Luonnon innoittamat metaheuristiikat

Daniel Rantala

Pro gradu -tutkielma

Kesäkuu 2023

Tarkastajat:

Prof. Marko Mäkelä

FT Stefan Emet

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

Daniel Rantala: Luonnon innoittamat metaheuristiikat
Pro gradu -tutkielma, 40 s., 14 liites.
Sovellettu Matematiikka
Kesäkuu 2023

Metaheuristiset menetelmät näyttäytyvät lupaavilta ratkaisumalleilta, joiden avulla voidaan nopeuttaa optimointia ja säästää resursseja. Luonto on monimuotoisuutensa ansiosta täynnä tehokkaita menetelmiä, jotka tarjoavat kattavan kirjon mahdollisuuksia optimointimenetelmien kehittämiseksi.

Tämän tutkielman tarkoitus on perehdyttää lukija kolmeen tunnettuun luonnon innoittamaan metaheuristiseen menetelmään: kiiltomatoalgoritmi, simuloitu jäähdytys ja geneettinen algoritmi. Kiiltomatoalgoritmi ja geneettinen algoritmi edustavat populaatiopohjaisia metaheuristiikoita ja simuloitu jäähdytys puolestaan on kehityskaaripohjainen metaheuristinen menetelmä. Menetelmien matemaattinen toimintaperiaate on vielä varsin keskeneräinen ja tämän tutkimuksen tavoitteena onkin luoda suomenkielinen pohja metaheuristiikoiden matemaattiseksi ymmärtämiseksi. Lisäksi tarkastellaan metaheuristiikoiden soveltuvuutta diskreettien optimointitehtävien ratkaisemiseen.

Metaheuristiikoiden nopea konvergoituminen kohti optimia näyttäytyy mahdollisuutena toimia ongelmaitsenäisenä ratkaisumenetelmänä, minkä takia metaheurististen menetelmien ja niiden sovellutusten tutkimus on ollut kiivasta. Lupaavista ominaisuuksistaan huolimatta metaheuristiikat eivät kuitenkaan toimi ongelmasta riippumatta parhaalla mahdollisella tavalla, vaikka ne siihen pyrkivätkin.

Asiasanat: Kiiltomatoalgoritmi, Simuloitu jäähdytys, Geneettinen algoritmi, Diskreetti optimointi, Luonnon innoittamat metaheuristiikat

Sisällys

1 Johdanto **1**

2 Matemaattinen optimointi **4**

3 Metaheuristiikat **6**

3.1 Metaheuristisen algoritmin rakenne 8

3.1.1 Hajauttaminen ja tehostaminen 9

3.1.2 Stokastisuus 10

4 Luonnon innoittamat metaheuristiikat **12**

4.1 Kiiltomatoalgoritmi 13

4.1.1 Toimintaperiaate 13

4.1.2 Menetelmän matemaattinen tarkastelu 14

4.1.3 Pseudokoodi 16

4.2 Simuloitu jäähdytys 18

4.2.1 Menetelmän matemaattinen tarkastelu 18

4.2.2 Pseudokoodi 21

4.3 Geneettinen algoritmi 22

4.3.1 Menetelmän matemaattinen tarkastelu 22

4.3.2 Pseudokoodi 27

5 Diskreetti optimointi **28**

5.1 Diskretisointimenetelmiä 28

5.1.1 Sigmoidaalinen funktio 29

5.1.2 Satunnaisavain 29

5.1.3 Pienimmän sijainnin arvo 30

5.2 Kauppamatkustajan ongelma 30

6 Metaheuristiikat ja kauppamatkustajan ongelma **32**

6.1 Kiiltomatoalgoritmin soveltaminen 32

6.2 Simuloidun jäähdytyksen soveltaminen 33

6.3 Geneettisen algoritmin soveltaminen 35

6.4 Tulokset 36

7 Yhteenveto **39**

Lähdeluettelo **41**

1 Johdanto

Heuristiikka sanana juontaa juurensa antiikin Kreikan kielen sanasta "eureka", joka tarkoittaa: "olen löytänyt" tai "olen selvittänyt". Sanalla on vahva assosiaatio antiikin Kreikan tiedemieheen Arkhimedekseen, joka sanoi kuuluisan "*eureka*!" -huudahduksen keksittyään nykyään Arkhimedeen lakina tunnetun nosteen fysikaalisen määritelmän. Matematiikan "heuristiikka" -sanalla tarkoitetaan yleisesti algoritmeja, joilla on myös tavoitteena "löytää" ratkaisu [1]. Heuristiikoilla on tapana olla ongelmakohtaisia menetelmiä, jotka pyrkivät yrityksen ja erehdyksen kautta löytyvään ratkaisuun [1, 2]. Heuristiikat perustuvat optimointitehtävän vahvaan tuntemukseen ja kokemuksen kautta saatavaan intuitioon, joiden avulla on tarkoituksena hyväksi käyttää tapauskohtaista tietoa tehtävän ratkaisusta tai siihen johtavasta lähtötilanteesta. Heuristiikoiden eräänä heikkoutena onkin niiden taipumus tyytyä lähtöpisteen läheltä löytyvään ratkaisuun. Tällöin menetelmän tuottamat tulokset ovat riippuvaisia valituista lähtöpisteistä [2].

Metaheuristiikat nimensä mukaisesti yhdistävät heuristisen ongelman ratkaisun ja tekevät siitä korkeatasoisemman menetelmän, "meta" -sanan mukaisesti [2, 3]. Korkeamman tason voidaan ajatella tarkoittavan sen soveltuvuutta ongelmasta itsenäiseksi ratkaisumenetelmäksi [3]. Metaheurististen menetelmien ymmärtämistä hankaloittaa jo lähtökohtaisesti metaheuristiikka-sanan ajan kuluessa kokema muutos sekä selkeän ja sovitun yksiselitteisen määritelmän puutteellisuus. Nimittäin metaheuristiikan voidaan ajatella tarkoittavan yksittäistä menetelmää tai menetelmien yhdistelmää, jonka tarkoitus on optimoida annettu tehtävä. Toisin sanoen voidaan ajatella metaheuristiikoiden tarkoittavan metaheuristista algoritmia. Toisaalta metaheuristiikoiden voidaan ajatella olevan nimenomaan ohjelmistokehys, jonka ympärille rakennetut menetelmät ja niiden yhdistelmät luokitellaan metaheuristiikoiksi. On siis tärkeää ymmärtää, että metaheuristiikalla voidaan tarkoittaa metaheuristista algoritmia tai metaheuristista ohjelmistokehystä. [4]

Metaheuristiikat ovat siis optimointimenetelmien kirjo, jonka avulla on tarkoitus löytää ongelmaan riittävän tarkka ratkaisu tai esimerkiksi teollisessa tuotannossa aiempaa parempi tuotantomalli. Metaheuristiikat eivät takaa optimaalisen ratkai-

sun löytämistä, eivätkä tutki hakuavaruutta kokonaisuudessaan. Mikäli metaheuristiikat eivät takaa optimaalista ratkaisua annettuun ongelmaan, herää kysymys niiden hyödyllisyydestä ja tarpeellisuudesta. Metaheuristiikoiden käyttäminen on kuitenkin varsin mielekästä, jos tehtävän optimaalisen ratkaisun löytäminen kuluttaa poikkeuksellisen paljon aikaa tai tarkka laskeminen osoittautuu mahdottomaksi. Tämänkaltainen tilanne tulee käytännössä vastaan, kun mahdollisten ratkaisujen joukko kasvaa valtavan suureksi ja kaikkien ratkaisujen läpikäyminen yksitellen ei ole enää ajallisesti kannattavaa tai edes mahdollista [5]. Käyttämällä metaheuristisia menetelmiä ongelman optimoimisessa on kuitenkin hyötynsä ja haittansa. Tehtävää ratkaistaessa nopean vastauksen saamiseksi on tehtävä uhrauksia: vastauksen tarkkuudessa (accuracy), täsmällisyydessä (precision) ja täydellisyydessä (completeness). Tilanteissa, joissa tarkan globaalisti optimaalisen vastauksen sijaan riittää tyytyä approksimaatioon, on oikea aika harkita metaheuristiikoiden hyödyntämistä. [6]

Matemaattista optimointia tehtäessä onkin siis oleellista pohtia muutakin kuin pelkkää ongelmanratkaisua. Keskeistä on esimerkiksi pohtia optimointiin asetettavia rajoitteita ja miten optimoinnin avulla saatuja tuloksia aiotaan hyödyntää. Toisaalta ongelmasta riippuen myös käytössä olevat resurssit, kuten aika, rahoitus, työvoima ja laskentateho voivat olla rajoittavia tekijöitä. Tavoitteista riippuen voidaan optimoinnissa pyrkiä absoluuttisesti parhaaseen lopputulokseen, joka tunnetaan globaalina optimina. Aina paras eli optimaalisin tulos ei kuitenkaan ole tavoittelemisen arvoinen. Rajoitukset asettavat mielenkiinnon vaihtoehtoisille optimointitavoitteille, jotka nimenomaisesti ovat ongelmakohtaisesti "riittävän hyviä" ratkaisuja. Ratkaisujen saavuttamiseksi käytettävän algoritmin täytyy soveltua tilanteeseen, missä esiintyy epätarkkuutta, epävarmuutta, osittaisia totuuksia ja arviointia. Edellä mainittuja rasitteita kestäviä algoritmeja kutsutaankin tietojenkäsittelytieteissä pehmolaskennaksi (eng. soft computing), joihin myös metaheuristiikat lukeutuvat. [5]

Luonto tarjoaa monimuotoisuudellaan valtavan kirjon erilaisia toimintamalleja ja menetelmiä, joista on otettu mallia metaheuristisia algoritmeja suunnitellessa. Luonnon innoittamat metaheuristiikat ovatkin siis eräs tunnettu alalaji, jossa luonnossa esiintyviä käytösmalleja, populaatioita, saalistustrategioita tai evoluutiota hyödyn-

täen on formalisoitu optimointiongelmiin yleisesti soveltuvia ratkaisualgoritmeja. Tutkielmassa perehdytään metaheuristiikoiden määrittelyyn ja niiden luokitteluun matemaattisessa optimoinnissa. Perehtymisen ohella tarkastellaan fundamentaalisesti erilaisia luonnon innoittamia metaheuristiikoita. Pohditaan luonnon innoittamien metaheuristiikoiden soveltuvuutta optimointiin. Erityisesti tutkitaan luonnon innoittamien metaheuristiikoiden soveltuvuutta maineikkaaseen kauppamatkustajan ongelmaan. Tutkielman tavoitteena on luoda lukijalle kattavat lähtökohdat metaheuristiikoiden epäjohdonmukaisen tutkimuskentän ymmärtämiselle. Lisäksi pohditaan kriittisesti metaheuristiikoiden heikkouksia ja vahvuuksia.

2 Matemaattinen optimointi

Matemaattisessa optimoinnissa tavoitteena on löytää paras mahdollinen lopputulos lukuisten mahdollisten vaihtoehtojen joukosta. Optimointi onkin varsin tärkeä aspekti lähes kaikenlaisessa päätöksenteossa, aina yhteiskunnassa elävien yksilöiden ja siellä toimivien organisaatioiden eri osa-alueissa. Yksilöt ja organisaatiot pyrkivät päivittäin minimoimaan esimerkiksi kustannuksia, hävikkiä ja energian kulutusta. Vastaavasti maksimoinnista ollaan kiinnostuneita, kun puhutaan tuottavuudesta, tuotannosta tai suorituskyvystä. Ei siis ole liioiteltua todeta, että optimointia tarvitaan lähes kaikkialla. Optimoinnista välttämätöntä tekee jatkuva pula resursseista, kuten ajasta, rahasta ja luonnonvaroista, jotka ovat käytännössä jatkuvasti rajoittavia tekijöitä. [7]

Optimointi täytyy ensin muuttaa matemaattiseksi tehtäväksi, jotta sen tarkastelu on mahdollista. Tätä varten on muodostettava käsiteet, joiden varaan optimointi rakentuu. Määritelläänkin seuraavaksi optimoinnin kannalta keskeiset käsitteet, jotka luovat pohjan matemaattiselle optimoinnille. Olkoon funktio $f : \mathbb{R}^n \to \mathbb{R}$. Optimoinnissa on tarkoitus minimoida funktiota f, eli ratkaista tehtävä:

$$\begin{aligned} &min \quad f(\boldsymbol{x}) \\ &s.t. \quad \boldsymbol{x} \in S. \end{aligned}$$

Tavoitteena on siis löytää paras mahdollinen piste $\boldsymbol{x} \in S$, missä S on sallittujen pisteiden joukko. Parhaalla pisteellä tarkoitetaan pistettä, jolla funktion arvo on mahdollisimman pieni. Mikäli on olemassa sellainen $\boldsymbol{x}^* \in S$, siten että

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \qquad \text{kaikilla } \boldsymbol{x} \in S$$

sanotaan, että $\boldsymbol{x}^*$ on tehtävän *optimiratkaisu* tai *globaali minimi*. Toisaalta, jos on olemassa piste $\boldsymbol{x}^* \in S$ ja $\varepsilon > 0$ siten, että

$$f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}), \qquad \text{kaikilla } \boldsymbol{x} \in S \;\; \text{joilla} \;\; ||\boldsymbol{x} - \boldsymbol{x}^*|| \leq \varepsilon,$$

niin $\boldsymbol{x}^*$ on tehtävän *lokaali optimi* tai *lokaali minimi*. [8] Metaheuristisesta algoritmista riippumatta on aiheellista määritellä algoritmin tutkima alue. Nimittäin

se ei ole yksiselitteinen silloin, kun ei ole mahdollista käydä läpi kaikkia sallittuja pisteitä joukossa S. Metaheuristisessa optimoinnissa algoritmin läpikäymät alkiot optimaalisuutta etsiessä muodostavat *hakuavaruuden* A. Jos globaali optimi kuuluu joukkoon A, sanotaan hakuavaruuden olevan *globaali hakuavaruus*. Tällöin A on kompakti joukko, jolle pätee

$$A = \{\boldsymbol{x} \mid \boldsymbol{x_l} \leq \boldsymbol{x} \leq \boldsymbol{x_u}\},$$

missä $\boldsymbol{x_l}, \boldsymbol{x_u}$ ovat eksplisiittiset ja äärelliset ylä- ja alarajat muuttujalle $\boldsymbol{x}$. Metaheuristisessa tavoiteoptimoinnissa, missä $f : \mathbb{R}^n \to \mathbb{R}$ voidaan tehtävä muuntaa muotoon:

$$min\ f(\boldsymbol{x})$$
$$s.t.\ \boldsymbol{x} \in A.$$

Mikäli globaali optimi $\boldsymbol{x^*} \notin A$, metaheuristinen algoritmi on *lokaalisti optimoiva algoritmi*.[9] Metaheuristiikoiden hakuavaruudesta puhuttaessa ei siis ole mahdollista tietää, onko metaheuristisen algoritmin peittämä osuus onnistunut sisällyttämään globaalin optimin vai ei. Metaheuristiset menetelmät eivät ole deterministisiä, jolloin ne eivät takaa ratkaisun olevan globaalisti optimi. Siitä huolimatta niitä voidaan käyttää globaaleina optimointimenetelminä.[2] Tällöin voidaan antaa metaheuristiikalle todennäköisyys

$$P(\boldsymbol{x^*}) > 0,\ \text{jos tiedetään, että}\ \boldsymbol{x^*} \in A,\ \text{muutoin}\ P(\boldsymbol{x^*}) \geq 0,$$

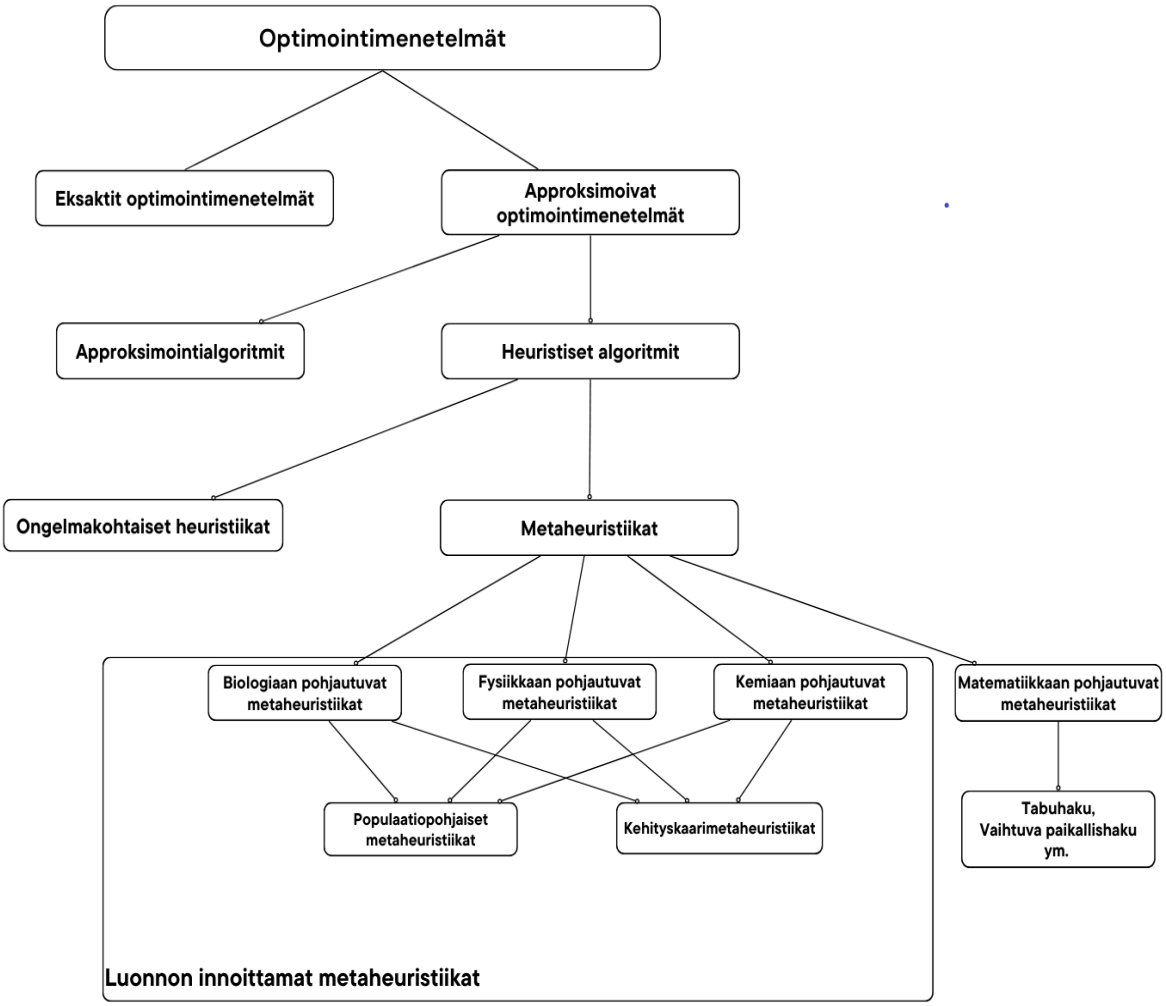
missä P tarkoittaa metaheuristisen algoritmin todennäköisyyttä löytää globaali optimi $\boldsymbol{x^*}$.

3 Metaheuristiikat

Metaheuristiikat tarjoavat laajan kirjon erilaisten strategioiden innoittamana toteutettuja optimointialgoritmeja. Selkeyden kannalta on syytä määritellä, mitä metaheuristiikalla tarkoitetaan ja miten ne voidaan luokitella erilaisten optimointimenetelmien kirjossa. Ymmärtämisen helpottamiseksi jaetaan optimointimenetelmät kahteen keskeiseen pääluokkaan, jotka ovat eksaktit eli deterministiset menetelmät ja approksimoivat eli stokastiset menetelmät [10]. Deterministiset menetelmät ovat tarkkoja menetelmiä, joiden tuloksille voidaan matemaattisesti todistaa saavutettu optimaalisuus. Stokastiset menetelmät, joihin metaheuristiikat kuuluvat, ovat puolestaan satunnaisuutta sisältäviä menetelmiä. Lisäksi stokastisten menetelmien saavuttama optimi ei ole yksiselitteinen tai taattu globaali optimi. Algoritmille annettavat parametrit määrittelevät metaheuristiikan suorituskykyä ja tuloksena saatavan vastauksen laatua optimointitehtävässä. Lisäksi parametrien valinta on ongelmallista, koska siihen ei ole olemassa yksiselitteistä parasta mahdollista tapaa [2]. Nimittäin metaheuristiikat toimivat ikään kuin mustina laatikkoina, joille määritellään parametrit ja tämän jälkeen ne tuottavat parhaimman approksimaation vastaukseksi, joka saattaa olla tai ei ole optimaalinen ratkaisu. Tuloksien tarkka matemaattinen todistaminen, että onko algoritmin tuottaman vastaus globaali- tai lokaali optimi yleisessä tapauksessa on toistaiseksi mahdotonta. Tämä johtuu siitä, että algoritmin toteuttaman optimoinnin matemaattinen analysointi on vielä nykytutkimuksen puitteissa puutteellista. [2]

Johdannossa jo ongelmalliseksi havaittu metaheuristiikka-sana tulee selkeyden vuoksi tutkielmassa käsittämään sekä algoritmisen määritelmän että ohjelmistokehityksellisen määritelmän. Näin ollen metaheuristiikoihin voidaan asiayhteydestä riippuen viitata algoritmina tai ohjelmistokehyksenä. Hyvän käsityksen siitä, miten metaheuristiikka-sana määritellään saadaan Sörensen ja Gloverin avulla.[3]

Määritelmä 1. Metaheuristiikka on yleisesti korkeatasoisen ongelmaitsenäisen algoritmin ohjelmointikehys, joka antaa suosituksia ja strategioita heuristisen optimointialgoritmin muodostamiseksi. Toisinaan yksittäisiä heuristisia algoritmejakin voidaan kutsua metaheuristiikoiksi. [3]



Kuva 1: Yhdistetty luokittelu metaheuristiikoiden sijainnista optimointimenetelmien kirjossa [6, 10, 11, 12, 13].

Metaheurististen menetelmien keskenäinen erottelu perustuu karkeasti algoritmin samanaikaisesti käsittelemien ratkaisujen lukumäärään. Kirjallisuudessa esiintyy paljon ristiriitaisia näkemyksiä siitä, miten luonnon innoittamat metaheuristiikat tulisi luokitella. Kuvassa 1 on yhdistetty toisistaan eroavia näkemyksiä metaheuristiikoiden luokittelussa ja pyritty muodostamaan mahdollisimman kattava käsitys menetelmien taksonomisesta luokittelusta. [6, 10, 11, 12, 13] Luokittelun sisältämä ristiriitaisuus vaikeuttaa metaheuristiikoiden määrittelyä ja aiheuttaa turhaa monimutkaisuutta niiden tutkimisessa [14].

Määritelmän 1 ohjelmointikehys antaa varsin monipuolisen tulkinnan metaheu-

ristiikan käsitteelle kuvastaen varsin osuvasti sitä, miten laaja yläkäsite metaheuristiikka itsessään on. Kuvasta 1 saadaan parempi käsitys siitä, miten laaja kirjo erilaisia metaheuristiikkoihin pohjautuvia optimointimenetelmien luokkia on olemassa. Karkeasti metaheuristiikat voidaan luokitella kehityskaarimetaheuristiikkoihin (eng. trajectory based metaheuristics) ja populaatiopohjaisiin metaheuristiikkoihin (eng. population based metaheuristics). Toisinaan kehityskaarimetaheuristiikoista puhutaan yksittäisratkaisumetaheuristiikkoina (eng. single solution metaheuristics). Metaheuristiikkojen alaluokat pitävät sisällään erikokoisia määriä eri metaforien pohjalta kehitettyjä menetelmiä. Kyse on siitä, mikä ilmiö on toiminut metaheuristisen menetelmän innoittajana. [5] Metaheuristiikat kärsivät puutteellisesta ja yleisesti hyväksytystä taksonomiasta. Tämä ilmenee tutkimuksien valtavasta tarpeesta muodostaa sellainen. Kilpailu voittavasta taksonomisesta määritelmästä kuitenkin esiintyy tutkimuksissa lähinnä vain keskenään riitelevinä ja puuttellisina määritelminä, kuten voidaan jo pelkästään tutkimuksissa [6, 11, 13, 15] olevista ristiriitaisista määrittelyistä ja ehdotuksista todeta.

3.1 Metaheuristisen algoritmin rakenne

Metaheurististen algoritmien rakenteissa on muutamia mainitsemisen arvoisia eroja, joiden avulla niiden erottelu helpottuu. Kehityskaarimetaheuristiikat nimensä mukaisesti ovat metaheuristisia algoritmeja, jotka käsittelevät yhtä iteroituvaa, eli "kehittyvää" ratkaisua. Yksinkertaistettuna algoritmilla on yksittäinen ratkaisu annettuun tehtävään, jota algoritmi pyrkii iteroiden päivittämään paremmaksi. Tunnetuimpia edellä mainittuun suunnittelufilosofiaan pohjautuvia algoritmeja ovat tabuhaku (Tabu search), simuloitu jäähdytys (simulated annealing), iteroitu paikallishaku (iterated local search) ja vaihtuva paikallishaku (variable neighbourhood search) [12]. Populaatiopohjaiset metaheuristiikat puolestaan pitävät sisällään useita keskenään kilpailevia ratkaisuja. Optimaalinen ratkaisukandidaattien määrä vaihtelee algoritmista ja optimoitavasta tehtävästä riippuen. Tätä ratkaisujen joukkoa, joka vaiheittain kehittyy algoritmin edetessä voidaan kutsua populaatioiksi. Populaatiopohjaisissa algoritmeissa voidaan ajatella mahdollisten ratkaisualkioiden tai -vektorien kilpailevan keskenään iteraatiosta toiseen. Lopuksi paras ratkaisu popu-

laation joukosta valitaan algoritmin tuottamaksi ratkaisuksi.[2, 12]

Hakuavaruus on reaaliavaruuden osa, joka toteuttaa optimointitehtävässä määritellyt rajoitteet. Metaheuristiset algoritmit tutkivat optimointia tehtäessä hakuavaruutta ja yrittävät löytää sieltä parhaimman mahdollisen ratkaisun. Optimointiongelmien monimutkaisesta luonteesta johtuen optimoitava hakuavaruus on yleensä äärimmäisen laaja, eikä laskentatehon puitteissa ole välttämättä mahdollista tutkia koko hakuavaruutta tai kaikkia mahdollisia vaihtoehtoja. Tällaisia ongelmia kutsutaan NP-täydellisiksi ongelmiksi [16] ja niissä on usein tyydyttävä hakuavaruuden osittaiseen tutkimiseen. Lisäksi ei aina ole matemaattisesti selvää, miten algoritmi on vastaukseen päätynyt tai miksi metaheuristinen menetelmä on toiminut. [17]

3.1.1 Hajauttaminen ja tehostaminen

Metaheuristisen menetelmien laajasta kirjosta huolimatta kaikki menetelmät hyödyntävät kahta keskeistä strategiaa hakuavaruuden tutkimisessa. Ensimmäisenä näistä määritellään hajauttaminen.

Määritelmä 2. Hajauttaminen (eng. diversification) on metaheuristisen algoritmien strategia, jonka tavoitteena on löytää hakuavaruudesta uusia potentiaalisia ratkaisuja sisältäviä alueita. Toisin sanoen kyseessä on hakuavaruuden sisällä tapahtuvaa globaalin optimin sisältävän alueen etsimistä. [17, 18]

Tarkasteltaessa NP-täydellisiä ongelmia on vaikea keksiä strategiaa, joka tehokkaasti tutkisi koko hakuavaruutta. Tällöin katseet kääntyvät hajauttamiseen, jonka tavoitteena on generoida uusia ratkaisukandidaatteja. Näiden ratkaisukandidaattien tulisi olla riittävän monipuolisia ja riittävän kaukana nykyisistä ratkaisuista. Toisin sanoen hajauttaminen on laajan hakuavaruuden globaalia optimointia, joka toteutetaan käytännössä satunnaisuuden avulla. Hajauttamisstrategian vahvuutena on tavoite globaalin optimin löytämisestä eli on epätodennäköistä, että hajauttamisprosessissa jumiuduttaisiin yksittäiseen lokaaliin optimiin. Hakuavaruuden ollessa äärimmäisen laaja voi hajauttamisen suppeneminen kohti globaalia optimia olla varsin hidasta. Lisäksi yksiselitteistä suppenemista kohti globaalia optimia ei voida taata, mikä voi helposti johtaa hukattuun laskentatehoon. Tavoitteena on välttää algortimin muodostamien ratkaisupopulaatioiden jumittumista yksittäiseen lokaaliin

optimiin. Globaalin optimoinnin kannalta hajauttaminen on varsin oleellista, sillä todennäköisyys löytää globaali optimi on kuitenkin aidosti nollaa suurempaa. [7]

Määritelmä 3. Tehostaminen (eng. intensification) on strategia, jossa paikallisesti lupaavaa hakuavaruuden osaa tutkitaan tarkasti. Näin saadaan parempi käsitys paikallisesti parhaasta ratkaisukandidaatista. [17, 19]

Tehostamisessa hyödynnetään tehtävästä saatavaa informaatioita uusien parempien ratkaisujen muodostamiseksi. Hajauttamisesta poiketen kyseessä on vahvasti lokaali strategia, joka suppenee varsin voimakkaasti. Selvänä heikkoutena on strategian voimakas taipumus jumittua lokaaliin optimiin. Tehostamista voi yksinkertaisestettuna ajatella gradienttipohjaisena menetelmänä, joka tyytyy läheltä löytyvään minimiin. [7]

Algoritmista riippuen hakuavaruutta voidaan tutkia hajauttamalla, tehostamalla tai molempien strategioiden yhdistelmänä. Yleisesti metaheuristiset algoritmit ovat yhdistelmä hajauttamista ja tehostamista. Keskeisenä haasteena on löytää sopiva suhde algoritmin tekemässä hajauttamisessa ja tehostamisessa, sillä nämä kaksi strategiaa ovat tärkeässä roolissa algoritmin tuottamien ratkaisujen ja tehokkuuden kanssa. Algoritmien sisällä tapahtuva taspainottaminen on vielä avoin ongelma metaheurististen algoritmien suunnittelussa ja toteuttamisessa. [7, 17]

3.1.2 Stokastisuus

Hajauttaminen ja tehostaminen strategioina pohjautuvat molemmat satunnaisuuteen. On olennaista tutustua metaheuristisissa algoritmeissa esiintyvään satunnaisuuten ja siihen, millä tavoin käytettävä satunnaisuus on saavutettu. [7]

Määritelmä 4. Pseudosatunnaisuus on matemaattisesti tuotettua satunnaisuutta, jonka avulla voidaan saavuttaa tilastollisesta näkökulmasta satunnaisia tuloksia. Pseudosatunnaisuus on kuitenkin matemaattisesti deterministinen. Tarkoittaen pseudosatunnaisuuden olevan toistettava prosessi.

Täydellisen satunnaisuuden saavuttamisen ollessa mahdotonta on aiheellista tyytyä pseudosatunnaisuuteen, Lévy-jakauman (Lévy distribution) ja normaali jakau-

man (normal distribution) ollessa esimerkkejä mahdollisesti käytettävistä todennäköisyysjakaumista. Muita kuuluisia sovellettavia todennäköisyysteorian tuloksia ovat satunnaiskulku (random walk), Markovin ketju (Markov chain) ja Lévy-kulku (Lévy flight), joita käytetään satunnaisuuden tuottamisessa [7].

4 Luonnon innoittamat metaheuristiikat

Luonnon innoittamat metaheuristiikat ovat saaneet nimensä luonnossa esiintyvistä ilmiöistä, joita ne pyrkivät mallintamaan. Keskeisenä ongelmana luonnon innoittamissa metaheuristiikoissa on uusien menetelmien runsaudenpula. Syynä äkillisesti kasvaneeseen metaheurististen algoritmien määrän on metaheuristiikka sanan kiistanalainen tulkinta. Aiemmin todettussa määritelmässä 1 esiintyy ensimmäinen ristiriita siitä, miten metaheuristiikat voidaan nähdä algoritmeina, tai strategioina ja ohjelmistokehyksenä [20]. Näin ollen luonnon innoittamat metaheuristiikat voidaan virheellisesti luokitella uusiksi metaheuristisiksi strategioiksi, vaikka parempi tapa kuvata luonnon innoittamia metaheuristiikoita olisi todeta niiden olevan jo olemassa olevien metaheuristisiten strategioiden uusia yhdistelmiä [20]. Luonnon innoittamilla metaheuristisilla algoritmeillä onkin samat ominaisuudet, kuten aiemmin metaheuristiikoille määriteltiin. Näin ollen voidaan todeta niiden olevan hajauttamisen, tehostamisen ja satunnaisuuden yhdistelmiä. Metaheuristisiten luokittelujen yhteydessä todettin, miten luonnon innoittamien metaheuristiikoiden määrittelystä ei ole selkeää tai yksiselitteistä käsitystä.

Keskeisen ristiriidan luonnon innoittamien metaheuristiikkojen nimeämisessä aiheuttaa se, miten luonto tulisi määritellä. Nimittäin määritelmästä riippuen miltei kaikki metaheuristiikat voivat kuulua luonnon innoittamiin metaheuristiikkoihin. Siitä huolimatta termi on varsin laajassa käytössä, eikä sen sivuuttaminen enää nykymääritelmien kirjossa olisi mielekästä. Paras käsitys luonnon innoittamista metaheuristiikoista saadaan ajattelemalla ne metaheuristiikka sanan jatkeeksi ja osaksi käsitteen määritelmää. Luonnon innoittamat metaheuristiikat ovat saanet myös osakseen vahvaa kritiikkiä siitä, miten niiden todellisesti pienet tai jopa merkityksettömät innovaatiot kätketään kekseliäiden metaforien taakse. Tästä syystä luonnon innoittamien metaheuristiikoiden tutkimus on vaikeaa, sillä yksittäisen menetelmän ymmärtäminen vie paljon aikaa ja uusien menetelmien kehittämisestä on valtavaa kilpailua. Eräs alan tutkija on todennut uusien luonnon innoittamien metaheuristiikoiden määrän olevan niin suuri, että se uhkaa jo koko metaheuristisen tutkimuksen tieteellistä eheyttä [14]. Toteamus on varsin ankara tutkimuskenttää kohtaan, mutta

uusien menetelmien lukumäärän lisääntyessä päivä päivältä entistäkin aiheellisempi. Tutustaan seuraavaksi yksityiskohtaisesti kolmeen filosofisesti vahvasti toisistaan eroavaan metaheuristiseen algoritmiin ja niiden toimintaperiaatteisiin.

4.1 Kiiltomatoalgoritmi

Kiiltomatoalgoritmi (Firefly algorithm) on metaheuristiikoiden saralla ansioituneen tutkijan Xin-She Yangin 2000-luvulla kehittämä metaheuristinen algoritmi [1, 2, 7]. Algoritmi matkii tropiikissa esiintyvien kovakuoriaisten eli kiiltomatojen käytöstä. Kiiltomadot kykenevät synnyttämään biokemiallisen prosessin avulla valoa. Prosessi tunnetaan nimeltä ja sitä kutsutaan bioluminesenssiksi. Kiiltomadon uskotaan käyttävän valoa esimerkiksi kumppanin tai saaliin houkuttelemiseen. Lisäksi on mahdollista, että ne varoittavat valolla saalistajiaan kiiltomatojen kitkerästä mausta. Tämä käytös toimii kiiltomatoalgoritmin filosofisena pohjana. Algoritmi on esimerkki populaatiopohjaisesta metaheuristiikasta, jossa kiiltomatoparven yksilöiden lokaali käyttäytyminen ohjaa koko populaation toimintaa. [21]

4.1.1 Toimintaperiaate

Algoritmin kiiltomadot ovat oikeasti optimointialgoritmin tuottamia yksittäisratkaisuja, jotka kilpailevat keskenään parhaasta ratkaisusta. Kiiltomatoalgoritmin toiminta perustuu yksittäisen kiiltomadon tuottaman bioluminesenssin voimakkuuden puoleensavetävyyden vaihtelun mallintamiseen. Kiiltomatoalgoritmin toimintaehdot ovat hieman yksinkertaistetumpia siitä, miten kiiltomadot käyttäytyvät luonnossa. Ensiksikin puoleensavetävyys nähdään samana tuli se sitten miltä kiiltomadolta tahansa, jolloin ei eritellä onko signaalin lähettänyt uros vai naaras. Toiseksi, vähemmän kirkas kiiltomato liikkuu aina kirkkaampaa kiiltomatoa kohti ja jos ympärillä ei ole kirkkaampia kiiltomatoja, yksilön liike tulkitaan sattumanvaraiseksi. Kolmanneksi, optimoitavan kohdefunktion pinnanmuodot vaikuttavat kiiltomadon kirkkauteen. Toisin sanoen kiiltomatoparven yksilöt, jotka ovat löytäneet potentiaalisen optimin loistavat kirkkaammin houkutellen muita yksilöitä luokseen.

Bioluminesenssin tuottama valo ajatellaan algoritmin kannalta varsin fysikaalisesti. Valon intensiteetin I voimakkuus heikkenee, kun etäisyys r kasvaa, verran-

nollisuutta merkitään $I \propto \frac{1}{r^2}$ mukaisesti. On oleellista todeta, että kirkkaampi valo houkuttelee tehokkaamin muita yksilöitä luokseen. Toisaalta kauempana oleva optimaalisempi kiiltomato ei välttämättä ole houkuttelevampi, kuin lähempänä oleva vähemmän optimaalinen kiiltomato. Tämä johtuu siitä, miten etäisyys vaikuttaa havaitsevana olevan kiiltomadon näkemään kirkkauteen. Lisäksi on tärkeä muistaa, että bioluminesenssin tuottama valo ei ole jatkuvaa, vaan kiiltomadot toimivat ikään kuin oskillaattoreina. Näin ollen tuotettu valo ja sen aiheuttama puoleensavetävyys on jaksollista. [1]

4.1.2 Menetelmän matemaattinen tarkastelu

Kiiltomatoalgoritmin kannalta valon intensiteetti, valon puoleensavetävyys ja satunnaisuuden hallinta ovat keskeisiä käsitteitä algoritmin toiminnassa. Käytännössä kiiltomadoilla on kirkkaus K, joka riippuu kiiltomadon sijainnista $\boldsymbol{x_i}$ optimoitavassa ympäristössä. Kiiltomadon kirkkaus ja kohdefunktion arvot ovat verrannollisia toisiinsa nähden. Esimerkiksi minimointitehtävässä yleensä valitaan, että $K(\boldsymbol{x_i}) \propto \frac{1}{1+f(\boldsymbol{x_i})}$, mikä tarkoittaa kiiltomadon kirkkauden olevan kääntäen verrannollinen optimoitavan kohdefunktion arvoon nähden. Tässä työssä kuitenkin tarkastellaan maksimointitehtävän tapausta, jossa valitaan kirkkauden ja kohdefunktion olevan suoraan verrannollisia toisiinsa nähden $K(\boldsymbol{x_i}) \propto f(\boldsymbol{x_i})$. Kirkkauden ja kohdefunktion suoraan verrannollisuuden tarkoituksena on, että kirkkaammin loistavan kiiltomadon sijainnissa kohdefunktion arvo on suurempi. Tällöin kirkkaamman kiiltomadon kyky vetää puoleensa muita himmeämpiä kiiltomatoja on korkeampi. Kirkkaus on kuitenkin subjektiivinen määre, eikä se kuvaa valon voimakkuuden absoluuttista määrää. Tällöin valolle on määriteltävä fysikaalisesti mitattava suure, minkä takia algoritmissä käytettävä kirkkauden K on määriteltävä etäisyydestä r riippuvana valon intensiteettinä I, joka on kvantitatiivinen suure. Valon intensiteetti väliaineelle on

$$I(r) = K_0 e^{-\gamma r^2}, \tag{1}$$

missä K_0 on alkuperäinen valon kirkkaus, eli kohdefunktion arvo pisteessä x_i etäisyydellä $r = 0$ ja γ on valitusta väliainesta riippuva absorptiokerroin. Valon intensiteetti siis riippuu kohdefunktion arvosta, valitusta väliaineesta ja etäisyydestä r. Kiiltomadon valon intensiteetti ei kuitenkaan kerro, miten valon intensiteetti näkyy toisen kiiltomadon perspektiivissä. Tätä varten on muodostettava kiiltomadoille puoleensavetävyys β, joka esittää funktio

$$\beta(r) = \beta_0 e^{-\gamma r^2}, \tag{2}$$

missä β_0 on puoleensavetävyys etäisyydellä $r = 0$ ja γ on absorptiokerroin. Tarvittaessa laskentatehokkuutta voidaan parantaa approksimoimalla

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2}. \tag{3}$$

On tilannekohtaista kumpaa funktioista (2) tai (3) kannattaa soveltaa. Kaavan (2) puoleensavetävyys kuvaa valon intensiteettiä havaitsevan kiiltomadon sijainnissa. Puoleensavetävyys β on siis suhteellinen arvo, sillä β määräytyy sen mukaan, missä valon havaitseva kiiltomato sijaitsee. Puoleensavetävyys heikkenee etäisyyden r kasvaessa, koska siihen vaikuttaa valon absorptio väliaineessa. Kahden kiiltomadon i ja j välinen etäisyys pisteissä $\boldsymbol{x_i}$ ja $\boldsymbol{x_j}$ on vastaavasti karteesinen etäisyys

$$r_{ij} = ||\boldsymbol{x_i} - \boldsymbol{x_j}|| = \sqrt{\sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2}, \tag{4}$$

missä $x_{i,k}$ on kiiltomadon $\boldsymbol{x_i}$ sijainnin komponentti k kiiltomadolle i. Kaksidimensioisessa tapauksessa:

$$r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{5}$$

Kiiltomadon i liike kohti puoleensavetävämpää, eli kirkkaampaa kiiltomatoa j määräytyy seuraavasti

$$\boldsymbol{x_i}^{t+1} = \boldsymbol{x_i}^t + \beta_0 e^{-\gamma r_{ij}^2} (\boldsymbol{x_j}^t - \boldsymbol{x_i}^t) + \alpha_t \boldsymbol{\epsilon}_i^t, \tag{6}$$

missä toinen termi johtuu puoleensavetävyydestä. Kolmas termi toimii satunnaistajana, jossa α_t on askelpituus ja $\boldsymbol{\epsilon_i}$ on satunnaislukuvektori, joka saadaan normaalijakaumasta satunnaisotantana.

Viimeisenä käsiteltävänä asiana on algoritmin satunnaisuus. Kiiltomatoalgoritmin tarkkuutta voidaan parantaa vaihtelemalla askelpituutta α_t. Tarkoituksena on saada α pienenemään optimiratkaisua lähestyttäessä. Näin ollen α_t voidaan päivittää esimerkiksi kaavalla

$$\alpha_t = \alpha_\infty + (\alpha_0 - \alpha_\infty)e^{-t}, \tag{7}$$

missä $t \in [0, t_{max}]$ on iteraatiolaskuri, t_{max} on iteraatioiden enimmäismäärä, α_0 on alkuperäinen askelpituus ja α_∞ sen lopullinen arvo. Vaihtoehtoisesti askelpituutta voidaan päivittää myös kaavalla:

$$\alpha_t = \alpha_0\theta^t, \tag{8}$$

missä $\theta \in (0, 1]$ on satunnaisuuden vähenemisnopeutta kuvaava vakio. Suurimmassa osassa tapauksia voidaan käyttää arvoja $0,95 \sim 0,99$ ja $\alpha_0 = 1$.

Kiiltomatoalgoritmin esitetyssä pseudokoodissa iteraatiokohtaista globaalia optimia $\boldsymbol{x^*}$ hyödynnetään ainoastaan lopullisen tuloksen muodossa. Algoritmin kehittäjän mukaan [7] sitä voidaan hyödyntää myös algoritmin tehostamisessa käyttämällä sitä kaavassa (6) ylimääräisen termin $\lambda_i\boldsymbol{\epsilon}_i^t(\boldsymbol{x^*} - \boldsymbol{x}_i^t)$ kanssa, missä λ on α:n ja β:n kaltainen parametri ja $\boldsymbol{\epsilon}_i^t$ on määritelty samalla tavalla kuin aikaisemmin. Huomionarvoista on todeta, että yhtälö (6) on satunnaiskulku, joka on painottunut kohti kirkkaampaa kiiltomatoa. Tapauksessa $\beta_0 = 0$ se käyttäytyy tavallisena satunnaiskulkuna. Lisäksi satunnaisuutta kuvaava termi voidaan laajentaa muihin satunnaisjakaumiin, kuten Lévy kulkuun. [7]

4.1.3 Pseudokoodi

Esitellään seuraavaksi kiiltomatoalgoritmin pseudokoodi, jotta voidaan ymmärtää paremmin algoritmin toimintaperiaatteita. Pseudokoodi ei ole tarkoitettu minkään ohjelmointikielen käännettäväksi, vaan algoritmisen toiminnan tarkastelemiseksi.

Kiiltomatoalgoritmi[7, 22]

Olkoon kohdefunktio $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ ja $\boldsymbol{x} \in S$.
Luodaan satunnaisesti [1] kiiltomatopopulaatio, joka koostuu d kappaleesta kiiltomatoja $\boldsymbol{x}_i^0 \ \ (i = 1, 2, \ldots, d)$.
Alustetaan kiiltomatojen valojen intensiteetit pisteissä $\boldsymbol{x_i}$ kaavan (1) mukaisesti.
Olkoon α_0 alkuperäinen askelpituus, α_{max} askelpituuden enimmäispituus, t_{max} iteraatioiden enimmäismäärä, β_0 on puoleensavetävyys etäisyydellä $r = 0$ ja $\theta \in (0, 1]$ satunnaisuuden vähenemisnopeutta kuvaava vakio.
Määritellään valon absorptiovakio γ (riippuu valittavasta väliaineesta) ja alustetaan iteraatiolaskuri $t = 0$.
while $(t < t_{max})$ **do**
  for $i = 1 : d$ (kaikki d kiiltomatoa) **do**
    for $j = 1 : d$ (kaikki d kiiltomatoa) **do**
      if $(\boldsymbol{x}_i, \boldsymbol{x}_j \in S)$ **then**
        Kiiltomato, jolla on pienempi valon intensiteetti I, liikkuu suuremman valon intensiteetin omaavaa kiiltomatoa kohti yhtälön (6) mukaisesti.
      else if $(\boldsymbol{x}_i \notin S$ ja $\boldsymbol{x}_j \in S)$ **or** $(\boldsymbol{x}_j \notin S$ ja $\boldsymbol{x}_i \in S)$ **then**
        Kiiltomato, joka ei kuulu sallittuun joukkoon S liikkuu sallitussa joukossa olevaa kiiltomatoa kohti yhtälön (6) mukaisesti.
      else if $(\boldsymbol{x}_i, \boldsymbol{x}_j \notin S)$ **then**
        Kiiltomato, joka toteuttaa vähemmän rajoitteita, liikkuu enemmän rajoitteita toteuttavaa kiiltomatoa kohti yhtälön (6) mukaisesti.
      end if
      if Liikkuneen kiiltomadon intensiteetti ei ole parempi kuin edellisessä paikassa **then**
        Liikkunut kiiltomato palaa takaisin vanhalle paikalleen.
      end if
      Korjaa yhtälön (2) valon puoleensavetävyys β kiiltomatojen i ja j välisen uuden etäisyyden r_{ij} avulla kaikille kiiltomadoille i.
      Tutkitaan kiiltomatojen uusia sijainteja ja päivitetään valon intensiteetti kaavan (1) mukaisesti.
    end for j
  end for i
  Vertaile kiiltomatoja keskenään ja etsi globaalisti paras yksilö $\boldsymbol{x}^*$, eli valon intensiteetiltään suurin yksilö.
  Päivitetään askelpituus kaavaa (8) käyttäen ja
  $t = t + 1$ (kasvatetaan iteraatiolaskuria t yhdellä).
end while
Kaikki algoritmin läpikäymät ratkaisut muodostavat hakuavaruuden A. Jälkiprosessoi tulokset ja muodosta visualisaatio tuloksesta.

4.2 Simuloitu jäähdytys

Simuloitu jäähdytys (simulated annealing) on kehitetty 1980-luvun alkupuolella ja on yksi vanhimmista tunnetuista luonnon innoittamista metaheuristiikoista. Algoritmi perustuu kiderakenteisen kiinteän aineen jäähtymisprosessin mallintamiseen. Tarkemmin sanottuna kiderakenteinen aine, esimerkiksi jokin metalli lämmitetään ja sen annetaan jäähtyä hitaasti. Hitaan jäähtymisprosessin aikana metallin hilarakenne muuntuu kaikista säännönmukaisimmaksi, tarkoittaen aineen saavuttavan pienimmän mahdollisen hilaenergian tilan. Vaatimuksena säännönmukaisen hilarakenteen saavuttamiseksi on, että jäähtymisprosessi ei tapahdu liian nopeasti. [7, 21] Algoritmi on pohjimmiltaan kehityskaarialgoritmi, jonka keskeisenä strategiana on vahva tehostaminen. Tehostamisen ollessa päästrategia on algoritmin suorituksessa suurempi riski jumittua yksittäiseen paikalliseen optimiratkaisuun. Simuloitu jäähdytys menetelmänä tiedostaa riskin lokaaliin optimiin jumittumisesta ja tämän välttämiseksi hyväksyy iteroinnissa ajoittain ratkaisua heikentäviä arvoja. Näin ollen algoritmi välttää lokaaliin optimiin jumittumista ja kykenee konvergoimaan kohti muita lupaavia ratkaisualueita. [7, 21]

4.2.1 Menetelmän matemaattinen tarkastelu

Simuloidun jäähdytyksen keskeisenä strategiana on käyttää satunnaishakua (eng. Random Search) Markovin ketjun tavoin. Keskeisenä näkökulmana tässä Markovin ketjun sovelluksessa on, että algoritmi hyväksyy parempien ratkaisujen lisäksi, myös huonompia ratkaisuvaihtoehtoja. Esimerkiksi minimointitehtävässä simuloitu jäähdytys hyväksyy kohdefunktiota pienentäviä arvoja, mutta myös todennäköisyydellä p algoritmi hyväksyy kohdefunktiota kasvattavia arvoja. Todennäköisyys valita optimoitavaa kohdefunktiota huonontavia arvoja määräytyy seuraavasti

$$p = e^{[-\frac{\triangle E}{k_b T}]}. \tag{9}$$

Kaavan (9) muuttuja k_b tarkoittaa Boltzmannin vakiota. Todennäköisyyden ymmärtämisen yksinkertaistamiseksi voidaan asettaa $k_b = 1$, sillä kyseessä on kuitenkin vain vakio. Muuttuja T puolestaan on lämpötila jolla hallitaan algoritmin tunnuso-

maiseksi piirteeksi kutsuttavaa jäähtymistä. Viimeisellä muuttujalla $\triangle E$ tarkoitetaan energiatasojen muutosta. Todennäköisyyttä p kutsutaan myös transitiotodennäköisyydeksi, joka pohjautuu Boltzmannin jakaumaan tilastollisessa mekaniikassa. Yksinkertaisin tapa liittää $\triangle E$ kohdefunktion muutokseen $\triangle f = f(\boldsymbol{x}_{t+1}) - f(\boldsymbol{x}_t)$ on kuvata energian muutosta

$$\triangle E = \gamma \triangle f. \tag{10}$$

Edellä mainitussa kaavassa γ on reaalinen vakio. Menettämättä tuloksien yleispätevyyttä voidaan edelleen valita $\gamma = 1$. Nyt voidaan redusoida todennäköisyys p muotoon:

$$p = e^{[-\frac{\triangle f}{T}]}. \tag{11}$$

Muutoksen hyväksymisessä käytetään apuna eräänlaista kynnysarvoa r, jolle pätee

$$p = e^{[-\frac{\triangle f}{T}]} > r. \tag{12}$$

Toisin sanoen jos $p > r$ niin transitio tapahtuu. Yleisesti r voidaan ottaa tasajakaumasta $[0, 1]$.

Algoritmin alussa valitaan alkulämpötila, joka on tärkein yksittäinen valittava parametri algoritmin toiminnan kannalta. Nimittäin tarkasteltavalle muutokselle $\triangle f$, jos lämpötila T on liian suuri ($T \to \infty$), niin tällöin $p \to 1$. Yhtälön (12) mukaan tässä tilanteessa se tarkoittaisi lähes poikkeuksetta kaikkien muutoksien hyväksymistä. Vastaavasti lämpötilan T ollessa liian pieni ($T \to 0$), jolloin $p \to 0$, mikä tahansa huonompi ratkaisu ($\triangle f > 0$) tulee hylätyksi. Nyt siis algoritmi hyväksyisi ainoastaan kohdefunktiota parantavia arvoja, jolloin riski päätyä yksittäiseen lokaaliin optimiin kasvaa. Tällöin algoritmi muuttuu strategialta ja toimintaperiaateeltaan täysin tehostavaksi, eikä siinä silloin tapahdu lainkaan ratkaisuavaruuden tutkimiseksi kutsuttavaa hajauttamista. Algoritmin toiminnan kannalta alkulämpötila on siis syytä valita tarkasti. Mikäli kohdefunktion arvojen vaihteluvälin suurin arvo $\max(\triangle f)$ tunnetaan, niin sitä voidaan käyttää alkulämpötilan muodostami-

seksi ja sitä voidaan arvioida yhtälöllä

$$T_0 \approx -\frac{\max(\triangle f)}{\ln p_0},$$

missä p_0 on ennalta määrätty todennäköisyys. Toisaalta vaihteluväli $\max(\triangle f)$ ei aina ole välttämättä tunnettu tai sen kokoluokasta ei voida tehdä tarkkaa arviota, jolloin T_0 täytyy valita suureksi. Tällöin, kun arvoa T_0 arvioidaan, on se yleensä niin suuri, että se hyväksyy kaikki iteraatioiden muutokset. Kuten aiemmin todettiin, kyseessä oleva tilanne ei ole tavoiteltu, jolloin se korjataan ottamalla suuria iteraatioaskelia. Iteraatioita käydään läpi, kunnes noin $50 - 60\%$ huonontavia ratkaisuja tulee algoritmin valitsemaksi. Toisin sanoen tällöin algoritmi toimii suunnitellusti eli se hyväksyy kohdefunktion arvoja kasvattavia ja huonontavia ratkaisuja.

Pelkkä alkulämpötila ei yksin riitä algoritmin optimaalisen toiminnan takaamiseksi. Lämpötilan lisäksi on vielä määriteltävä jäähtymisaikataulu. Toisin sanoen, miten hitaasti tai nopeasti algoritmin mallintama jäähtyminen tapahtuu. Kaksi yleisimmin käytettyä jäähtymisaikataulua ovat lineaarinen jäähtyminen ja geometrinen jäähtyminen. Lineaarinen jäähtyminen esitetään muodossa,

$$T = T_0 - \beta t, \tag{13}$$

missä T_0 on alkulämpötila ja t on iteraatiolaskuri. Parametri β puolestaan kuvaa jäähtymisnopeutta, joka tulisi valita siten että $T \to 0$, pseudoajan lähestyessä viimeistä iteraatiota N. Toisin sanoen $t \to N$, mistä yleisesti saadaan $\beta = \frac{T_0 - T_f}{N}$. Kaavan T_f parametrilla tarkoitetaan lämpötilaa, mihin algoritmin suorittaminen lopetetaan. Toinen vaihtoehto jäähtymiselle on geometrinen jäähtyminen, jossa jäähtyminen tapahtuu jäähtymistekijän α avulla, jolle pätee $0 < \alpha < 1$. Yleisesti tässä jäähtymisaikataulussa T korvataan termillä αT

$$T_t = T_0 \alpha^t, \qquad t = 1, 2, \ldots, N. \tag{14}$$

Geometrinen jäähtyminen ei vaadi iteraatiomäärän määrittelyä, kun $T \to 0$ niin $t \to \infty$. Tällöin algoritmin suoritus loppuu, kun algoritmissa ei tapahdu enää muu-

toksia. Edellisestä lauseesta voi jäädä käsitys, että algoritmi iteroituu äärettömästi. Näin ei kuitenkaan tapahdu käytännössä, sillä simuloitu jäähdytys voi muodostaa uuden iteraatiopisteen kahdella tavalla. Ensimmäinen vaihtoehto on, kun kaavan (10) energiatasojen muutos on $\triangle E \leq 0$ [23]. Tällöin uusi ratkaisu on aiempaa parempi ja iteraatio hyväksytään. Toinen vaihtoehto on $\triangle E > 0$, jolloin iteraatio hyväksytään, jos kaavan (9) transitiotodennäköisyys on suurempi kuin kynnysarvo r, niin iteraatio hyväksytään [23]. Toisin sanoen, kun uusi ratkaisu ei ole aiempaa parempi, eikä se voi enää ylittää kynnysarvoa r algoritmin suoritus pysähtyy. Jäähtymisprosessin tulisi tapahtua riittävän hitaasti, jotta algoritmin löytämä optimi on mielekäs. Käytännössä yleensä parametri α on suuruudeltaan $\alpha = 0,7 \sim 0,99$. [7, 21, 23, 24]

4.2.2 Pseudokoodi

Simuloidun jäähdytyksen ymmärtämisen syventämiseksi tutustutaan seuraavaksi metaheuristiikan pseudokoodin. Pseudokoodista on helpompi ymmärtää, miten algoritmin toteuttama tehostaminen ja hajauttaminen käytännössä toimii, kun iteraatioprosessi puretaan helpommin seurattaviin vaiheisiin.

Simuloitu jäähdytys[7]

Kohdefunktio $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ ja $\boldsymbol{x} \in S$.
Alustetaan alkulämpötila T_0, iteraatiolaskuri $t = 0$ ja lähtöarvo $\boldsymbol{x}_0$, joka on mikä tahansa sallittu ratkaisu.
Alustetaan parhaat ratkaisut $\boldsymbol{x}^*$ ja f^*
Asetetaan lopetuslämpötila T_f ja iteraatioiden maksimimäärä N, lisäksi valitaan jäähtymistekijä α.
while $T_t > T_f$ ja $t < N$ **do**
  Määritellään jäähtymisnopeus $T_{t+1} = \alpha T_t$, $(0 < \alpha < 1)$
  Valitaan satunnaisvektori $\boldsymbol{\epsilon}$ normaalijakaumasta
  Siirrytään satunnaisesti uuteen sijaintiin: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{\epsilon}$
  Lasketaan $\triangle f = f(\boldsymbol{x}_{t+1}) - f(\boldsymbol{x}_t)$
  Hyväksytään uusi ratkaisu, jos se on aiempaa parempi
  if vastaus ei parantunut **then**
    Luodaan satunnaisluku r
    Hyväksytään jos $p = e^{[\frac{-\triangle f}{T_t}]} > r$
  end if
  Päivitetään paras ratkaisu $\boldsymbol{x}^*$ ja f^*
  $t = t + 1$
end while

4.3 Geneettinen algoritmi

Geneettiseksi algoritmiksi (Genetic algorithm) kutsuttu metaheuristiikka on saanut ideansa luonnossa tapahtuvasta evoluutioksi kutsutusta biologisesta prosessista. Ajatuksena geneettisen algoritmin taustalla on hyödyntää Darwinin teoriaa luonnonvalinnasta. Geneettinen algoritmi esiteltiin ensimmäisen kerran vuonna 1992 ja sen kehitti J.H. Holland. Menetelmässä algoritmi mallintaa luonnonvalintaa ja populaation kelpoisuuden kehitystä geneettisen ajautumisen seurauksena. Keskeisinä elementteinä geneettisessä algoritmissa ovat: ratkaisujen koodaaminen kromosomeiksi, kromosomien valinta, risteytys, mutaatio ja kelpoisuusfunktio. Kromosomit ovat koodattuja yksittäisratkaisuja, jotka kokevat algoritmin iteraatioissa risteytymistä ja mutaatioita. Yhdessä kaikki kromosomit muodostavat populaation, eli keskenään kilpailevien kandidaattiratkaisujen joukon. Keskenään kilpailevista ratkaisuista valituiksi tulevat kromosomit, jotka ovat todennäköisemmin kelpoisuudeltaan parhaita ja niiden avulla toteutetaan kromosomien risteytys. Kromosomien kvalitatiiviseksi arvioimiseksi käytetään kelpoisuusfunktiota, joka ei suoranaisesti vaikuta mutaation tapahtumisen todennäköisyyteen. Kromosomiksi kutsuttu merkkijono tullaan tässä työssä esittämään binäärimuotoisena yksinkertaisuuden vuoksi, vaikka käytännössä koodaustapoja on useita. Iteraatioita kutsutaan geneettisen algoritmin yhteydessä sukupolviksi. Lähtötilanteessa on alkupopulaatio, joka sisältää N kappaletta kromosomeja, jotka alustetaan satunnaisesti. [7, 25]

4.3.1 Menetelmän matemaattinen tarkastelu

Kromosomien esittäminen voidaan toteuttaa binäärimuotoisena, heksadesimaalimuotoisena, oktaalimuotoisen tai muilla esitystavoilla. [25] Kuten aiemmin mainittiin tässä työssä tarkastellaan ainoastaan binääristä esitysmuotoa. Esitysmuodon valinta vaikuttaa käytännössä siihen minkälaisia kromosomeja algoritmi käsittelee. Paras esitysmuoto riippuu optimointitehtävästä ja tavoiteltavasta algoritmin käyttäytymisestä. Kromosomit muuttuvat sukupolvien, eli iteraatioiden kuluessa. Tämä johtuu kromosomien algoritmissa kokemasta valinnasta, risteytyksestä ja mutaatiosta.

Ennen risteytyksen ja mutaation tarkempaa tarkastelua, keskitytään geneetti-

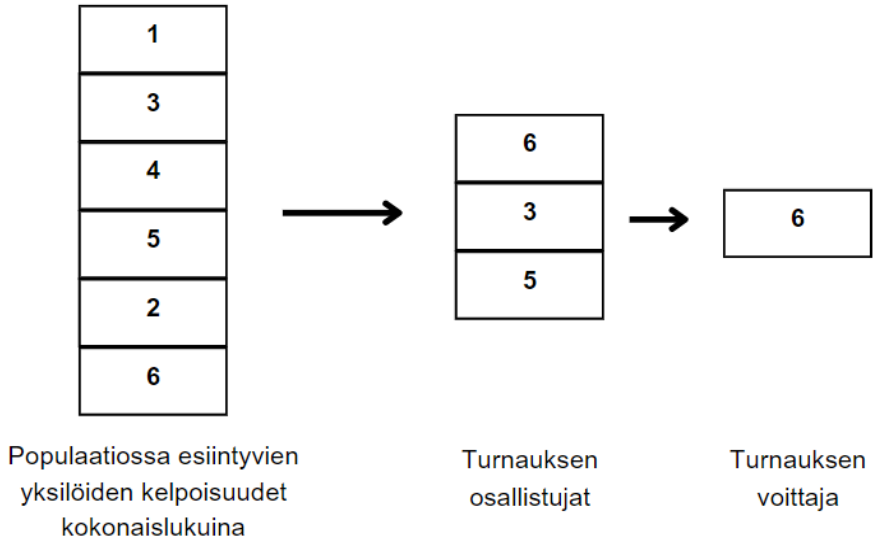
sen algoritmin toteuttamaan valinnaksi kutsuttuun evoluutio-operaatioon. Kromosomien lukuisien esitysvaihtoehtojen tavoin mahdollisia kromosomien valintatapoja on useita. Vaihtoehdoista yleisimmin käytettyjä ovat turnausvalinta ja onnenpyörävalinta. Onnenpyörävalinnassa jokaiselle kromosomille i määritellään kelpoisuusfunktion avulla kelpoisuus. Näin ollen kromosomilla i on olemassa kelpoisuus F_i, joka suoraan vaikuttaa kromosomin valintatodennäköisyyteen. Nimittäin kromosomille i voidaan määritellä valintatodennäköisyys p_i, joka määräytyy seuraavasti

$$p_i = \frac{F_i}{\sum_{j=1}^{N} F_j}.$$

Turnausvalinta on valintamenetelmänä varsin monimuotoinen ja sen määrittelyllä on mahdollista vaikuttaa tulevien populaatioiden monimuotoisuuteen. Turnausvalinnassa on onnenpyörävalinnan tavoin jokaiselle kromosomille määritelty kelpoisuus F_i. Kromosomien kelpoisuuksien ohella täytyy määritellä valintaa suorittavien turnauksien koko k. Turnauksia pidetään niin monta, että saadaan uusi N:n kokoinen sukupolvi, joka on yhtäsuuri alkupopulaation kanssa. Populaation yksilöistä arvotaan turnaukseen osallistuvat yksilöt, jotka kilpailevat keskenään. Turnauksen osallistujien arvonta toteutetaan usein esimerkiksi onnenpyörävalinnalla, sillä tällöin ei tule karsittua liikaa lupaavia ratkaisukandidaatteja. Turnauksen voittaja voidaan yksinkertaisuuden vuoksi todeta olevan suurimman kelpoisuuden omaava yksilö, vaikka voittajan valintaan on olemassa lukuisia tapoja. Turnausvalintaprosessi kuvassa 2 on yksinkertainen esimerkki algoritmin toteuttamasta valinnasta. Mikäli turnauksessa $k = 1$ tarkoittaa se turnausvalinnan kontekstissa satunnaisvalintaa alkuperäisestä populaatiosta. Turnaukseen osallistuvien yksilöiden valinta on oleellinen ja yleisimmin käytetään binääristä turnausvalintaa, missä valintatodennäköisyydelle p_i saatiin lähteessä [26] muoto

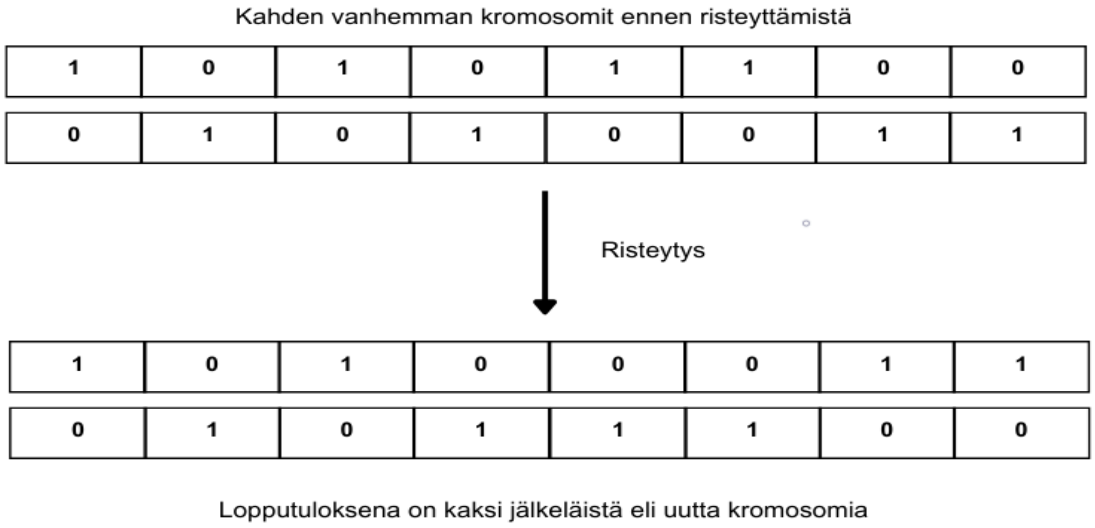
$$p_i = \frac{1}{N^k}((i)^k - (i-1)^k),$$

missä k on turnauksen dimensio. [27]



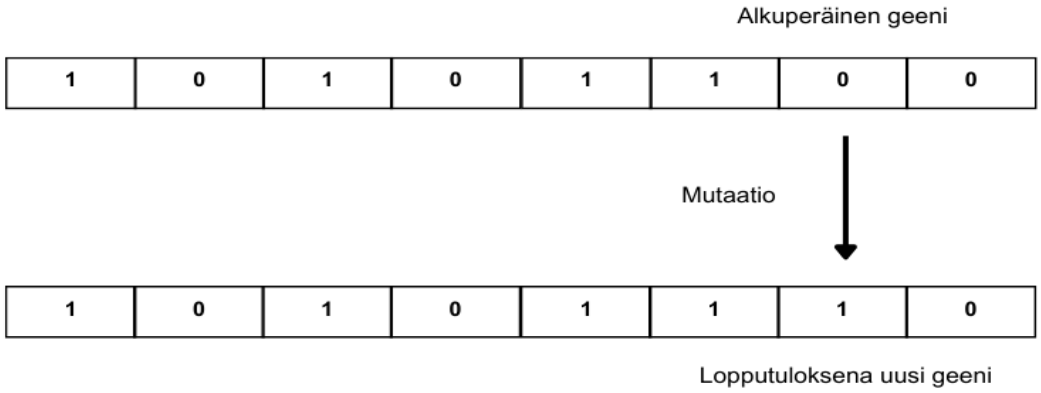
Kuva 2: Turnausvalinta, kun $N = 6$ ja $k = 3$.

Risteytys tarkoittaa käytännössä kahden valituksi tulleen kromosomin osien vaihtamista keskenään. Yhden osan keskenään vaihtamisessa on kyseessä 1-pisteristeytys, joka tapahtuu kromosomeille kuvassa 3 esitetyllä tavalla. Risteyttämisen toteuttamiseen on olemassa lukuisia tapoja, joista ainakin yhdeksän erilaista on mainittu lähteessä [25]. Menetelmien joukosta oikean valitseminen perustuu jälleen haluttuun algoritmin toimintaan, sekä ratkaistavaan optimointiongelmaan. Risteytyksen ohella tapahtuu mutaatioita, jotka muuttavat geenejä kuvassa 4 näkyvällä tavalla. Risteytyksen perustelujen tapaan mutaatiolle on lukuisia toteutustapoja ja niiden valitseminen on tilannekohtaista.



Kuva 3: Geneettisen algoritmin toteuttama risteytys, joka tapahtuu yhdessä kohdassa kromosomia. Vanhemman geenien oikeassa reunassa oleva järjestys on muuttunut risteytyksen seurauksena. Kromosomien muutokset tapahtuvat samanaikaisesti, vaikka muutoksia tulisi useammassa eri kohdassa. [7]

Metaheuristiikoille yleisesti ja geneettiselle evoluutiolle erityisesti parametrien valitseminen on äärimmäisen tärkeää. Erityisesti geneettisessä algoritmissa korostuu, millä tavoin optimointitehtävä muutetaan algoritmin käsittelemään kromosomimuotoon. Nimittäin algoritmin tekemä optimointi pohjautuu vertailuun kromosomien ja



Kuva 4: Geneettisen algoritmin toteuttama mutaatio, joka tapahtuu yhdessä kohdassa kromosomia. Alkuperäisen geenin sisältämä informaatio muuttuu mutaation seurauksena. Samalla tavalla kuin risteytyksessä, myös mutaatio voi tapahtua useammassa paikassa samaan aikaan. [7]

valitun kelpoisuusfunktion välillä. Kuvissa 3 ja 4 esitettiin yksinkertaisuuden vuoksi vain binäärimuotoisessa kromosomissa tapahtuvia muutoksia. Todellisuudessa genettisen algoritmin merkkijonot valitaan ongelmakohtaisesti ja tarpeen vaatiessa ne voidaan ilmaista esimerkiksi reaalilukujen avulla.

Aiemmin mainitun kelpoisuusfunktion rooli algoritmissa on tärkeä, sillä se toimii vertailuarvona, jonka suhteen kromosomien käytöstä optimoidaan. Minimointia tehtäessä eräs yksinkertainen ilmaisuasu kelpoisuusfunktiolle on $F(x) = A - f(x)$, missä A voi olla joko suuri vakio tai 0, jos kelpoisuudelta ei vaadita positiivista arvoa.

4.3.2 Pseudokoodi

Simuloidusta jäähdytyksestä poiketen geneettinen algoritmi käsittelee lukuisia ratkaisukandidaatteja samanaikaisesti. Esitellään seuraavaksi yksityiskohtainen geneettisen algoritmin pseudokoodi, jonka avulla tiivistetään algoritmin toimintaperiaate.

Geneettinen algoritmi[7, 25]

Kohdefunktio $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ ja $\boldsymbol{x} \in S$.
Luodaan satunnaisesti alkupopulaatio, joka koostuu N kappaleesta kromosomeja $\boldsymbol{x}_i^0$ $(i = 1, 2, \ldots, N)$.
Määritellään kromosomien kelpoisuudet $F_i = A - f(\boldsymbol{x}_i^0)$
Alustetaan iteraatiolaskuri $t = 0$ ja todennäköisyydet
p_c risteytykselle, p_m mutaatiolle
while ($t <$ iteraatioiden maksimimäärä) **do**
  Järjestetään kelpoisuuksia käyttäen N kappaletta turnauksia
  Korvataan aiemman sukupolven kromosomit turnauksien voittajista koostuvalla N kokoisella sukupolvella
  Luodaan N kappaletta uusia ratkaisuja risteyttämällä ja mutatoimalla
  Risteytys tapahtuu risteytystodennäköisyydellä p_c
  Mutaatio tapahtuu mutaatiotodennäköisyydellä p_m
  Päivitetään risteytyksen ja mutaation seurauksena muodostuneiden kromosomien kelpoisuudet F_i
  Päivitetään iteraatiolaskuri $t = t + 1$
end while
Palautetaan ratkaisuna korkeimman kelpoisuuden omaava yksilö F_i^*

---

5 Diskreetti optimointi

Varsin usein populaatiopohjaisissa metaheuristisissa algoritmeissa, kuten esimerkiksi kiiltomatoalgoritmissa, optimoinnin tulokset voidaan esittää reaaliarvoisina muuttujina tai vektoreina. Diskreettinen optimointi on matemaattisen optimoinnin alahaara, jossa optimoitava funktio on korkeintaan osittain jatkuva tai yleisesti täysin epäjatkuva eli diskreettinen.

Esimerkki 1. Yksinkertainen esimerkki diskreetistä kohdefunktiosta on kokonaislukujen avulla määritelty funktio $f : \mathbb{Z} \to \mathbb{R}$. Funktio on diskreettinen, kun se on määritelty vain kokonaislukujen joukossa.

Diskreetissä optimoinnissa reaaliarvoisten muuttujien ja vektoreiden käsittely on haasteellista. Syynä tähän on esimerkiksi tilanne, missä metaheuristiseen algoritmiin koodattu ratkaisuvektori esimerkiksi kiiltomato, toteuttaa sallittujen pisteiden joukon rajoitteet, mutta ei enää välttämättä ole kelvollinen ratkaisu alkuperäiseen ongelmaan. Pohditaan tapausta, jossa optimoinnin ratkaisu on jokin kokonaislukujen permutaatio. Metaheuristiikka voi ratkaisua etsiessään muuttaa vektorin arvoja, niin että edellä mainittu permutaatio voi rikkoutua. Tämä on ratkaisun kannalta haastavaa, sillä on mahdollista, että algoritmi tuottaa vain kelvottomia ratkaisuja tarkasteltavalle tehtävälle. Diskreettisyys voi siis olla äärimmäisen ongelmallista metaheuristisessa algoritmissa, jonka tarkoituksena on etsiä ratkaistavaa optimia reaalilukujen avulla. Tällöin pelkästään sallittujen pisteiden joukkoon kuuluminen optimointitehtävässä ei ole enää riittävää vastauksen tuottamiseksi, vaan menetelmästä riippuen täytyy soveltaa diskretisointimenetelmiä. [28]

5.1 Diskretisointimenetelmiä

Diskreetissä optimoinnissa törmätään ongelmiin populaatiopohjaisten metaheuristiikkojen kanssa, koska monet niistä on alun perin suunniteltu jatkuvien eli reaaliarvoisten muuttujien optimoimiseen. Konkreettiset diskreetit ongelmat ovat laskennallisesti erittäin haastavia ja tehtävän dimension kasvaessa niiden ratkaiseminen tarkkoja menetelmiä käyttäen käy usein mahdottomaksi. Täten populaatiopohjaiset metaheuristiikat ovat näyttäytyneet mielenkiintoisina vaihtoehtoina tyydyttävien rat-

kaisujen saamiseksi. Populaatiopohjaisten algoritmien ominaisuudet, kuten yksittäiset keskenään kilpailevat ratkaisut, ovat osoittautuneet hyödyllisiksi diskreettisessä optimoinnissa. [28]

Sovellettaessa populaatiopohjaista menetelmää diskreettiin ongelmaan tulee edellisessä kappaleessa esitetyn ongelman välttämiseksi menetelmä diskretisoida. Populaatiopohjainen metaheuristiikka voidaan diskretisoida usealla eri tavalla. Vaihtoehtoisista menetelmistä tunnetuimpia ja käytetyimpiä ovat sigmoidaalista funktiota (Sigmoid function), satunnaisavain (Random-Key), pienimmän sijainnin arvo (Smallest position value), muokattu sijaintiyhtälö (Modified position equation), suuren arvon priorisointi (Great value priority) ja lähin kokonaisluku (Nearest Integer). Perehdytään tarkemmin sigmoidaaliseen funktioon, satunnaisavainmenetelmään ja pienimmän sijainnin arvoon, jotka tutkimuksen mukaan vastaavat yhdessä 90% käytetyistä diskretisointimenetelmistä.[28]

5.1.1 Sigmoidaalinen funktio

Sigmoidaalista funktiota voidaan käyttää jatkuvan hakuavaruuden binääriseksi muuttamiseen ja se on edellä mainituista menetelmistä suosituin. Muunnos tehdään kaikille vektorin alkioille. Vektori diskretisoidaan binääriseksi seuraavasti. Oletetaan, että

$$x_{ij} = \begin{cases} 1, & \text{jos } rand() \leq \frac{1}{1+\exp(-x_{ij})} \\ 0, & \text{muutoin,} \end{cases} \tag{15}$$

jossa $i = 1,\ldots,N$ ja $j = 1,\ldots,n$, missä N on ratkaisuvektoreiden lukumäärä ja n on ratkaisuvektorin ulottuvuus. Funktio $rand()$ tuottaa satunnaisluvun väliltä $[0, 1]$. [28]

5.1.2 Satunnaisavain

Satunnaisavain on sigmoidaalisen funktion jälkeen käytetyin menetelmä populaatiopohjaisten metaheuristiikoiden diskretisoinnissa. Menetelmässä jatkuva vektori voidaan muuttaa diskreetiksi luokittelemalla alkiot suurusjärjestykseen pienimmästä alkaen. Esimerkiksi jatkuva ratkaisuvektori $\vec{x_i} = (0.92, 0.24, 0.05, 0.22, 0.78)$ voidaan

muuttaa diskreettiin muotoon $\vec{x_i} = (5, 3, 1, 2, 4)$. [28]

5.1.3 Pienimmän sijainnin arvo

Pienimmän sijainnin arvon menetelmässä kuvataan ratkaisuvektorin komponentit pienimmästä alkaen. Esimerkiksi ratkaisuvektorille $\vec{x_i} = (0.92, 0.24, 0.05, 0.22, 0.78)$ pienimmän sijainnin arvon ratkaisuvektori on muotoa $\vec{x_i} = (3, 4, 2, 5, 1)$. [28]

5.2 Kauppamatkustajan ongelma

Diskreetin optimoinnin kirjallisuuden tunnetuinta ongelmaa kutsutaan *kauppamatkustajan ongelmaksi* (Travelling salesman problem tai TSP). Ongelma on täydellinen metaheuristiikoiden syvempään tarkasteluun. Ongelmasta on tehty mittavasti tutkimusta metaheuristiikkoja hyödyntäen ja se vie algoritmien suorituskyvyn äärimmilleen. Määritellään seuraavaksi kauppamatkustajan ongelma.

Määritelmä 5. *Kauppamatkustajan ongelma*: Alussa on olemassa lista kaupungeista ja niiden välisistä etäisyyksistä. Tarkoituksena on selvittää, mikä on lyhin mahdollinen matka, kun kuljetaan kaikkien kaupunkien kautta täsmälleen kerran ja lopuksi palataan lähtökaupunkiin.

Varsin yksinkertaisesti muotoiltava tehtävä osoittautuu kuitenkin ratkaistaessa laskennalliseksi painajaiseksi. Ongelma on käytännönläheinen, mutta se konkretisoituu varsin erilaisissa käytännön sovelluksissa, kuten esimerkiksi verkko- ja piirilevysuunnittelussa sekä logistisissa toteutuksissa. Ongelmaan ei ole vielä löydetty kaupunkimäärästä polynomisesti riippuvaa ratkaisumenetelmää ja sitä pidetäänkin NP-täydellisenä (NP-complete) ongelmana [16]. Polynomiaikaista algoritmia on etsitty jo vuosikymmeniä, eikä varmuutta sen olemassaolosta tai -olemattomuudesta ole vielä onnistuttu todistamaan. Mikäli polynomiaikainen algoritmi löytyisi, seurauksena olisi logistisen optimoinnin merkittävä parantuminen. Tämä tarkoittaisi käytännössä lyhyempiä kuljettavia matkoja, jotka parantaisivat logistiikka-alan yrityksien kannattavuutta ja potentiaalisesti vähentäisivät myös ympäristön kuormitusta. [29] Tehtävä on helppo ilmaista, mutta pienikin kaupunkien määrän lisääminen kasvattaa mahdollisten reittien vaihtoehtojen määrää kertoman verran. Tarkastellaan

seuraavaksi kauppamatkustajan ongelmaa suhteellisen pienellä kaupunkimäärällä.

Esimerkki 2. Kauppamatkustajan ongelman kaupunkien määrän ollessa vain 50 löytyy mahdollisia reittejä 50! kappaletta, mikä tarkoittaa noin $3.041 \cdot 10^{64}$ vaihtoehtoa.

Yllä olevan esimerkin avulla saadaan parempi käsitys, miten kaikkien mahdollisten vaihtoehtojen yksittäinen läpikäyminen ei ole ajallisesti mielekästä suurilla kaupunkimäärillä. Tällöin metaheuristiikat esiintyvät äärimmäisen houkuttelevina vaihtoehtoina. Niiden avulla saadaan erinomaisia tuloksia yksittäisille kaupunkimäärille. Eräs motivaattori kauppamatkustajan ongelman tutkimiseen ja muihin laskentatehokkuuden tutkimuksiin on $P = NP$ -ongelma. Mikäli kyseisen ongelman ratkaisu voidaan tarkistaa polynomisessa ajassa, niin sen ratkaiseminen polynomisessa ajassa saattaa olla mahdollista. Kuuluisaa $P = NP$ -ongelmaa ei vielä tähän päivään asti ole osoitettu todeksi tai epätodeksi ja se onkin eräs seitsemästä avoimesta miljoonan dollarin ongelmasta.[16]

6 Metaheuristiikat ja kauppamatkustajan ongelma

Kauppamatkustajan ongelma on varsin kattavasti tutkittu ongelma, jota on optimoitu miltei kaikilla tunnetuilla metaheuristisilla algoritmeilla. Metaheuristisen menetelmien suosio juuri kauppamatkustajan ongelmassa johtuu sen NP-täydellisyydestä, mihin metaheuristisen menetelmien laskenta-aikaa ja laskentatehoa säästävät ominaisuudet soveltuvat täydellisesti. Kauppamatkustajan ongelma soveltuu loistavasti metaheuristisen algoritmien suorituskykytestiksi johtuen siihen jo olemassa olevasta kattavasta dokumentoinnista ja tutkimuksesta. Esimerkiksi Deniz et al [30] käyttivät kauppamatkustajan ongelmaa nimenomaisesti suorituskykytestinä metheuristisen algoritmien tutkimuksessa.

6.1 Kiiltomatoalgoritmin soveltaminen

Kiiltomatoalgoritmi on alunperin suunniteltu jatkuviin optimointitehtäviin ja niiden optimoimiseen. Alkuperäistä kiiltomatoalgoritmia ei voida sellaisenaan hyödyntää, koska kauppamatkustajan ongelma on diskreettinen. [28, 31] Ainoaksi vaihtoehdoksi jää tällöin algoritmin diskretisoiminen, mitä käsiteltiin aiemmin. Kiiltomatoalgoritmin tapauksessa jatkuvat rajoitteet, jotka liittyvät puoleensavetävyyteen ja etäisyysfunktioon, tulee diskretisoida [1]. Epäjatkuvien rajoitteiden avulla toimivaa algoritmia kutsutaan kirjallisuudessa diskreetiksi kiiltomatoalgoritmiksi (Discrete firefly algorithm, DFA). Näin saatua algoritmia voidaan soveltaa diskreetteihin ongelmiin, kuten esimerkiksi kauppamatkustajan ongelmaan ja erilaisiin aikataulutusongelmiin [1, 32].

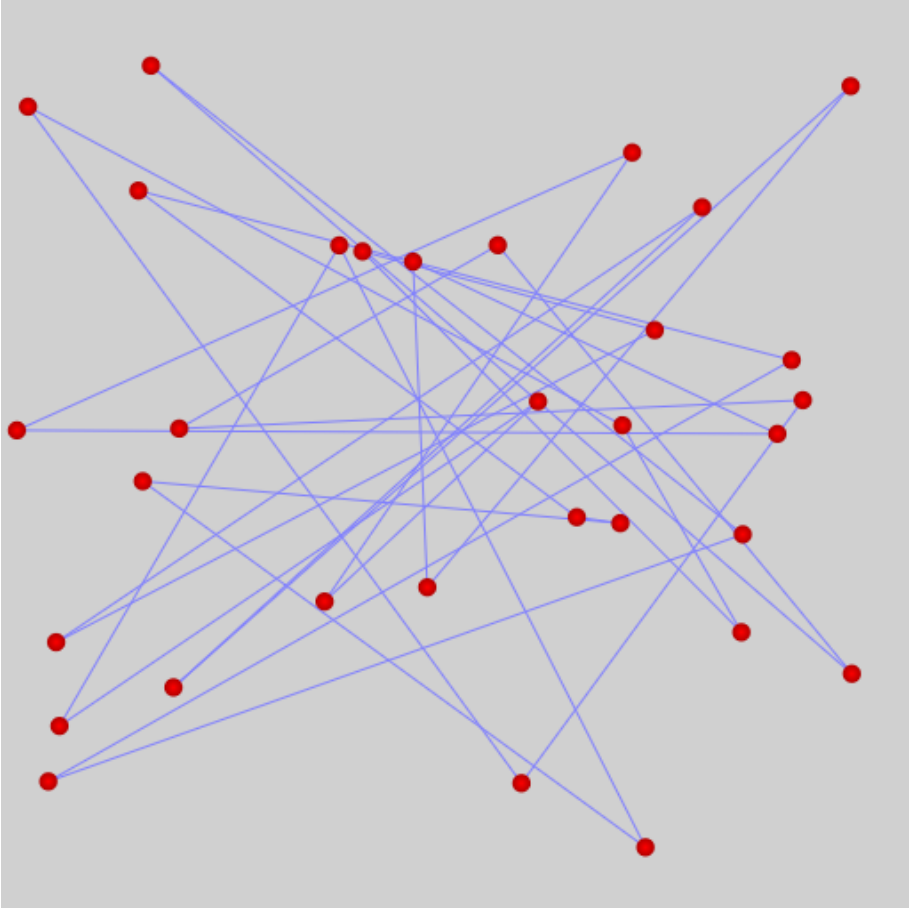
Pelkkä algoritmin rajoitteellinen yhteensopivuus ei takaa mitään ongelman ratkaisun laadusta. Vaikka kiiltomatoalgoritmi onkin osoittanut paikkansa tunnettuna menetelmänä, sen suorituskykyä voidaan parantaa muiden menetelmien avulla. Metaheuristisen strategioiden yhdistelemisestä syntyneitä menetelmiä kutsutaan *hybridialgoritmeiksi*. Tämä yhdisteleminen voidaan nähdä ikään kuin algoritmin virittämisenä. Hybridisoinnin määrittelyä ei ole virallisesti rajoitettu, mikä tarkoittaa, että mikä tahansa osa metaheuristiikasta voidaan korvata eri suunnittelufilosofian komponentilla paremman tuloksen aikaansaamiseksi. Esimerkiksi kiiltoma-

toalgoritmissa hybridisaatiota voidaan toteuttaa kiiltomatojen alustamisessa, niiden liikkumisessa tai ratkaisun hyväksymisessä. Tunnettuja kiiltomatoalgoritmin kanssa hybridisoituja menetelmiä ovat muun muassa geneettiset algoritmit (GA), differentiaalievoluutio (DE) ja parannettu paikallishaku (local search). Kaikkien näiden hybridisointien avulla on tarkoitus kehittää alkuperäisen algoritmin soveltuvuutta optimoitavaan ongelmaan ja mahdollisesti parantaa sen suorituskykyä [1]. Eräs tä kiiltomatoalgoritmin hybridisointia kauppamatkustajan ongelman ratkaisemiseksi tutkittiin evolutiivisen diskretisoidun kiiltomatoalgoritmin avulla lähteessä [32].

6.2 Simuloidun jäähdytyksen soveltaminen

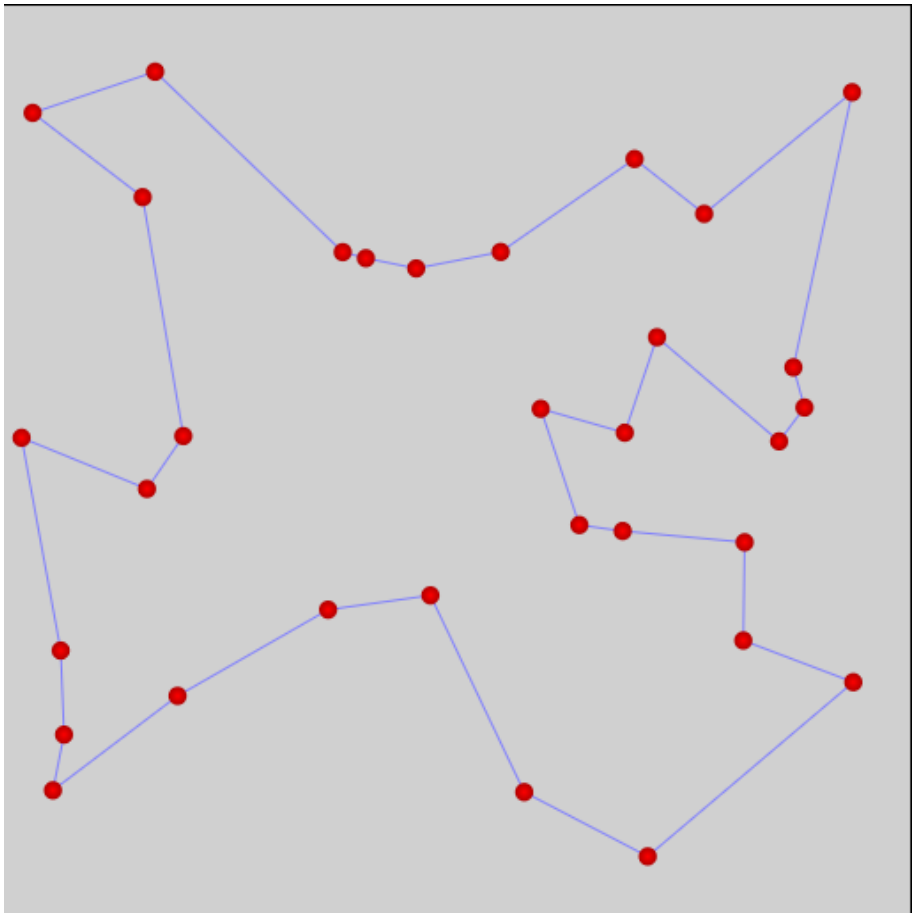
Kiiltomatoalgoritmista poiketen, simuloitua jäähdytystä ei tarvitse muokata sopimaan diskreettiin ongelmaan, koska se on alunperin suunniteltu diskreettiin optimointiin. Tämän suoran soveltuvuuden voisi ajatella tekevän simuloidusta jäähdytyksestä poikkeuksellisen houkuttelevan menetelmän esimerkiksi kauppamatkustajan ongelman ratkaisuun. Simuloitua jäähdytystä kauppamatkustajan ongelmaan sellaisenaan soveltavien tutkimuksien määrä on kuitenkin yllättävän vähäinen. Erääna syynä tähän on metaheuristiikan hidas suppeneminen kohti optimaalista ratkaisua [33, 34]. Ensisilmäykseltä huonona pidettävä ominaisuus simuloidussa jäähtymisessä ei sitä kuitenkaan aina välttämättä ole. Simuloidun jäähdytyksen hidas konvergenssi nimittäin mahdollistaa parempia lopputuloksia, mitä enemmän aikaa algoritmi saa operaatioiden suorittamiseen [35]. Algoritmin todetaan myös olevan varsin herkkä muutoksille, jolloin parametrien vaikutus korostuu saatavien lopputuloksien valossa varsin merkittävästi. Näin ollen tietotaito algoritmin käyttämisessä on keskeisessä osassa selittämässä keskenään varsin ristiriitaisia tuloksia, joita simuloidulla jäähtymisellä on saatu.

Simuloidulla jäähdytyksellä on kuitenkin paljon potentiaalia tehokkaana metaheuristiikkana. Algoritmin heikkouksia on koitettu kiertää esimerkiksi muuttamalla jäähtymisaikatauluja, kuten tehtiin lähteissä [36, 37]. Näin toimittaessa on saatu lupaavia tuloksia kauppamatkustajan ongelmaan.



Kuva 5: Lähtötilanne 32 kaupungin kokoisessa kauppamatkustajan ongelmassa. Kaupunkien väliset yhteydet ovat täysin satunnaiset, eikä lyhintä mahdollista reittiä ole vielä lähdetty optimoimaan.

Kuvista 5 ja 6 nähdään simuloidun jäähdytyksen visualisointi alkutilanteessa ja optimoinnin jälkeen. Kuten metaheuristiikoilla yleisesti ongelmana on, simuloidun jäähdytyksen tapauksessakaan ei ole takeita, että kuvan 6 reitti olisi lyhin mahdollinen. Kuitenkin logistiikassa reitin voidaan ajatella olevan "tarpeeksi" hyvä esimerkiksi polttoaineen kulutuksen tai ajamiseen kuluvan ajan kannalta.



Kuva 6: Eräs simuloidun jäähdytyksen tuotattama ratkaisu 32 kaupungin kokoiselle tapaukselle. Ratkaisun optimaalisuutta ajatellen ei voida todeta, onko kyseinen muoto kaikista optimaalisin, mutta alkutilanteeseen (kuva 5) verrattuna parannus on varsin merkittävä. Ratkaisu voisi mahdollisesti olla riittävän hyvä esimerkiksi logistiikka-alan sovelluksissa, kuten kuljetusreittien optimoinnissa.

6.3 Geneettisen algoritmin soveltaminen

Geneettisen algoritmin soveltuvuutta kauppamatkustajan ongelmaan on ajan saatossa tutkittu varsin kattavasti. Geneettinen algoritmi soveltuu hyvin kauppamatkustajan ongelman ratkaisevaksi menetelmäksi. Menetelmä kuuluu simuloidun jäähdytyksen ohella vanhimpiin tunnettuihin metaheuristisiin menetelmiin. Geneettisessä algoritmissa paikallishaku (local search) on varsin tehokasta ja jopa parempaa kuin iteratiivisessa paikallishaussa, jossa vanha tulos päivitetään paremmalla ratkaisulla. Tämä johtuu paikallishaun ominaisuudesta jumittua paikalliseen optimiin. Simuloidun jäähdytyksen voidaan todeta toimivan geneettisen algoritmin tavoin paikallishaussa, mutta geneettisen algoritmin suorituskyky on osoittautunut

paremmaksi. Tämä johtuu algoritmin ratkaisukomponenttien eli geenien periytymisestä uusiin ratkaisuihin. Uudet ratkaisut saavat kauppamatkustajan ongelman yhteydessä aiemmilta ratkaisuilta hyviä osaratkaisuja. Yhdistelemällä näin saatuja hyviä osaratkaisuja onnistuu geneettinen algoritmi luomaan parempia ratkaisuja kuin simuloidun jäähdytyksen jatkuva iteratiivinen lähestyminen lyhimmän reitin löytämiseksi. Lisäksi geneettisen algoritmin väitetään olevan ainoa metaheuristiikka, joka on onnistunut ratkaisemaan kauppamatkustajan ongelman yli 3000 kaupungin tapauksissa. [38]

Mielenkiintoinen näkökulma genettisen algoritmin soveltamisessa kauppamatkustajan ongelmaan on myös tehtävään valittava koodaustapa. Aiemmin geneettisen algoritmin yhteydessä tarkasteltiin yksityiskohtaisemmin binääristä koodaustapaa. Binäärisen koodauksen ohella geneettisessä algoritmissa voidaan soveltaa geenien käsittelyä listana. Näistä kahdesta binäärinen koodaus on osoittautunut hyväksi, mutta vain pienille kaupunkimäärille [39]. Näin ollen paras tapa koodata reittejä geneettiselle algoritmille on geenien listaaminen.

6.4 Tulokset

Aloitetaan tuloksien tarkastelu ensin käymällä läpi tutkimus [35], jossa vertailtiin geneettistä algoritmia ja simuloitua jäähdytystä kauppamatkustajan ongelmassa. Tutkimuksen mukaan simuloitu jäähdytys suoriutuu tarkasteltavista ongelmista geneettistä algoritmia paremmin. Asian ilmaiseminen niin, että toinen menetelmä on toista parempi ei metaheuristisessa optimoinnissa ole mielekästä. Esimerkiksi tutkimuksessa käytetty TSPlib on internetissä sijaitseva kauppamatkustajan ongelmien kirjasto, joka on kauppamatkustajan ongelmaa tutkittaessa varsin yleisesti käytetty [40]. Yksittäisiä kauppamatkustajan ongelman instansseja tutkittaessa ei voida muodostaa yleistä näkemystä metaheuristiikan suorituskyvystä annetussa tehtävässä, vaan vain nimenomaisesti sillä hetkellä tarkasteltavien tehtävien tapauksista. Tästä seuraa ongelma siitä, että metaheuristiikkojen ollessa yleisiä optimointimenetelmiä, niiden suorituskyky vaihtelee tehtävästä riippuen. Edelleen tämä vaihtelevuus tuo ongelmia ja saattaa edesauttaa muodostamaan väärinkäsityksiä siitä, miten jokin menetelmä saattaisi olla toista menetelmää parempi. Toisin sanoen va-

litsemalla tarkasteltavat tehtävät oikealla tavalla voidaan mikä tahansa menetelmä saada näyttämään toista paremmalta. Perehdytään seuraavaksi tutkimukseen, joka vahvistaa aikaisemmin esitetyn argumentin.

Tutkimuksessa [41] verrataan jälleen simuloitua jäähdytystä geneettiseen algoritmiin ja lähimmän naapurin (nearest neighbour) menetelmään. Tuloksien mukaan geneettinen algoritmi toimii selvästi simuloitua jäähdytystä paremmin ja kykenee merkittävästi parempiin lopputuloksiin. Näin olemme kahta eri tutkimusta [35, 41] vertaamalla saavuttaneet ristiriidan geneettisen algoritmin ja simuloidun jäähdytyksen suorituskyvyn kanssa. Tapaus ei metaheuristisessa tutkimuksessa ole mitenkään ainutlaatuinen, vaan nimenomaisesti esimerkki siitä, minkälainen tieto on oleellista. Metaheuristiseen tutkimukseen tulisi suhtautua äärimmäisellä varovaisuudella ja kriittisyydellä, sekä selvittää minkälaiset tulokset alan tutkimuksessa ovat oikeasti mielekkäitä ja jalostuskelpoisia.

Artikkelissa [30] puolestaan tutkittiin metaheurististen algoritmien tehokkuutta ja yleistä toimivuutta kauppamatkustajan ongelman tapauksessa. Tuloksista havaitaan, miten riippumatta metaheuristiikasta ne kaikki onnistuvat optimoimaan annettuja kauppamatkustajan ongelmia tehokkaasti. Tulokset ovat yleisesti muutaman prosenttiyksikön etäisyydellä tunnetusta optimaalisesta ratkaisusta. Näin voidaan todeta metaheuristiikkojen tuottavan luvattuja "riittävän hyviä" lopputuloksia myös kauppamatkustajan ongelmassa. Metaheuristisen algoritmien tuoma satunnaisuus samalla myös vähentää laskennallisesti raskaiden ongelmien ratkaisuun käytettävää aikaa.

Viimeisenä mainitsemisen arvoisena tuloksena on kiiltomatoagloritmin suorituskyky kauppamatkustajan ongelmassa. Tutkimuksissa [32, 35] muutamilla eri algoritmeilla testattu kauppamatkustajan ongelma sattuu olemaan keskenään identtinen. Aineistot Ulysses 16 ja Ulysses 22 ovat vastaavasti 16 ja 22 kaupungin kokoisia kauppamatkustajan ongelman aineistoja, joka on nimenomaisesti tarkoitettu ongelman tutkimukseen. Näin tutkimuksia ristiin vertaamalla voidaan todeta, miten onnistuneesti diskretisoidulla kiiltomatoalgoritmilla ja sen eräällä evolutiivisella hybridimetaheuristiikalla on keskenään vertailukelpoiset tulokset simuloidun jäähdytyksen ja geneettisen algoritmin kanssa. Kaikki metaheuristiikat onnistuivat ratkaisemaan

annetun kauppamatkustajan ongelman tapauksen alle minuutissa, mikä on tehtävän kompleksisuuden huomioon ottaessa mainitsemisen arvoinen laskenta-ajallinen suoritus.

7 Yhteenveto

NP-täydellisten ongelmien ratkaisemiseen ei ole olemassa hyvää algoritmia, joka kykenisi murtamaan tehtävätyyppiä. Kirjallisuudessa esiintyy lukuisia optimointialgoritmeja, eikä yksikään näistä algoritmeista kykene ratkaisemaan kaikkia olemassaolevia optimointiongelmia. Siitä huolimatta tehokkaiden optimointialgoritmien etsimiseen panostetaan merkittävästi alan tutkimuksen keskuudessa. Tämä pyhän "graalin maljan" kaltaisen algoritmin etsiminen jatkuu varmasti, ellei analyyttisesti onnistuta osoittamaan tulos universaalin optimointityökalun olemattomuudesta. [7] Metaheuristiikat pyrkivät olemaan universaaleja monitoimityökaluja, jotka tarjoavat hyviä tai jopa optimaalisia ratkaisuja laskennallisesti haastaviin optimointitehtäviin.

Kunnianhimoisista tavoitteistaan huolimatta eivät metaheuristiikatkaan kykene välttymään D. H. Wolpertin ja W. G. Macreadyn ilmaisten lounaiden teoreemalta (No free lunch theorem, NFL) [42]. NFL - teoreeman yksinkertaista, mutta kerta toisensa jälkeen näytettyä tulosta ei vielä pakene yksikään tunnettu optimointimenetelmä. Tuloksen mukaan optimointialgoritmien tehokkuudet lähestyvät toisiaan, mitä suurempi niiden vertailussa käytettävien tehtävien määrä on. Lisäksi teoreeman avulla on osoitettu, ettei yksikään optimaalinen tulos löydy vahingossa. Aivan kuin ei lounaskaan ilmaiseksi tyhjästä ilmesty.

Valitettavasti siis metaheuristiikoiden tuomat työmäärälliset helpotukset ovat optimoinnin mullistamisen kannalta varsin vähäisiä. Tämä tarkoittaa, että optimoitavan ongelman mallintaminen, parametrien valinta ja niiden soveltaminen tulevat jatkossakin olemaan työläitä sekä keskeisiä optimaalisten tuloksen löytämisessä. Lisäksi algoritmien ja menetelmien ongelmakeskeisyys tulee näkymään parempien tuloksien muodossa sekä siinä, miten ongelmaan paremmin hioutuneet menetelmät tulevat suoriutumaan yleisiä menetelmiä paremmin. Toisaalta on huomionarvoista muistaa, miten metaheuristiset menetelmät ovat osoittaneet ratkaisutaitonsa laskennallisesti raskaissa tehtävissä. Metaheuristiikoiden ansiosta voidaan ratkaista tehtäviä, jotka eivät ole klassisten menetelmien avulla optimoitavissa. Tulevaisuudessa on mielenkiintoista nähdä, miten tekoälyllä voidaan helpottaa algoritmien käyttämien rajoitteiden määrittelyä. Lisäksi tuloksien parantamisen kannalta tehtävän algorit-

mien "virittelyn" sijaan voidaan keskittyä optimointitehtävän ratkaisuun ja menetelmien ymmärtämiseen sekä toimivuuden osoittamiseen.

Lähdeluettelo

[1] Fister, I., Fister, I. Jr., Yang, X.-S., Brest J. (2013) *A Comprehensive Review of Firefly Algorithms*, Swarm and evolutionary computation, Vol. 13, s. 34–46

[2] Yang, X. (2018) *Mathematical Analysis of Nature-Inspired Algorithms*, Nature-Inspired Algorithms and Applied Optimization, Vol. 744, s. 1-25

[3] Sörensen, K., Glover, F. (2013) *Metaheuristics*, Encyclopedia of Operations Research and Management Science, 3rd edition, s. 960-970

[4] Sörensen, K., Sevaux M., Glover, F. (2018) *A History of Metaheuristics*, Handbook of Heuristics, 2nd edition, s. 791-808

[5] Nesmachnow, S. (2014) *An Overview of Metaheuristics: Accurate and Efficient Methods for Optimisation*, International Journal of Metaheuristics, Vol. 3, No. 4, s. 320-347

[6] Fausto, F., Reyna-Orta, A., Cuevas, E., Andrade, Á. G., Perez-Cisneros, M. (2020) *From Ants to Whales: Metaheuristics for All Tastes*, Artificial Intelligence Review, Vol. 53, No. 1, s. 753-810

[7] Yang, X.-S. (2014) *Nature-Inspired Optimization Algorithms*, 1st edition, Elsevier

[8] Mäkelä, M. (2015) *Matemaattinen optimointi 1*, Opintomoniste, Turun yliopisto

[9] Stork, J., Eiben, A. E., Bartz-Beielstein, T. (2022) *A new taxonomy of global optimization algorithms*, Natural Computing, Vol. 21, No. 2, s. 219-242

[10] Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., Juan, A. A. (2015) *Rich Vehicle Routing Problem: Survey*, ACM Computing Surveys, Vol. 47, No. 2, s. 1-28

[11] De León-Aldaco, S. E., Calleja H., Aguayo Alquicira, J. (2015) *Metaheuristic Optimization Methods Applied to Power Converters: A Review*, IEEE Transactions on Power Electronics, Vol. 30, No 12, s. 6791-6803

[12] Boschetti, M. A., Maniezzo, V. (2022) *Matheuristics: Using Mathematics for Heuristic Design*, Handbook of Heuristics, Vol. 20, No. 2, s. 173-208

[13] Rana, N., Latiff, M. S. A., Abdulhamid, S. M., Chiroma, H. (2020) *Whale optimization algorithm: a systematic review of contemporary applications, modifications and developments*, Neural Computing and Applications, Vol. 32, s. 16245-16277

[14] Sörensen, K. (2015) *Metaheuristics—the Metaphor Exposed*, International Transactions in Operational Research, Vol. 22, No. 1, s. 3-18

[15] Stegherr, H., Heider, M., Hähner, J. (2022) *Classifying Metaheuristics: Towards a Unified Multi-level Classification System*, Natural computing, Vol. 21, No. 2, s. 155-171

[16] Orponen, P. (2007) *P = NP -ongelma ja laskennan vaativuusteoria*, Tietojenkäsittelytiede Vol. 26, s. 54-67

[17] Yang, X.-S., Deb, S., Fong, S. (2013) *Metaheuristic Algorithms: Optimal Balance of Intensification and Diversification*, Applied Mathematics & Information Sciences, Vol. 8, No. 3, s. 977-983

[18] Blum, C., Roli, A. (2003) *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*, ACM computing surveys, Vol. 35, No. 3, s. 268-308

[19] Lozano, M., García-Martínez, C. (2010) *Hybrid Metaheuristics with Evolutionary Algorithms Specializing in Intensification and Diversification: Overview and Progress Report*, Computers & operations research, Vol. 37, No. 3, s. 481-497

[20] Lones, M. (2014) *Metaheuristics in Nature-Inspired Algorithms*, 10.1145/2598394.2609841

[21] Henderson, D., Jacobson S. H., Johnson, A. W. (2006) *The Theory and Practice of Simulated Annealing*, Handbook of Metaheuristics, International Series in Operations Research & Management Science, Vol. 57, s. 287-319

[22] Deshpande, A. M., Phatnani, G. M., Kulkarni, A. J. (2013) *Constraint Handling in Firefly Algorithm*, IEEE International Conference on Cybernetics, s. 186-190

[23] Kirkpatrick, S., Gelatt, C.D. Jr., Vecchi, M.P. (1983) *Optimization by Simulated Annealing*, Science (New York, N.Y.), Vol. 220, s. 671-680

[24] Henderson, D., Jacobson, S. H., Johnson, A. W. (2003) *The Theory and Practice of Simulated Annealing*, Handbook of Metaheuristics, s. 287-319

[25] Katoch, S., Chauhan, S. S., Kumar, V. (2021) *A review on genetic algorithm: past, present, and future*, Multimedia Tools and Applications, Vol. 80, No. 5, s. 8091-8126

[26] Hussain, A., Riaz, S., Amjad, M. S., ul Haq, E. (2022) *Genetic Algorithm with a New Round-Robin Based Tournament Selection: Statistical Properties Analysis*, PloS one, Vol. 17, No. 9

[27] Razali N. M., Geraghty J. (2011) *Genetic Algorithm Performance with Different Selection Strategies in Solving TSP*, Proceedings of the World Congress on Engineering, Vol. 2

[28] Krause, J., Cordeiro, J., Parpinelli, R., S., Lopes, H. S. (2013) *A survey of swarm algorithms applied to discrete optimization problems*, Swarm Intelligence and Bio-Inspired Computation, s. 169-191.

[29] Dündar, A. O., Sahman, M. A., Tekin, M., Kıran, M. S. (2019) *A Comparative Application Regarding the Effects of Traveling Salesman Problem on Logistics Costs*, International Journal of Intelligent Systems and Applications in Engineering, Vol. 7, s. 207-215

[30] Tosoni, D., Galli, C., Hanne, T., Dornberger, R. (2022) *Benchmarking Metaheuristic Optimization Algorithms on Travelling Salesman Problems*, Proceedings of the 8th International Conference on e-Society, e-Learning and e-Technologies, s. 20-25

[31] Wang, M.-B., Fu, Q., Tong, N., Li, M., Zhao, Y. (2015) *An Improved Firefly Algorithm for Traveling Salesman Problems*, Proceedings of the 2015 4th National Conference on Electrical, Electronics and Computer Engineering, s. 1085-1092

[32] Jati, G. K., Manurung, R., Suyanto. (2013) *Discrete Firefly Algorithm for Traveling Salesman Problem: A New Movement Scheme*, Swarm Intelligence and Bio-Inspired Computation: Theory and Applications, s. 295-312

[33] Dréo, J., Siarry, P., Pétrowski, A., Taillard, E., Chatterjee, A. (2006) *Metaheuristics for Hard Optimization: Methods and Case Studies*, Springer

[34] Oliveira, H. A., Petraglia, A. (2011) *Global optimization using dimensional jumping and fuzzy adaptive simulated annealing*, Applied Soft Computing, Vol. 11, No. 6, s. 4175-4182

[35] Botsalı, A., Alakyrian, K. (2020) *Analysis of TSP: Simulated Annealing and Genetic Algorithm Approaches*, International Journal of Computational and Experimental Science and Engineering, Vol. 6, No. 1, s. 23-28

[36] Yang, W., Wang, Y. (2012) *Improved simulated annealing algorithm for GTSP*, IET Conference Proceedings, Vol. 2012, No. 598, s. 1202-1205

[37] Zhan, S.-H., Lin, J., Zhang, Z., Zhong, Y. (2016) *List-Based Simulated Annealing Algorithm for Traveling Salesman Problem*, Computational Intelligence and Neuroscience, Vol. 2016

[38] Scholz, J. (2018) *Genetic algorithms and the traveling salesman problem: A historical review*, arXiv preprint arXiv:1901.05737

[39] Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., Dizdarevic, S. (1999) *Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators*, Artificial Intelligence Review, Vol. 13, No. 2, s. 129-170

[40] Applegate, D. L., Bixby, R. E., Chvátal, V. (2006) *The Traveling Salesman Problem: A Computational Study*, Princeton Series in Applied Mathematics, Princeton University Press

[41] Rao, T. S. (2017) *A comparative evaluation of GA and SA TSP in a supply chain network*, Materials Today: Proceedings, Vol. 4, No. 2, s. 2263-2268

[42] Wolpert, D. H., Macready, W. G. (1997) *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, s. 67-82