittunut samaan pisteeseen.

Jokaiselle geenityypille voidaan määritellä useita mutaatio-operaatioita. Tavallisin binääriluvuille tehtävä operaatio yksinkertaisesti kääntää yhden bitin todennäköisyydellä p_b, joka on tyypillisesti välillä 0.001 ja 0.1. Kokonaisluvuille mutaatiot voidaan tehdä muuttamalla luku todennäköisyydellä p_i täysin satunnaisesti toiseksi luvuksi arvoalueen rajoissa.

Reaaliluvut. Reaalilukujen mutaatiot tehdään evoluutiostrategioissa tyypillisesti käyttämällä:

$$x' = x + N(0, \sigma), \tag{2.18}$$

jossa σ on mutaatio-varianssi reaalilukugeeneille ja $N(0, \sigma)$ on normaalisti jakautunut satunnaisluku, jonka keskiarvo on nolla ja varianssi σ. Toinen, geenien välisiä korrelaatioita hyväksi käyttävä tapa esitetään myöhemmin itsesäätelyiden yhteydessä.

2.5 Rekombinaatiot

Rekombinaatio (risteytys) on EA:eissa geneettinen operaattori $\omega_r : I^p \rightarrow I^q$, joka muodostaa q jälkeläisgenomia (tyypillisesti yksi tai kaksi) yhdistämällä p:n yksilön genomit (tyypillisesti kaksi) sopivalla tavalla.

EA:iden rekombinaatio-ajatus perustuu kahteen biologisten organismien rekombinaatiomekanismiin: *tekijäinvaihtoon* (crossover) ja *riippumattomaan valikointiin* (independent assortment). Tekijäinvaihdossa kaksi kromatidia risteävät yhdestä tai useammasta kohdasta ja vaihtavat samalla osia (ks. kuva 2.2). Riippumaton valikointi tarkoittaa vastinkromosomien satunnaista jakautumista kahteen haploidiin gameettiin (sukusoluun) meioosin aikana. Hedelmöityksessä nämä haploidit gameetit yhtyvät ja muodostavat diploidisen alkion.

Suurin osa EA:iden rekombinaatio-operaattoreista perustuu tekijäinvaihdon ajatukseen, joka on esitetty edellä kuvassa 2.2. Binäärilukuvektoreita käsittelevillä algoritmeilla se onkin helpoin tapa. Evoluutiostrategioiden käyttämille reaalilukugeeneille on kuitenkin mahdollista — ja hyödyllistä — määritellä omat sisäiset rekombinaatio-operaattorinsa. Yleisiä operaattoreita liukulukugeeneille ovat seuraavat:

$$x'_i = \begin{cases} x_{S,i} & \text{ei rekombinaatiota} & r_- \\ x_{S,i} \text{ tai } x_{T,i} & \text{diskreetti} & r_d \\ x_{S,i} + (x_{T,i} - x_{S,i})/2 & \text{keskiarvo} & r_k \\ x_{S,i} + \chi_i \cdot (x_{T,i} - x_{S,i})/2 & \text{yleistetty keskiarvo} & r_y \end{cases} \quad (2.19)$$

jossa $\vec{x}_S$ ja $\vec{x}_T$ ovat jälkeläisen $\vec{x}'$ kahden vanhemman genomit. Diskreetti rekombinaatio toimii siis samoin kuin tekijäinvaihto binäärigenomissa.

Rekombinaatio toimii myös muissakin tietorakenteissa kuin vektoreissa. Esimerkiksi *geneettisessä ohjelmoinnissa* (genetic programming) genomi on puumuotoinen, jossa solmut ovat tyypillisesti binäärisiä tai unäärisiä operaatioita ja rekombinaatio tapahtuu leikkaamalla kahdesta vanhemmasta oksa pois ja vaihtamalla ne keskenään. Näin saadaan aikaiseksi kaksi jälkeläistä.

Radcliffe (1992) on kehittänyt rekombinaatio-operaattoreita myös joukkotyyppisille genomeille. Rekombinaatio-operaattori valitsee jälkeläiseen ensinnäkin kaikki molemmissa vanhemmissa olevat geenit. Loput jälkeläisen geenit valitaan satunnaisesti vain toisella vanhemmista olevista geeneistä sekä geeneistä, joita kummallakaan ei ole. Radcliffe on esittänyt rekombinaatio-operaattoristaan kaksi versiota: toisessa genomin koko pysyy samana, toisessa se voi muuttua.

Luku 3

Strategiaparametrien säätö

Tarkastelemme tässä luvussa evolutiivisten algoritmien strategiaparametrien säätöä siten kuin Hinterding et al. (1997) ovat sen esittäneet. Evolutiivisen algoritmien sisäisten parametrien säätöä voidaan tarkastella kahdelta kannalta. Ensimmäinen näkökulma on *säätötapa*, joka tarkastelee säätämistä algoritmiselta kannalta: mitä sääntöä käyttäen säätö tehdään. Toinen näkökulma on *säädön taso*, jossa tarkastellaan sitä, kuinka paikallista tai globaalia säätö on. Tästä johdettu säädön luokitus kuvataan taulukossa 3.1.

Evolutiivisissa algoritmeissa on yleensä varsin paljon säätöä kaipaavia parametreja. Selkeimpiä ovat populaation koko λ, valintamenetelmän parametrit (μ, η^+, q, jne.), mutaatioiden määrä (p_m, σ) ja tyyppi, sekä rekombinaatioiden määrä (p_r) ja tyyppi. Lähes kaikki nämä parametrit ovat lisäksi jossain määrin riippuvia toisistaan, mikä vaikeuttaa niiden määrittämistä.

3.1 Säätötavat

Tarkastelemme seuraavassa säätötapoja niiden sopeutumismekanismin kannalta. Menetelmät voidaan jakaa kahteen osaan sen mukaan, saako mekanismi palautetta EA:lta (Hin-

Tapa / *Taso*	Kiinteä	Muuttuva		
		Deterministinen	Sopeutuva	Itsesäätelevä
Ympäristö	St	Ym-D	Ym-S	Ym-IS
Populaatio	St	P-D	P-S	P-IS
Yksilö	St	Yk-D	Yk-S	Yk-IS
Komponentti	St	K-D	K-S	K-IS

Taulukko 3.1: Säädön luokitus evolutiivisissa algoritmeissa (Hinterding 1997).

terding, Michalewicz, and Eiben 1997); kiinteä ja yksinkertainen muuttuva säätö eivät palautetta käytä, sopeutuva ja itsesäätelevä taas käyttävät.

3.1.1 Kiinteä säätö

Kiinteitä säätöjä käytettäessä strategiaparametrit pysyvät samoina koko EA-prosessin ajan. Tästä syystä tarvitaan jokin tapa, jolla parametrit määritellään sopivan suuruisiksi ennen prosessin suoritusta.

Parametrien määritystapoja on viime vuosikymmenien aikana pyritty kehittämään aktiivisesti, ja monille parametreille on esitetty useita peukalosääntöjä. Esimerkiksi binäärigeenien mutaatiotaajuudelle käytetään usein arvoja $p_m = 0.001$ (Jong 1975), $p_m = 0.01$ (Grefenstette 1986) tai yleisemmin $p_m \in [0.0005, 0.01]$ (Schaffer, Caruna, Eshelman, and Das 1989), mikä riippuu genomin koosta, yms. Schafferin et al. tulos on määritelty myös kaavalla

$$p_m \approx +\frac{1.75}{\lambda\sqrt{l}}, \tag{3.1}$$

jossa λ on populaation koko ja l on genomin koko (bitteinä). Schaffer et al. ovat todenneet kaavan hyväksi kokeellisesti.

Eräs tapa määrittää EA:n parametrit on soveltaa niihin hakualgoritmia, esim. EA:ta. Tällaista hakua kutsutaan *meta-evoluutioksi*. Vaikka sen avulla voidaankin löytää sopivat parametrit tietylle ongelmatyypille, eivät löydetyt parametrit välttämättä enää sovellu, kun ongelma muuttuu merkittävästi. Meta-evoluutio myös vaatii, että optimoitavan EA:n haku voidaan suorittaa varsin monta kertaa. Tämä johtaa helposti täysin kohtuuttomiin aikaresurssivaatimuksiin; haluamme tehokkaan hakualgoritmin, joka tarvitsee ajaa vain kerran.

Kiinteät säädöt ovat useimmin käytetty tapa useimpien strategiaparametrien säätämiseksi.

3.1.2 Muuttuvat säädöt

Muuttuvalla säädöllä tarkoitetaan sellaista säätelyä, joka tapahtuu algoritmin sisällä, algoritmin suorituksen aikana. Se ei edelleenkään välttämättä poista säädettäviä strategiaparametreja, mutta toivon mukaan muuntaa ne toisiksi, vähemmän herkiksi parametreiksi.

3.1.2.1 Deterministinen säätely

Deterministisellä säätelyllä tarkoitetaan johonkin deterministiseen sääntöön perustuvaa parametrien muuttamista. Tämä menetelmä muuttaa parametreja ilman palautetta EA:lta. Tyypillisin säätötekijä on aika, joka ilmaistaan sukupolvien lukumäärällä tietystä hetkestä (yleensä evoluution alusta) laskettuna. Jo itse Holland (1975) ehdotti aikariippuvaista säätelyä, vaikkakaan ei itse antanut tarkkoja parametreja tälle menetelmälle. Yksinkertainen kaava voi olla esimerkiksi

$$p_m = 0.01 - 0.009 \cdot \frac{g}{G}, \tag{3.2}$$

jossa g on sukupolven numero välillä $1...G$. Tässä esimerkissä mutaatiotodennäköisyys p_m pienenee 0.01:stä 0.001:een, kun sukupolvi kulkee kohti G:tä. Sekä Hollandin että Fogartyn (1989) lähestymistavoissa $\lim_{t\to\infty} p_m(t) = 0$. Hesser ja Männer (1991) ovat teoreettisesti määrittäneet mutaatiotodennäköisyyden säätökaavan eräälle GA:n versiolle:

$$p_m(t) = \sqrt{\frac{\alpha}{\beta}} \cdot \frac{e^{-\gamma \frac{t}{2}}}{\lambda\sqrt{l}}, \tag{3.3}$$

jossa α, β ja γ ovat vakioita. Kaikissa näissä menetelmissä mutaatiotaajuus on koko populaatiossa globaali parametri, eli kaikki yksilöt käyttävät samaa arvoa.

3.1.2.2 Sopeutuva säätely

Sopeutuvaksi säätelyksi kutsutaan sellaista muuttuvaa säätelyä, joka tapahtuu vastaanottamalla palautetta EA:lta. Tätä palautetta käytetään strategiaparametrin muutoksen suunnan tai suuruuden tai molempien määrittämiseen.

Eräs esimerkki tästä säätelytavasta on Rechenbergin (1973) *»1/5 - onnistumissääntö»*, jota käytetään mutaatioiden suuruuden määrittämiseen. Säännön mukaan onnistuneiden mutaatioiden osuuden tulisi olla 1/5. Jos onnistuneita on vähemmän, kasvatetaan mutaatioiden suuruutta, ja jos onnistuneita on enemmän, pienennetään mutaatioiden suuruutta. Rechenberg sovelsi menetelmää alunperin evoluutiostrategioihin.

Davis (1989) on esittänyt »sopeutuvan operaattorikelpoisuuden» mallin, jossa eri lisääntymisoperaattorien käytön todennäköisyyttä säädetään niiden suorituskyvystä saatavan palautteen avulla.

Sopeuttamista voi käyttää myös kelpoisuusfunktion muuttamiseen säätämällä rikottujen rajoitusten rangaistuskertoimia. Esimerkiksi:

$$F(\vec{x}) = f(\vec{x}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\vec{x}), \tag{3.4}$$

jossa λ:ta päivitetään joka sukupolvi seuraavasti:

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t) & \text{, jos } \vec{b}(i) \in F \text{ kaikille } t-k+1 \leq i \leq t \\ \beta_2 \cdot \lambda(t) & \text{, jos } \vec{b}(i) \in S - F \text{ kaikille } t-k+1 \leq i \leq t \\ \lambda(t) & \text{muutoin} \end{cases}, \tag{3.5}$$

jossa $\vec{b}(i)$ on parhaimmaksi arvioitu yksilö.

3.1.2.3 Itsesäätely

Evolutiivista optimointia voi käyttää myös EA:n omien, sisäisten parametrien optimointiin. Tällöin säädettävät parametrit koodataan kunkin yksilön genotyyppiin, jolloin mutaatiot ja rekombinaatiot vaikuttavat niihin siinä missä muihinkin geeneihin. Tällä tavoin koodatut parametrit eivät vaikuta yksilöiden kelpoisuuteen suoraan, vaan »paremmat» parametrit johtavat »parempiin» yksilöihin, jotka lisääntyvät helpommin ja edistävät parametriensa edustusta populaatiossa.

Itsesäätelyä on käytetty erityisesti evoluutiostrategioissa (ES), jossa sen avulla säädetään reaalilukugeenien mutaatioiden varianssin suuruutta. ES:ssa on käytetty sekä yksilötasolla että komponenttitasolla tapahtuvaa itsesäätelyä. Bäck (1991) ja Hinterding (1995) ovat laajentaneet itsesäätelyn koskemaan myös binääriluvuilla operoivia geneettisiä algoritmeja.

Paitsi, että säädettävät parametrit voivat olla numeerisia strategiaparametreja, voivat ne olla myös valintoja tai todennäköisyyksiä vaihtoehtoisten algoritmin yksityiskohtien käyttämiselle. Esimerkiksi Fogel et al. (1991) itsesäätelevät viiden eri mutaatio-operaattorin käytön todennäköisyyksiä.

Itsesäätelymenetelmien yksityiskohdista kerrotaan enemmän luvussa 4.

3.2 Säädön tasot

Kuten edellä jo mainittiin, evoluutioparametrin säätö voi tapahtua neljällä eri tasolla: ympäristön, populaation, yksilöiden ja komponenttien (Hinterding, Michalewicz, and Eiben 1997).

3.2.1 Ympäristötason säätö

Korkein säätötaso on EA:n »ympäristö» eli ongelma. Tällä säätötasolla vaikutetaan siihen tapaan, jolla yksilön kelpoisuus määritellään. Kelpoisuusfunktioon määritellyt rangaistustekijät ja niiden painot voivat muuttua, ja yksilön kelpoisuus voi muuttua lokerointimenetelmien (niching) perusteella.

Esimerkiksi Beasley et al. (1993) muokkaavat ympäristöä löytääkseen multimodaaliseen[1] ongelmaan useita erilaisia ratkaisuja. EA suoritetaan menetelmässä useaan kertaan. Kelpoisuusfunktioon on lisätty termi, joka alentaa sellaisten yksilöiden kelpoisuutta, jotka ovat lähellä edellisissä ajoissa löytyneitä optimeja. Myös ympäristö siis sopeutuu.

3.2.2 Populaatiotason säätö

Kaikissa evolutiivissa algoritmeissa on aina joitain strategiaparametreja, jotka ovat globaaleja koko populaatiolle. Näiden parametrien muuttamista kutsutaan populaatiotason säädöksi.

Populaatiotason parametrien muuttuva säätö on useimmin joko determinististä tai sopeutuvaa. Populaatiotason parametrien itsesäätelyä ei toistaiseksi ole tehty. Edellä esitetyt mutaatiotaajuuden deterministiset muutokset ja Rechenbergin »1/2-onnistumissääntö» ovat esimerkkejä tämän tason säätömenetelmistä.

Populaatiotason säätö kattaa myös tapaukset, joissa useita alipopulaatioita evolvoidaan samanaikaisesti. Lis (1996) käyttää alipopulaatioista saatavaa palautetta säätääkseen sopeuttavasti joitain (tai kaikkia) alipopulaatioiden kokoja.

3.2.3 Yksilötason säätö

Yksilötason säädössä jokaisella yksilöllä on omat henkilökohtaiset versionsa joistain strategiaparametreista, jotka vaikuttavat tuolloin vain yksilön omaan käyttäytymiseen. Tyypillisiä tällä tasolla säädettäviä parametreja ovat mutaatiotodennäköisyys (GA:t) ja –ja-

[1] Multimodaalinen ongelma on ongelma, jossa on useita paikallisia tai globaaleja maksimikohtaa. Vastaavasti unimodaalisessa ongelmassa on vain yksi maksimikohta.

kauma (ES:t) sekä rekombinaatioiden todennäköisyys ja sijainti genomissa (Schaffer and Morishima 1987).

Arabas et al. (1994) ovat kuvanneet menetelmän populaation koon säätämiseksi yksilötason sopeutumisella. Menetelmä perustuu yksilöiden ikään. Populaation koko määräytyy kaavalla

$$P(t{+}1) = P(t) + N(t) - D(t), \tag{3.6}$$

jossa $N(t)$ on iteraatiolla t syntyvien jälkeläisten ja $D(t)$ kuolevien yksilöiden lukumäärä (Michalewicz 1996). Jälkeläisten määrä on suhteessa populaation kokoon, mutta kuolevien määrä riippuu yksilöiden iästä. Yksilöiden iän määräämiselle on useita heuristisia menetelmiä (Arabas, Michalewicz, and Mulawka 1994), jotka kaikki käyttävät palautetta evolutiivisen haun kunkin hetkisestä tilasta.

3.2.4 Komponenttitason säätö

Alimman säätötason muodostavat genomin sisäiset komponentit, »geenit». Geenillä voidaan tarkoittaa sekä atomaarista geeniä (GA:ssa yksi bitti, ES:ssä yksi reaaliluku) että suurempaa ryhmää (esimerkiksi tiettyä kokonaislukuparametria koodaavaa bittisarjaa). Komponenttitason säädössä voi esimerkiksi kullakin genomin bitillä tai reaaliluvulla olla omat parametrinsa.

Tunnetuin ja pisimmälle tutkittu komponenttitason säädön muoto on mutaatioiden varianssin ja rotaatiokulmien (ks. luku 4.2.3) itsesäätely evoluutiostrategioissa. Lisäksi Fogel et al. (1995) ovat pohtineet mekanismeja tilakoneiden komponenttien mutaatioiden todennäköisyyksien säätämiseksi.

Luku 4

Itsesäätely

Käsittelemme tässä luvussa itsesäätelyä geneettisissä algoritmeissa ja evoluutiostrategioissa sekä yksilötasolla että komponenttitasolla. Esitämme menetelmät aluksi siten kuin menetelmien kehittäjät ovat ne esittäneet, ja lopussa esitämme ne käyttäen edellä esiteltyä oliosuuntautunutta pseudokoodia.

4.1 Geneettisissä algoritmeissa

Bäck (1991) kuvaa mutaatiotaajuuden itsesäätelyn geneettisissä algoritmeissa seuraavasti.

Optimoimme jatkuvia parametrejä n-muuttujaisille ongelmille $f : \prod_{i=1}^{n}[a_i, b_i] \mapsto \Re$. Koodaamme joko yhden tai n mutaatiotaajuutta genotyyppiin, joka kuuluu joukkoihin

$$Y_{Y-IS} = \{0,1\}^l \times \{0,1\}^{\hat{l}} = \{0,1\}^{l+\hat{l}} \text{ tai} \tag{4.1}$$

$$Y_{K-IS} = \{0,1\}^l \times \left(\{0,1\}^{\hat{l}}\right)^n = \{0,1\}^{l+\hat{l}\cdot n}, \tag{4.2}$$

jossa l on bittien määrä kaikissa kohdemuuttujissa yhteensä (eli $l = \sum_{i=1}^{n} l_i$) ja $\hat{l}$ on bittien määrä yhden mutaatiotaajuuden geneettisessä koodauksessa. Joukko Y_{Y-IS} on siis yksilötasolla itsesäätelevien yksilöiden joukko ja jälkimmäinen Y_{K-IS} komponenttitasolla itsesäätelevien yksilöiden joukko. Käsittelemme nämä tapaukset alla erikseen.

4.1.1 Yksilötason itsesäätely

Bäck (1991) kuvaa yksilötasolla toimivan mutaatiotaajuuden itsesäätelyn seuraavasti. Yksilöt ovat muotoa $(\vec{a}, \vec{p}) \in Y_{Y-IA}$, jossa $\vec{a}$ on kaikkia kohdemuuttujia ja $\vec{p}$ mutaatiotodennäköisyyksiä koodaava binäärivektori. Bäck käyttää binääriesityksen dekoodaukseen funktiota

$$\Gamma_{u,v,l}(a_1 ... a_l) = u + (v - u) \cdot \frac{\sum_{i=0}^{l-1} \left(\bigoplus_{j=1}^{i+1} a_j \right) 2^i}{2^l - 1}, \tag{4.3}$$

joka on kuvaus $\Gamma_{u,v,l} : \{0,1\}^l \mapsto [a, b]$, eli binäärivektorista reaalivälille $[a, b]$. Tämä funktio on sama kuin sivulla 8 esitettyjen funktioiden (2.11) ja (2.12) yhdistelmä.

Mutaatiot tehdään seuraavasti:

1. Dekoodataan mutaatiotaajuus: $p_m = \Gamma_{1/l,0.5,l'}(\vec{p})$.
2. Mutatoidaan itse mutaatiogeenejä:

$$p'_i = \begin{cases} p_i & \text{, jos } \chi > p_m \\ 1 - p_i & \text{, jos } \chi \leq p_m, \end{cases} \tag{4.4}$$

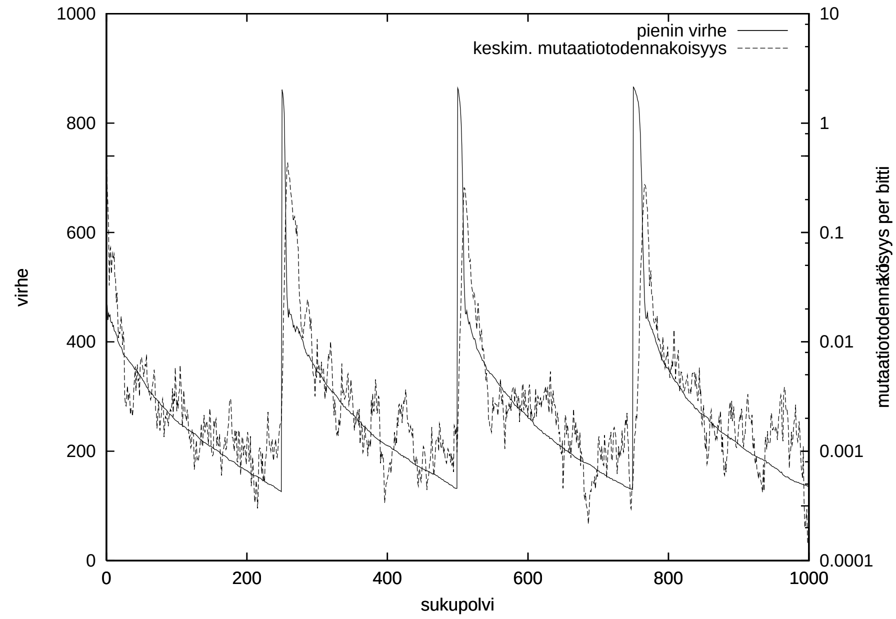
jossa χ on tasaisesti jakautunut satunnaisluku välillä $[0, 1]$. Tästä satunnaisluvusta otetaan uusi näyte jokaiselle bitille erikseen.

3. Dekoodataan mutatoitu mutaatiotaajuus: $p'_m = \Gamma_{1/l,0.5,l'}(\vec{p})$.
4. Nyt voidaan mutatoida varsinaiset kohdemuuttujia koodaavat binäärigeenit:

$$a'_i = \begin{cases} a_i & \text{, jos } \chi > p'_m \\ 1 - a_i & \text{, jos } \chi \leq p'_m \end{cases} . \tag{4.5}$$

Bäck on esittänyt yllä kuvatulle myös vaihtoehtoisen menetelmän, joka on hybridi geneettisten algoritmien ja evoluutiostrategioiden välillä. Kohdemuuttujat koodataan binäärisesti, kuten yllä, mutta mutaatiotaajuus koodataan reaalilukuna, kuten evoluutiostrategioissa. Yksilö on tuolloin $(\vec{a}, p) \in \{0,1\}^l \times [0, 1]$. Mutaatiotaajuutta mutatoidaan reaalilukujen mutaatiofunktiolla

$$p'_m = \left(1 + \frac{1 - p_m}{p_m} \cdot e^{-\gamma \cdot N(0,1)} \right)^{-1}, \tag{4.6}$$



Kuva 4.1: Esimerkki mutaatiotaajuuksien itsesäätelystä yksinkertaisella ykköstenlaskenta-ongelmalla (Kuva: Grönroos, 1998).

jossa γ on sopeutumisnopeus, joka vastaa τ:tä evoluutiostrategioissa. Tämän jälkeen mutatoidaan kohdemuuttujia koodaavia binäärilukuja kuten yllä.

4.1.1.1 Esimerkki

Kuva 4.1 esittää yksilötason itsesäätelevän GA:n äärimmäisen yksinkertaisella testifunktiolla (ns. ykkösten laskenta):

$$f(\vec{a}) = \sum_{i=1}^{n} a\_i\text{-}o, \tag{4.7}$$

jossa geneettisesti evolvoitavan binäärilukuvektorin $\vec{a}$ kohdearvo o muuttuu joka 250:s sukupolvi (vaihdellen $o = 0$ ja $o = 1$). Kun $o = 1$, löytyy optimi siis vain kääntämällä jokainen vektorin biteistä ykköseksi. Vektorin dimensio on $n = 1000$. Testiajon strategiaparametrit ovat $(15, 100)$-valinta eikä lainkaan rekombinaatiota.

4.1.2 Komponenttikohtaiset mutaatiot

Bäck (1991) kuvaa komponenttikohtaisen itsesäätelyn GA:ssa seuraavasti. Oletamme edelleen, että käytössämme on injektiivinen dekoodausfunktio $\Gamma_{a,b,l} : \{0,1\}^l \mapsto [a, b]$, joka siis kuvaa pituutta l olevan bittisarjan reaalivälille $[a, b]$. Bäck kuvaa yksilön $a \in Y_{K-IA}$ seuraavasti:

$$a = (\alpha_{1,1}...\alpha_{1,l_1}...\alpha_{n,1}...\alpha_{n,l_n},\ \hat{\alpha}_{1,1}...\hat{\alpha}_{1,\hat{l}}...\hat{\alpha}_{n,1}...\hat{\alpha}_{n,\hat{l}}). \tag{4.8}$$

Saamme tällöin koodauksen $(\hat{\alpha}_{i,1}...\hat{\alpha}_{i,\hat{l}})$ esittämän mutaatiotaajuuden p_i käyttämällä dekoodausta $p_i = \Gamma_{0,1,\hat{l}}(\hat{\alpha}_{i,1}...\hat{\alpha}_{i,\hat{l}})$. Uusi mutaatiomekanismi $m : I \mapsto I$ toimii seuraavasti:

$$m((\alpha_{1,1}...\alpha_{n,l_n}...\hat{\alpha}_{1,1}...\hat{\alpha}_{1,\hat{l}})) = (\beta_{1,1}...\beta\alpha_{n,l_n}...\hat{\beta}_{1,1}...\hat{\beta}_{1,\hat{l}}), \tag{4.9}$$

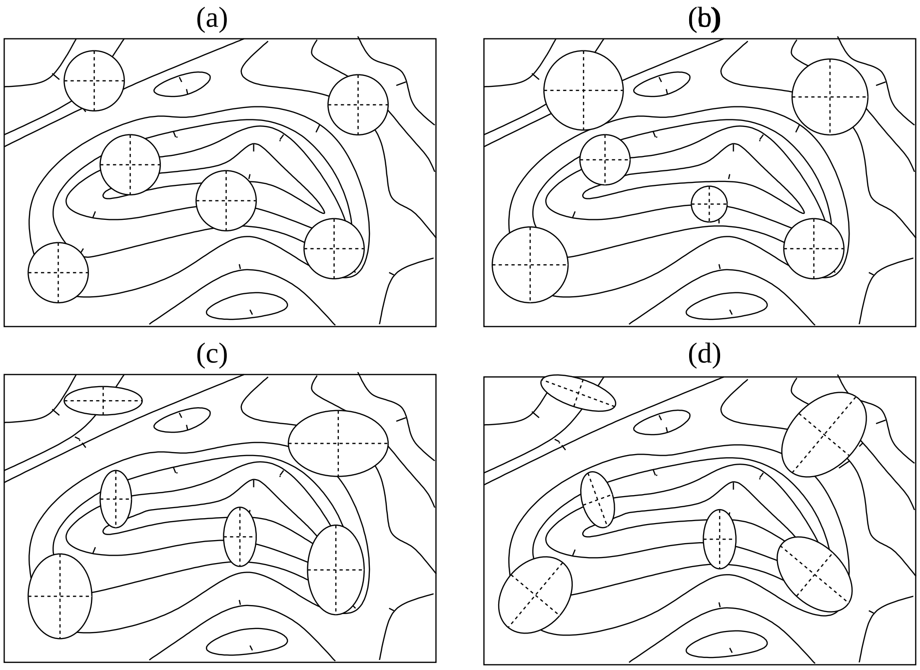
jossa $\forall i \in \{1, ..., n\} \forall k \in \{1, ..., \hat{l}\}$:

$$\begin{array}{l} \hat{\beta}_{i,k} = \begin{cases} \hat{\alpha}_{i,k} & , \chi \leq \Gamma_{0,1,\hat{l}}(\hat{\alpha}_{i,1}...\hat{\alpha}_{i,\hat{l}}) \\ \kappa \in \{0,1\} & , \chi > \Gamma_{0,1,\hat{l}}(\hat{\alpha}_{i,1}...\hat{\alpha}_{i,\hat{l}}) \end{cases} \\ \beta_{i,k} = \begin{cases} \alpha_{i,k} & , \chi \leq \Gamma_{0,1,\hat{l}}(\hat{\beta}_{i,1}...\hat{\beta}_{i,\hat{l}}) \\ \kappa \in \{0,1\} & , \chi > \Gamma_{0,1,\hat{l}}(\hat{\beta}_{i,1}...\hat{\beta}_{i,\hat{l}}) \end{cases} \end{array}, \tag{4.10}$$

jossa χ on tasaisesti jakautunut satunnaisluku välillä $[0, 1]$, josta otetaan uusi näyte jokaiselle bitille. Myös $\kappa \in \{0, 1\}$ on satunnaismuuttuja, joka antaa satunnaisen binäärisen alleeliarvon.

4.2 Evoluutiostrategioissa

Evoluutiostrategioissa (Bäck 1996, s. 66-91) on käytetty sekä mutaatioiden varianssin σ (ks. kaava (2.8)) yksilötason että komponenttitason itsesäätelyä. Itsesäätelyllä voidaan lisäksi säädellä mutaatioiden rotaatiokulmia, jotka kuvaavat muuttujien välisiä korrelaatioita. Kuvissa 4.2b–d on esitetty esimerkki eri itsesäätelymenetelmien vaikutuksesta mutaatioiden jakaumaan. Esimerkissä on kuusi yksilöä, joiden mutaatioiden takauma on ilmaistu yhtäläisen todennäköisyyden hyperpintana. Komponenttitason (kuvat 4.2c–d) itsesäätelyssä se on ellipsoidin muotoinen.



Kuva 4.2: Mutaatioiden jakaumat eri itsesäätelytasoilla. **(a)** Ei itsesäätelyä - globaali mutaatiovarianssi. **(b)** Yksilötason mutaatiovarianssin itsesäätely. **(c)** Komponenttitason mutaatiovarianssin itsesäätely. **(d)** Komponenttitason mutaatiovarianssin ja rotaatiokulmien itsesäätely.

4.2.1 Mutaatioiden varianssin itsesäätely

Itsesäätelevä yksilö kuvataan evoluutiostrategioissa seuraavasti:

$$\begin{array}{rl} (\vec{x}, \sigma) \in Y_{Y-IS} = \Re^n \times \Re & \text{yksilötason IA : ssa ja} \\ (\vec{x}, \vec{\sigma}) \in Y_{K-IS} = \Re^n \times \Re^n & \text{komponenttitason IA : ssa.} \end{array} \tag{4.11}$$

ES:n mutaatiot esiteltiin kaavassa (2.8), jossa mutaatioiden varianssi ilmaistiin parametrilla σ. Mutaatioiden varianssia säätelevän geenin tai geenien mutaatiot tehdään erikseen seuraavasti:

$$\sigma' = \sigma \cdot e^{N(0,\tau')} \quad \text{yksilötasolla} \tag{4.12}$$

$$\sigma_i' = \sigma_i \cdot e^{N_i(0,\tau)} \quad \text{komponenttitasolla} \tag{4.13}$$

$$\sigma_i' = \sigma_i \cdot e^{N(0,\tau')+N_i(0,\tau)} \quad \text{molemmilla tasoilla,} \tag{4.14}$$

jossa τ ja τ' ovat mutaatiogeeneille tehtävien mutaatioiden »meta-taajuuksia», $N(0,\sigma)$ on kaikille komponenteille yhteinen satunnaismuuttuja[1] ja $N_i(0,\sigma)$ on kullekin komponentille yksilöllinen satunnaismuuttuja. Kaavassa (4.14) on yhdistetty yksilö- ja komponenttitason itsesäätely. Se on sama kuin kaavassa (4.12), mikäli $\tau = 0$ ja varianssit σ_i on alustettu yhtä suuriksi, ja sama kuin kaavassa (4.13), mikäli $\tau' = 0$. Schwefel (1977; myös Bäck, 1996, s. 72) ehdottaa metataajuuksien suuruudeksi $\tau' \approx \left(\sqrt{2n}\right)^{-1}$ ja $\tau \approx \left(\sqrt{2\sqrt{n}}\right)^{-1}$, missä n on genomin reaalilukugeenien määrä. Kohdemuuttujien mutaatiot tehdään tämän jälkeen käyttäen σ':aa:

$$\vec{x}' = \vec{x} + N(0,\sigma'). \tag{4.15}$$

Mutaatioiden hyperellipsoidien muoto tämänkaltaisissa yksilöissä on esitetty kuvissa 4.2b ja c.

4.2.2 Esimerkki

Kuva 4.3 esittää itsesäätelevän evoluutiostrategian virheen konvergoitumisen *pallo*-testifunktiolle

$$f_{pallo}(\vec{x}) = \sum_{i=1}^{n} (x_i - o)^2, \tag{4.16}$$

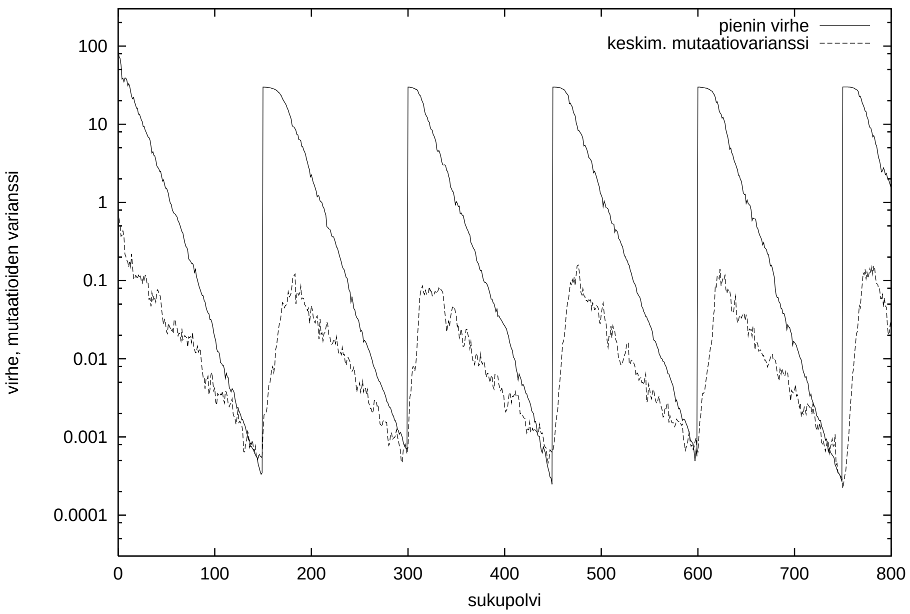
jossa kohdearvo o vaihtuu joka 150:s sukupolvi (vaihdellen $o = 0$ ja $o = 1$). Reaalilukuvektorin dimensio on $n = 30$. Tämän hyvin yksinkertaisen funktion hakumaisema sisältää vain yhden optimipisteen. Populaation optimaalinen koko olisi kaksi yksilöä.

4.2.3 Korreloidut mutaatiot

Mutaatioiden varianssin lisäksi voidaan itsesäädellä mutaatioiden korrelaatioita, jotka esitetään rotaatiokulmina (Bäck 1996, s. 68-73). Vaikutus mutaatioiden jakaumaan on esitetty kuvassa 4.2d. Yksilöt määritellään tällöin seuraavasti:

$$(\vec{x},\vec{\sigma},\vec{\alpha}) \in Y_{K-IA} = \Re^n \times \Re^n \times [-\pi,\pi]^{n\cdot(n-1)/2}. \tag{4.17}$$

[1]Ensimmäinen parametri on satunnaislukujen keskiarvo, toinen varianssi.



Kuva 4.3: Esimerkki mutaatiotaajuuksien itsesäätelystä evoluutiostrategioissa f_{pallo}-testifunktiolla. (Kuva: Grönroos, 1998)

Varianssien $\vec{\sigma} \in \Re^n$ mutaatiot tehdään kuten edellä kaavassa (4.14). Rotaatiokulmien $\vec{\alpha} \in [-\pi, \pi]^{n \cdot (n-1)/2}$ (radiaaneja) mutaatiot tehdään vastaavasti:

$$\alpha'_j = \alpha_j + N_j(0, \beta), \tag{4.18}$$

jossa β on varianssi, jonka arvoksi Schwefel (1977; Bäck, 1996, s. 72) ehdottaa $\beta \approx 0.0873$. Nyt voidaan muodostaa kovarianssimatriisi $\mathbf{C}^{-1} = (c_{ij})$:

$$\begin{aligned} c_{ii} = & \ \sigma_i^2, \\ c_{ij,(i \neq j)} = & \ \tfrac{1}{2}(\sigma_i^2 - \sigma_j^2) \cdot \tan(2\alpha_{ij}). \end{aligned} \tag{4.19}$$

Kohdemuuttujien mutatointi tapahtuu tällöin seuraavasti:

$$\vec{x}' = \vec{x} + \vec{N}(\vec{0}, \mathbf{C}(\vec{\sigma}', \vec{\alpha}')), \tag{4.20}$$

jossa $\vec{N}(\vec{0}, \mathbf{C})$ on n-ulotteinen satunnaismuuttuja, jonka jakauma noudattaa yleistettyä n-ulotteista normaalijakaumaa keskiarvolla $\vec{0}$ ja kovarianssimatriisilla $\mathbf{C}^{-1} = \mathbf{C}^{-1}(\vec{\sigma}, \vec{\alpha})$. Tämän normaalijakauman tiheysfunktio on

$$p(\vec{z}) = \frac{\exp(-\frac{1}{2}\vec{z}^T \mathbf{C}^{-1}\vec{z})}{\sqrt{(2\pi)^n \cdot \det \mathbf{C}}}. \tag{4.21}$$

Mutaatioellipsoidien (yhtäläisen todennäköisyyden hyperpintojen) akselit ovat koordinaatiston akselien kanssa myötäiset vain **C**:n ollessa suorakulmainen. Yleisemmässä tapauksessa ellipsoideilla voi olla mielivaltainen suunta hakuavaruudessa ja kohdemuuttujien mutaatiot ovat lineaarisesti korreloidut. Yksilöt voivat tällä tavoin sopeutua edulliseen hakusuuntaan, mikä on hyödyllistä esimerkiksi kapeiden laaksojen tapauksissa, joissa sopivin suunta mutaatioellipseille on laaksojen myötäinen (Bäck 1996, s. 69).

Itsesäätelyssä käytetään rotaatiokulmia α_{ij}, koska ei ole käyttökelpoista koodata kovariansseja geneettisesti ja itsesäädellä niitä. Tämä johtuu siitä, että koordinaattijärjestelmän pysyminen suorakulmaisena olisi tuolloin vaikea taata.

4.3 Itsesäätely oliomallissa

Eri tasoilla tapahtuvan sopeutumisen ja itsesäätelyn määrittely on oliomallissa hyvin helppoa.

4.3.1 Binäärigeenien mutaatiot

Määrittelemme binääristen geenien mutaatiot (reaalilukugeenien itsesäätely tapahtuu vastaavasti). Ensimmäiseksi täytyy määritellä *Evolutiivinen*-luokka uudelleen:

$$Evolutiivinen_{IS} = (k_m \in \Re \cup Geeni_{\Re};\ \omega_m \in \Omega) \Rightarrow Evolutiivinen, \tag{4.22}$$

jossa k_m on mutaatiotodennäköisyyksien kerroin. Luokassa määritellään mutaatio-operaatio uudelleen siten, että se kertoo parametrina saamansa mutaatiotaajuuden p_m kertoimella k_m ja kutsuu rekursiivisesti alempia rakenteitaan, eli:

$$Evolutiivinen_{IS}[\omega_m(p_m)] \equiv \{\forall 1 \le i \le n : o_i[\omega_m(p_m \cdot k_m);\ \}. \tag{4.23}$$

Tällöin mutaatiotaajuus voi itsesäätyä kaikilla arkkitehtuurin tasoilla. Jos evolutiivisen rakenteen $k_m \in \Re$ (oletusarvo $k_m = 1$), se on vakio eikä itsesäätelyä ei tapahdu. Itsesäätyvyys toteutetaan polymorfismilla $k_m \in Geeni_{\Re}$. Yksilötasolla tämä tapahtuu määrittelemällä itsesäätelevän yksilön genomi:

$$Genomi_{IS} = (k_m \in Geeni_{\Re EIA}) \Rightarrow Evolutiivinen, \tag{4.24}$$

jossa $Geeni_{\Re EIS}$ on ei-itsesäätelevä reaalilukugeeni, jossa mutaatiot tehdään esimerkiksi kaavan (4.13) mukaan.

Komponenttitason itsesäätely toteutetaan vastaavasti asettamalla jokaiseen binääriseen reaalilukugeeniin (ks. Kaava 2.9) mutaatiotaajuutta koodaava geeni k_m:

$$Geeni_{\Re bIS} = (k_m \in Geeni_{\Re b}) \Rightarrow Geeni_{\Re b}. \tag{4.25}$$

Tämä geeni voidaan asettaa vielä alhaisemmallekin tasolle, jossa jokaisella bitillä on tämä geeni.

4.3.2 Muut tyypit

Rakenteellisen itsesäätelyn voi myös laajentaa koskemaan useiden erityyppisten geenien mutaatioita. Yllähän oli toteutettu vain binäärilukuja käyttävien geneettisten algoritmien itsesäätely. Reaalilukuja käyttäville ES:lle määritellään vastaavasti kerroin

$$k_{\Re m} \in \Re \cup Geeni_{\Re EIS}. \tag{4.26}$$

Oliomalli mahdollistaa myös useiden eri tyyppisten geenien käytön samassa genomissa. Tällöin rekursiiviselle mutaatio-operaatiolle tulee antaa parametrina kaikkien eri geenityyppien vaatimat kertoimet erikseen.

Grönroos (1998) on käyttänyt oliomalliin perustuvaa, pääasiassa binääri- ja kokonaislukugeenien mutaatioiden itsesäätelymenetelmää evolvoidessaan hermoverkkorakenteita. Menetelmässä käytettyjä mutaatioparametreja ovat: binäärigeenien mutaatiotodennäköisyys p_m, kokonaislukugeenien (jotka mutatoidaan täysin satunnaisesti) mutaatiotodennäköisyys p_k, reaalilukugeenien varianssi σ sekä mutaatiotodennäköisyys $p_\Re$. Jälkimmäisen parametrin vuoksi on mahdollista, etteivät reaalilukugeenit välttämättä aina mutatoidu lainkaan, päinvastoin kuin evoluutiostrategioissa. Menetelmässä voidaan kullekin geenille määrittää myös oma mutaatiofunktionsa.

Edellä esitetty reaalilukumutaatioiden rotaatiokulmien itsesäätely on oliomallissa vaikea toteuttaa sen rakenteellista ideologiaa noudattaen. Se pakottaa toteuttamaan mutaatioiden käsittelyn genomitasolla.

Luku 5

Yhteenveto

Olemme esittäneet evolutiivisten algoritmien toimintaperiaatteet, joihin liittyy useita hakumenetelmän sisäisiä parametreja, kuten mutaatiotodennäköisyys ja -jakauma. Näitä parametreja voidaan säätää varsin monella eri tavalla, ja olemme esittäneet säätelyn luokittelun näille tavoille. Luokittelu jakautuu kahteen osaan: säätötapaan ja säätötasoon. Säätötapoja ovat kiinteä säätö, deterministisesti muuttuva säätö, sopeutuva säätö ja itsesäätelyn avulla sopeutuva säätö. Säädön tasoja ovat hakuongelma itse, populaatio, yksilö ja alimpana geneettisen perimän komponenttitaso.

Kohdistimme tässä tutkielmassa huomiomme erityisesti itsesäätelyyn yksilö- ja komponenttitasolla. Esittämämme menetelmät kohdistavat itsesäätelyn geneettisten rakenteiden mutaatiotodennäköisyyksiin tai -hajontoihin.

Itsesäätely auttaa ensinnäkin löytämään ongelmalle sopivat parametrit. Toisekseen, kyseiset parametrit voivat muuttua evoluutioprosessin aikana haun kohdistuessa geneettisen hakuavaruuden yhä vain pienempiin osiin. Kolmanneksi, mikäli ongelma eli ympäristö muuttuu haun aikana syystä tai toisesta, voivat parametrit muuttua jatkuvasti uutta ympäristöä vastaaviksi.

Itsesäätely on monia mahdollisuuksia sisältävä komponentti evolutiivisissa algoritmeissa. Se voidaan nähdä alkuna evolutiivisten algoritmien itseohjelmoinnille.

Viitteet

Arabas, J., Z. Michalewicz, and J. Mulawka (1994). GAVaPS — a genetic algorithm with varying population size. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pp. 73–78. IEEE Press.

Bäck, T. (1991, 11-13 December). Self-adaptation in genetic algorithms. In F. J. Varela and P. Bourgine (Eds.), *Proceedings of the First European Conference on Artificial Life. Toward a Practice of Autonomous Systems*, Paris, France, pp. 263–271. MIT Press, Cambridge, MA.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice.* Oxford University Press.

Beasley, D., D. Bull, and R. Martin (1993). An overview of genetic algorithms. *University Computing 15*(4), 170–181.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann.

Fogarty, T. C. (1989). Varying the propability of mutation in the genetic algorithm. In *Proceedings of the 3rd International Conference on Genetic Algorithms and Their Applications*, pp. 104–109. Morgan Kaufmann Publishers, San Mateo, CA.

Fogel, D. B., L. J. Fogel, and J. W. Atmar (1991). Meta-evolutionary programming. In R. R. Chen (Ed.), *Proceedings of 25th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, California, pp. 540–545.

Fogel, L. J., P. J. Angeline, and D. B. Fogel (1995). An evolutionary programming approach to self-adaptation on finite state machines. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel (Eds.), *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, Massachusetts, pp. 355–365. MIT Press.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics 16*(1), 122–128.

Grönroos, M. A. (1998). Evolutionary design of neural networks. Master's thesis, Computer Science, Department of Mathematical Sciences, University of Turku, Finland.

Hesser, J. and R. Männer (1991, 1-3 October). Towards an optimal mutation probability for genetic algorithms. In H. P. Schwefel and R. Männer (Eds.), *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, Volume 496 of *Lecture Notes in Computer Science*, Dortmund, Germany, pp. 23–32. Springer-Verlag, Berlin, Germany.

Hinterding, R. (1995). Gaussian mutation and self-adaptation in numeric genetic algorithms. In *IEEE International Conference on Evolutionary Computation*, pp. 384–389. IEEE Press.

Hinterding, R., Z. Michalewicz, and A. E. Eiben (1997). Adaptation in evolutionary computation: A survey. In *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence.* IEEE Press.

Holland, J. (1975). *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, MI.

Jong, K. A. D. (1975). *An analysis of the behaviour of a class of genetic adaptive systems.* Ph. D. thesis, University of Michigan.

Lis, J. (1996). Parallel genetic algorithm with dynamic control parameter. In *proceedings of the 1996 IEEE Conference on Evolutionary Computation*, Piscataway, NY. IEEE Press.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structure = Evolutionary Programs.* New York: Springer-Verlag.

Radcliffe, N. J. (1992, July). Genetic set recombination. In D. Whitley (Ed.), *Foundations of Genetic Algorithms Workshop (FOGA-92)*, Vail, Colorado.

Rechenberg, R. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Stuttgart: Frommann-Holzboog.

Schaffer, J. D., A. Caruna, L. J. Eshelman, and R. Das (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, San Mateo, California, pp. 51–60. Morgan Kaufmann Publishers.

Schaffer, J. D. and A. Morishima (1987). An adaptive crossover distribution mechanism for genetic algorithms. In *Proceedings of the 2nd International Conference on Genetic Algorithms*, pp. 36–40. Lawrence Erlbaum Associates.

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionstrategie*, Volume 26 of *Interdisciplinary Systems Research*. Basel/Stuttgart: Birkhäuser.