AUTO2050

Optimointi geneettistä algoritmia käyttäen

Touko Hallasmaa

Sisältö

1 Johdanto

Työssä tutustutaan Geneettiseen Algoritmin (GA) toimintaan ja sovelletaan sitä Scala -ohjelmointiympäristössä. Lopputuloksena syntyy ohjelma, jota voidaan käyttää yhtälöiden, sekä maksimointi ja minimointi ongelmien ratkaisuun.

Työssä annetaan joitain esimerkkejä geneettisen algoritmin käytöstä ja tarkastellaan annettujen esimerkkitapausten implementoimista ja eri tekniikoiden vaikutusta geneettisen algoritmin etenemisessä. Luotua työkalua ei sellaisenaan ole mahdollista käyttää kaikkiin mahdollisiin geneettisen algoritmin sovelluksiin.

2 Geneettinen algoritmi

Geneettinen algoritmi on luonnollisen valinnan inspiroima stokastinen, eli satunnaisuuteen perustuva, etsimis- ja optimointiprosessi. Sen aikana käytetään *periytymisen*, *mutaatioiden* ja *risteytysten* prosesseja, joilla pyritään löytämään riittävän soveltuva ratkaisu tarkasteltavaan ongelmaan.

Geneettinen algoritmi eroaa muista tavanomaisemmista haku ja optimointimenetelmistä siten, että sen aikana käsitellään pikemminkin parametreille asetettuja arvoja, kuin parametreja itsessään. Näin ollen ratkaisut tehdään parametrien avulla lasketun tuloksen perusteella sen sijaan, että käytettäisiin apuna jotain johdettua informaatiota, kuten derivaattoja. Tavanomaisesti parametrien avulla lasketun arvon perusteella määritellään kyseisen arvojoukon *kelvollisuus*, ja siten mahdollisuus siirtää tietoa seuraavaan optimointisykliin.

GA kannattaa käyttää ongelmissa, joissa ei välttämättä tarvita parasta mahdollista ratkaisua, vaan riittävän hyvän tuloksen saavuttaminen katsotaan sopivaksi. Menetelmä sopii hyvin myös ongelmiin, joissa erillaisten ratkaisujen arvojoukko on laaja ja monimutkainen. Näin ollen menetelmää voidaan käyttää esimerkiksi NP-täydellisten ongelmien lähes optimaalisten ratkaisuje etsimiseen. Geneettisen algoritmin tukena on lisäksi mahdollista käyttää muita menetelmiä. Geneettinen algoritmi on hyvin skaalautuva, ja sitä voidaan käyttää useista muuttujista riippuvien ratkaisujen etsimiseen. GA:n eduksi voidaan katsoa myös se, että menetelmässä löydetään useita hyviä ratkaisuja.

Mutaatioilla on iso rooli GA:n toiminnassa, sillä ne ovat ainoa tapa tuoda uutta informaatiota populaatioon. Mutaation aikana joitain *genotyypin*, eli tietokonemuotoisen ratkaisun, arvoja muutetaan sattumanvaraisesti. Risteytyksessä luodaan uusia alkioita yhdistelemällä kahden tai useamman populaation jäsenen sisältämää informaatiota eri menetelmin.

2.1 Komponentit

2.1.1 Populaatio

Geneettisessä algoritmissa ratkaisujen joukkoa kutsutaan populaatioksi. Populaation jäseniä manipuloimalla ja yhdistelemällä luodaan niistä uusia sukupolvia, jotka tavoittelevat edelleen tulevissa sukupolvissa parempia ratkaisujoukkoja kunnes geneettiselle algoritmille asetettu ehto on täyttynyt. Populaation koko on syytä valita huolella; liian iso populaatio hidastaa algoritmin etenemistä ja liian pienellä populaatiolla ei välttämättä saada tarpeeksi hajontaa seuraavan populaation ratkaisujen muodostamiseen.

2.1.2 Genotyyppi

Kussakin genotyypissä esillään populaation jäsen tietokoneen ymmärtämässä muodossa. Genotyypit voidaan edelleen kääntää *fenotyypeiksi*, eli yksilöksi tai ratkaisuksi.

2.1.3 Alleeli

Alleeli käsittää tiedon yhden genotyypin tekijän paikasta ja arvosta. Alleelin arvot voidaan valita sovelluskohteesta riippuen binääri-, kokonais- tai liukulukuvuiksi.

2.2 Algoritmin eteneminen

1. Luodaan populaatio, jonka alkioiden arvot asetetaan tässä työssä alussa satunnaisiksi
2. Populaatiosta luodaan uusi sukupolvi **(Tämä kohta ohitetaan ensimmäisellä iteraatiolla)**
   (a) Alkioihin voidaan tehdä satunnaisia mutaatioita
   (b) Tämän jälkeen niiden välillä saatetaan tehdään ristetys, josta syntyy uusi alkio. Alkio voi myös siirtyä muuttumattomana seuraavaan sukupolveen, jos esimerkiksi populaatiossa tehtävää risteytysten määrää on rajattu.
3. Genotyyppi → fenotyyppi
4. Kunkin fenotyypin perusteella lasketaan alkiolle kelvollisuusarvo
5. Fenotyyppi → genotyyppi

6. Kelvollisuusarvon perusteella määritellään genotyypin todennäköisyys siirtää geenejään seuraavan sukupolveen.

7. Palataan kohtaan 2, ellei keskeytykselle asetettu ehto toteudu.

3 Työn suunnittelu

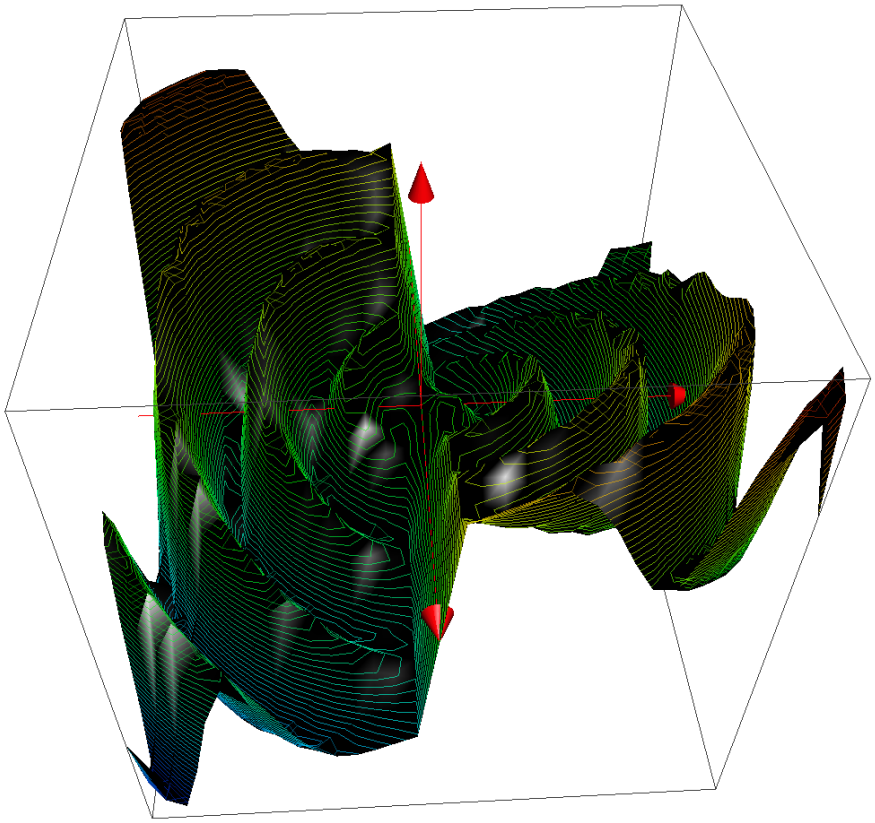
Työssä luodaan sovellus, jolla on mahdollista laskea ohjelmakoodissa määriteltyjä ongelmia geneettisen algoritmin avulla. Ongelmaa varten tulee määritellä funktio kelvollisuusarvon laskemista varten, sekä siinä esiintyvät muuttujat. Lisäksi voidaan määrätä geneettinen algoritmi käyttämään jotain ennalta luotua valintaprosessia, jolla valitaan kunkin syklin aikana populaation jäsenet, jotka siirtävät informaatiota seuraavaan populaatioon.

Ennen sovelluksen kehittämistä valittiin yhtälöt 1 ja 2, jolla geneettisen algoritmin aikana luotujen populaatioiden sopivuutta olisi helppo tarkastella. Yhtälössä 1 voidaan visualisoida populaatioiden kehitystä graafisesti kuvaajalla 1. Yhtälön 2 teoreettinen suurin ja pienin voidaan helposti johtaa, kun on määritelty siinä käytettävien muuttujien suurimmat ja pienimmät mahdolliset arvot. Esitetyistä yhtälöistä luodaan maksimointiongelmat, jotka voidaan suorittaa ohjelmasta.

$$f(x,y) = y \sin\left(\sqrt{x^2+y^2}\right) + x\,(sign(y)) \tag{1}$$

$$f(\vec{a}) = a_1 - a_2 + a_3 - a_4 + a_5 - a_6 + a_7 - a_8 \qquad \{\vec{a} \in \mathbb{R}^8\} \tag{2}$$

Kuva 1: Yhtälössä 1 esitetyn funktion kuvaaja xyz-tasossa välillä $(x, y, z) = ([-20, 20], [-20, 20], [-40, 40])$



Koska tarkasteltaville muuttujille annetaan arvo niille määritetyn maksimi- ja minimiarvon väliltä, päädyttiin genotyyppien alleelit esittämään liukulukuarvoin. Alleelit voivat saada lukuarvoja väliltä $[0, 1]$, ja kun genotyypit käännetään fenotyypeiksi, skaalataan kunkin muuttujaan liitetyn alleelin arvon perusteella luku muuttujan maksimi ja minimiarvon välillä. Kun fenotyypin arvot on johdettu, voidaan ne edelleen syöttää funktioon, joka laskee alkiolle kelvollisuusarvon.

Mutaation todennäköisyys on syytä pitää matalana, sillä muutoin geneettinen algoritmi muuttuu liian satunnaiseksi. Jos alleelien arvot olisivat muodoltaan binäärilukuja, mutaatio voitaisiin toteuttaa siten, että mutatoituvissa alleeleissa arvo 0 muutetaisiin arvoon 1. Koska työssä olevien alleelien arvot ovat liukulukuja väliltä $[0, 1]$, toteutetaan mutaatio siten, että mutatoitavaan alleeliin lisätään satunnainen arvo joka voi olla myös negatiivinen. Jotta mutaatiosta saadaan hienovaraisempaa,

satunnainen arvo lasketaan siten, että mitä lähempänä nollaa arvo on, sen todennäköisempää sen valinta on. Työssä kaikille genotyypin alleeleille annetaan mahdollisuus mutaatioon, joten yhden syklin aikana alkioon voi tapahtua teoriassa yhtä monta mutaatiota, kun siinä on alleeleja.

Sovellus toteutetaan oliopohjaisena Scala -ohjelmointikieltä käyttäen. Kutakin ongelmaa varten luodaan valvoja (*Overseer*) luokan olio, joka vastaa geneettisen algoritmin suorittamisesta ja sen parametrien hallinnasta. Ohjelmaan ei tule graafista käyttöliittyymää ja sitä ohjataan komentoriviltä. Ohjelman käyttäjän tulee voida valita ratkaistava ongelma, populaation koko, mutaation ja risteytyksen todennäköisyys, sekä pystyä hallinnoimaan ongelmaa varten määriteltyjen muuttujien arvoja.

4 Työn toteutus

Ohjelmointi aloitettiin luomalla valikot ohjelmassa navigoimista varten, tätä varten luotiin apufunktioita objektiin console.scala. Valikkoihin lisättiin pääsy mm. ongelman valintaan, muuttujien ja parametrien hallintaan sekä GAn toimintojen suorittamiseen. Valikkojen luominen sujui ongelmitta, ja onnistuminen kokeilemalla navigointia konsoliohjelmassa. Valikkojen toiminnallisuuksiin ei puututtu vielä tässä vaiheessa.

Tämän jälkeen alettiin luomaan luvussa 2.1 esiteltyjä komponentteja. Alkioita varten, joka on geno- ja fenotyypin yläluokka, luotiin erillinen luokka muuttujia varten (*Variable*), minkä jäseniin alkion alleelien arvot liitettiin. Muuttuja luokan oliot sisältävät vapaasti määriteltävät nimi ja yksikkö kentät, jotka selventävät ohjelman käyttäjälle muuttujan käyttöä. Luokassa on tieto myös kullekkin muuttuja luokan oliolle laskettavan arvon ylä- ja alarajasta, joita sovelluksen käyttäjä voi muuttaa ajon aikana. Sovellus tukee myös vakioita, jotka voidaan ottaa käyttöön määrittelemällä ylä- ja alaraja samaksi.

Kun komponentit oltiin saatu luotua, tehtiin luokka geneettisen algoritmin hallintaa varten. Luokkaan lisättiin metodi valvojan parametrien (mutaation todennäköisyys, risteytyksen todennäköisyys, populaation koko ja elitismi) hallintaan sekä metodi valvojan käsittämän ongelman muuttujien hallintaan. Luokkaan tehtiin funktiot myös risteytystä ja uuden populaation generoimista varten. Uuden populaation generoimista varten tulee valita nykyisestä populaatiosta jäsenet, jotka siirtävät tähän informaatiota. Tätä varten luotiin oma piire (*Trait, ParentSelection*), jonka jälkeen luotiin edelleen tästä periytyviä erilaisia valintametodeja käsittäviä objekteja.

Työn aikana luotiin kolme eri valintatapaa. Jokaisessa näissä valintatavoissa valitaan riittävä määrä vanhempia seuraavan populaation generoimsta varten. *RouletteSelection*:issa alkiot järjestetään kelvollisuusarvon mukaiseen järjestykseen, ja jonon

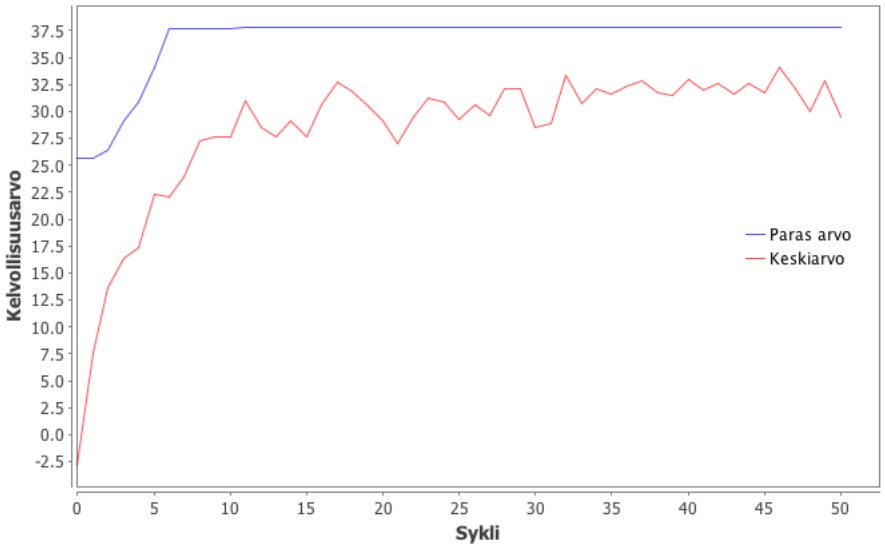
kullakin jäsenellä on tämän kelvollisuusarvosta johdettu todennäköisyys ($\frac{\text{alkion kelvollisuusarvo}}{\text{Populaation yhteenlasketut kelvollisuusa}}$ päätyä vanhemaksi. *RankSelection:ion* ei suosi yhtä paljoa korkean kelvollisuusarvon omaavia yksilöitä, mikä vähentää GA:n jämähtämäistä lokaaleihin maksimeihin tai minimeihin. Edellä esitelty valintamenetelmä eroaa RouletteSelection:ista siten, että todennäköisyys alkion valinnalle määritellään sen paikasta järjestetystä populaatiosta ($\frac{N-n}{\Sigma n}$), jossa n alkion indeksi, indeksien yhteenlaskettu summa $\Sigma n = N(N+1)/2$ ja N populaation koko. Lisäksi luotiin *RankedTournamentSelection*, joka toimii samoin tavoin edellä mainitun menetelmän kanssa, mutta vanhemmat valitaan satunnaisesta otoksesta koko populaation sijaan.

Tämän jälkeen luotiin luvussa 3 esitetyille ongelmille valvoja luokan oliot. Kelvollisuusfunktioita määriteltäessä tulee ottaa huomioon, että mitä suurempi siitä palautettu arvo on, sitä suurempi todennäköisyys alkion on päätyä seuraavaan populaatioon. Maksimointi ongelmissa kelvollisuusarvon määrittäminen on suoraviivaista, sillä se voidaan suoraan määrittää ongelman "yhtälöksi". Jos on tarve suorittaa minimointi ongelma, niin esimerkiksi yhtälön 2 kelvollisuusarvo voitaisiin laskea seuraavasti: $(-1) \times (a_1 - a_2 + a_3 - a_4 + a_5 - a_6 + a_7 - a_8)$. Jos taas halutaan etsiä kohtia, missä jokin yhtälö saavuttaa tietyn arvon, voidaan esimerkiksi yhtälön 1 tapauksessa funktio määrittää $a - \left| y \sin\left(\sqrt{x^2 + y^2}\right) + x\,(sign(y)) \right|$, jossa a on tavoiteltava arvo.

5 Tulokset

Havaittiin että yhtälön 1 maksimointiongelmassa jämähdettiin joillain suorituskerroilla lokaaleihin maksimeihin käytettävästä valintamenetelmästä riippumatta. Tähän ratkaisuna olisi valintaprosessin luominen, jossa valituksi tulemisen todennäköisyydeksi kelvollisuusarvon rinnalla otettaisiin eroavaisuus jo valituista populaation jäsenistä.

Kuva 2:


Kelvollisuusarvo
37.5
35.0
32.5
30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
-2.5
0
5
10
15
20
25
30
35
40
45
50
Sykli
Paras arvo
Keskiarvo