Juha Reinikainen

Evolutionaariset monitavoiteoptimointialgoritmit

Tietotekniikan kandidaatintutkielma

17. toukokuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Juha Reinikainen

Yhteystiedot: juha.a.reinikainen@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: Evolutionaariset monitavoiteoptimointialgoritmit

Title in English: Evolutionary multi-objective optimization

Työ: Kandidaatintutkielma

Opintosuunta: Kaikki opintosuunnat

Sivumäärä: 26+0

Tiivistelmä: Tässä tutkielmassa selvitetään evolutionaaristen monitavoiteoptimointialgoritmien (MOEA) toimintaa. Tutkielmassa käydään evolutionaaristen menetelmien lisäksi läpi monitavoiteoptimointia. MOEA:ia kuvaillaan yleisellä tasolla, ja esitellään joitain tunnettuja algoritmeja pyrkien antamaan mahdollisimman kattavan kokonaisukuvan algoritmeista. Tutkielmassa vertaillaan myös algoritmeja toisiinsa ja tutkitaan niiden tehokkuutta.

Avainsanat: monitavoiteoptimointi, evolutionaarinen monitavoiteoptimointi, geneettinen algoritmi, Pareto-optimaalisuus, skalarisointi, NSGA-II, MOEA/D

Abstract: Goal of this thesis is to study evolutionary multiobjective optimization algorithms (MOEA). Multiobjective optimization is also presented. MOEAs are examined on abstract level and known algorithms are presented to give a comprehensive view of the algorithms. Algoritms are also compared to each other and their efficiency is explored.

Keywords: multi-objective optimization, evolutionary multi-objective optimization, genetic algorithm, Pareto optimality, scalarization, NSGA-II, MOEA-D

Sisältö

1 Johdanto

Tässä kandidaatintutkielmassa tutkitaan evolutionaarisia monitavoiteoptimointialgoritmeja (MOEA) selvittäen niiden keskeisiä piirteitä yleisesti etsien kirjallisuudesta niille ominaisia piirteitä ja tarkemmin konkreettisten algoritmien avulla. Algoritmien piirteiden lisäksi selvitetään, mitkä asiat vaikuttavat näiden algoritmien tehokkuuteen.

Ensimmäinen tutkimus evoluutioteorian soveltamisesta monitavoiteoptimointiin on 1960-luvulta, mutta David Schafferin 1980-luvun puolivälissä kehittämää VEGA:aa pidetään ensimmäisenä MOEA:na (Coello Coello 2006). Monitavoiteoptimoinnille löytyy useita sovelluksia, kuten tuotantoketjut (Čuček ym. 2012) ja aikataulutus (Johnston ja Giuliano 2011). Menetelmiä löytyy myös paljon, mutta eri menetelmät sopivat eri tehtäviin ja huonosti soveltuvan menetelmän valitseminen voi johtaa ongelman kannalta epäoptimaaliseen lopputulokseen.

Tutkielma on järjestetty seuraavasti. Luvussa 2 käydään läpi monitavoiteoptimoinnin tärkeimmät käsitteet ja esitetään monitavoiteoptimointiongelman ratkaisumenetelmiä. Luvussa 3 käsitellään evolutionaarista monitavoiteoptimointi yleisesti ja luvussa 4 esitellään konkreettisia algoritmeja. Luvussa 5 selvitetään, miten algoritmien tehokkuutta voi tutkia ja, mikä siihen vaikuttaa. Lopuksi havainnot kootaan yhteen luvussa 6.

2 Monitavoiteoptimointi

Tässä luvussa esitellään monitavoiteoptimointia yleisesti. Luvussa käydään läpi monitavoiteoptimointiprosessia ja monitavoiteoptimointiin liittyviä käsitteitä. Ensin määritellään monitavoiteoptimointiongelma formaalisti matemaattisessa muodossa sitten määritellään optimaalisuuden käsite. Käsitteiden esittelyn jälkeen tutustutaan vielä optimoinnin rooliin osana päätöksentekoa ja vielä viimeisenä esitellään perinteisiä monitavoiteoptimointialgoritmeja, joilla monitavoiteoptimointiongelma voidaan ratkaista.

2.1 Monitavoiteoptimointiongelma

Ennen kuin optimointi voidaan aloittaa, ongelma pitää muotoilla matemaattiseksi optimointimalliksi. Optimointimallilla pyritään esittämään optimoinnin kohteena olevaa ilmiötä halutulla tarkkuudella riippuen ilmiön monimutkaisuudesta (Branke ym. 2008, preface).

Monitavoiteoptimointiongelma voidaan mallintaa seuraavassa muodossa

$$\begin{aligned} &\text{min/max } \{f_1(x), f_2(x), ..., f_k(x)\} \\ &x \in \mathbf{S} \end{aligned}$$

Tässä $f_i,\ i = 1, ..., k,\ k \geq 2$ ovat optimoitavat *objektifunktiot* eli tavoitteet ja vaaditaan, että *päätösmuuttujavektori* $x = (x_1, ..., x_n)^T$ kuuluu *sallittujen ratkaisujen joukkoon S*, joka koostuu niistä ratkaisuista, jotka kyseisen ongelman kannalta kelpaavat sen ratkaisuiksi. (Branke ym. 2008, preface). Kukin objektifunktio voidaan asettaa joko minimoitavaksi tai maksimoitavaksi. Optimointiongelmaa, jolla on enemmän kuin kolme tavoitetta kutsutaan runsastavoitteiseksi optimointiongelmaksi (many-objective optimization problem) (Yang ym. 2013).

Monet optimointitehtävät voidaan esittää käyttäen *rajoitefunktioita* seuraavassa muodossa:

$$\min/\max \{f_1(x), f_2(x), ..., f_k(x)\} \tag{2.1}$$

$$g_j(x) \geq 0 \qquad j = 1, ..., J \tag{2.2}$$

$$h_m(x) = 0 \qquad m = 1, ..., K \tag{2.3}$$

$$x_i^{(l)} \leq x_i \leq x_i^{(u)} \qquad i = 1, ..., n \tag{2.4}$$

Tässä (2.1) ovat objektifunktiot, (2.2) epäyhtärajoitteet, (2.3) yhtälörajoitteet ja (2.4) laatikkorajoittet. Sallittujen ratkaisujen joukko **S** määräytyy rajoitefunktioiden perusteella (Branke ym. 2008, luku 3).

2.2 Optimaalisuus

Yksitavoitteisen optimointitehtävän tavoite on löytää päätösmuuttujavektori, jolla tehtävän objektifunktio saa pienimmän tai suurimman arvonsa. Monitavoiteoptimointitehtävän ratkaiseminen eroaa yksitavoitteisesta siinä, että optimoidaan montaa ristiriitaista objektifunktiota yhtä aikaa ja objektifunktioiden optimoiminen yksitellen ei riitä monitavoiteoptimointiongelman ratkaisemiseksi. Monitavoiteoptimoinnille on ominaista, että sille ei ole olemassa yhtä yksiselitteistä optimiratkaisua vaan joukko matemaattisesti yhtä hyviä kompromissiratkaisuja. Näiden optimiratkaisujen joukolle on useita nimityksiä, kuten Pareto-optimaaliset (Pareto optimal), ei-hallitut (nondominated), tehokkaat (efficient) ja ei-huonommat (noninferior) ratkaisut (Branke ym. 2008, luku 1).

Määritellään optimiratkaisut monitavoiteoptimointiongelmalle seuraavasti:

Päätösvektorin $x' \in \mathbf{S}$ sanotaan olevan Pareto-optimaalinen, jos ei ole yhtään toista päätösvektoria x, jolle $f_i(x) < f_i(x')$ kaikilla $i = 1, ..., k$ ja $f_j(x) < f_j(x')$ ainakin jollain j:n arvolla (Branke ym. 2008, preface).

Päätösmuuttujavektorit muodostavat päätösavaruuden (decision space) ja niitä vastaavat objektivektorit $z = f(x) = (f_1(x), ..., f_k(x))$ muodostavat objektiavaruuden (objective space) (Branke ym. 2008, preface). Pareto-optimaalisten päätösmuuttujavektorien kuvausta objektiavaruuteen kutsutaan Pareto-optimaaliseksi rintamaksi (Pareto optimal front) (Zhang ja Li

2007).

Pareto-optimaalisuudesta seuraa, että ratkaisun paraneminen yhden tavoitteen suhteen johtaa huononemiseen ainakin yhden toisen tavoitteen suhteen (Ruiz ym. 2015). Kaikkien Pareto-optimaalisten ratkaisujen määrittäminen on yleisesti erittäin aikaa vievää tai jopa mahdotonta, koska niitä voi olla ääretön määrä (Zhang ja Li 2007) tai käytetty menetelmä ei pysty aina löytämään kaikkia ratkaisuja, kuten alaluvussa 2.4 esitettävä painokerroinmenetelmä.

2.3 Päätöksenteko

Päätöksenteko on tärkeä osa optimointia. Optimointi voidaan nähdä prosessina, jolla tuetaan päätöksentekijän päätöksentekoa eli toisaalta optimiratkaisun etsimistä (Branke ym. 2008, luku 1.1). Monitavoiteoptimointiongelmalle voidaan yleensä saada monta ratkaisua, ja päätöksentekijän vastuulle jää valita näistä mieluisin (Branke ym. 2008, luku 3.3.1).

Optimointi tapahtuu päätöksentekijän ja analyytikon vuorovaikutuksen avulla. Päätöksentekijä on henkilö, jolla oletetaan olevan ymmärrystä optimoinnin kohdealueesta. Päätöksentekijä ohjaa optimointiprosessia antamalla tietoa liittyen päätösmuuttujien ja objektifunktioiden arvoihin (Branke ym. 2008, luku 1.1). Päätöksentekijän lisäksi optimoinnissa tarvitaan analyytikko, joka on henkilö tai ohjelma, jonka tehtävä on mallintaa ja ratkaista optimointitehtävä (Branke ym. 2008, luku 1.1).

Päätöksenteon kannalta on tärkeää, että vaikuttavat seikat, kuten objektifunktioiden arvot, esitetään muodossa, jonka päätöksentekijä voi ymmärtää. Päätöksenteon tukena voi käyttää esimerkiksi tilastomenetelmiä, jotka pyrkivät esittämään tiedon helposti ymmärrettävässä muodossa (Branke ym. 2008, luku 8). Valintojen tekemisen helpottaminen lisää analyytikon ja päätöksentekijän vuorovaikutuksen mielekkyyttä ja lisää päätöksentekijän motivaatiota jatkaa optimointiprosessia, kunnes mieluisa ratkaisu on löytynyt (Ruiz ym. 2015).

Päätöksentekijä voi osallistua prosessiin eri vaiheissa riippuen käytetystä menetelmästä. Monitavoiteoptimointimenetelmät luokitellaan usein päätöksentekijän osallistumisen luonteen mukaan. Esimerkiksi priori-menetelmissä päätöksentekijä osallistuu vain alussa ja interaktiivisissa menetelmissä koko optimointiprosessin ajan (Branke ym. 2008, luku 1.1).

2.4 Skalarisointi

Monitavoiteoptimointiongelmat ratkaistaan perinteisesti skalarisoimalla. Skalarisointi tarkoittaa, että monitavoitteinen optimointiongelma muutetaan yksitavoitteiseksi koostamalla objektifunktiot yhteen. Saatu funktio voidaan ratkaista yksitavoitteisille optimointiongelmille tarkoitetuilla algoritmeilla. Menetelmille pystyy todistamaan, että ne löytävät Pareto-optimaalisia ratkaisuja monitavoiteoptimointiongelmille (Branke ym. 2008, luku 1). Yhdellä skalarisointimenetelmän suorituksella löydetään yksi ratkaisu, mutta parametreja muuttamalla voidaan tuottaa myös useita ratkaisuja. Saatuja ratkaisuja voidaan verrata suoraan objektifunktion arvon perusteella (Zhang ja Li 2007). Skalarisointia voidaan hyödyntää myös evolutionaarisissa menetelmissä, kuten luvussa 4.2 esitettävä MOEA/D tekee.

Painokerroinmenetelmää (Weighting method) voidaan käyttää Pareto-optimaalisien ratkaisujen etsimiseen monitavoitteisille ongelmille. Siinä ongelma muutetaan yksitavoitteiseksi käyttämällä painokertoimia (Branke ym. 2008, luku 1).

$$\begin{cases} \min \quad \Sigma_{i=1}^{k} w_i f_i(x) \\ x \in \mathbf{S} \\ w_i \geq 0, \ \forall i = 1, ..., k \end{cases}$$

Menetelmällä voidaan tuottaa eri Pareto-optimaalisia ratkaisuja objektifunktioiden painoja muuttelemalla kunhan $w_i > 0$. Menetelmän huonona puolena on se, että se pystyy luotettavasti tuottamaan minkä tahansa Pareto-optimaalisen pisteen vain, jos ongelma on konveksinen. Jos ongelma ei ole konveksinen, menetelmän tuottama Pareto-optimaalinen joukko ei välttämättä anna hyvää kokonaiskuvaa koko joukosta (Branke ym. 2008, luku 1).

Epsilon-rajoitemenetelmässä yksi objektifunktio valitaan optimoitavaksi ja muut asetetaan rajoitteiksi tietyillä ylärajoilla.

$$\begin{cases} \min \quad f_i(x) \\ f_j(x) \leq \varepsilon_j, \ \forall j = 1,...,k, j \neq i \\ x \in \mathbf{S} \end{cases}$$

Menetelmälle pätee:

$x^* \in \mathbf{S}$ on Pareto-optimaalinen $\iff$ x^* ratkaisee Epsilon-rajoitetehtävän $\forall i = 1,...,k, \ \varepsilon_j = f_j(x^*), \ j = 1,...,k, j \neq i$

ja

x^* on yksikäsitteinen $\implies$ x^*on Pareto-optimaalinen, $\varepsilon_j \in \mathbf{R}, j = 1,...,k$

Menetelmä pystyy löytämään minkä tahansa Pareto-optimaalisen ratkaisun riippumatta siitä onko tehtävä konveksinen toisin kuin painokerroin menetelmä. Menetelmän huonona puolena on se, että on vaikea määrittää sellaiset ylärajat ε_j, että ongelmalle löytyy ratkaisu (Branke ym. 2008, luku 1).

3 Evolutionaarinen monitavoiteoptimointi

Evolutionaariset monitavoiteoptimointialgoritmit (MOEA) pyrkivät ratkaisemaan monitavoiteoptimointiongelman. MOEA:t ovat populaatioon pohjautuvia stokastisia hakumenetelmiä, jotka iteratiivisesti painottavat parhaita alkioitaan yhdistellen ja muunnellen niitä yrittäen luoda yhä parempia ja parempia populaatioita, kunnes ollaan saavutettu tarpeeksi hyvä ratkaisu (Branke ym. 2008, luku 3.2). Menetelmät pohjautuvat luonnonevoluution matkimiseen, ja varsinkin Darwinin luonnonvalinta-ideologiasta on otettu inspiraatiota (Branke ym. 2008, preface).

MOEA:n päätehtävät ovat ohjata populaatiota kohti Pareto-optimaalista rintamaa varmistaen suppenemisen ja säilyttää populaation alkioiden monipuolisuuden antaen hyvän kokonaiskuvan kaikista ratkaisuvaihtoehdoista eli Pareto-optimaalisen rintaman koosta ja muodosta (Branke ym. 2008, luku 3; Liu ym. 2017).

MOEA:t ovat yleensä suorahakumenetelmiä. Se tarkoittaa, että funktioiden arvojen laskeminen on keskeinen keino ohjata algoritmin etenemistä. MOEA:t eivät yleensä käytä gradienttitietoa hyväkseen. Gradientti kertoo, mihin suuntaan funktion arvot muuttuvat nopeimmin. Gradienttia hyödyntävät menetelmät soveltuvat paremmin kuin suorahakumenetelmät ongelmille, joissa käsiteltävät funktiot ovat hyvin käyttäytyviä. Toisaalta se, että ei käytetä gradienttia, mahdollistaa sellaisien optimointitehtävien ratkaisemisen, joissa esimerkiksi objektifunktio ei ole differentioituva (Branke ym. 2008, luku 3.2; Bosman ja Jong 2005).

Alla oleva pseudokoodi 1 kuvaa MOEA:n yleisen rakenteen. Alaluvuissa kerrotaan tarkem-

min, mitä kukin operaatio tekee (Branke ym. 2008, luku 3).

Algorithm 1: Algoritmin rakenne

t = 0;
AlustaPopulaatio(P_t);
while *Ei valmis* **do**
    evaluointi(P_t);
    $P_t^{'}$ = valinta(P_t);
    $P_t^{''}$ = muuntelu($P_t^{'}$);
    P_{t+1} = elitismi($P_t^{'}$, $P_t^{''}$);
    t = t + 1;
end

3.1 Populaatio

Yleensä optimointialgoritmit optimoivat yhtä arvoa kohti optimia, jolloin yhtä alkuarvausta kohti saadaan yksi optimaalinen ratkaisu. Tämä on toimiva ratkaisu silloin, kun ongelmalle on vain yksi ratkaisu, mutta monitavoiteoptimointiongelmille on yleensä useita matemaattisesti yhtä hyviä ratkaisuja tai jopa äärettömän monta. Tällöin voi olla mielekästä yrittää optimoida yhden arvon sijaan useita arvoja yhtä aikaa. Tätä usean arvon joukkoa kutsutaan populaatioksi. Populaation käyttäminen yhden pisteen sijaan lisää laskennallista kuormitusta, mutta laskentaa voidaan nopeuttaa esimerkiksi rinnakkaistamalla algoritmin osia.

Algoritmin alussa luotava alkupopulaatio voidaan muodostaa satunnaisista alkioista huomioiden kuitenkin päätösmuuttujien mahdolliset ala- ja ylärajat. Alkupopulaation muodostuksessa voidaan hyödyntää myös aiemmin löydettyjä ratkaisuja ja asiantuntijan tuntemusta. Hyvän alkuratkaisun valitseminen voi nopeuttaa optimointia merkittävästi (Branke ym. 2008, luku 3).

3.2 Evaluointi

Evaluointivaiheessa määritetään populaation alkioiden kelpoisuus käyttäen kaikkea saatavilla olevaa tietoa (Wang ym. 2008). Luvussa 2.2 määriteltiin Pareto-optimaalisuus kuvaamaan

monitavoiteoptimointiongelman ratkaisuja. Löysentämällä Pareto-optimaalisuuden käsitettä koskemaan vain populaation alkioita, se saadaan järjestettyä osittain järjestetyksi joukoksi käyttämällä objektifunktion arvoja (Rudolph ja Agapie 2000). Evolutionaarisia menetelmiä käsittelevässä kirjallisuudessa Pareto-optimaalisuuden tilalla käytetään yleensä hallitsevuuden käsitettä. Pareto-optimaalisia ratkaisuja populaatiossa kutsutaan ei-hallituiksi ratkaisuiksi.

Jos ei haluta, että joukossa on alkioita, jotka ovat määritelmän suhteen yhtä hyviä, voidaan käyttää lisäksi esimerkiksi rajoitefunktioita ja monipuolisuuden säilytystekniikoita ratkaisujen tarkempaan järjestämiseen (Wang ym. 2008). Näitä tekniikoita on esitelty luvussa 4.

Jotta ratkaisut olisivat sallittuja, kannattaa ratkaisuille laskea myös rajoitefunktioiden arvot. Sallittujen ratkaisujen asettaminen kelpoisimmiksi ohjaa hakua kohti sallittuja ratkaisuja. Toisaalta jotkut ei-sallitut ratkaisut voivat ohjata hakua tehokkaasti kohti Pareto-optimaalista rintamaa, joten niitä ei ole pakko hylätä kokonaan (Wang ym. 2008).

3.3 Valinta

Valintavaiheessa populaatiosta yritetään määrittää parhaat ratkaisut. Kuten evaluointiosassa todettiin, ratkaisujen paremmuus ei ole yksikäsitteistä, mutta valinnassa kannattaa painottaa ei-hallittuja ratkaisuja. Muillekin kuin ei-hallituille ratkaisuille voidaan antaa mahdollisuus tulla valituiksi. Yksinkertainen tapa toteuttaa valinta on turnajaisvalinta (tournament selection). Siinä kaksi alkiota valitaan sattumanvaraisesti populaatioista ja näistä parempi valitaan (Branke ym. 2008, luku 3).

(Zitzler, Laumanns ja Thiele 2001) jakaa valinnan parittelu- ja ympäristövalintaan (mating and environment selection). Paritteluvalinnalla ohjataan hakua kohti Pareto-optimaalista rintamaa määrittämällä alkioille kelpoisuuden ja valitsemalla parhaat alkiot uuden populaation luomista varten. Ympäristövalinta käsittelee, mitkä ratkaisut pidetään tallessa sukupolvien välillä. Ympäristövalinta toteutetaan yleensä algoritmeissa pitämällä yllä arkistoa (archive) löydetyistä ei-hallituista ratkaisuista.

Kaikkia löydettyjä ratkaisuja ei voida tallettaa arkistoon aika ja tallennustilasyistä, joten pi-

tää tehdä valintoja, mitkä alkiot säilytetään (Zitzler, Laumanns ja Thiele 2001). Teoriassa rajoittamattoman kokoisen arkiston käyttäminen parantaisi ratkaisujen tarkkuutta, mutta koska tämä ei ole käytännöllistä, samaa tavoitetta voidaan tavoitella vahvalla eliittisäilytyksellä (Laumanns, Zitzler ja Thiele 2001, luku 5.1).

3.4 Muuntelu

Muuntelun idea on muodostaa aiemman populaation avulla uusi toivottavasti parempi populaatio P_t''. Muuntelu sisältää useita geneettisiä operaattoreita, kuten risteytyksen ja mutaation. Muuntelun tavoitteena on lisätä populaation monipuolisuutta samalla ohjaten hakua kohti Pareto-joukkoa. (Zhang ja Li 2007) ehdottavat korjausheuristiikkojen (repair heuristic) käyttämistä rajoitefunktioiden huomioimisessa.

Risteytys luo uuden populaation käyttäen aiemmin löydettyjä ratkaisuja. Aiemmista vanhempialkioista luodaan yhdistelemällä uusia lapsialkioita. Alkiot valitaan risteytykseen todennäköisyydellä $p_c \in [0,1]$, joka kuvaa risteytykseen osallistuvan osuuden kokoa.

Pääsääntöisesti, jos valitut vanhempialkiot ovat hyvin erilaisia toisistaan, luodut lapsialkiot ovat myös erilaisia kuin vanhempansa ja, jos vanhemmat ovat samanlaisia, lapsetkin ovat lähellä vanhempiaan. Tämä johtaa siihen, että erilaisia vanhempia risteytettäessä, ratkaisujen monipuolisuus säilyy ja samanlaisia risteytettäessä tarkennutaan jo löytyneeseen toivottavasti hyvään alueeseen. Tämä ominaisuus tekee MOEA:ista itsestään sopeutuvia (Branke ym. 2008). Toisaalta eri tutkimuksissa on saatu ristiriitaisia tuloksia sen suhteen, kannattaako hyvin erilaisia alkioita risteyttää (Rudenko ja Schoenauer 2004).

Risteytyksessä luotua populaatiota prosessoidaan vielä lisää mutatoimalla. Mutaatio mahdollistaa paikallisen, muista populaation alkioista riippumattoman vaihtelun lisäämisen hakuun. Toisin kuin risteytys, mutaatio tehdään yleensä kaikille populaation alkioille. Mutatoitavat päätösmuuttujat ratkaisussa valitaan todennäköisyydellä p_m, joka taas kuvaa ratkaisun eli päätösmuuttujavektorin mutatoitumisen määrää (Branke ym. 2008, luku 3).

3.5 Eliitti-säilytys

Eliitti-säilytyksellä tarkoitetaan algoritmin kykyä pitää yllä parhaita ratkaisuja kaikista jo tutkituista alkioista (Coello Coello 2006). Eliitti-säilytys toteutetaan yleensä pitämällä yllä ulkoista populaatioita, joka koostuu sillä hetkellä löydetyistä ei-hallituista ratkaisuista suhteessa kaikkiin jo tutkittuihin ratkaisuihin. Laumanns, Zitzler ja Thiele (2001) määrittelevät, että MOEA on elitistinen, jos ulkoisen populaation eli arkiston alkiot osallistuvat uusien alkioiden tuottamiseen.

Useissa lähteissä on huomattu, että elitismin käyttäminen lisää algoritmin suorituskykyä ja sillä varmistetaan algoritmin monotonisesti ei-heikkenevä suorituskyky. (Branke ym. 2008; Kalyanmoy Deb ym. 2002; Laumanns, Zitzler ja Thiele 2001). Coello Coello (2006) toteaa, että elitismi on tärkeä myös suppenemisen varmistamiseksi. MOEA:n ei ole pakko käyttää ulkoista populaatioita eliitti-säilytykseen, vaan se voidaan toteuttaa dynaamisemmin esimerkiksi säilyttämällä edellisestä ja uudesta populaatiosta parhaat. Tällä tavalla saadaan vähennettyä algoritmin tilantarvetta ja toisaalta myös aikavaativuutta, jos arkiston koko on suuri suhteessa populaation kokoon (Coello Coello 2006). Toisaalta se, että ei käytä ulkoista populaatioita mahdollistaa hyvien ratkaisujen menettämisen, koska jossain aiemmassa populaatioissa on voinut olla alkio, joka hallitsee jotain nykyisen populaation alkioista, jolloin löydetty populaatio ei ole paras suhteessa kaikkiin läpikäytyihin alkioihin.

3.6 Lopetusehto

MOEA:ille voidaan asettaa erilaisia lopetusehtoja. Joissain tapauksissa tiedetään suoraan ratkaisusta, että se on tarpeeksi hyvä asetettujen tavoitteiden perusteella. Voidaan määritellä lopetusehto peräkkäisten populaatioiden arvojen suppenemisen perusteella hieman samaan tapaan kuin yhden objektifunktion tapauksessa, jossa verrataan, kuinka kaukana peräkkäisten ratkaisujen objektifunktion arvot ovat toisistaan.

Teoreettisia optimaalisuusehtoja on myös käytetty optimaalisuuden määrittämiseen, kuten Karush-Kuhn-Tucker-ehtoja (KKT) (Branke ym. 2008, luku 3). Menetelmien heuristisen luonteen takia (Branke ym. 2008, s. 64) on vaikea sanoa, miten lähelle ratkaisut päätyvät Pareto-optimaalista rintamaa tai hajautuvatko ratkaisut jopa poispäin optimista (Sindhya,

Deb ja Miettinen 2011). Tästä syystä on hyvä vahvistaa algoritmin päättyminen esimerkiksi lopettamalla ohjelman suoritus, kun tarpeeksi suuri määrä populaatioita on luotu.

4 Evolutionaariset monitavoiteoptimointialgoritmit

Tässä luvussa esitellään evolutionaarisia monitavoiteoptimointialgoritmeja painottaen niiden eroja luvussa 3 esitettyyn perusalgoritmiin 1 verrattuna. Usein algoritmit on esitelty kirjallisuudessa siten, että vain suurimmat erot edellisiin näkyvät, eikä esimerkiksi kaikkien geneettisten operaattorien toteutukseen ole otettu kantaa. Tässä luvussa on otettu sama näkökulma, koska algoritmin yksityiskohdat riippuvat paljon käyttökohteesta. Esimerkiksi, jos päätösmuuttujat ovat kokonaislukuja, reaalilukuja käsittelevä risteytysmenetelmä ei tuottaisi sallittua ratkaisua. Toisaalta yksityiskohtien jättäminen huomioimatta mahdollistaa löydetyn hyvän pohja-algoritmin pohjalta uusien algoritmien toteuttamisen, joka näkyy esimerkiksi NSGA-II:n nimestä. Algoritmien osien komponenttimaisuus on mahdollistanut niiden kokoamisen yhteen viitekehyksiksi. Esimerkiksi Liefooghe ym. (2007) esittelevät ParasEO-MOEA nimisen viitekehyksen, jonka tavoitteena on helpottaa ja nopeuttaa evolutionaaristen monitavoiteoptimointisovellusten kehittämistä tuottaen tehokkaita sovelluksia vähentäen ohjelmointivaivaa ja maksimoiden ohjelmakoodin uudelleenkäytön. Alaluvuissa esitellään kaksi tunnettua MOEA:aa, joista NSGA-II ratkaisee optimointitehtävän suoraan monitavoitteisena ja MOEA/D, joka ratkaisee tehtävän jakamalla sen osiin.

4.1 NSGA-II

Kalyanmoy Deb ym. (2002) esittelemä ei-hallitsevasti järjestävä geneettinen algoritmi II (Nondominated sorting genetic algorithm II (NSGA-II)) löytää nopeasti ja tehokkaasti kattavan määrän ratkaisuja läheltä Pareto-optimaalista rintamaa. Se on paranneltu version NSGA:sta. Menetelmä pärjää hyvin suhteessa muihin aikaisiinsa MOEA:ihin, kuten PAES:iin ja SPEA:an ratkaisujen monipuolisuuden ja suppenemisen perusteella. Menetelmä on aikavaativuudeltaan $O(MN^2)$, jossa M on objektifunktioiden lukumäärä ja N populaation koko.

Algoritmi järjestää alkiot ei-hallitsevuustasoihin (nondomination levels) parhaiden löytämiseksi populaatiosta. Perusidea tasoihin järjestämisessä on etsiä ensin kaikki populaation ei-hallitut ratkaisut. Ensimmäinen taso koostuu niistä alkioista, joita ei hallitse yksikään populaation alkio. Sitten ensimmäiselle tasolle kuulumattomista alkioista etsitään jälleen ei-

hallitut ja tätä toistetaan, kunnes kaikille alkioille on saatu asetettua taso. NSGA-II nopeuttaa tätä toimenpidettä määrittelemällä ensin jokaiselle alkiolle sen hallitsemat alkiot S_p ja sitä hallitsevien alkioiden lukumäärän n_p. Ei-hallituilla ratkaisuilla n_p on nolla, joten saamme ensimmäisen hallitsevuustason. Seuraavat tasot saadaan etsimällä aina edellistason alkioiden hallitsemista alkioista alkiot, joita hallitsee vain yksi alkio ja jatkamalla tätä, kunnes kaikki alkiot on saatu järjestettyä.

NSGA-II käyttää ratkaisujen monipuolisuuden parantamiseen tungos-vertailumenetelmää (crowded-comparison approach). Sitä käytetään valinnassa ja populaation pienennysvaiheessa. Osalla valinnassa valituista ratkaisuista saattaa olla sama ei-hallitsevuustaso, joten tarvitaan menetelmä näiden ratkaisujen vertailuun. Samalla tasolla oleville alkioille lasketaan tiheysestimaatti, joka kertoo, kuinka lähellä sitä on muita ratkaisuja. Kullekin ratkaisulle x_i etsitään kutakin k:ta objektifunktiota kohden kaksi lähintä ratkaisua, ja tiheysestimaatti on näiden lähimpien ratkaisuparien etäisyyksien summa. Pieni tiheysestimaatin arvo tarkoittaa, että ratkaisu on ruuhkaisella alueella, ja algoritmi suosii niitä ratkaisuja, joiden tiheysestimaatin arvo on suuri eli lähimmät ratkaisut ovat mahdollisimman kaukana.

Algoritmi toimii seuraavasti. Aluksi luodaan alkupopulaatio, joka järjestään paremmuusjärjestykseen. Jokaisella kierroksella edellisen ja nykyisen populaation unionista muodostetaan uusi populaatio. Se muodostetaan valitsemalla unionista järjestyksessä parhaat alkiot ei-hallitsevuuden ja tungosvertailun avulla. Kierroksen lopuksi muodostetaan seuraava populaatio geneettisillä operaatioilla. Ensimmäisellä kierroksella edellinen populaatio on tyhjä joukko, joten kaikki N alkiota osallistuu geneettisiin operaatioihin.

Algoritmin tehokkuutta voi parantaa kiinnittämällä huomiota populaation kokoon. Jos ei-hallitsevuus riittää määrittämään parhaat alkiot, tungosvertailumenetelmää ei tarvita, mikä säästää aikaa. Kaikkia alkioita ei myöskään tarvitse järjestää, koska tarvitaan unionista, jonka koko on 2N, vain N alkiota.

Algoritmi ei käytä ulkoista populaatiota elitismin toteuttamiseksi vaan uuden populaation luomisessa hyödynnetään edellisen populaation ei-hallittuja ratkaisuja. Koska algoritmi suosii luonnostaan ei-hallittuja ratkaisuja, edellispopulaation parhailla alkioilla on suurempi todennäköisyys selvitä. Tämä menetelmä voi olla joissain tapauksissa tehokkaampi kuin ul-

koinen populaatio riippuen ongelmasta.

4.2 MOEA/D

Zhang ja Li (2007) esittelevät MOEA:n nimeltä MOEA/D (Multiobjective evolutionary algorithm based on decomposition), joka perustuu monitavoitteisen ongelman hajottamiseen (decomposition) useaan skalaariseen alioptimointiongelmaan, jotka ratkaistaan samanaikaisesti. Ongelman skalarisointiin voidaan käyttää luvussa 2.4 esitettyjä skalarisointimenetelmiä, mutta muitakin menetelmiä on olemassa, kuten Tchebycheffin menetelmä (Zhang ja Li 2007).

MOEA/D ratkaisee monitavoiteoptimointiongelman etsimällä kullekin N:lle alioptimointiongelmalle optimiratkaisun $x_i^*, i = 1,2,...,N$. MOEA/D ottaa parametrina N painokerroinvektoria $\{\lambda^1, \lambda^2, ..., \lambda^N\}$, ja niillä saadaan muodostettua N alioptimointiongelmaa skalarisoimalla kukin ongelma yhdellä painokerroinvektorilla. Kun ongelmat on saatu muodostettua niitä aletaan iteratiivisesti ratkaisemaan.

MOEA/D:n monipuolisuuden säilytysmekanismi perustuu aliongelmien väliseen etäisyyteen. Se määrittää aliongelmien välisen etäisyyden niihin assosioitujen painokerroinvektorien avulla. Algoritmi etsii jokaiselle painokerroinvektorille lähimmät T naapuria eukliidisen etäisyyden perusteella. Näihin naapuripainokerroinvektoreihin löydyneiden optimiratkaisujen oletetaan olevan lähellä toisiaan. Löydettyjen aliongelmien ratkaisut monipuolisuuden varmistamiseksi painokerroinvektoreiksi $\lambda^1, ..., \lambda^N$ kannattaa valita tasaisesti jakautuneita vektoreita.

Algoritmi toimii seuraavalla tavalla. Ensin algoritmi alustaa tyhjän arkiston ja alkupopulaation $\{x_1, x_2, ..., x_N\}$. Sitten määritetään jokaiselle alkiolle T lähintä naapuria painokertoimien perusteella. Alustuksen jälkeen algoritmi lähtee geneettisesti muuntelemaan kutakin naapurustoa etsien sen parasta ratkaisua. Naapurustoa ja ulkoista populaatiota päivitetään, jos löydetään parempia ratkaisuja.

5 Suorituskyvyn testaaminen

Monitavoiteoptimointialgoritmin tehokkuuden määrittäminen on itsessään monitavoiteoptimointiongelma. Voidaan optimoida tehokkuutta esimerkiksi tarvittavien funktion evaluointien lukumäärän, ylimääräisen tilantarpeen tai algoritmin parametrien lukumäärän suhteen, mutta näiden ristiriidaton optimoiminen yhtä aikaa ei ole todennäköisesti mahdollista monimutkaisien riippuvuuksien takia. Algoritmin voidaan haluta toimivan erityisesti tietyn tyyppisille ongelmille, jolloin pitää tietää, mitkä asiat vaikuttavat suorituskykyyn siitä näkökulmasta. Algoritmin tehokkuutta ei ole myöskään helppo määrittää, koska todellista ratkaisua ei välttämättä tunneta tai vaikuttavia tekijöitä ei osata erotella toisistaan. Tässä luvussa käsitellään algoritmien testaamisen periaatteita suorituskyvyn näkökulmasta ja tarkastellaan, miten algoritmin ominaisuudet vaikuttavat suorituskykyyn.

5.1 Suorituskykytestit

Algoritmien tehokkuutta testataan usein käyttäen testiongelmia, joiden ratkaisu tiedetään entuudestaan, jolloin saatua tulosta voidaan verrata oikeaan ratkaisuun. Testiongelma pitäisi olla helposti ymmärrettävissä ja muotoiltavissa optimointiongelmaksi, jotta saadut tulokset olisi helppo toistaa ja todentaa. Ideaalisesti testi mallintaa jotain kategoriaa tosielämän ongelmista, jotta saadaan tietoa, miten hyvin se toimii kyseisen kategorian ongelmille yleisesti (Zitzler ja Thiele 1998). Teknisestä näkökulmasta testin päätösmuuttujien ja objektifunktioiden määrää pitäisi olla helppo muuttaa ja tehokkuuden eri osa-alueiden testaaminen pitäisi olla mahdollista kontrolloidusti. MOEA:ien testaamista on tutkittu paljon ja on kehitetty testisarjoja, joilla voidaan testata algoritmin tehokkuutta eri näkökulmista (K. Deb ym. 2002).

Testiongelma voidaan luoda yksitavoitteisen ongelman pohjalta (K. Deb ym. 2002). Perinteisessä 0/1-reppuongelmassa (0/1 knapsack problem) halutaan valita reppuun pakattavat tavarat siten, että sen arvo on mahdollisimman suuri, kun jokaisella tavaralla on jokin arvo ja asetettua painorajaa ei saa ylittää. Ongelma saadaan muutettua monitavoitteiseksi käsittelemällä useita reppuja eli maksimoimalla kaikkien reppujen arvoja yhtä aikaa (Zitzler ja Thiele 1998). Ongelma sopii hyvin testiongelmaksi, koska se on helppo ymmärtää, muotoilla op-

timointitehtäväksi ja sen todellinen Pareto-optimaalinen rintama osataan estimoida tarkasti. Lisäksi ongelmaa voidaan mukauttaa helposti muuttamalla reppujen ja päätösmuuttujien määriä.

Testiongelman muotoilu voidaan aloittaa myös määrittämällä ensin Pareto-optimaalinen rintama matemaattisesti, jolloin ratkaisujen optimaalisuuden selvittäminen on helppoa. Kun Pareto-optimaalinen rintama on saatu muotoiltua, sitä voidaan lähteä laajentamaan halutulla tavalla muodostaen hakuavaruuden. Pareto-optimaalinen rintama voisi olla esimerkiksi osa pallopinnasta ja hakuavaruus voitaisiin määritellä kyseisen pallon ja toisen suuremman pallon välisenä alueena (K. Deb ym. 2002). Hakuavaruuden muotoa ja paikkaa on helppo säätää tällaisessa lähestymistavassa muuttamalla funktioiden kertoimia, mutta monimutkaisien avaruuksien esittäminen matemaattisesti voi olla hankalaa. Evolutionaarisia menetelmiä voidaan käyttää myös muodon optimointiin, jossa etsitään esimerkiksi mahdollisimman aerodynaamista muotoa. Tällaisissa ongelmissa hakuavaruuden muodon määrittäminen on luonnostaan vaikeaa, koska kappale voi olla vaikea esittää optimointiin sopivalla tavalla (Branke ym. 2008, luku 11.2.2).

5.2 Algoritmin parametrien vaikutus suorituskykyyn

Algoritmin parametrien valitseminen suorituskyvyn maksimoimiseksi on myös itsessään optimointiongelma. Sen lisäksi, että parametrit vaikuttavat itsenäisesti suorituskykyyn, myös niiden vaikutus toisiinsa pitää ottaa huomioon (Zitzler ja Thiele 1998). Esimerkiksi eliittisäilytyksen ja mutatoinnin voimakkuuksien on havaittu vaikuttavan suorituskykyyn toisistaan riippuvasti (Laumanns, Zitzler ja Thiele 2001).

Parametreja säätämällä voi vaikuttaa merkittävästi algoritmin suorituskykyyn. Esimerkiksi luotavien sukupolvien ylärajan asettaminen liian pieneksi johtaa helposti siihen, että populaatio ei ole ehtinyt kehittyä vielä tarpeeksi, jolloin ratkaisut eivät ole optimaalisia. Epäoptimaalisilla asetuksilla on myös helppo vääristää vertailun tuloksia asettamalla toiselle algoritmille epäsuotuisat parametrin arvot.

MOEA:t ottavat syötteenä optimoitavan mallin lisäksi useita parametreja, kuten populaation koon, risteytys- ja mutaatiotodennäköisyyden ja sukupolvien enimmäismäärän. Jotta para-

metrien säätäminen ei olisi liian vaikeaa, kannattaa pyrkiä minimoimaan tarvittavien parametrien määrä. Esimerkiksi luvussa 4.1 esitetty algoritmi ei käytä edellisversion jakamisparametria.

Algoritmien käyttämissä parametreissa on paljon eroja, mutta myös paljon yhtäläisyyksiä (Zitzler ja Thiele 1998). Näiden erottelu helpottaa algoritmien vertailua, koska voidaan asettaa yhteisen parametrit samoiksi ja keskittyä niihin, joita halutaan tutkia. Esimerkiksi Zhang ja Li (2007) asettavat populaation koon ja sukupolvien ylärajan samaksi verrattaville algoritmeille.

5.3 Algoritmien vertaileminen

Algoritmeja luokitellaan niiden ominaisuuksien mukaan ja niitä vertaillaan parhaan algoritmin löytämiseksi kullekin ongelmatyypille. Samantyyppisten algoritmien vertailu voi olla mielekästä, koska niillä on yhteisiä ominaisuuksia, mikä vähentää esimerkiksi parametrien eroista syntyvää haastetta. Esimerkiksi luvussa 4.2 esitetty MOEA/D ottaa parametrina painokerroinvektoreita, ja vertaamalla sitä toiseen painokertoimia käyttävään menetelmään, ei tarvitse ottaa painokertoimien valinnan vaikutusta huomioon testeissä.

Erityyppisiä algoritmeja vertaillessa voidaan hyödyntää geneettisien operaattorien yhtenäistämistä. Molemmat algoritmit voidaan esimerkiksi asettaa käyttämään turnajaisvalintaa. Algoritmeja on myös muokattu testiasetelman tasapainottamiseksi. Zhang ja Li (2007) muokkaa algoritmiaan olemaan käyttämättä ulkoista populaatiota NSGA-II:seen vertailun aikana, jotta vertailu olisi mahdollisimman reilu.

5.4 Algoritmin ominaisuuksien vaikutus suorituskykyyn

Algoritmien suunnittelussa painotetaan yleensä optimaalisuutta ratkaisujen monipuolisuuden sijaan (Liu ym. 2017). Molemmat luvussa 4 esitetyt algoritmit painottavat valinnassa ensisijaisesti ei-hallittuja ratkaisuja, ja vasta sitten huomioivat ratkaisujen monipuolisuuden painottamalla vähemmän tiheillä alueilla olevia alkioita. Jos tehokkuutta tarkastellaan monipuolisuuden näkökulmasta, nämä algoritmit eivät ole kovin optimaalisia (Liu ym. 2017).

Luvussa 3 todettiin, että monipuolisien ratkaisujen suosiminen auttaa antamaan mahdollisimman hyvän kuvan kaikista ratkaisuvaihtoehdoista. Jotta vaihtoehdot eivät antaisi vääristynyttä kuvaa vaihtoehdoista, kannattaa tavoitella ratkaisujen monipuolisuutta optimaalisuuden lisäksi.

Ulkoisen populaation eli arkiston implementointi vaikuttaa algoritmin suorituskykyyn tilan ja ajan suhteen. Luvussa 3.3 todettiin, että kaikkia ratkaisuja ei voi tallentaa tallennustilan rajallisuuden takia arkistoon. Toisaalta arkiston koon kasvaessa aikavaativuus kasvaa, kun pitää käsitellä aina vain suurempia ja suurempia määriä alkioita. Arkiston koon kasvaessa myös todennäköisyys, että ratkaisut ovat jakautuneet epätasaisesti kasvaa, koska arkistossa säilyvät alkiot valitaan yleensä optimaalisuuden perusteella monipuolisuuden sijaan (Laumanns, Zitzler ja Thiele 2001).

6 Yhteenveto

Tässä tutkielmassa selvitettiin evolutionaaristen monitavoiteoptimointialgoritmien toimintaa. Algoritmien päätavoite on löytää useita ratkaisuja monitavoiteoptimointiongelmalle kiinnittäen huomiota ratkaisujen optimaalisuuteen ja monipuolisuuteen. MOEA:illa on paljon yhteisiä piirteitä, kuten samojen parametrien ja geneettisten operaattoreiden käyttäminen. MOEA:t pystyvät löytämään useita monipuolisia ratkaisuja läheltä Pareto-optimaalista rintamaa jopa hankalille ongelmille. MOEA:ien tehokkuutta voi testata testiongelmien avulla ja niiden vertailu on mahdollista kunhan huomioi niiden erojen ja yhtäläisyyksien vaikutuksen tehokkuuteen.

Tutkielmassa muodostetaan yleiskuvaa MOEA:ista. Toisaalta paljon yksityiskohtia sivuutettiin, kuten geneettisten operaatioiden toteutusyksityiskohtia. Luvussa 4 esitellään kaksi suosittua ja yleiskäyttöistä algoritmia, mutta toki ne eivät anna koko kuvaa kaikista MOEA:ista.

Evolutionaarinen monitavoiteoptimointi on laaja tutkimusalue ja sitä voi tutkia monesta eri näkökulmasta. Päätöksenteon näkökulmasta päätöksentekijän rooli osana optimointiprosessia pitäisi pystyä huomioimaan algoritmien suunnittelussa. Ohjelmistokehittäjä taas haluaisi tarkemman kuvauksen geneettisten operaattoreiden toteutuksesta. Menetelmien teoreettinen suppeneminen on myös haastava tutkimusaihe. Tulevaisuus tuo mukanaan skaalautuvuushaasteet, kun halutaan mallintaa yhä suurempia ja suurempia ongelmia, joista selviytymiseen pitää kehittää tehokkaampia algoritmeja.

Lähteet

Bosman, Peter A. N., ja Edwin D. de Jong. 2005. “Exploiting Gradient Information in Numerical Multi–Objective Evolutionary Optimization”. Teoksessa *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation,* 755–762. GECCO ’05. Washington DC, USA: Association for Computing Machinery. ISBN: 1595930108. https://doi.org/10.1145/1068009.1068138.

Branke, Jürgen, Jurgen Branke, Kalyanmoy Deb, Kaisa Miettinen ja Roman Slowiński. 2008. *Multiobjective optimization: Interactive and evolutionary approaches.* Nide 5252. Springer Science & Business Media.

Coello Coello, C. A. 2006. “Evolutionary multi-objective optimization: a historical view of the field”. *IEEE Computational Intelligence Magazine* 1 (1): 28–36. ISSN: 1556-6048. doi:10.1109/MCI.2006.1597059.

Čuček, Lidija, Petar Sabev Varbanov, Jiří Jaromír Klemeš ja Zdravko Kravanja. 2012. “Total footprints-based multi-criteria optimisation of regional biomass energy supply chains”. Integration and Energy System Engineering, European Symposium on Computer-Aided Process Engineering 2011, *Energy* 44 (1): 135–145. ISSN: 0360-5442. doi:https://doi.org/10.1016/j.energy.2012.01.040.

Deb, K., L. Thiele, M. Laumanns ja E. Zitzler. 2002. “Scalable multi-objective optimization test problems”. Teoksessa *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No.02TH8600),* nide 1, 825–830 vol.1. doi:10.1109/CEC.2002.1007032.

Deb, Kalyanmoy, Amrit Pratap, Sameer Agarwal ja TAMT Meyarivan. 2002. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. *IEEE transactions on evolutionary computation* 6 (2): 182–197.

Johnston, Mark D., ja Mark E. Giuliano. 2011. “Multi-Objective Scheduling for Space Science Missions”. *Journal of Advanced Computational Intelligence and Intelligent Informatics* 15 (8): 1140–1148. doi:10.20965/jaciii.2011.p1140.

Laumanns, Marco, Eckart Zitzler ja Lothar Thiele. 2001. “On The Effects of Archiving, Elitism, and Density Based Selection in Evolutionary Multi-objective Optimization”. Teoksessa *Evolutionary Multi-Criterion Optimization,* 181–196. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-44719-1.

Liefooghe, Arnaud, Matthieu Basseur, Laetitia Jourdan ja El-Ghazali Talbi. 2007. “ParadisEO-MOEO: A Framework for Evolutionary Multi-objective Optimization”. Teoksessa *Evolutionary Multi-Criterion Optimization,* toimittanut Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu ja Tadahiko Murata, 386–400. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-70928-2.

Liu, H., L. Chen, K. Deb ja E. D. Goodman. 2017. “Investigating the Effect of Imbalance Between Convergence and Diversity in Evolutionary Multiobjective Algorithms”. *IEEE Transactions on Evolutionary Computation* 21 (3): 408–425.

Rudenko, Olga, ja Marc Schoenauer. 2004. “Dominance Based Crossover Operator for Evolutionary Multi-objective Algorithms”. Teoksessa *Parallel Problem Solving from Nature - PPSN VIII,* toimittanut Xin Yao, Edmund K. Burke, José A. Lozano, Jim Smith, Juan Julián Merelo-Guervós, John A. Bullinaria, Jonathan E. Rowe, Peter Tiňo, Ata Kabán ja Hans-Paul Schwefel, 812–821. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-30217-9.

Rudolph, G., ja A. Agapie. 2000. “Convergence properties of some multi-objective evolutionary algorithms”. Teoksessa *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512),* nide 2, 1010–1016 vol.2.

Ruiz, Ana B., Mariano Luque, Kaisa Miettinen ja Rubén Saborido. 2015. “An Interactive Evolutionary Multiobjective Optimization Method: Interactive WASF-GA”. Teoksessa *Evolutionary Multi-Criterion Optimization,* toimittanut António Gaspar-Cunha, Carlos Henggeler Antunes ja Coello, 249–263. Cham: Springer International Publishing. ISBN: 978-3-319-15892-1.

Sindhya, Karthik, Kalyanmoy Deb ja Kaisa Miettinen. 2011. “Improving convergence of evolutionary multi-objective optimization with local search: a concurrent-hybrid algorithm”. *Natural Computing* 10 (4): 1407–1430.

Wang, Y., Z. Cai, Y. Zhou ja W. Zeng. 2008. “An Adaptive Tradeoff Model for Constrained Evolutionary Optimization”. *IEEE Transactions on Evolutionary Computation* 12 (1): 80–92.

Yang, Shengxiang, Miqing Li, Xiaohui Liu ja Jinhua Zheng. 2013. “A grid-based evolutionary algorithm for many-objective optimization”. *IEEE Transactions on Evolutionary Computation* 17 (5): 721–736.

Zhang, Q., ja H. Li. 2007. “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition”. *IEEE Transactions on Evolutionary Computation* 11 (6): 712–731.

Zitzler, Eckart, Marco Laumanns ja Lothar Thiele. 2001. “SPEA2: Improving the strength Pareto evolutionary algorithm”. *TIK-report* 103.

Zitzler, Eckart, ja Lothar Thiele. 1998. “Multiobjective optimization using evolutionary algorithms — A comparative case study”. Teoksessa *Parallel Problem Solving from Nature — PPSN V,* toimittanut Agoston E. Eiben, Thomas Bäck, Marc Schoenauer ja Hans-Paul Schwefel, 292–301. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-540-49672-4.