Parviälyyn perustuvat metaheuristiset optimointialgoritmit

LuK Julia von Hertzen

Pro gradu -tutkielma
Tammikuu 2023

Tarkastajat:
Prof. Marko Mäkelä
FT Stefan Emet

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

Julia von Hertzen: Parviälyyn perustuvat metaheuristiset optimointialgoritmit
Pro gradu -tutkielma, 44 s., 2 liites.
Sovellettu matematiikka
Tammikuu 2023

Tässä Pro gradu -tutkielmassa käsitellään yleisesti optimointia, metaheuristiikkoja ja parviälyä. Tutkielmassa esitellään muutamia luonnon inspiroimia parviälyalgoritmeja sekä vertaillaan niitä keskenään teoreettisesti ja numeerisesti. Tutkielmassa esitellyt vertailut on toteutettu kirjallisuuskatsauksena.

Luonnon inspiroimat parviälyalgoritmit ovat metaheuristiikkoja, jotka perustuvat luonnossa elävien eläinlajien parvikäyttäytymiseen. Parvikäyttäytyminen juontaa juurensa parvessa elävien yksilöiden välisestä parviälystä, joka perustuu parven yksilöiden yhteiseen tavoitteeseen. Parviälyn periaatteena on, että parven yksilöiden välisen käyttäytymisen ja tiedonsaannin avulla syntyy toimiva kokonaisuus. Tutkielmassa esitellyt parviälyalgoritmit ovat muurahaisyhdyskuntaoptimointi, partikkeliparvioptimointi, mehiläisyhdyskuntaoptimointi, lepakkoalgoritmi, käkihaku ja tulikärpäsalgoritmi. Kyseiset parviälyalgoritmit ovat monikäyttöisiä erilaisten optimointitehtävien ratkaisemiseen, muun muassa niiden populaatiopohjaisuuden ansiosta. Lisäksi algoritmien numeerinen vertailu osoittaa muutaman algoritmin tehokkaan tavan saavuttaa jopa globaali optimi.

Asiasanat: optimointi, metaheuristiikka, parviäly, luonnon inspiroimat algoritmit

Sisällys

1 Johdanto

Jokainen ihminen törmää arjessaan ongelmiin, joiden ratkaisu voidaan ajatella perustuvan optimointiin. Esimerkiksi lyhyimmän reitin etsiminen työpaikalle on optimointia eli parhaan ratkaisun etsimistä kaikkien ratkaisujen joukosta. Matemaattisesti tämän kaltaiset ongelmat voidaan esittää optimointitehtävinä. Optimointitehtäviä voidaan jakaa eri alalajeihin esimerkiksi kohdefunktion, päätösmuuttujien ja rajoitteiden ominaisuuksien perusteella.

Optimointitehtävässä ongelma mallinnetaan ensin matemaattisesti ja sen jälkeen ratkaistaan jollakin algoritmilla. Mallintamisessa optimointitehtävälle määrätään kohdefunktio ja tavoite. Tavoitteena pidetään yleensä kohdefunktion minimointia tai maksimointia. Kohdefunktion lisäksi optimointitehtäville määrätään päätösmuuttujat ja rajoitteet. Ratkaisuna voidaan hakea optimaalisinta arvoa eli globaalia optimia tai vaihtoehtoisesti lähintä paikallista eli lokaalia optimia.

Riittävän hyvään ratkaisuun voidaan joutua tyytymään esimerkiksi suurten optimointitehtävien kanssa. Tällöin optimaalisen ratkaisun löytäminen voisi viedä kohtuuttomasti aikaa ja resursseja, joka ei välttämättä itse ongelman ratkaisemiseksi olisi järkevää. Tällaisissa tilanteissa voidaan hyödyntää heuristiikkoja. Heuristiikat ovat optimointialgoritmeja, jotka eivät välttämättä tuota optimaalista ratkaisua, mutta pääsevät silti yleensä riittävän hyvään ratkaisuun. Ne voidaan jakaa kahteen eri luokkaan: tavallisiin heuristiikkoihin ja metaheuristiikkoihin. Tavalliset heuristiikat ovat tiettyyn ongelmaan soveltuvia menetelmiä, kun metaheuristiikat ovat puolestaan ongelmasta riippumattomia. Metaheuristiikat ovat tehokkaita iteraatiomenetelmiä, jotka pyrkivät tiettyjä yksinkertaisia sääntöjä noudattaen löytämään globaalin optimin.

Tunnetuimpia metaheuristisia menetelmiä ovat tabuhaku, simuloitu jäähdytys ja geneettiset algoritmit. Näiden menetelmien toimivuus perustuu siihen, että ratkaisua etsittäessä sallitaan jopa tietyin ehdoin huonontavat askeleet. Geneettiset algoritmit ovat biologista periytymistä jäljitteleviä optimointimenetelmiä, jotka käyttävät hyödyksi luonnonvalintaa. Geneettisten algoritmien käyttökelpoisuus johtuu niiden kyvystä käsitellä useampia ratkaisuja samalla iteraatiokierroksella. Tunnetuimmat geneettiset operaattorit ovat valinta, risteytys ja mutaatio. Geneettisten operaattoreiden avulla voidaan vaikuttaa siihen, mitkä yksilöistä valitaan ja miten yksilöiden hyvät ominaisuudet saadaan säilytettyä seuraavaan sukupolveen. Ominaisuuksien säilymisen lisäksi on tärkeää huolehtia sukupolven monimuotoisuudesta. Geneettisten algoritmien lisäksi muita tunnettuja luonnon inspiroimia algoritmeja ovat esimerkiksi parviälyyn perustuvat algoritmit.

Tutkielman pääaiheena toimivat parviäly ja siihen liittyvät optimointialgoritmit. Parviäly perustuu parven yksilöiden yhteiseen tavoitteeseen. Vaikka parven yksilöiden suoriutuminen tehtävästä olisi huono, voi silti kokonainen parvi päästä lähelle tavoitetta. Parviälyä esiintyy ihmisillä, mutta myös monilla eläinlajeilla. Varsinkin parvissa elävien eläinlajien käyttäytymisessä on huomattavissa parviälyä. Esimerkiksi linnut ja kalat osaavat hämmästyttävän hyvin kulkea suurissakin parvissa törmäämättä toisiinsa. Vaikka liikkuminen saattaa vaikuttaa sattumanvaraiselta, senkin taustalla on tutkittu olevan tietynlainen optimointialgoritmi. Tällaisia luonnon inspiroimia parviälyalgoritmeja on kehitetty tähän päivään mennessä kymmeniä

erilaisia. Ne ovat myös suosittuja aiheita kirjallisuudessa, mikä näkyykin lähteiden runsaudessa.

Tähän tutkielmaan valitut parviälyalgoritmit mallintavat lintujen ja kalojen, muurahaisten, mehiläisten, lepakkojen, käkien sekä tulikärpästen käyttäytymistä. Kaikki valitut algoritmit ovat metaheuristiikkoja, jotka hyödyntävät algoritmin populaatiopohjaisuutta. Tämän ansiosta kyseiset parviälyalgoritmit ovat monikäyttöisiä erilaisten optimointiongelmien ratkaisemiseen. Vaikka parviälyalgoritmit kuuluvat metaheuristiikkojen joukkoon, varsinkin tuoreimmat kehitetyistä algoritmeista ovat erittäin tehokkaita ja täyttävät jopa globaalit konvergenssivaatimukset. Tästä hyvä esimerkki on käkihaku, joka löytää numeerisessa vertailussa todennäköisimmin globaalin optimin verrattuna muihin algoritmeihin.

Tutkielmaan valittujen parviälyalgoritmien lisäksi on mielenkiintoista ottaa vertailuun mukaan myös muita luonnon inspiroimia algoritmeja. Tässä vertailussa ovat mukana muun muassa geneettinen algoritmi, differentiaalievoluutio, gravitaatiohakualgoritmi ja harmaasusioptimointi. Näiden yhteneväisyydet ja eroavaisuudet johtuvat pitkälti samoista ominaisuuksista kuin pelkkien parviälyalgoritmien teoreettisessa vertailussa. Luonnon inspiroimien algoritmien suorituskyvyn numeerinen vertailu on myös yhdenmukainen aiemmien saatujen tulosten ja kirjallisuuden havaintojen nojalla: käkihaulla ja differentiaalievoluutiolla on selkeästi parempi suorituskyky kuin muilla algoritmeilla ja ne tuottavat todennäköisimmin globaalin optimin.

Tutkielma rakentuu viidestä luvusta. Toisessa luvussa käsitellään pohjatietona optimoinnista ja konveksisuudesta vaadittavia määritelmiä ja lauseita. Asiat havainnollistetaan lineaarisen ja epälineaarisen optimointitehtävän esimerkkien avulla. Kolmannessa luvussa käsitellään heuristisia ja metaheuristisia algoritmeja. Metaheuristisista menetelmistä esitellään tabuhaku, simuloitu jäähdytys ja geneettiset algoritmit. Neljäs luku koostuu parviälystä sekä parviälyalgoritmien esittelystä ja vertailusta. Luvun teoreettinen ja numeerinen vertailu toteutetaan kirjallisuuskatsauksena. Neljännen luvun lopussa vertaillaan lisäksi vielä yleisesti luonnon inspiroimia algoritmeja.

2 Optimointi yleisesti

Optimointi tarkoittaa parhaan ratkaisun etsimistä kaikkien mahdollisten ratkaisujen joukosta. Optimointitehtävät ovat erittäin tärkeitä matematiikassa, mutta ne myös esittävät merkittävää roolia arkielämässä. Optimoinnilla voidaan esimerkiksi yrittää löytää vastaus seuraaviin arkielämän kysymyksiin: Miten minimoida polttoaineen kulutus työmatkalla? Miten kylmälaukun saa pakattua niin, että kylmyys säilyy mahdollisimman pitkään? Tai mikä on nopein reitti lomakohteeseen? Näiden lisäksi esimerkiksi kuljetusten, aikataulutusten, pakkausten ja tietoverkkojen suunnittelu pohjautuu pitkälti optimointiongelman mallintamiseen ja ratkaisuun. [23, s. 4]

Optimointitehtävät koostuvat tehtävän muodostamisesta eli matemaattisesta mallintamisesta ja ratkaisemisesta. Useat käytännön optimointitehtävistä vaativat runsaasti yleistyksiä. Optimointitehtävälle voidaan yrittää löytää tarkka globaali optimiratkaisu, mutta toisinaan voidaan joutua tyytymään riittävän hyvään ratkaisuun. [5, s. 32] Seuraavissa aliluvuissa kerrataan optimointiin liittyvät tärkeimmät käsitteet ja esitetään esimerkit lineaarisesta ja epälineaarisesta optimointitehtävästä.

2.1 Tärkeimmät käsitteet

Optimointiin liittyy aina jokin funktio f, jota kutsutaan *kohdefunktioksi*. Kohdefunktio kertoo miten valittavassa olevia ratkaisuja eli *päätösmuuttujia* voidaan vertailla. Päätösmuuttujia kuvataan yleensä vektorilla $\mathbf{x}$. Päätösmuuttujien arvot ovat optimointitehtävän alussa toistaiseksi tuntemattomia. Optimointitehtävässä voi esiintyä myös *rajoitteita*, jotka rajoittavat päätösmuuttujien valintaa. [20, s. 1] Rajoitteet määrittävät sallitun joukon S, joka on yksinkertaistuksen vuoksi jatkossa koko $\mathbb{R}^n$. Rajoitteet voidaan jakaa luonnollisiin ja varsinaisiin rajoitteisiin. Luonnolliset rajoitteet seuraavat päätösmuuttujiin liittyvistä ominaisuuksista. Tavallisia luonnollisia rajoitteita ovat esimerkiksi ei-negatiivisuusrajoitukset. Varsinaiset rajoitteet ovat muita ehtoja, jotka rajoittavat päätösmuuttujien valintaa, esimerkiksi kysyntään ja kapasiteettiin liittyvät rajoitukset. *Parametreiksi* kutsutaan optimointitehtävän mallintamiseen tarvittavia muita suureita. Parametreilla on tehtävässä yleensä numeerinen arvo, esimerkiksi hinta. [15, s. 15-16]

Optimointitehtävän tavoitteena on yleensä kohdefunktion minimointi tai maksimointi. *Minimoinnissa* pyritään löytämään sallituista päätösmuuttujan $\mathbf{x}$ arvoista se, jonka kohdefunktion arvo on pienin. Siis sellainen $\mathbf{x}^*$, että kaikilla sallitun joukon alkioilla $\mathbf{x}$ pätee

$$f(\mathbf{x}^*) \leq f(\mathbf{x}).$$

Kohdefunktion arvoa $f(\mathbf{x}^*)$ kutsutaan tällöin minimiarvoksi. Vastaavasti *maksimoinnissa* pyritään löytämään sallituista päätösmuuttajan $\mathbf{x}$ arvoista se, jonka kohdefunktion arvo on suurin. Siis sellainen $\mathbf{x}^*$, että kaikilla sallitun joukon alkioilla $\mathbf{x}$ pätee

$$f(\mathbf{x}^*) \geq f(\mathbf{x}).$$

Kohdefunktion arvoa $f(\mathbf{x}^*)$ kutsutaan tällöin maksimiarvoksi. [20, s. 3] Minimointi- ja maksimointitehtävät on mahdollista ratkaista samoilla menetelmillä, koska koh-

defunktion f maksimointi tarkoittaa samaa kuin funktion $-f$ minimointi. Tästä syystä usein käsitelläänkin vain funktion minimointia. [5, s. 34][6, s. 332]

Optimoinnin peruskysymyksenä on siis mikä päätösmuuttujista tuottaa optimaalisen, tilanteesta riippuen lokaalin tai globaalin optimin, kohdefunktion arvon [20, s. 3]. Määritellään vielä mitä tarkoittavat lokaali ja globaali optimi.

Määritelmä 2.1. Optimointitehtävän *lokaali optimi* on piste $\mathbf{x}^* \in \mathbb{R}^n$, jossa pätee

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \text{ kaikilla } \mathbf{x} \in \mathbb{R}^n \text{ siten että } \parallel \mathbf{x}^* - \mathbf{x} \parallel < \epsilon,\ \epsilon > 0.$$

Minimointitehtävässä lokaalia optimia voidaan kutsua *lokaaliksi minimiksi.*

Määritelmä 2.2. Optimointitehtävän *globaali optimi* on piste $\mathbf{x}^* \in \mathbb{R}^n$, jossa pätee

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \text{ kaikilla } \mathbf{x} \in \mathbb{R}^n.$$

Minimointitehtävässä globaalia optimia voidaan kutsua *globaaliksi minimiksi* ja se on pienin kaikista lokaaleista optimeista. Globaalien optimointitehtävien ratkaiseminen on yleensä vaikeaa, koska tällöin optimointitehtävän tulee olla joko konveksi tai joudutaan käyttämään globaaliin optimointiin soveltuvia menetelmiä [5, s. 34].

Optimointitehtävissä käytetään usein hyödyksi konveksisuutta. Tästä syystä määritellään vielä, mitä tarkoittavat konveksi joukko ja konveksi funktio sekä todistetaan niihin liittyvät lauseet.

Määritelmä 2.3. Joukko S on *konveksi joukko*, jos kahden mielivaltaisen joukon pisteen yhdysjana kuuluu kokonaisuudessaan joukkoon S eli

$$\lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \in S,$$

kaikilla $\mathbf{x}_1, \mathbf{x}_2 \in S$ ja $\lambda \in (0, 1)$. [14, s. 2]

Määritelmä 2.4. Olkoon funktio $f : S \to \mathbb{R}$, missä $S \neq \emptyset$ on konveksi avaruuden $\mathbb{R}^n$ osajoukko. Sanotaan, että funktio f on *konveksi funktio* joukossa S, jos

$$f(\lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1 - \lambda) f(\mathbf{x}_2),$$

kaikilla $\mathbf{x}_1, \mathbf{x}_2 \in S$ ja $\lambda \in (0, 1)$. [14, s. 27]

Lause 2.5. *Olkoon $S \neq \emptyset$ avoin konveksi joukko ja funktio $f : S \to \mathbb{R}$ kahdesti differentioituva joukossa S. Funktio f on konveksi, jos ja vain jos Hessen matriisi $\boldsymbol{H}$ on positiivisemidefiniitti jokaisessa joukon S pisteessä.*

Todistus löytyy *Konveksi analyysi ja optimointi* -luentomonisteesta [14, s. 40-41] ja se sivuutetaan. Hessen matriisilla $\mathbf{H}$ tarkoitetaan matriisia, joka koostuu funktion f toisista osittaisderivaatoista eli

$$\mathbf{H}(\mathbf{x}) = \left(\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \right),$$

kaikilla $i, j = 1, \ldots, n$ [14, s. 39].

Lause 2.6. *Olkoon $S \neq \emptyset$ konveksi joukko ja funktio $f : S \to \mathbb{R}$. Tarkastellaan tehtävää*

$$\min_{\boldsymbol{x} \in S} f(\boldsymbol{x}).$$

Olkoon $\boldsymbol{x}^* \in S$ tehtävän lokaali optimi. Silloin jos f on konveksi funktio, niin piste $\boldsymbol{x}^*$ on globaali optimi.

Todistus. Oletetaan, että f on konveksi funktio. Koska oletuksen mukaan piste $\mathbf{x}^*$ on lokaali optimi, niin määritelmän 2.1 mukaan tällöin on olemassa piste $\mathbf{x}$, siten että $f(\mathbf{x}^*) \leq f(\mathbf{x})$. Tehdään vastaoletus, ettei piste $\mathbf{x}^*$ olisikaan globaali optimi. Tällöin on olemassa piste $\mathbf{y} \in S$ siten, että $f(\mathbf{y}) < f(\mathbf{x}^*)$. Koska lisäksi oletuksen mukaan funktio f on konveksi, saadaan määritelmän 2.4 mukaan

$$f(\lambda \mathbf{y} + (1-\lambda)\mathbf{x}^*) \leq \lambda f(\mathbf{y}) + (1-\lambda) f(\mathbf{x}^*) < \lambda f(\mathbf{x}^*) + (1-\lambda) f(\mathbf{x}^*) = f(\mathbf{x}^*),$$

kaikilla $\lambda \in (0,1)$. Jos nyt parametri λ valitaan riittävän pieneksi, saadaan

$$\lambda \mathbf{y} + (1-\lambda)\mathbf{x}^* \in S,$$

mikä on ristiriidassa sen oletuksen kanssa, että piste $\mathbf{x}^*$ on lokaali optimi. Siis piste $\mathbf{x}^*$ on myös globaali optimi. [14, s. 43-44] □

Optimointitehtävät voidaan yleisesti jakaa useisiin eri alalajeihin. Yleisin jako riippuen kohdefunktion ominaisuuksista on *lineaarisiin* ja *epälineaarisiin optimointitehtäviin* [15, s. 1]. Lineaarisissa optimointitehtävissä kohdefunktio ja rajoitteet ovat lineaarisia päätösmuuttujien suhteen. Epälineaarisissa optimointitehtävissä jokin kohdefunktio tai rajoite on epälineaarinen. Lineaarisista ja epälineaarisesta optimointehtävistä esitetään esimerkit aliluvuissa 2.2 ja 2.3. Optimointitehtävässä voi olla myös sekä lineaarisia että epälineaarisia kohdefunktioita ja rajoitteita. Esimerkiksi lineaarisesti rajoitetussa epälineaarisessa optimointitehtävässä on lineaariset rajoitukset, mutta kohdefunktio on epälineaarinen. Tästä eräs alalaji on *kvadraattinen optimointitehtävä*, jossa kohdefunktio on kvadraattinen eli kohdefunktion termit ovat yksittäisiä päätösmuuttujia, päätösmuuttujien neliöitä tai päätösmuuttujien tuloja kerrottuna vakiolla [8, s. 11]. Päätösmuuttujista riippuva jako on esimerkiksi *diskreetteihin* ja *jatkuviin optimointitehtäviin*. Diskreeteissä optimointitehtävissä muuttujat saavat arvoja numeroituvista joukoista. Yleisimpiä diskreettejä optimointitehtäviä ovat *kokonaislukuoptimointi- ja binäärioptimointitehtävät*. Kokonaislukuoptimoinnissa päätösmuuttujat saavat ainoastaan kokonaislukuarvoja, kun taas binäärioptimoinnissa arvoja 0 tai 1. Jatkuvissa optimointitehtävissä muuttujien arvot voivat olla mitä tahansa reaalilukuarvoja. [6, s. 333] Optimointitehtävissä on myös mahdollista, että mukana on sekä diskreettejä että jatkuvia päätösmuuttujia, jolloin kyseessä on *sekalukuoptimointi*. Mikäli optimointitehtävässä on mukana useampi kuin yksi kohdefunktio, sitä kutsutaan *monitavoiteoptimoinniksi* [23, s. 4]. Monitavoiteoptimoinnissa on usein tarkoituksena löytää kaikki mahdolliset ratkaisut siten, että ratkaisujoukon ulkopuolelta ei löydy ratkaisua, joka olisi parempi kuin joukon ratkaisut kaikkien tavoitteiden suhteen. Monitavoiteoptimoinnissa pyritään ratkaisujen löytymisen lisäksi löytämään paras mahdollinen kompromissiratkaisu [8,

s. 5]. Näiden lisäksi optimointitehtävät voidaan muun muuassa jakaa vielä *sileään* ja *epäsileään optimointiin* sekä *konveksiin* ja *epäkonveksiin optimointiin*. Sileässä optimoinnissa kohdefunktion ja rajoitusten oletetaan olevan kaikkialla jatkuvasti derivoituvia, kun epäsileässä optimoinnissa ei. Konveksisissa optimoinnissa kohdefunktio ja rajoitukset ovat konvekseja. Epäkonveksi optimointi sisältää kaikki epälineaariset optimointimallit, jotka eivät täytä konveksin optimoinnin oletuksia. [8, s. 12]

2.2 Esimerkki: lineaarinen optimointitehtävä

Lineaarisissa optimointitehtävissä kohdefunktio ja rajoitteet ovat siis päätösmuuttujien suhteen lineaarisia. Koska lineaariset funktiot ovat aina selvästi konvekseja, on lauseen 2.6 mukaan mahdollista löytää globaali optimi. Seuraavassa esimerkissä muodostetaan lineaarinen optimointitehtävä ja ratkaistaan se. Lopuksi tarkastellaan löydetyn ratkaisun globaalisuutta. Kyseinen optimointiongelma on esitetty alunperin vuonna 2021 kurssilla Matemaattinen optimointi I [15].

Esimerkki 2.7. Puusepänliike valmistaa viisijalkaista sohvapöytää kahtena eri versiona. Perusmallin valmistaminen kestää 0.6 tuntia ja siitä saadaan 200 €/kpl. Luksusmallissa on peruskannen sijaan lasikansi, sen valmistaminen kestää 1.5 tuntia ja siitä saadaan 350 €/kpl. Seuraavan viikon aikana on käytettävissä 300 pöydänjalkaa, 50 peruskantta, 35 lasikantta ja 63 tuntia työaikaa. Muodostetaan optimointitehtävä, jolla saadaan maksimoitua pöytien valmistuksessa saatava tuotto, kun oletetaan että kaikki pöydät saadaan myytyä. Ratkaistaan tehtävä tarkasti CPLEX:illä. CPLEX on optimointiohjelma, jolla voidaan ratkaista muun muassa lineaarisia optimointitehtäviä [11].

Aloitetaan tehtävä määräämällä sille päätösmuuttujat. Olkoon muuttuja x peruspöytien lukumäärä ja muuttuja y luksuspöytien lukumäärä. Tehtävän sallitun joukon S määräävät sen rajoitteet. Luonnolliset rajoitteet ovat einegatiivisuusrajoitteet, koska perus- ja luksuspöytien lukumäärä ei voi olla negatiivinen. Eli siis $x \geq 0$ ja $y \geq 0$. Varsinaiset rajoitteet saadaan muista annetuista parametreista. Pöydissä olevien jalkojen kokonaismäärä ei voi ylittää 300, koska pöydänjalkoja ei ole sen enempää. Tästä saadaan rajoite $5(x + y) \leq 300$. Seuraavat rajoitteet $x \leq 50$ ja $y \leq 35$ saadaan pöydänkansien käytettävissä olevista määristä. Lisäksi yksi rajoite saadaan käytettävissä olevista työtunneista eli $0.6x + 1.5y \leq 63$.

Tehtävässä halutaan maksimoida pöytien valmistuksessa saatava tuotto eli merkitään tätä tavoitetta $200x + 350y$. Nyt saamme mallinnettua tehtävänannossa olevan ongelman lineaariseksi optimointitehtäväksi

$$\begin{aligned} \max \quad & 200x + 350y \\ \text{s.t.} \quad & 5(x + y) \leq 300 \\ & x \leq 50 \\ & y \leq 35 \\ & 0.6x + 1.5y \leq 63 \\ & x \geq 0, y \geq 0. \end{aligned} \tag{1}$$

Ratkaistaan optimointitehtävä CPLEX:illä (liite 1) ja saadaan kohdefunktion arvoksi $f = 16500$ sekä päätösmuuttujiksi $x = 30$ ja $y = 30$. Siis maksimoitaessa

pöydistä saatava tuotto, molempia pöytiä pitää tehdä 30 kappaletta. Tällöin saatu kokonaistuotto on 16500 €.

Tarkastellaan vielä löydetyn lokaalin optimin $(x, y) = (30, 30)$ globaalisuutta. Koska muodostettu optimointitehtävä (1) on lineaarinen ja täten konveksi, on ratkaistu lokaali optimi lauseen 2.6 mukaan myös globaali optimi.

2.3 Esimerkki: epälineaarinen optimointitehtävä

Epälineaarisissa optimointitehtävissä kohdefunktio tai jotkut rajoitteista ovat epälineaarisia. Epälineaariset funktiot eivät aina ole konvekseja, joten löydetty lokaali optimi ei välttämättä ole globaali optimi. Seuraavassa esimerkissä muodostetaan epälineaarinen optimointitehtävä ja ratkaistaan se. Lopuksi tarkastellaan löydetyn ratkaisun globaalisuutta. Kyseinen optimointiongelma on esitetty alunperin vuonna 2021 kurssilla Matemaattinen optimointi I [15].

Esimerkki 2.8. Yrityksen tehtävänä on valmistaa lieriön muotoinen puolen litran virvoitusjuomatölkki, johon kuluu mahdollisimman vähän materiaalia. Muodostetaan optimointitehtävä ja ratkaistaan se derivoimalla.

Aloitetaan tehtävä määräämällä päätösmuuttujat. Olkoon muuttuja r tölkin säde ja muuttuja h tölkin korkeus. Tehtävän sallitun joukon S määräävät sen rajoitteet. Luonnolliset rajoitteet ovat positiivisuusrajoitteet, koska säde tai korkeus eivät voi olla negatiivisia tai nollia. Eli siis $r > 0$ ja $h > 0$. Tehtävällä on lisäksi yksi varsinainen rajoite koskien tölkin tilavuutta. Suoran lieriön tilavuuden kaavan avulla saadaan rajoitteeksi $\pi r^2 h = 0.5$. Tehtävässä halutaan minimoida tölkkiin käytetyn materiaalin määrä eli käyttäen hyväksi suoran lieriön pinta-alan kaavaa saadaan kohdefunktioksi $2\pi r^2 + 2\pi rh$. Nyt saamme mallinnettua tehtävänannossa olevan ongelman epälineaariseksi optimointitehtäväksi

$$\begin{aligned} \min \quad & 2\pi r^2 + 2\pi rh \\ \text{s.t.} \quad & \pi r^2 h = 0.5 \\ & r > 0, h > 0. \end{aligned}$$

Ratkaistaan tehtävä seuraavaksi derivoimalla. Muunnetaan ensin tehtävän rajoite $\pi r^2 h = 0.5$ muotoon $\pi rh = \frac{0.5}{r}$. Muodostetaan tämän jälkeen kohdefunktio $f(r)$ sijoittaen siihen rajoitteen uusi muoto

$$f(r) = 2\pi r^2 + 2\pi rh = 2\pi r^2 + 2 \cdot \frac{0.5}{r} = 2\pi r^2 + \frac{1}{r} = 2\pi r^2 + r^{-1}.$$

Tämän jälkeen ratkaistaan kohdefunktion derivaatta

$$f^{'}(r) = 4\pi r - r^{-2} \tag{2}$$

ja sen nollakohta

$$r = \frac{1}{(4\pi)^{1/3}} \approx 0.4301 \text{ (dm)}.$$

Ratkaistaan tämän jälkeen muuttujan h arvo

$$\pi r h = \frac{0.5}{r} \Rightarrow h = \frac{0.5}{\pi r^2} = \frac{0.5}{\pi(\frac{1}{(4\pi)^{1/3}})^2} \approx 0.8603 \text{ (dm)}.$$

Viimeisenä voidaan laskea kohdefunktion arvo

$$f(\frac{1}{(4\pi)^{1/3}}) = 2\pi(\frac{1}{(4\pi)^{1/3}})^2 + (\frac{1}{(4\pi)^{1/3}})^{-1} \approx 3.4873 \text{ (dm}^2\text{)}.$$

Näin ollen optimaalisen tölkin säde $r \approx 4.3$ cm, korkeus $h \approx 8.6$ cm ja tällöin materiaalia kuluu $f \approx 34.9$ cm^2.

Tarkastellaan vielä löydetyn lokaalin optimin $(r, h) \approx (4.3, 8.6)$ globaalisuutta. Derivoimalla kaava (2) saadaan funktion f toinen derivaatta $f''(r) = 4\pi + \frac{2}{3}r$. Selvästi $f''(r) \geq 0$, joten Hessen matriisi $\mathbf{H}$ on positiivisemidefiniitti ja tällöin funktio f on lauseen 2.5 mukaan konveksi. Ratkaistu lokaali optimi on siis lauseen 2.6 mukaan myös globaali optimi.

3 Heuristiset ja metaheuristiset algoritmit

Matemaattisesti *algoritmi* tarkoittaa vaiheittaista prosessia laskelmien ja ohjeiden antamiseksi [27, s. 1]. Se on siis lista käskyjä ja operaatioita, jotka suoritetaan annetussa järjestyksessä. Algoritmi on yksityiskohtainen kuvaus siitä, miten tehtävä ratkaistaan tai kuinka ratkaisua approksimoidaan. Arkielämässä algoritmeja tulee vastaan esimerkiksi suunnistaessa paikasta toiseen navigaattorin avulla, luettaessa pesukoneen käyttöohjeita tai kokkaillessa reseptin mukaan. Myös esimerkiksi Facebookin toiminnan taustalla on algoritmi. Matematiikassa ja tietojenkäsittelytieteessä algoritmit ovat keskeinen käsite. Usein algoritmit kuvataan *pseudokoodilla*, jossa esitetään lukijalle oleelliset asiat. Esimerkkejä pseudokoodeista esitetään parviälyalgoritmien yhteydessä luvussa 4. Tietokoneelle annettu algoritmi on kuitenkin ohjelmointikielen ansiosta aina paljon yksityiskohtaisempi ja tarkempi kuin lukijalle annettu pseudokoodi. [6, s. 29] Erilaisia algoritmeja on olemassa todella monia. Tässä tutkielmassa keskitytään optimointialgoritmeihin, jotka generoivat uuden ratkaisun tunnetusta ratkaisusta iteroimalla. Optimointialgoritmeissa tavoitteena on löytää lopulta paras mahdollinen ratkaisu. [28, s. 8]

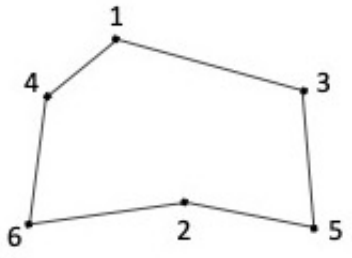
Heuristiikat ovat optimointialgoritmeja, mutta eroavat varsinaisista optimointialgoritmeista siinä, etteivät ne välttämättä tuota parasta mahdollista ratkaisua [20, s. 4]. Joskus optimointiongelmassa on järkevää luopua tavoitteesta löytää optimaalinen ratkaisu ja tyytyä etsimään riittävän hyviä ratkaisuja. Näin käy usein esimerkiksi suurien kombinatoristen optimointitehtävien kanssa. [23, s. 7] Kombinatorisia optimointitehtäviä ovat esimerkiksi diskreetit kokonaislukutehtävät, joissa ratkaisujen määrä saattaa kasvaa todella suureksi. Näin ollen vähänkään isommissa tehtävissä ei ole mahdollista ja tehokasta käydä kaikkia ratkaisuja läpi, vaan kannattaa soveltaa heuristisia tai metaheuristisia menetelmiä. Tunnetuimmat kombinatoriset optimointitehtävät, joiden ratkaisemiseen usein käytetään heuristiikkaa ovat kauppamatkustajan ongelma ja selkäreppuongelma. [5, s. 35]

Kauppamatkustajan ongelma on yksi tunnetuimmista reititysongelmista. Se on hyvin ymmärrettävissä oleva lyhimmän reitin ongelma, jota on tutkittu paljon. Kauppamatkustajan ongelmaksi kutsutaan tehtävää, jossa etsitään lyhin mahdollinen suljettu reitti, joka kulkee kerran kaikkien annettujen kaupunkien kautta ja palaa lopuksi takaisin lähtökaupunkiin. Kaupunkien väliset etäisyydet ovat tiedossa. [5, s. 164] Ongelma voidaan esittää graafina $G = (N, A)$, missä N on kaupunkeja esittävien solmujen joukko ja A kaupunkeja yhdistäviä teitä esittävien kaarien joukko. Jokaiseen kaareen $(i, j) \in A$ on liitetty arvo d_{ij}, joka on kaupunkien i ja j välinen etäisyys. Ongelmassa tarkoituksena on löytää solmujen $1, 2, \ldots, n, (n = |N])$ permutaatio π, jolle

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}$$

on pienin. [9, s. 17] Esimerkiksi kuvasta 1 nähdään kuuden kaupungin välillä kulkeva lyhin reitti, joka alkaa ja loppuu kaupungista 1. Kaupunkien välisiä etäisyyksiä ei ole yksinkertaisuuden vuoksi kuvattu.

Kauppamatkustajan ongelmalle on olemassa monia käytännön sovelluksia, sillä monet ongelmat voidaan mallintaa sopimaan kauppamatkustajan ongelmalla rat-



Kuva 1: Kauppamatkustajan ongelma kuudella kaupungilla.

kaistavaksi. Esimerkiksi ajoneuvojen reititys, piirilevyjen poraus, tietokoneiden johdotus ja robotin liikkeiden ohjaus ovat käytännön sovelluksia kauppamatkustajan ongelmasta. [18, s. 7] Lisäksi myöhemmin esiteltävillä luonnon inspiroimilla parviälyalgoritmeilla on läheinen yhteys kauppamatkustajan ongelmaan.

Selkäreppuongelma on hyvin yleinen kombinatorinen optimointiongelma, jossa jokaisella tavaralla on paino ja arvo. Valitut tavarat laitetaan reppuun, jolla on maksimipaino. Tarkoituksena on valita paras yhdistelmä tavaroita, joilla yhteenlaskettu arvo on suurin, mutta joiden yhteenlaskettu paino ei ylitä repun maksimipainoa. Selkäreppuongelma on aito lineaarinen kokonaislukuoptimointitehtävä, jossa on mukana vain yksi rajoite. Usein selkäreppuongelman päätösmuuttujat valitaan binäärisiksi, jolloin muuttuja saa arvon 0 tai 1. Seuraavassa esimerkissä esitellään eräs tapa ratkaista selkäreppuongelma käyttäen hyödyksi heuristiikkaa. [7, s. 15]

Esimerkki 3.1. Vaeltajan täytyy päättää mitkä neljästä mahdollisesta tavarasta hän pakkaa reppuunsa, kun repun maksimipaino on 20. Esineen i paino on w_i ja siitä saatava hyöty on u_i. Tarvittavat tiedot ovat taulukossa 1.

i	1	2	3	4
u_i	4	8	2	3
w_i	10	12	8	6

Taulukko 1: Tavaroiden painot ja arvot

Tehtävän tavoitteena on maksimoida pakattavien esineiden hyöty eli merkitään tätä tavoitetta $4x_1 + 8x_2 + 2x_3 + 3x_4$. Tehtävän sallitun joukon S määräävät sen rajoitteet. Tehtävässä on vain yksi varsinainen rajoite, joka saadaan repun maksimipainosta, eli kaikkien mukaan pakattavien esineiden paino on oltava korkeintaan 20. Merkitään tätä $10x_1 + 12x_2 + 8x_3 + 6x_4 \leq 20$. Luonnolliset rajoitteet koskevat päätösmuuttujien binäärisuusehtoa, jolloin saadaan $x_1, x_2, x_3, x_4 \in \{0, 1\}$. Mallinnetaan nyt tehtävä selkäreppuongelmaksi

$$\begin{aligned} \max \quad & 4x_1 + 8x_2 + 2x_3 + 3x_4 \\ \text{s.t.} \quad & 10x_1 + 12x_2 + 8x_3 + 6x_4 \leq 20 \\ & x_1, x_2, x_3, x_4 \in \{0, 1\}. \end{aligned} \tag{3}$$

Ratkaistaan tehtävä käyttäen hyötysuhteisiin perustuvaa heuristiikkaa. Lasketaan ensin jokaisen esineen hyötysuhde kaavalla $h_i = \frac{u_i}{w_i}$. Saadut hyötysuhteet ovat laskevassa suuruusjärjestyksessä $h_2 = 0.67$, $h_4 = 0.5$, $h_1 = 0.4$, ja $h_3 = 0.25$. Valitaan sitten esineet reppuun tässä järjestyksessä niin kauan kunnes maksimipaino on saavutettu. Valitaan esineet 2 ja 4. Tällöin paino on 18, joten esineet 1 ja 3 eivät enää mahdu. Paras mahdollinen hyöty saadaan siis kun valitaan esineet 2 ja 4. Tällöin repun paino on 18 ja esineiden kokonaishyöty 11. Koska optimointitehtävä (3) on lineaarinen ja täten konveksi, on saatu ratkaisu lauseen 2.6 mukaan globaali optimi. Optimointitehtävästä saadaan myös sama tulos CPLEX:illä (liite 2).

Heuristiikalla voidaan tarkoittaa siis joko *tavallista heuristiikkaa* tai *metaheuristiikkaa*. Tavallinen heuristiikka on tiettyyn ongelmaan soveltuva menetelmä, joka pyrkii riittävän hyvään ratkaisuun. Tämä saattaa toki joskus olla myös lokaali tai jopa globaali optimi. [20, s. 5] Yksi tavallisista heuristiikoista on *naapurihaku*. Siinä jotain olemassa olevaa ratkaisua parannetaan iteratiivisesti lähentyen jokaisella iteraatiolla parempaa ratkaisua ennalta määrättyjen operaatioiden mukaisesti, kunnes saavutetaan ratkaisu, jota ei voi enää tällä haulla parantaa. [23, s. 7]

Metaheuristiikat on yleisiä, ongelmasta riippumattomia menetelmiä, jotka pyrkivät yksinkertaisia sääntöjä noudattaen löytämään optimin. Ne ovat tehokkaita stokastisia iteraatiomenetelmiä, jotka sopivat monelle eri ongelmalle. *Stokastisuuden* eli satunnaisuuden ansiosta metaheuristiset menetelmät ohjaavat ja muokkaavat paikallista parantavaa hakua ja ylemmän tason heuristiikkoja hakuavaruuden tutkimisessa. Metaheuristiikassa hyvänä puolena on useamman ratkaisun käsitteleminen yhtäaikaa kullakin iteraatiokierroksella. [8, s. 14] Lisäksi metaheuristisilla menetelmillä ratkaistaessa sallitaan tietyin ehdoin jopa huonontavat askeleet. Näiden ominaisuuksien ansiosta metaheuristiikkojen on mahdollista löytää jopa globaali optimi. Metaheuristiikkoja on olemassa paljon ja niiden toimintatavat vaihtelevat. Tunnetuimpia menetelmiä ovat esimerkiksi tabuhaku, simuloitu jäähdytys ja geneettiset algoritmit, jotka esitellään seuraavissa aliluvuissa. [16, s. 129]

3.1 Tabuhaku

Tabuhaku on tehokas metaheuristinen menetelmä, joka toimii hyvin monissa vaikeissa diskreeteissä optimointiongelmissa. Haun on kehittänyt Fred Glover vuonna 1986. [4, s. 52] Perusideana tabuhaussa on estää siirrot takaisin jo aiemmin käytettyihin ratkaisuihin julistamalla ne väliaikaisesti tabuiksi eli kielletyiksi. Eteneminen nykyisestä ratkaisusta seuraavaan tehdään paikallisen haun avulla. Pääperiaatteena on pyrkiä pois lokaalista optimista sallimalla kohdefunktion arvoa huonontavia hakuaskelia. Palaaminen edellisiin ratkaisuihin estetään pitämällä tabulistaa, johon kirjataan viimeisin hakuhistoria. Listan avulla pystytään estämään lyhyellä aikavälillä ikuiset silmukat, koska joudutaan valitsemaan myös listaan kuulumattomia ratkaisuehdokkaita. Jokaisen siirron jälkeen lista päivitetään. [16, s. 129]

Tabuhaun suunnittelussa tärkeintä on hakuavaruuden ja naapuruston valinta. Hakuavaruudeksi voidaan valita esimerkiksi kaikki vain sallitut ratkaisut, mutta toisinaan on tarpeellista valita hakuavaruuteen myös ei-sallittuja ratkaisuja. Valitun hakuavaruuden rakenne voi mahdollistaa monta erilaista naapuruston rakennetta. [8, s. 18-19]

3.2 Simuloitu jäähdytys

Simuloitu jäähdytys on yksi yksinkertaisimmista ja tunnetuimmista metaheuristisista menetelmistä. Se on esitelty ensimmäisen kerran Kirkpatrickin ym. työssä [13] vuonna 1983. Menetelmä on saanut nimensä metallien lämpökäsittelystä, jossa on kaksi vaihetta. Ensin metalli kuumennetaan niin korkeaan lämpötilaan, että sen rakenne alkaa sulamaan. Tämän jälkeen metalli jäähdytetään hitaasti takaisin kiinteäksi, parantaen samalla sen lujuusominaisuuksia. Mikäli jäähdytys tehdään liian nopeasti, ei metallille ehdi muodostua lujaa rakennetta. [4, s. 2] Sama ilmiö on havaittavissa myös optimoinnissa. Liian nopeaa jäähdytystä vastaa optimointi, jossa hyväksytään vain parantavat ratkaisut. Tällöin voidaan edetä liian nopeasti lokaaliin optimiin ja tyytyä siihen, vaikka ympärillä olisi vieläkin parempia ratkaisuja. Hidasta jäähdytystä vastaa optimointi, jossa sallitaan tietyin ehdoin myös huonontavat ratkaisut. [8, s. 18]

Simuloidussa jäähdytyksessä systeemin tilaa vastaa ensin jokin ratkaisuehdokas. Iteraation edetessä tätä kyseistä ratkaisuehdokasta muutetaan eli valitaan jokin lähellä oleva ratkaisuehdokas nykyisen sijaan. Algoritmin alustuksessa valitaan alkulämpötila, jäähdytysfunktio ja kussakin lämpötilassa käytettävä iteraatioiden lukumäärä. Alussa jäähdytyksen lämpötila on korkea, eli voidaan suurella todennäköisyydellä tulla valitseneeksi nykyistä ratkaisuehdokasta huonompiakin ratkaisuja. Algoritmissa iteraatiokierroksella valittu ratkaisu hyväksytään aina, jos se on parempi kuin nykyinen ratkaisu. Nykyistä huonompi ratkaisu hyväksytään todennäköisyydellä, joka pienenee algoritmin edetessä. [8, s. 18] Lopulta löydetään jähmettynyt ratkaisu. Simuloidun jäähdytysmenetelmän tehokkuus riippuu jäähdytysnopeuden valinnasta. [5, s. 175-176]

3.3 Geneettiset algoritmit

Geneettiset algoritmit ovat biologista periytymistä jäljitteleviä optimointimenetelmiä, joiden tarkoituksena on pyrkiä löytämään optimiratkaisu käyttäen hyödyksi luonnonvalintaa [7, s. 2]. Algoritmin pääajatuksena on muodostaa suuri joukko mahdollisimman erilaisia ratkaisuja, jotka ilmaistaan joukkoina lukuja. Geneettisella algoritmilla ei ole yhtä tiettyä rakennetta, vaan se muovautuu optimointiongelman mukaan. Algoritmien eräs hyödyllinen ominaisuus on sen populaatiopohjaisuus. Tämä tarkoittaa sitä, että kokoajan käsitellään joukkoa ratkaisuja eli populaatiota, eikä keskitytä vain yhteen ratkaisuun kerrallaan. Tällöin voidaan etsiä useaa parempaa ratkaisua samanaikaisesti yhdellä iteraatiokierroksella. [8, s. 27] Geneettiset algoritmit eivät myöskään vaadi kohdefunktion tai rajoitteiden jatkuvuutta tai konveksisuutta, minkä takia niitä voidaan hyödyntää vaikeasti ratkaistaviin optimointiongelmiin [5, s. 163]. Edellisten lisäksi geneettisilla algoritmeilla hyödyllisenä ominaisuutena on vielä se, että populaation yksilöiden välillä esiintyy kommunikaatiota ja tietojen vaihtoa valinnan ja risteytyksen takia. Hyödyllisten ominaisuuksien lisäksi algoritmeilla on myös heikkouksia. Ratkaisujen tuottaminen voi vaatia paljon laskemista ja olla siten erittäin hidasta. Yleinen ongelma on myös se, että päätöksentekijälle voi olla vaikeaa löytää mieleinen ratkaisu suuresta ratkaisujen joukosta, varsinkin jos tavoitteita on useampi. [8, s. 27]

Geneettiset algoritmit koostuvat *genotyypeistä* ja *fenotyypeistä*. Genotyypit koos-

tuvat geeneistä ja ne koodaavat fenotyyppejä. Toisinaan genotyyppejä voidaan kutsua myös *kromosomeiksi*. Genotyypin voidaan ajatella esittävän usein binäärimerkkijonoa, kun taas fenotyyppi vastaa sitä lukua, jota binäärimerkkijono esittää. Esimerkiksi genotyypin 00110010 fenotyyppi on luku 2. Genotyypin sisällä olevaa sijaintia kutsutaan *lokukseksi* ja sijainnissa olevaa objektia *alleeliksi*. Esimerkiksi binäärimerkkijonossa geenillä voi olla kaksi eri alleelia, luvut 0 tai 1. Geneettisessa algoritmissa populaatiota ohjataan tietyillä valinnoilla kohti parempia ratkaisuja [5, s. 164]. Tunnetuimmat geneettiset pääoperaattorit ovat *valinta*, *risteytys* ja *mutaatio*, joita kaikkia tapahtuu luonnossa. Jokaista näiden operaattoreiden jälkeen saatua uutta populaatiota kutsutaan *sukupolveksi*. [7, s. 2]

Ensimmäisessä vaiheessa päätetään, miten yksilöitä valitaan seuraavaan vaiheeseen eli risteytykseen ja mutaatioon. Valintaprosessi on tärkeä vaihe, koska se määrittää, mitkä yksilöt tuottavat seuraavan sukupolven. Valintatapa määrittää yksilön todennäköisyyden tulla valituksi tuottamaan jälkeläisiä. Jotta sukupolvi paranisi, pitäisi paremmat ominaisuudet omaavat yksilöt valita suuremmalla todennäköisyydellä. Valintatapoja on olemassa monia. Yksinkertaisin tapa on *rulettivalinta*, jossa yksilöille annetaan mahdollisuus valikoitua seuraavaan vaiheeseen sen mukaan, kuinka hyvä vaihtoehto se on verrattuna muihin yksilöihin. Toinen tapa on käyttää *stokastista valintaa*, jossa yksilöt valitaan joukosta tasaisin välimatkoin. Kolmas vaihtoehto on *turnausvalinta*, jossa valitaan joukko yksilöitä ja verrataan niitä toisiinsa. Vertailun jälkeen paras yksilö etenee seuraavaan vaiheeseen. [7, s. 6-7] Monissa valintatavoissa on kuitenkin ongelmana se, ettei paras vaihtoehto välttämättä tule aina valituksi. Tällöin ratkaisuna toimii *elitismi*, jossa parhaat saavutetut ratkaisut säilytetään seuraavaan sukupolveen [8, s. 35-36]. Tällöin siis tietty määrä yksilöitä saa jatkaa seuraavaan vaiheeseen ilman varsinaisesti valituksi tulemista.

Valintojen jälkeen on aika siirtyä seuraavaan vaiheeseen eli yleensä suorittamaan risteytys ja mutaatio. Risteytyksellä tarkoitetaan kahden yksilön piirteiden yhdistämistä yhdeksi jälkeläiseksi. Risteytyksen tuloksena syntyneitä jälkeläisiä kutsutaan *lapsiksi* ja risteytettyjä yksilöitä *vanhemmiksi*. [7, s. 5-7] Risteyksen tavoitteena on tuottaa uusia lapsia, jotka toivon mukaan säilyttävät vanhempiensa hyvät ominaisuudet. Risteytystapoja on olemassa erilaisia riippuen siitä, mistä kohtaa yksilöiden kromosomit katkaistaan ja yhdistetään. *Yhden-kohdan* risteytystapa on katkaista vanhempien kromosomit kahtia samasta, satunnaisesti valitusta kohdasta ja liittää näistä saadut kromosomipätkät yhteen kahdeksi uudeksi lapseksi. Esimerkiksi jos yksilöt $A = 0000$ ja $B = 1111$ risteytetään puolesta välistä, saadaan lapsiksi 0011 ja 1100. Toinen tapa on *m-kohdan* risteytys, jolloin kromosomeista valitaan m-määrä kohtia risteytettäviksi. Esimerkiksi 2-kohdan risteytyksellä lokuksen 3 kohdalta molemmin puolin, saadaan edellä oleville yksilöille A ja B lapsiksi 0010 ja 1101. [7, s. 7-8] Risteyksessä on tärkeää huomata, että populaation yksilöt ovat tarpeeksi erilaisia, jotta niistä syntyvät jälkeläiset kehittyvät paremmiksi [16, s. 134]. Risteytyksen ongelmana on lopulta kuitenkin populaation suppeneminen niin, että kaikki populaation yksilöt alkavat muistuttamaan toisiaan [8, s. 30]. Tämän vuoksi tarvitaan myös toista operaattoria eli mutaatiota.

Mutaatiossa tarkoituksena on muokata valittujen yksilöiden kromosomeja niin, että sattumanvaraisuus säilyy ja monimuotoisuus ei katoa. Yleensä mutaatio on pieni ja riippuu kromosomin pituudesta. Mutaation jälkeen uusi kromosomi ei poikkea

paljoa alkuperäisestä, mutta saa aikaan tarvittavaa vaihtelua yksilöissä. Mutaation ansiosta syntyneiden jälkeläisten ei tarvitse siis olla täysin omien vanhempien kombinaatioita. Tämä on tärkeää, jotta yksilöt eivät liian aikaisin keskity lähelle vain yhtä optimia, kun läheltä saattaisi löytyä vieläkin parempi ratkaisu. Yleisin mutaatiotapa on *bitinvaihtomutaatio*, jossa binäärikoodatun kromosomin jokainen bitti vaihdetaan toiseen tietyllä todennäköisyydellä. Esimerkiksi binäärimerkkijono 0010 voisi mutaation jälkeen olla 1010. [8, s. 31]

4 Parviäly

Tyypillinen esimerkki ihmisten välisestä parvikäyttäytymisestä on taputtaminen teatteriesityksen päättyessä. Kukaan ei varsinaisesti kerro yleisölle, milloin taputtaminen alkaa ja loppuu, vaan se tapahtuu spontaanisti seuraten muita ihmisiä. Tätä kutsutaan *parviälyksi*. [12] Parviälyssä yksilöiden tekemistä päätöksistä syntyy toimiva kokonaisuus. Toinen tunnettu ihmisten parviälyyn liittyvä esimerkki on Sir Francis Galtonin vuonna 1907 tekemä härkäkoe. Kokeessa Galton pyysi 787 kyläläistä arvaamaan härän painon. Kukaan kyläläisistä ei arvannut painoa oikein, mutta kun Galton laski kaikkien kyläläisten arvaamista painoista keskiarvon, hän päätyi lähes täydelliseen arvoon. [31]

Samaan tapaan monien eläinlajien käyttäytymisessä voidaan havaita parveilua. Eläimien käytöstä ohjaavat tietyt yksinkertaiset säännöt esimerkiksi liikkuminen samaan suuntaan törmäämättä toisiin lähellä oleviin yksilöihin [20, s. 1]. Parvi toimii usein ilman johtajaa. Yksi kuuluisimmista lintumaailman parveilijoista on kottarainen. Kuten kuvasta 2 nähdään, kottaraiset voivat lentää jopa kymmenien tuhansien lintujen joukkona, parven ollessa läpimitaltaan kymmeniä tai satoja metrejä. Kottaraisparvessa parven jäsenet liikkuvat pienellä viiveellä toisiinsa nähden, saaden parven liikehdinnän näyttämään aaltomaiselta. [19]

Kuva 2: Kottaraisparven muodostama kuvio. Kuva Juha Rahkonen [19].

Parviälyn perustana on parven yksilöiden pyrkimys johonkin yhteiseen tavoitteeseen. Parven muodostavat yksilöt, jotka saavat jonkinlaista tietoa tavoitteesta ympäristöstä ja toisilta parven yksilöiltä. Saadessaan tiedon, ne muuttavat käyttäytymistään tietojen pohjalta. Parviälyn periaatteena on, että kokonaisuutena parvi voi löytää hyvän ratkaisun, vaikka parven yksilöillä yksinään olisi erittäin huono

kyky löytää hyvä ratkaisu. [20, s. 1]

Seuraavissa aliluvuissa esitellään aluksi muutamia eri eläinlajien inspiroimia parviälyalgoritmeja. Tämän jälkeen algoritmeja verrataan toisiinsa ensin sanallisesti ja lopuksi numeerisesti testaten.

4.1 Luonnon parviälyalgoritmit

Parviälyyn perustuvia algoritmeja on kehitetty tähän päivään mennessä kymmeniä erilaisia. Yleisimpiä niistä ovat jonkin tietyn eläinlajin parvikäyttäytymisen inspiroimina syntyneet algoritmit. Tutkielmaan on valittu eläinlajeista tarkasteltaviksi linnut ja kalat, muurahaiset, mehiläiset, lepakot, käet ja tulikärpäset. Näistä kolme ensimmäistä ovat vanhimpia ja tunnetuimpia, kun taas kolme viimeistä hieman uudempia algoritmeja. Esitellään aluksi jokainen algoritmi tarkemmin. Algoritmeista esitetyt pseudokoodit ovat muunnelmia Pro gradu -tutkielmasta *Parviälykkyys optimointiongelmien ratkaisemisessa* [23].

4.1.1 Partikkeliparvioptimointi

Partikkeliparvioptimointi (Particle Swarm Optimization) on James Kennedyn ja Russell Eberhartin vuonna 1995 kehittämä parviälyyn perustuva metaheuristinen optimointialgoritmi. Partikkeliparvioptimointi mallintaa luonnossa esiintyvää parvikäyttäytymistä esimerkiksi lintu- ja kalaparvissa. On tutkittu, että eläinten parvikäyttäytymisessä avainmekanismeja ovat parven jäsenten vuorovaikutukset, mikä on myös perusmekanismi partikkeliparvioptimoinnin pohjalla. [23, s. 22-23] Esimerkiksi iso kalaparvi muodostaa tehokkaan tavan etsiä ruokaa, kun jokainen parven kala muuttaa hakumallia jatkuvasti oman ja muiden kalojen kokemusten pohjalta [2, s. 1]. Samoin tapahtuu esimerkiksi lintujen lentäessä parvessa paikasta toiseen. Taidokkaat käännökset ja nopeat suunnanmuutokset syntyvät yleensä muutaman yksilön päätöksestä, mihin muut reagoivat.

Kuten geneettisessa algoritmissa, myös partikkeliparvioptimoinnissa on joukko ratkaisukandidaatteja, jotka yhdessä pyrkivät optimiratkaisuun [20, s. 5]. Partikkeliparvioptimoinnissa parvikäyttäytymistä mallinnetaan käyttäen perusyksiköinä hakuavaruudessa liikkuvia partikkeleita, jotka vastaavat optimointiongelman ratkaisukandidaatteja. Parviälyn tavoin yksi partikkeli on käytännössä hyödytön ja siksi parveen kuuluu monia partikkeleita. Merkitään parven kokoa eli partikkeleiden lukumäärää parvessa kirjaimella N. Parvi voidaan jakaa partikkelijoukkoihin, joita kutsutaan partikkelin ympäristöksi. Jokaisella partikkelilla on sijainti eli hakuavaruuden piste, joka tulkitaan vektoriksi. Lisäksi jokaisella partikkelilla on muisti, joka sisältää yleensä partikkelin parhaan aikaisemman sijainnin. Näiden lisäksi jokaisella partikkelilla on vielä nopeus, jonka mukaan ne liikkuvat. Toistensa ympäristössä olevat partikkelit ovat tietoisia toistensa sijainnista ja muistista. Liikkuessaan parvessa partikkelien on laskettava seuraava sijaintinsa niin, että ne ottavat huomioon oman sijaintinsa ja muistinsa lisäksi ympäristössä olevien muiden partikkelien optimaaliset sijainnit. [20, s. 9]

Algoritmi alkaa partikkelien sijainnin alustamisella ennen ensimmäistä iteraatiota. Yleisemmin alustus tehdään valitsemalla sijaintivektorit tasaisesti jakautuneesti jostain hakualueen osajoukosta. Hakualue on hakuavaruuden äärellinen osajoukko,

jonka sisältä optimaalista pistettä etsitään. Alustuksen jälkeen aloitetaan varsinainen iterointi eli partikkelien sijainnin päivitys. Partikkelit päivittävät sijaintiaan iteratiivisesti hakeutumalla kohti parasta tuntemaansa ratkaisua sekä parasta niiden naapuruston tuntemaa ratkaisua. Partikkelien nopeuden $\mathbf{v}_i$ ja sijainnin $\mathbf{x}_i$ päivitys tapahtuvat kaavoilla

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \boldsymbol{\varphi}_1 \otimes (\mathbf{p}_i - \mathbf{x}_i^t) + \boldsymbol{\varphi}_2 \otimes (\mathbf{p}_g - \mathbf{x}_i^t) \tag{4}$$

ja

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}, \tag{5}$$

missä $\mathbf{p}_i$ vastaa kyseisen partikkelin parasta tähän asti löytämää sijaintia ja $\mathbf{p}_g$ parhaan partikkelin sijaintia sen naapurustossa. Lisäksi $\boldsymbol{\varphi}_1 = c_1\mathbf{R}_1$ ja $\boldsymbol{\varphi}_2 = c_2\mathbf{R}_2$, missä $\mathbf{R}_1$ ja $\mathbf{R}_2$ ovat funktioita, jotka palauttavat arvoinaan lukuväliltä [0,1] poimittuja satunnaislukuvektoreita. Parametrit c_1 ja c_2 vastaavat kiihtyviskertoimina toimivia säätöparametreja. Operaattori $\otimes$ tarkoittaa alkioittain tehtävää vektorikertolaskua. Parametri t tarkoittaa iteraatiolaskuria. [23, s. 23] Partikkelin i liikenopeus $\mathbf{v}_i^{t+1}$ koostuu siis kaavan (4) mukaisesti kolmesta eri osasta: liikemäärästä sekä kognitiivisesta ja sosiaalisesta osasta. Liikemääräosa eli $\mathbf{v}_i^t$ kuvastaa partikkelin liikenopeutta edellisellä iteraatiokierroksella ja se ohjaa sen samaan suuntaan kuin se oli aikaisemmin matkalla. Kognitiivinen osa eli $\boldsymbol{\varphi}_1 \otimes (\mathbf{p}_i - \mathbf{x}_i^t)$ ohjaa partikkelia kohti parhaita sijainteja, joita se on jo aikaisemmin matkalla kartoittanut. Sosiaalinen osa eli $\boldsymbol{\varphi}_2 \otimes (\mathbf{p}_g - \mathbf{x}_i^t)$, ohjaa partikkelia kohti parven löytämiä sijainteja.

Jotta algoritmi pysähtyy äärellisen ajan kuluessa, sille pitää antaa myös lopetusehto. Lopetusehto on kriteeri, jonka jälkeen algoritmi pysähtyy ja antaa lopullisen vastauksen. Partikkeliparvioptimoinnissa voidaan käyttää lopetusehtona esimerkiksi iteraatioiden maksimimäärää C_{max}. Lopullinen vastaus on aina kaikkien parven partikkelien tuntemista ratkaisuista paras. [20, s. 9] Partikkeliparvioptimoinnin pseudokoodi on esitetty algoritmissa 1.

Algoritmi 1 Partikkeliparvioptimointi

Syötteet: Partikkeliparvi $\mathbf{x}_i$ $(i = 1, \ldots, N)$, kohdefunktio f, iteraatioiden maksimimäärä C_{max}
Tuloste: Paras löydetty ratkaisu $\mathbf{x}^*$

```
Alusta(x_i, v_i)
  while t < C_max do
    for i = 1, ..., N do
      if f(x_i) < f(p_i) then
        p_i = x_i
        p_g = max(N_i(p))
        PäivitäLiikenopeus(i)
        PäivitäSijainti(i)
      end if
    end for
  end while
```

Partikkeliparvioptimoinnin algoritmissa verrataan jokaisella iteraatiolla partikkelin nykyistä sijaintia kyseisen partikkelin parhaaseen tähän asti löytäneeseen sijaintiin. Jos nykyinen sijainti $\mathbf{x}_i$ on parempi kuin aiemmin tähän mennessä löydetty paras sijainti $\mathbf{p}_i$, korvataan se uudella sijainnilla ja päivitetään naapuruston parhaan partikkelin sijainti $\mathbf{p}_g$, jossa $N_i(p)$ ilmaisee naapuruston positiot. Tämän jälkeen toteutaan proseduurit *PäivitäLiikenopeus(i)* kaavalla (4) ja *PäivitäSijainti(i)* kaavalla (5).

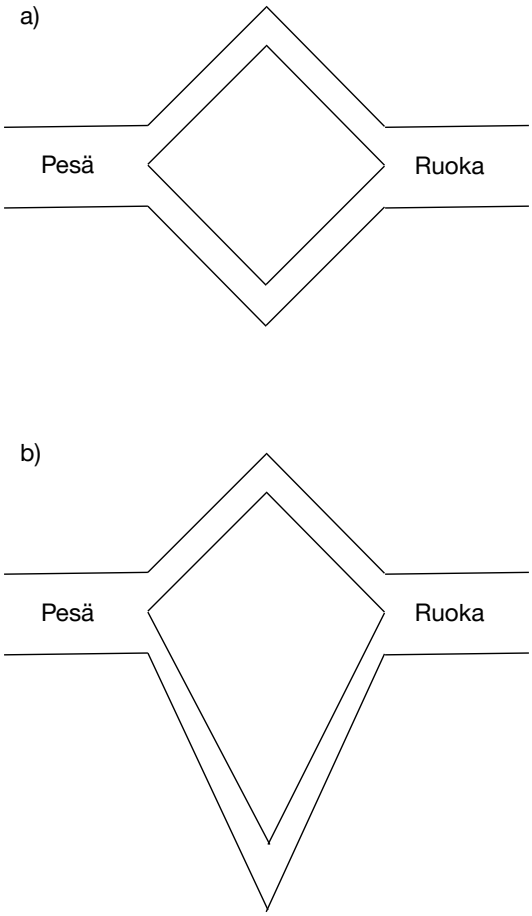
4.1.2 Muurahaisyhdyskuntaoptimointi

Muurahaisyhdyskuntaoptimointi (Ant Colony Optimization) jäljittelee nimensä mukaisesti muurahaisten käyttäytymistä. Kyseinen menetelmä on yksi vanhimmista luonnon inspiroimista optimointialgoritmeista ja sen kehitti Marco Dorigo vuonna 1991 [3]. Ensimmäinen menetelmä kehitettiin kombinatoristen optimointitehtävien ratkaisuun, samalla luoden muurahaisyhdyskuntaoptimoinnin rungon eli säännöt. Myöhemmin vuonna 1995 George Bilchev ja Ian C. Parmee loivat jatkuvien optimointitehtävien ratkaisuun tarkoitetun menetelmän, mutta sen sopimisesta Dorigon määrittelemiin muurahaisyhdyskuntaoptimoinnin sääntöihin kiisteltiin. Vasta vuonna 2005 Dorigo itse kehitti Krzysztof Sochan kanssa jatkuvien optimointitehtävien ratkaisuun soveltuvan menetelmän, joka täytti Dorigon alunperin kehittämän muurahaisyhdyskuntaoptimoinnin säännöt. [9, s. 3-4]

Muurahaisyhdyskuntaoptimoinnin inspiraation lähteenä ovat erityisesti ruokaa etsivät muurahaiset, jotka jättävät jälkeensä feromonijälkiä. Koska monien muurahaisten näkökyky on hyvin heikko, feromonijäljet ovat tärkeä osa niiden kommunikointia. Tunnettu feromoni on esimerkiksi polkuferomoni, jota muurahaiset käyttävät kulkemansa reitin merkitsemiseen. Kommunikointi feromonin avulla tarkoittaa sitä, että ruokaa etsivät muurahaiset seuraavat muiden jättämiä feromonijälkiä pesältä ruoan luo. Mitä enemmän feromonia reitillä on, sitä todennäköisemmin muurahaiset valitsevat kyseisen reitin. [9, s. 5] Tätä käyttäytymistä on tutkittu paljon muun muassa kaksoissiltakokeella, jonka on esittänyt Jean-Louis Deneubourg, Serge Aron, Simon Goss ja Jacques Pasteels vuonna 1990 [23, s. 13].

Kaksoissiltakokeessa muurahaisilla oli käytettävissä kaksi eri polkua pesältä ruoan luokse. Kuten kuvasta 3 nähdään, ensimmäisessä kaksoissiltakokeessa polut olivat yhtä pitkät kun taas toisessa kokeessa eripituiset. Ensimmäisen kokeen alussa muurahaiset valitsivat molemmat polut yhtä suurella todennäköisyydellä, koska feromoneja ei ollut vielä kertynyt kummallekaan polulle. Satunnaisen vaihtelun vuoksi kokeen edetessä yleensä toisen reitin valitsi hieman suurempi määrä muurahaisia, jolloin feromonia kertyi sille reitille enemmän. Suurimmassa osassa kokeita tämä johti lopussa siihen, että lähes kaikki muurahaiset käyttivät samaa polkua ja vain muutamissa kokeissa molemmilla poluilla oli yhtä paljon liikennettä, vaikka polut olivat saman pituisia. Toisessa kokeessa toinen poluista oli toista puolet pidempi. Samoin kuin ensimmäisen kokeen alussa muurahaiset valitsivat molemmat polut yhtä suurella todennäköisyydellä. Lyhyemmän polun valinneet muurahaiset ehtivät kuitenkin aikaisemmin ruoan luo kuin pidemmän polun valinneet. Palatessaan pesälle ne valitsivat jälleen lyhyemmän polun, koska siinä oli jo feromoneja. Alussa pidemmän polun valinneet muurahaiset eivät olleet vielä ehtineet ruoan luo, jolloin pidemmällä polulla ei ollut vielä feromoneja. Näin ollen pesälle palaavat muurahaiset valitsivat

aina todennäköisemmin lyhyemmän polun ja sille polulle alkoi kerääntyä enemmän feromonia. Kokeessa huomattiin, että ajan kuluessa suurin osa muurahaista valitsi lyhyemmän polun. [9, s. 5-7]



Kuva 3: Kaksoissiltakoe, jossa a) polut yhtäpitkiä b) polut eripituisia.

Muurahaisyhdyskuntaoptimointiin on kehitetty monia erilaisia algoritmeja. Usein algoritmit esitellään kauppamatkustajan ongelman avulla. Tämä johtuu siitä, että muurahaisyhdyskuntaoptimoinnin menetelmiä on helppo soveltaa kauppamatkustajan ongelmaan. Usein myös optimointimenetelmän hyvää käyttäytymistä kauppamatkustajan ongelmassa pidetään todisteena menetelmän käyttökelpoisuudesta. [9, s. 17] Kauppamatkustajan ongelma esiteltiin aiemmin heuristiikkojen yhteydessä luvussa 3. Muurahaisyhdyskuntaoptimoinnin runko voidaan sovittaa kauppamatkustajan ongelmaan esimerkiksi siten, että muurahaiset lähtevät jostakin kaupungista eli solmusta i ja valitsevat lähimmän naapurin menetelmällä seuraavan kaupungin j kunnes ovat käyneet jokaisessa kaupungissa kerran. Lähimmän naapurin menetelmässä kaupungista i kuljetaan aina lähimpänä olevaan kaupunkiin j, jossa ei ole vielä käyty [9]. Kaupunkien välissä tehdään aina feromonijälkien päivitys. Tämä tarkoittaa sitä, että ensin haihdutetaan feromonijälkeä kaikilta kaarilta, minkä jälkeen lisätään feromonia niille kaarille, joita juuri käytettiin. Feromonijälkien päivitys on

tärkeää, koska tällöin reittien etsintä ohjataan vierailemattomille kaarille. Parempi reitti löydetään todennäköisemmin uusia kaaria pitkin kuin seuraamalla jo vierailtuja kaaria. Feromonipäivityksen ansiosta muurahaiset eivät kulje siis kokoajan samoja reittejä, löytämättä ollenkaan uusia. [21, s. 31-32]

Ensimmäinen muurahaisyhdyskuntaoptimointiin kehitetyistä algoritmeista oli *muurahaisjärjestelmä (Ant System)* ja sen kehittämisestä vastasi Marco Dorigo vuonna 1992 [23, s. 17]. Algoritmista on olemassa kolme erilaista versiota, jotka eroavat toisistaan feromonipäivityksen osalta. Yleensä muurahaisjärjestelmällä tarkoitetaan algoritmia, jossa feromonipäivitys tehdään vasta kun kaikki muurahaiset ovat rakentaneet ratkaisunsa. Muurahaisjärjestelmässä N muurahaista asetetaan aluksi satunnaisesti valittuihin kaupunkeihin. Tämän jälkeen algoritmi koostuu kahden suoritusvaiheen iteroinnista. Ensimmäiseksi vaiheeksi kutsutaan *tilasiirtymäsääntöä*, jonka avulla muurahaiset siirtyvät seuraavaan kaupunkiin. Kaupungissa i sijaitseva muurahainen k valitsee kaupungin j todennäköisyydellä $p_k(i,j)$, joka saadaan kaavalla

$$p_k(i,j) = \begin{cases} \frac{\tau(i,j)^\alpha \eta(i,j)^\beta}{\sum_{l \in J_k(i)} \tau(i,l)^\alpha \eta(i,l)^\beta}, & \text{jos } j \in J_k(i) \\ 0, & \text{jos } j \notin J_k(i), \end{cases} \tag{6}$$

missä feromonijälki $\tau(i,j)$ kuvaa feromonin määrää kaarella (i,j). Lisäksi valitaan $\eta(i,j) = \frac{1}{d(i,j)}$, jossa $d(i,j)$ tarkoittaa kaupunkien i ja j etäisyyttä. Joukko $J_k(i)$ on niiden kaupunkiin i liittyvien kaupunkien joukko, joissa muurahainen k ei ole vielä käynyt. Säätöparametrit α ja β määrittelevät feromonijäljen ja heuristisen informaation eli etäisyyden suhteellista vaikutusta. Jos $\alpha = 0$, valitsevat muurahaiset seuraavan kaupungin lyhimmän etäisyyden perusteella. Jos taas $\beta = 0$, valitsevat muurahaiset seuraavan kaupungin vahvimman feromonijäljen perusteella. [21, s. 29]. Kun jokainen muurahainen on siirtynyt seuraavaan kaupunkiin kaavan (6) mukaisesti, päivittävät muurahaiset kaarille asetetut feromoniarvot. Tätä algoritmin toista vaihetta kutsutaan *feromonipäivityssäännöksi*. Sääntö toteutetaan muurahaisjärjestelmässä kaavoilla

$$\tau(i,j) = (1-\rho) \cdot \tau(i,j) + \sum_{k=1}^{N} \Delta\tau_k(i,j) \tag{7}$$

ja

$$\Delta\tau_k(i,j) = \begin{cases} \frac{1}{L_k}, & \text{jos } (i,j) \in T_k \\ 0, & \text{jos } (i,j) \notin T_k, \end{cases}$$

missä L_k on muurahaisen k rakentaman kierroksen pituus, $0 < \rho \leq 1$ feromonin haihtumista ohjaava säätöparametri ja T_k muurahaisen k löytämä ratkaisukandidaatti. Kaavassa (7) on siis mukana samanaikaisesti feromonien vähentäminen aikaisemmilta reiteiltä ja lisääminen juuri kuljetuille reiteille. Toistamalla näitä kahta vaihetta, tilasiirtymäsääntöä ja feromonipäivityssääntöä, päästään lopulta samaan lopputulokseen kuin kaksoissiltakokeessakin: lyhyille reiteille kertyy enemmän feromonia, jolloin muurahaiset valitsevat jatkossa todennäköisemmin lyhyempiä reittejä. [23, s.17-18] Muurahaisyhdyskuntaoptimoinnin pseudokoodi on esitetty algoritmissa 2.

Algoritmi 2 Muurahaisyhdyskuntaoptimointi

Syötteet: Muurahaispopulaatio $\mathbf{x}_i$ $(i = 1, \ldots, N)$, kohdefunktio f, iteraatioiden maksimimäärä C_{max}
Tuloste: Lyhin reitti T^+ ja lyhimmän reitin pituus L^+

```
Alusta(x_i, τ(i, j))
  while t < C_max do
    RakennaRatkaisut()
    LaskeReitinPituus()
    if ReittiLyhyempi() then
      Päivitä(T+)
      Päivitä(L+)
    end if
    PäivitäFeromonijäljet()
  end while
```

Muurahaisyhdyskuntaoptimoinnin algoritmissa jokaisella iteraatiolla jokainen muurahainen rakentaa oman ratkaisunsa proseduurilla *RakennaRatkaisut()* siirtymällä seuraavaan kaupunkiin kaavan (6) mukaan. Tämän jälkeen lasketaan jokaisen muurahaisen kulkeman reitin pituus proseduurilla *LaskeReitinPituus()*. Jos proseduurin *ReittiLyhyempi()* mukaan nykyinen reitti on lyhyempi kuin aiemmin tallennettu lyhyin reitti, päivitetään lyhin reitti T^+ ja lyhimmän reitin pituus L^+ proseduureilla *Päivitä*(T^+) ja *Päivitä*(L^+). Lopuksi päivitetään kaikkien reittien feromonijäljet proseduurilla *PäivitäFeromonijäljet()* kaavan (7) mukaan.

4.1.3 Mehiläisyhdyskuntaoptimointi

Mehiläisalgoritmit ovat mehiläisten inspiroimia algoritmeja, joista lähes kaikki ovat saaneet vaikutteita mehiläisten ravinnonhakukäyttäytymisestä luonnossa. Algoritmit eroavat toisistaan siinä, että ne käyttävät hyödyksi erilaisia mehiläisten käyttäytymisen ominaisuuksia. Näitä ominaisuuksia ovat esimerkiksi heilutustanssi, polarisaatio ja nektarin maksimointi, joita käytetään usein simuloimaan mehiläisten jakautumista ravinnonlähteille ja siten eri hakualueille. Osassa algoritmeista mehiläiset käyttävät hyödyksi myös feromoneja, kuten muurahaiset muurahaisyhdyskuntaoptimoinnissa. [28, s. 14] Tunnetuimpia mehiläisyhdyskuntaoptimointiin liittyviä algoritmeja ovat Phamin ym. vuonna 2006 kehittämä *mehiläisalgoritmi (Bee Algorithm)* ja Dervis Karabogan vuonna 2007 kehittämä *mehiläisyhdyskuntaoptimointialgoritmi (Artificial Bee Colony Algorithm)*. [23, s. 29] Mehiläisalgoritmissa mehiläiset jaetaan kahteen ryhmään: työ- ja tarkkailijamehiläisiin. Mehiläisyhdyskuntaoptimointialgoritmi on hieman kehittyneempi versio mehiläisalgoritmista, sillä siinä mehiläisryhmiä on kolme: työmehiläiset, tarkkailijamehiläiset ja partiolaiset. Kumpikin algoritmi on erittäin käyttökelpoinen erilaisten kombinatoristen optimointiongelmien ratkaisemiseen, aivan kuten muurahaispohjaiset algoritmitkin. [28, s. 14] Tässä tutkielmassa tarkastellaan tarkemmin mehiläisyhdyskuntaoptimointialgoritmia.

Mehiläisten ravinnonetsintäkäyttäytyminen luonnossa mallintaa muiden parvieläinten tapaan parviälykkyyttä. Keskinäisen vuorovaikutuksen avulla mehiläiset jakavat yhdyskunnassa tehtäviä ja reagoivat ympäristössä tapahtuviin muutoksiin.

Ravinnonetsintäkäyttäytymisen lisäksi niillä on paljon muitakin käyttäytymismekanismeja kuten esimerkiksi mehiläistanssi, tehtävien allokointi, kuningatarkäyttäytyminen, pesäpaikan valintamekanismi, parittelu, pölyttäminen ja navigointi. [23, s. 29-30] Mehiläisyhdyskuntaoptimointialgoritmissa mallinnettavana on erityisesti mehiläisten ravinnonetsintämekanismi. Mehiläiset etsivät ravintoaan luonnossa kasvien tarjoamasta nektarista, jonka ne muuttavat hunajaksi. Mehiläisyhdyskuntaoptimointialgoritmi koostuu kahdesta käyttäytymismuodosta ja kolmesta pääkomponentista. Käyttäytymismuodot ovat ravinnonhakuun värvääminen ja ravinnonlähteen hylkääminen. Kolme pääkomponenttia ovat:

1. Ravinnonlähde: mehiläiset arvottavat ravinnonlähteen arvon riippuen muun muassa sen hyödyntämisen helppoudesta, etäisyydestä pesästä ja sen sisältämän ravinnon laadusta.

2. Aktiiviset ravinnonetsijät: pitävät hallussaan tietoa käsittelemästään ravinnonlähteestä. Tietoon lukeutuu muun muassa lähteen sijainti, etäisyys ja käsiteltävyys. Aktiivisia ravinnonetsijöitä kutsutaan työmehiläisiksi ja ne jakavat informaatiota muun parven kanssa.

3. Passiiviset ravinnonetsijät: tehtävänä etsiä uusia hyödynnettäviä ravinnonlähteitä. Passiiviset ravinnonetsijät jaetaan tarkkailijamehiläisiin ja partiolaisiin. Tarkkailijamehiläiset tutkivat ympäristöään etsien uusia ravinnonlähteitä, kun partiolaiset odottavat pesässä tavoittaen ravinnonlähteitä muilta mehiläisiltä saadun tiedon perusteella.

Mehiläisten välinen kommunikointi ja tiedon jakaminen ravinnonlähtestä tapahtuu mehiläistanssin avulla. Tämä tarkoittaa käytännössä sitä, että mehiläisyhdyskunnalla on parven kommunikaatiolle varattu tanssialue, johon mehiläiset saapuvat löydettyään ravinnonlähteen. Mehiläistanssi toteutetaan tanssialueella lentämällä kahdeksikkoa. Tanssin muotoja on erilaisia ja sen perusteella mehiläiset välittävät tietoa toisilleen löytämistään ravinnonlähteistä. [23, s. 30] Tanssin muodosta mehiläiset saavat esimerkiksi tiedon, missä suunnassa ja kuinka kaukana ravinnonlähde sijaitsee. Tanssin saattaa nähdä kuitenkin vain osa seuraavalle ravinnonlähteelle lähtevistä mehiläisistä, joten ne hyödyntävät lentäessäään parviälyä lintuparvien tavoin. Muutaman mehiläistanssin nähneen mehiläisen johdolla koko parvi löytää siis perille, vaikkeivat kaikki mehiläiset itse tietäisi reittiä. [21, s. 11]

Luonnossa ravinnonetsintäprosessin alussa kaikki ravinnonetsijät ovat passiivisia. Ravinnonetsijästä tulee joko tarkkailijamehiläinen, joka alkaa tutkia ympäristöä ravinnonlähdettä etsien tai partiolainen, joka jää pesään seuraamaan mehiläistanssia ja värväytyy tämän jälkeen tanssilla kuvatun ravinnonlähteen hakuun. Kun mehiläinen löytää ravinnonlähteen, se säilöö siihen liittyvän informaation ja hyödyntää ravinnonlähdettä eli tutkii sen käyttömahdollisuuksia. Tämän jälkeen se palaa pesälle ja tekee jonkun kolmesta toimintavaihtoehdosta:

1. hylkää ravinnonlähteen, jolloin siitä tulee jälleen passiivinen ravinnonetsijä.

2. suorittaa mehiläistanssin, jolloin se värvää partiolaisia tälle ravinnonlähteelle.

3. jatkaa löytämänsä ravinnonlähteen hyödyntämistä, mutta ei vielä suorita mehiläistanssia.

Näitä mekanismeja hyödyntäen mehiläisyhdyskunta saavuttaa ravintoa etsiessään ominaisuudet, joihin perustuu parven parviälykkyysmäinen itseorganisointi ilman yksittäistä johtajaa. [23, s. 30-31]

Mehiläisyhdyskuntaoptimointialgoritmin alustus eroaa hieman edellä kuvatusta luonnossa tapahtuvasta ravinnonetsintäprosessista. Algoritmin alussa puolet yhdyskunnan mehiläisistä määritellään työmehiläisiksi ja jokainen työmehiläinen sijoitetaan yhteen satunnaisesti valittuun ravinnonlähteeseen. Alustuksessa ravinnonlähteelle määritellään laatu. Toinen puolikas yhdyskunnan mehiläisistä määritellään partiolaisiksi, jotka jäävät odottamaan mehiläistanssia. Ravinnonlähteiden laadunmäärityksen jälkeen työmehiläiset palaavat pesälle ja esittävät mehiläistanssin. Tanssin jälkeen työmehiläiset palaavat löytämälleen ravinnonlähteelle mukaansa saamien partiolaisten kanssa. Saapuessaan ravinnonlähteelle osasta partiolaisista tulee työmehiläisiä ja osasta tarkkailijamehiläisiä, jotka tarkkailevat ravinnonlähteen ympäristössä olevia muita mahdollisia ravinnonlähteitä. Algoritmissa korkeintaan yksi tarkkailijamehiläinen kerrallaan etsii uutta ravinnonlähdettä ja työmehiläisten sekä partiolaisten lukumäärä on sama. [23, s.31]

Mehiläisyhdyskuntaopimointialgoritmissa ravinnonlähteiden sijainnit vastaavat ratkaisukandidaatteja. Mehiläisryhmän koko vastaa ravinnonlähteiden eli täten siis myös ratkaisukandidaattien määrää. Alustuksessa generoidaan satunnaiset ratkaisukandidaatit eli ravinnonlähteet populaationa P, joka koostuu ratkaisukandidaattien joukosta kooltaan N. Merkitään ratkaisukandidaattien paikkoja n-ulottuvuuksisilla vektoreilla $\mathbf{x}_i$, $i = 1, ..., N$. Alustuksen jälkeen algoritmissa iteroidaan ratkaisuja $\mathbf{x}_i \in P$ siihen asti kunnes lopetusehto toteutuu. Lopetusehtona voidaan käyttää partikkeliparvioptimoinnin tapaan iteraatioiden maksimimäärää C_{max}. Algoritmin suorituksen aikana mehiläiset muokkaavat ratkaisupopulaatiota sitä mukaan, kun löytävät uusia ravinnonlähteitä. Mikäli uusi ravinnonlähde on parempi kuin aikaisempi, korvaa mehiläinen muistissaan aiemmin tallennetun ratkaisun uudella löytyneellä ratkaisulla. Partiolaiset toimivat samoin mikäli ne tulevat värvätyksi jonkun ravinnonlähteen hakuun. Partiolainen valitsee ravinnonlähteen i mehiläistanssissa todennäköisyydellä p_i, joka saadaan kaavalla

$$p_i = \frac{y_i}{\sum_{j=1}^{N} y_j}, \tag{8}$$

missä y_i työmehiläisen arvioiman ratkaisukandidaatin eli sijainnin i ravinnonlähteen sopivuusarvo. Sopivuusarvo on suoraan verrannollinen sijainnin j ravinnonlähteen arvoon. Uuden ratkaisun päivitys eli paremman ravinnonlähteen sijainnin $\mathbf{x}_i$ tallentaminen tapahtuu kaavalla

$$\mathbf{x}_i = \mathbf{v}_i + \phi_i(\mathbf{v}_i - \mathbf{v}_j), \tag{9}$$

missä satunnaislukuparametri ϕ_i on satunnaisluku väliltä $[-1, 1]$, joka ohjaa uuden ratkaisukandidaatin generointia tunnetun ravinnonlähteen $\mathbf{v}_i$ ympärillä. Algoritmin suoritusta voidaan ohjata kahdella säätöparametrilla, jotka ovat populaation koko N ja iteraatioiden maksimimäärä C_{max}. [23, s. 32-33] Mehiläisyhdyskuntaoptimoinnin pseudokoodi on esitetty algoritmissa 3.

Mehiläisyhdyskuntaoptimoinnin algoritmissa jokaisella iteraatiolla lähetetään aluksi työmehiläiset ravinnonlähteille proseduurilla *TyömehiläisetRavinnonlähteille()*. Jos työmehiläisten löytämän uuden ravinnonlähteen sijainti $\mathbf{x}_i$ on parempi

Algoritmi 3 Mehiläisyhdyskuntaoptimointi

Syötteet: Mehiläisyhdyskunta $\mathbf{x}_i$ $(i = 1, \ldots, N)$, kohdefunktio f, iteraatioiden maksimimäärä C_{max}
Tuloste: Paras löydetty ratkaisu $\mathbf{x}^*$

```
Alusta(x_i, y_i)
  while t < C_max do
    TyömehiläisetRavinnonlähteille()
    if f(x_i) < f(v_i) then
      PartiolaisetRavinnonlähteille()
      PäivitäRavinnonlähde()
      LähetäTarkkailijamehiläiset()
    end if
  end while
```

kuin aiemmin löydetyn ravinnonlähteen sijanti $\mathbf{v}_i$, työmehiläiset palaavat pesälle esittämään mehiläistanssin. Tämän jälkeen toteutetaan proseduurit *PartiolaisetRavinnonlähteille()* kaavalla (8) ja *PäivitäRavinnonlähde()* kaavalla (9). Lisäksi lähetetään tarkkalijamehiläiset tarkkailemaan ravinnonlähteen ympäristössä olevia uusia ravinnonlähteitä proseduurilla *LähetäTarkkailijamehiläiset()*.

4.1.4 Lepakkoalgoritmi

Lepakkoalgoritmi (Bat Algorithm) on aiemmin esiteltyihin algoritmeihin verrattuna suhteellisen uusi parviälykkyyteen perustuva metaheuristiikka. Sen kehitti Xin-She Yang vuonna 2010, tavoitteenaan yhdistellä aiemmin tunnetuista heuristiikoista hyviä puolia sekä minimoida niissä esiintyneet puutteet [23, s. 38]. Lepakkoalgoritmissa nimensä mukaisesti mallinnetaan lepakoiden käyttäytymistä. Lepakot käyttävät liikkuessaan kaikuluotaimen kaltaista navigointia, minkä avulla ne pystyvät havaitsemaan saaliin, välttämään esteitä ja paikantamaan yöpymispaikat pilkkopimeässä, jopa ilman muita aisteja. Navigoidessaan lepakot päästävät erittäin korkean äänen ja kuuntelevat kaikuja, jotka palaavat takaisin ympärillä olevista pinnoista. Lepakon lajista ja tilanteesta riippuen niiden päästämissä äänissä on eroja ja jopa eri taajuuksia [28, s. 14]. Yleensä lepakon päästämän ääneen taajuus on välillä 24 kHz $-$ 100 kHz eli niin korkea, että ihmiskorva harvoin kuulee sitä. Lepakon päästämän yksittäisen ääni-impulssin kesto on noin $5 - 20$ millisekuntia ja parhaimmillaan lepakot voivat päästää tällaisia ääni-impulsseja jopa 200 kertaa sekunnissa. Tämä tapahtuu yleensä silloin, kun lepakot lentävät saaliinsa lähellä. Pulssien äänenvoimakkuus voi olla korkeimmillaan 110 dB ja täten kantaa muutaman metrin päähän. Äänenvoimakkuus vaihtelee sen mukaan, kuinka lähellä lepakko on saaliista. [29, s. 3-4] Lepakoiden käyttäessä hyödyksi kaikuluotausta, ne tulkitsevat saatua tietoa muutamien eri menetelmien avulla. Näitä ovat muun muassa aikaviive äänen päästämisen ja kaiun vastaanottamisen välillä, kaikujen voimakkuuksien vaihtelut ja korvien keskinäisten ääniaistimusten väliset aikaerot. Kyseisten menetelmien ansiosta lepakot pystyvät rakentamaan ympäristöstään kolmiulotteisen kuvan ja siten tunnistamaan kohteen etäisyyden ja suunnan, mutta myös saaliin tyypin ja liikkumisnopeuden. [29, s. 4]

Lepakkoalgoritmi perustuu kaikuluotausperustaiseen mekanismiin ja sen mallin-

nuksessa pohjataan muutamaan taustaolettamukseen:

1. Kaikki lepakot käyttävät kaikuluotausta etäisyyden aistimiseen ja ne myös kykenevät erottamaan toisistaan ruoan, saaliin ja muut esteet.

2. Lepakot lentävät ympäristössään satunnaisesti seuraavilla ominaisuuksilla: nopeus $\mathbf{v}_i$, sijainti $\mathbf{x}_i$, äänen taajuus g_{min}, sekä äänen aallonpituus λ ja voimakkuus A_0. Lepakot päästävät ääniä kiinteällä taajuudella ja voivat säätää päästämänsä äänen aallonpituutta sekä äänen päästämisen tiheyttä suhteessa saaliin etäisyyteen. Äänen päästämisen tiheyttä merkitään parametrilla $r \in [0, 1]$.

3. Oletetaan, että äänenvoimakkuus vaihtelee välillä $[A_{min}, A_0]$, missä A_{min} tarkoittaa äänen minimivakioarvoa ja A_0 suurta positiivista vakioarvoa. Äänen taajuus vaihtelee välillä $[0, g_{max}]$.

4. Oletetaan, että taajuusaluetta $[g_{min}, g_{max}]$ vastaa tietty aallonpituuksien arvoalue $[\lambda_{min}, \lambda_{max}]$. Äänen aallonpituus λ saadaan kaavasta

$$\lambda = \frac{v}{g},$$

missä $v = 340$ m/s äänen nopeus ja g äänen taajuus. Siis esimerkiksi taajuusaluetta [25 kHz $-$ 150 kHz] vastaa aallonpituuksien alue [2 mm $-$ 14 mm]. [29, s. 4-5]

Lepakkojen liikettä mallinnetaan algoritmissa vektoreina, samaan tapaan kuin partikkeliparvioptimoinnissa. Lepakoiden käyttämän taajuuden g_i, nopeuden $\mathbf{v}_i$ ja sijainnin $\mathbf{x}_i$ päivitykset toteutetaan kaavoilla

$$g_i = g_{min} + (g_{max} - g_{min})\beta, \tag{10}$$

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + (\mathbf{x}_i^t - \mathbf{x}^*)g_i, \tag{11}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}, \tag{12}$$

missä parametri β on satunnaismuuttuja tasaisesti jakautuneelta lukuväliltä $[0, 1]$ ja $\mathbf{x}^*$ paras tähän hetkeen mennessä löydetty ratkaisu. [28, s. 15] Näiden liikkeiden ohella lepakoille generoidaan paikallisen satunnaiskulun avulla uudet sijainnit kaavalla

$$\mathbf{x}_{uusi} = \mathbf{x}_{vanha} + \epsilon A^t, \tag{13}$$

missä ϵ on satunnaisluku väliltä $[-1, 1]$ ja A^t koko lepakkopopulaatiosta laskettu keskimääräinen äänenvoimakkuus hetkellä t. Satunnaiskulun tavoitteena on etsinnän tehostaminen, joka saadaan aikaa säädettävien parametrien avulla. Esimerkiksi

kun lepakko löytää saaliin, sääntelyssä vähennetään äänenvoimakkuutta A ja kasvatetaan äänen päästötiheyttä r. Parametrien päivitykset toteutetaan kaavoilla

$$A_i^{t+1} = \alpha A_i^t, \tag{14}$$

$$r_i^{t+1} = r_i^0[1 - e^{-\gamma t}], \tag{15}$$

missä α ja γ kuvaavat päivitystä ohjaavia vakioparametreja. Kyseisille parametreille pätevät seuraavat ominaisuudet

$$A_i^t \to 0, \text{ kun } t \to \infty, \alpha \in (0, 1)$$

ja

$$r_i^t \to 0, \text{ kun } t \to \infty, \gamma > 0.$$

Huomionarvoista on kuitenkin muistaa, että parametreja päivitetään vain jos uusia ratkaisuja parannetaan eli mennään kohti optimaalista ratkaisua [29, s. 6-7]. Lepakkoalgoritmin pseudokoodi on esitetty algoritmissa 4.

Algoritmi 4 Lepakkoalgoritmi

Syötteet: Lepakkopopulaatio $\mathbf{x}_i$ $(i = 1, \ldots, N)$, kohdefunktio f, iteraatioiden maksimimäärä C_{max}
Tuloste: Paras löydetty ratkaisu $\mathbf{x}^*$

```
Alusta(x_i, v_i, g_i)
Alusta(r_i, A_i)
  while t ≤ C_max do
    PäivitäTaajuus()
    PäivitäNopeus()
    PäivitäSijainti()
    if β > r_i then
      SatunnaisKulku(x_i)
    end if
    EtsiSijanti()
    if β < A_i ja f(x_i) < f(x*) then
      HyväksyRatkaisut()
      VähennäÄänenvoimakkuus()
      KasvataPäästötiheys()
    end if
    x_* = Paras(x_i)
  end while
```

Lepakkoalgoritmissa jokaisella iteraatiolla päivitetään aluksi lepakon taajuus, nopeus ja sijainti eli toteutetaan proseduurit *PäivitäTaajuus()* kaavalla (10), *PäivitäNopeus()* kaavalla (11) ja *PäivitäSijainti()* kaavalla (12). Tämän jälkeen verrataan

valittua satunnaismuuttujaa lepakon päästämän äänen päästötiheyteen. Jos satunnaismuuttuja β on suurempi kuin äänen päästötiheys r_i, suoritetaan lepakon nykyisestä sijannista satunnaiskulku proseduurilla *SatunnaisKulku*($\mathbf{x}_i$). Tämän jälkeen generoidaan lepakolle uusi sijainti eli toteutetaan proseduuri *EtsiSijanti()* kaavalla (13). Jos tämän jälkeen satunnaismuuttuja β on pienempi kuin lepakon äänenvoimakkuus A_i ja lepakon nykyinen sijainti $\mathbf{x}_i$ parempi kuin paras aiemmin löydetty sijainti $\mathbf{x}^*$, hyväksytään ratkaisut proseduurilla *HyväksyRatkaisut()*. Tämän jälkeen toteutaan proseduurit *VähennäÄänenvoimakkuus()* kaavalla (14) ja *KasvataPäästötiheys()* kaavalla (15). Iteraation lopuksi tallennetaan parhaaksi löydetyksi sijanniksi $\mathbf{x}^*$ nykyinen sijainti $\mathbf{x}_i$ proseduurilla *Paras*($\mathbf{x}_i$).

4.1.5 Käkihaku

Käkihaku (Cuckoo Search) on yksi uusimmista luonnon inspiroimista metaheuristisista algoritmeista. Sen on kehittäneet Xin-She Yang ja Suash Deb vuonna 2009 [28, s. 17]. Käkihaku perustuu joidenkin käkilajien tyypilliseen pesäloisintakäyttäytymiseen. Pesäloisinnassa käki munii munansa toisten lintujen pesään, jolloin se itse säästyy munien hautomiselta ja hoivaamiselta. Käen muniminen tapahtuu usein samaan aikaan kuin pesän omistajan, jolloin sitä on vaikeampi havaita etukäteen. Käen munien kuoriutumisaika on yleensä myös lyhyempi kuin pesän omistajan. Kun käen munat kuoriutuvat, käenpoikaset poistavat muut toisen linnun kuoriutumattomat munat pesästä, jolloin niille riittää enemmän pesän omistajan hoivasta. [25, s. 6]

Pesäloisinnan lisäksi algoritmia tehostaa *Lévy-kävely (Lévy flight)*, joka kuvaa joidenkin lintu- ja hyöteislajien taipumusta liikkua luonnossa kerrallaan pitkiä yksittäisiä liikkeitä useiden, lyhyempien ja satunnaisempien liikkeiden jaksottelemana. Lévy-kävely on nimetty ranskalaisen matemaatikon Paul Lévyn mukaan. Se määrittelee satunnaiskävelyitä, joiden askelpituudet poimitaan Lévyn satunnaisjakaumasta. Satunnaisjakauman ominaispiirteisiin kuuluvat potenssilain mukaan paksuhäntäinen todennäköisyysjakauma. Luonnossa useilla elänlajeilla ja jopa joillakin ihmisheimoilla ilmenee liikkumisessa piirteitä, jotka mukailevat Lévy-kävelyä. [23, s. 42-43]

Käkihaun algoritmi perustuu kolmeen taustaolettamukseen:

1. Jokainen käki munii yhden munan kerrallaan ja valitsee munintapesän satunnaisesti, eli pesien, munien ja käkien lukumäärä on sama.

2. Parhaat pesät, joista syntyy laadukkaita munia, siirtyvät seuraaville sukupolville.

3. Käytettävissä olevien pesien lukumäärä on vakio. Pesän omistaja havaitsee käen munan todennäköisyydellä $p_a \in [0, 1]$, jolloin se voi joko poistaa munan pesästä tai hylätä pesän ja rakentaa kokonaan uuden.

Mallinnuksessa ratkaisukandidaatteja esittävät pesiin sijoitetut munat ja uusia ratkaisuja käen munat, joilla pyritään tarjoamaan parempia vaihtoehtoja valmiiksi tunnettuja ratkaisuja edustaville pesässä valmiiksi oleville munille. Algoritmissa käytetään Lévy-kävelyitä ratkaisujen generoinnissa. Lévy-kävelyt toimivat Markovin ket-

jujen mukaisella prosessilla, missä seuraava tila riippuu ainoastaan nykyisestä tilasta ja siirtymän todennäköisyydestä. Käkihaussa sijainnin $\mathbf{x}_i$ päivitys saadaan Lévy-kävelyn mukaan kaavalla

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \alpha \otimes \text{Lévy}(\lambda), \tag{16}$$

missä $\alpha > 0$ on ongelman kokoon suhteuttava säätöparametri, joka kuvaa askelmittaa. Yleensä valitaan $\alpha = 1$. Operaattori $\otimes$ tarkoittaa alkioittaista tuloa ja Lévy(λ) parametrilla λ Lévyn jakaumasta poimittua satunnaisvektoria. Lévyn jakaumalla on ääretön varianssi sekä keskiarvo ja jakaumaa kuvataan kaavalla

$$\text{Lévy} \sim u = t^{-\lambda},$$

missä $1 < \lambda \leq 3$. [23, s. 43] Hyvien tulosten saavuttamiseksi osa uusista ratkaisuista tulisi generoida parhaiden tunnettujen ratkaisujen ympärille, mutta huomattava osa myös satunnaisesti riittävän pitkillä etäisyyksillä parhaasta tunnetusta ratkaisusta. Tämä siksi, että hakumenettely toimisi oikein eikä jäätäisi jumiin ainoastaan lokaaleihin optimeihin. [25, s. 8] Käkihaun pseudokoodi on esitetty algoritmissa 5.

Algoritmi 5 Käkihaku

Syötteet: Isäntäpesien populaatio $\mathbf{x}_i$, $(i = 1, \ldots, N)$, kohdefunktio f, iteraatioiden maksimimäärä C_{max}
Tuloste: Paras löydetty ratkaisu $\mathbf{x}^*$

```
Alusta(x_i)
  while t ≤ C_max do
    käki =Lévy() {kaava (16)}
    F_i = f(käki)
    F_j = PoimiPesäSatunnaisesti(n)
    if F_i < F_j then
      j = i
    end if
    KorvaaPesätUusilla(p_a)
    SäilytäParhaat(x_i)
    Järjestä(x_i)
    x* = EtsiParas(x_i)
  end while
```

Käkihaun algoritmissa jokaisella iteraatiolla aluksi kukin käki päivittää oman sijaintinsa Lévy-kävelyn mukaan, jota vastaan proseduuri *Lévy()* kaavan (16) mukaisesti. Tämän jälkeen valitaan isäntäpesien sijainnit satunnaisesti proseduurilla *PoimiPesäSatunnaisesti(n)*. Jos nykyinen käen munan pesän sijainti F_i on parempi kuin isäntäpesän munan sijainti F_j, korvataan aiempi ratkaisu uudelle ratkaisulla. Tämän jälkeen isäntäpesän omistaja korvaa pesän uudella pesällä proseduurilla *KorvaaPesätUusilla*(p_a). Proseduureilla *SäilytäParhaat*($\mathbf{x}_i$), *Järjestä*($\mathbf{x}_i$) ja *EtsiParas*($\mathbf{x}_i$) päivitetään kokonaisratkaisu korvaamalla huonoimmat ratkaisut paremmalla ratkaisulla.

4.1.6 Tulikärpäsalgoritmi

Tulikärpäsalgoritmi (Firefly Algorithm) perustuu tulikärpästen vilkkumiseen ja käyttäytymiseen. Algoritmin on kehittänyt Xin-She Yang vuosina 2007-2008. [28, s. 16] Huolimatta nimestään tulikärpäset ovat todellisuudessa kovakuoriaisia ja kuuluvat samaan heimoon kiiltomatojen kanssa. Kiiltomatoihin verrattuna niillä on kuitenkin siivet, joilla ne pystyvät lentämään. Jokaisella tulikärpäsellä on vatsansa alla oma valo, joka loistaa pimeässä. Lajista riippuen tulikärpästen valonkäyttötarkoitukset vaihtelevat. Toiset käyttävät valoa houkutellakseen luokseen toisia tulikärpäsiä ja toiset puolestaan houkuttelevat valon avulla saaliita. Osa tulikärpäsistä saattaa käyttää valoa myös varoittaakseen muita esimerkiksi epämiellyttävästä mausta. [22, s. 221]

Algoritmin perusideana on käyttää tulikärpästen valoa kumppaneiden etsimiseen ja optimaalisen ratkaisun löytämiseen. Algoritmissa tulikärpästen liikkuminen tapahtuu valon avulla. Valon kirkkaudella eli luminanssilla on merkittävä rooli, koska kirkkaammin loistavat tulikärpäset houkuttelevat luokseen vähemmän kirkkaita yksilöitä parittelemaan. Tulikärpäsen liikkumisen aikana sen sijaintia päivitetään jatkuvasti parhaan sijainnin selvittämiseksi. Optimaalisen ratkaisun löytymisprosessiin liittyy siis tulikärpästen valon luminanssi, keskinäinen houkuttelevuus ja sijainnin päivitys. Pohjimmiltaan algoritmi perustuu kolmeen seuraavaan taustaoletukseen:

1. Tulikärpästen liike perustuu vain kirkkauteen. Tulikärpäset ovat yksisukuisia, joten yksi tulikärpänen houkuttelee muita tulikärpäsiä sukupuolesta riippumatta.

2. Houkuttelevuus on verrannollinen kirkkauteen ja molemmat pienenevät etäisyyden kasvaessa. Eli kun valitaan mitkä tahansa kaksi tulikärpästä, vähemmän kirkkaampi siirtyy aina lähemmäs kirkkaampaa tulikärpästä. Jos lähellä ei ole kirkkaampaa tulikärpästä, se liikkuu satunnaisesti ympäriinsä.

3. Tulikärpästen kirkkaus määräytyy kohdefunktion arvon mukaan. [22, s. 223]

Tulikärpäsen valon luminanssi I saadaan laskettua seuraavilla kaavoilla

$$I = I_0 e^{-\gamma r_{ij}^2} \tag{17}$$

ja

$$r_{ij} = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^2},$$

missä I_0 kuvaa tulikärpäsen valon maksimiluminanssia, kun $r = 0$. Parametri γ kuvaa valon absorptiokerrointa ja muuttuja r_{ij} tulikärpästen i ja j välistä etäisyyttä. Koska tulikärpäsen houkuttelevuus on verrannollinen toisen tulikärpäsen näkemään valon kirkkauteen, saadaan houkuttelevuudelle β etäisyydellä r seuraava kaava

$$\beta = \beta_0 e^{-\gamma r_{ij}^2}, \tag{18}$$

missä β_0 kuvaa maksimihoukuttelevuutta, kun $r = 0$. [24, s. 9-10] Näiden avulla saadaan tulikärpäsen i sijainnin $\mathbf{x}_i$ päivitykselle kaava

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \beta_0 e^{-\gamma r_{ij}^2}(\mathbf{x}_j^t - \mathbf{x}_i^t) + \alpha\epsilon_i^t, \tag{19}$$

missä tulikärpänen j on kirkkaampi yksilö kuin tulikärpänen i. Kaavan toinen termi kuvaa houkuttelevuutta kaavan (18) mukaan. Kolmas termi eli $\alpha\epsilon_i^t$ kuvaa satunnaistamista, missä α on satunnaistamisparametri ja ϵ on Gaussin jakaumasta tai tasaisesta jakaumasta hetkellä t valittu satunnaislukuvektori. Jos $\beta_0 = 0$ eli tilanne, jossa lähellä ei ole kirkkaampaa tulikärpästä, on tällöin kyseessä Lévy-kävely. Lévy-kävely esiteltiin aiemmin käkihaun yhteydessä luvussa 4.1.5. Toisaalta jos $\gamma = 0$, pelkistyy tulikäspäsalgoritmi partikkeliparvioptimoinniksi. [25, s. 13] Tulikärpäsalgoritmin pseudokoodi esitetty algoritmissa 6, joka on muunnelma lähteestä *Performance of Firefly Algorithmfor Null Positioning in Linear Arrays* [1, s. 386].

Algoritmi 6 Tulikärpäsalgoritmi

Syötteet: Tulikärpästen populaatio $\mathbf{x}_i, (i = 1, \ldots, N)$, kohdefunktio f, iteraatioiden maksimimäärä C_{max}
Tuloste: Paras löydetty ratkaisu $\mathbf{x}^*$
Alusta$(\mathbf{x}_i, \beta, I)$

```
while t < C_max do
  for i = 1, ..., N do
    for j = 1, ..., i do
      if I_j < I_i then
        PäivitäHoukuttelevuus()
        PäivitäSijainti()
      end if
    end for
  end for
end while
```

Tulikärpäsalgoritmissa jokaisella iteraatiolla verrataan tulikärpäsen i sijantia toisen tulikärpäsen j sijantiin. Jos tulikärpäsen valon luminanssi I_i on parempi kuin toisen tulikärpäsen valon luminanssi I_j, toteutetaan proseduurit *PäivitäHoukuttelevuus()* kaavalla (18) ja *PäivitäSijanti()* kaavalla (19).

4.2 Algoritmien vertailua

Luonnon inspiroimien parviälyalgoritmien esittelyn lisäksi on mielekästä tutkia algoritmien välisiä yhteneväisyyksiä ja eroja. Seuraavissa aliluvuissa vertaillaan aiemmin esiteltyjä algoritmeja ensin teoreettisesti ja lopuksi numeerisesti.

4.2.1 Teoreettinen vertailu

Kuten algoritmien esittelyistä käy ilmi, ovat kaikki esitellyt algoritmit samantyyppisiä geneettisten algoritmien kanssa. Jokaisessa algoritmissa korostuu populaatiopohjaisuus, joka mahdollistaa useamman ratkaisun etsimisen samanaikaisesti. Millään

algoritmeista ei ole merkittäviä rajoitteita, jonka takia niiden soveltaminen optimointiongelmiin on suoraviivaista. Lisäksi jokainen algoritmi hyödyntää yksilöiden välistä kommunikointia ja muistia. Kommunikaatiosta syntyy parviälyllistä käyttäytymistä, jota käytetään hyödyksi optimointiongelmien ratkaisemisessa. Muistin avulla voidaan tallentaa informaatiota ratkaisuvaihtoehdoista ja näin auttaa parhaan vaihtoehdon löytämisessä. [23, s. 3, 12] Näiden lisäksi algoritmeja yhdistävänä tekijänä voidaan pitää niiden inspiraatiota tiettyä eläinlajia kohtaan. Jokaisessa algoritmissa hyödynnetään eläinlajin tyypillistä käyttäytymistä luonnossa.

Algoritmien välisiä eroja ovat muun muassa niiden keskittyminen erilaisiin ominaisuuksiin ja eri ongelma-alueiden ratkaisemiseen. Esimerkiksi muurahaisyhdyskuntaoptimointi keskittyy lyhimmän reitin löytämiseen ja laskemiseen, kun muut algoritmit pyrkivät löytämään parhaan sijainnin muun muassa ravinnonlähteelle tai pesälle. Vaikka kaikki algoritmit ovat saaneet innoituksensa luonnosta, niiden mallinnuspohjana toimii erilainen toimintamekanismi. Partikkeliparvioptimointi mallintaa lintu- ja kalaparvissa tapahtuvaa parveutumiskäyttäytymistä ja muurahaisyhdyskuntaoptimointi muurahaisten ruuanhakua feromonien avulla. Mehiläisyhdyskuntaoptimointi mallintaa mehiläisten ravinnonhakua mehiläistanssien avulla ja lepakkoalgoritmi lepakoiden kaikuluotaukseen perustuvaa navigointia. Käkihaku puolestaan mallintaa käkien pesänloisintakäyttäytymistä ja tulikärpäsalgoritmi tulikärpästen vilkkumista etsiessään kumppania.

Kaikki tutkielmassa esitellyt algoritmit kuuluvat metaheuristiikkojen joukkoon. Koska metaheuristiikat ovat yleistasoisia, ongelmasta riippumattomia menetelmiä, niille on vaikeaa todistaa mitään konvergenssituloksia. Konvergenssituloksilla tarkoitetaan tässä kohtaa arvioita siitä, kuinka nopeasti algoritmi löytää lokaalin tai globaalin optimin. Konvergenssitulosten vaikean todistamisen takia algoritmeja ei pystytä vertailemaan täysin aukottomasti tai laittamaan paremmuusjärjestykseen. Algoritmien noudattamista tietyistä periaatteista johtuen voidaan niistä kuitenkin tehdä muutamia havaintoja. [9, s. 31]

Yksi merkittävä ero algoritmien välillä aiheutuu väistämättäkin niiden iästä. Mitä uudempi algoritmi on, sitä kehittyneempi se on yleensä edellisestä algoritmista. Esimerkiksi tutkielmaan valitut vanhimmat algoritmit partikkeliparvioptimointi, muurahaisyhdyskuntaoptimointi sekä mehiläisyhdyskuntaoptimointi häviävät uudemmille algoritmeille nopeudessa ja tehokkuudessa yksinkertaisesti siitä syystä, että uudet algoritmit ovat usein syntyneet näiden algoritmien pohjalta. Täysin uusien algoritmien lisäksi kyseisiä vanhimpia algoritmeja on jatkokehitetty, varioitu ja tehostettu runsaasti [23, s. 45]. Kehitettäessä uudempia algoritmeja on voitu korjailla aikaisempien algoritmien puutteita ja lisätä esimerkiksi satunnaistamiseen liittyviä parametreja.

Lähtökohtaisesti kaikki esitellyt algoritmit yrittävät suorittaa jonkinlaisen lokaalin tai globaalin haun. Jos haku on pääasiassa lokaalia, se lisää todennäköisyyttä juuttua lokaaliin optimiin. Jos haku taas keskittyy liikaa globaaliin hakuun, se voi hidastaa konvergenssia. [26, s. 12-13] Esitellyistä algoritmeista partikkeliparvioptimointi hyödyntää eniten lokaalia hakua. Tästä syystä sen yleisenä haittana on jumiutuminen liian nopeasti lokaaliin optimiin, jolloin mahdollinen globaali optimi voi jäädä löytymättä. [26, s. 7] Algoritmeista globaalia hakua taasen hyödyntävät erittäin tehokkaasti lepakkoalgoritmi, käkihaku ja tulikärpäsalgoritmi.

Lepakkoalgoritmin tärkeimmät edut seuraavat sen kolmesta ominaisuudesta. Nämä ominaisuudet ovat sen taajuusvaihteluiden käyttö, automaattinen zoomaus ja parametrien ohjaus. Taajuusvaihteluiden käyttö tarjoaa samoja toimintoja kuin partikkeliparvioptimoinnissakin. Automaattinen zoomaus tarkoittaa algoritmin kykyä zoomata automaattisesti alueelle, jossa sallittuja ratkaisuja on. Tämän seurauksena algoritmilla on nopeampi konvergenssinopeus verrattuna muihin algoritmeihin. Parametrien ohjaus tarkoittaa algoritmin kykyä vaihdella parametrien arvoja iteraation edetessä. Tämä tarjoaa tavan vaihtaa etsinnästä hyödyntämiseen, kun optimaalinen ratkaisu lähestyy. Lisäksi on analysoitu, että lepakkoalgoritmi täyttää globaalit konvergenssiominaisuudet oikeissa olosuhteissa ja algoritmi pystyy ratkaisemaan suuren mittakaavan ongelmia tehokkaasti. [30, s. 7]

Käkihaku täyttää globaalit konvergenssivaatimukset ja voi täten lähestyä globaalia optimia. Lisäksi käkihaulla on kaksi hakuominaisuutta, lokaali ja globaali haku. Tämä mahdollistaa sen, että hakuavaruutta voidaan tutkia tehokkaammin globaalissa mittakaavassa ja siten löytää globaali optimi suuremmalla todennäköisyydellä. Käkihaun lisäetuna on, että sen globaali haku käyttää Lévyn jakaumaa tavallisen satunnaisjakauman sijaan. Koska Lévyn jakaumalla on ääretön keskiarvo ja varianssi, käkihaku voi tutkia hakuavaruutta tehokkaammin kuin algoritmit, jotka eivät käytä Lévyn jakaumaa. [25, s. 8-9] Näiden ominaisuuksien ansiosta tutkimukset ovatkin osoittaneet, että käkihaku voi toimia merkittävästi paremmin kuin muut algoritmit monissa sovelluksissa [25, s. 12].

Tulikärpäsalgoritmilla on kaksi suurta etua muihin algoritmeihin verrattuna. Nämä edut ovat sen kyky jakautua osaparviin sekä kyky käsitellä useampaa lokaalia optimia samanaikaisesti. Kuten jo aiemmin esiteltiin, tulikärpäsalgoritmi perustuu houkuttelevuuteen, joka vähenee etäisyyden myötä. Tämä johtaa siihen, että parvi jakautuu automaattisesti osaparviin, joista jokainen osaparvi keskittyy lähelle tiettyä lokaalia optimia. Tämän ansiosta on mahdollista löytää melko nopeasti lokaalien optimien joukosta globaali optimi. Tästä syystä tulikärpäsalgoritmia pidetään erittäin sopivana epälineaarisiin multimodaalisiin optimointiongelmiin. Lisäksi tulikärpäsalgoritmissa olevat parametrit voidaan virittää ohjaamaan satunnaisuutta iteraatioiden edetessä, jolloin voidaan nopeuttaa konvergenssia säätämällä näitä parametreja. [25, s. 16-17]

4.2.2 Numeerinen vertailu

Käsitellään seuraavaksi muutamien tutkielmassa esiteltyjen algoritmien tehokkuutta ja luotettavuutta numeerisesti. Testaus on toteutettu kirjallisuuskatsauksena yhdistellen artikkelien *Personal Best Cuckoo Search Algorithm for Global Optimization* [10] ja *Modified Grey Wolf Optimizer for Global Engineering Optimization* [17] tuloksia. Yksinkertaisuuden vuoksi tutkituista algoritmeista käytetään jatkossa niiden virallisia lyhenteitä, jotka löytyvät taulukosta 2.

Usein algoritmien suorituskykyä testataan tunnetuilla matemaattisilla funktioilla, joiden globaali optimi tunnetaan. Tätä varten vertailuun on valittu 6 vertailufunktiota. Vertailufunktioiksi on valittu sekä unimodaalisia että multimodaalisia funktioita. Unimodaalisten funktioiden ($F_1 - F_3$) avulla voidaan vertailla algoritmien tehokkuutta, koska niillä on vain yksi globaali optimi eikä lainkaan lokaalisia optimeja. Valitut unimodaaliset funktiot ovat myös konvekseja. Multimodaalisilla

Algoritmin nimi	Lyhenne
Partikkeliparvioptimointi	PSO
Mehiläisyhdyskuntaoptimointi	ABC
Lepakkoalgoritmi	BA
Käkihaku	CS

Taulukko 2: Tutkittujen algoritmien lyhenteet

funktioilla ($F_4 - F_6$) taas on suuri määrä lokaaleja optimeja, joten niiden avulla voidaan vertailla algoritmien luotettavuutta eli kykyä löytää joukosta globaali optimi ja toisaalta kykyä välttää jumiutuminen lokaaliin optimiin. Valitut multimodaaliset funktiot ovat epäkonvekseja. [17, s. 3] Käytettyjen vertailufunktioiden matemaattinen muotoilu on esitetty taulukossa 3.

Vertailufunktio	Kaava	Arvoalue	f_{min}
Sphere	$F_1(x) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]$	0
Quartic	$F_2(x) = \sum_{i=1}^{n} i x_i^4$	$[-1.28, 1.28]$	0
Schwefel 2.22	$F_3(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \sum_{i=1}^{n} \lvert x_i \rvert$	$[-10, 10]$	0
Ackley	$F_4(x) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(cx_i)\right) + a + \exp(1)$	$[-32, 32]$	0
Rastrigin	$F_5(x) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	$[-5.12, 5.12]$	0
Griewank	$F_6(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600, 600]$	0

Taulukko 3: Vertailufunktiot lähteiden [10] ja [17] mukaan

Heurististen algoritmien ollessa stokastisia optimointimenetelmiä, niitä on iteroitava vähintään 10 kertaa, jotta on mahdollista saada tilastollisesti mielekkäitä tuloksia [17, s. 3-4]. Valituista lähteistä toisessa on käytetty iteraatiomääränä 1500 ja toisessa 3000. Suorituskyvyn mittarina käytetään viimeisen iteroinnin parhaan ratkaisun keskiarvoa ja keskihajontaa tai kaikkien iteraatioiden keskiarvoa. Vertailtavuuden vuoksi kummankin lähteen tuloksista on poimittu vain ne arvot, joissa dimensio on ollut 30.

Lähteestä [17] on valittu vertailuun PSO-, BA- ja CS-algoritmien tulokset. Testauksessa on käytetty 30 hakuagenttia ja 3000 iteraatiota jokaiselle algoritmille. Numeeriset tulokset on esitetty taulukossa 4 ja tuloksista on selkeyden vuoksi tummennettu paras ratkaisu. Tulokset osoittavat käkihaun olevan selkeästi luotettavin valituista algoritmeista. Jokaisella vertailufunktiolla tarkastellessa käkihaun tuottaman parhaan ratkaisun keskiarvo on lähimpänä globaalia optimia. Käkihaun jälkeen algoritmeista toiseksi paras on partikkeliparvioptimointi ja huonoin lepakkoalgoritmi kaikilla vertailufunktioilla. Huomionarvoista on se, että lepakkoalgoritmilla saadut tulokset jäävät jokaisella vertailufunktiolla hyvin kauas globaalista optimista ja eroavat täten käkihaun ja partikkeliparvioptimoinnin tuloksista merkittävästi. Vaikka lepakkoalgoritmilla onkin nopea konvergenssinopeus, se toimii paremmin suuremmissa optimointiongelmissa. Muuten saadut tulokset tukevat hyvin luvun

4.2.1 teoreettisessa vertailussa esiteltyjä havaintoja.

Vertailufunktio	Algoritmit	Keskiarvo	Keskihajonta
F_1	PSO	5.31E−11	9.76E−11
	BA	2.25E+04	6.67E+03
	CS	**2.89E-16**	3.33E−16
F_2	PSO	5.42E−02	1.76E−02
	BA	5.05E+00	1.62E+00
	CS	**4.67E-02**	2.15E−02
F_3	PSO	5.67E−07	1.26E−06
	BA	2.86E+03	9.44E+03
	CS	**5.92E-09**	5.43E−09
F_4	PSO	1.08E+01	9.17E+00
	BA	1.72E+01	1.11E+00
	CS	**9.97E-01**	4.91E−01
F_5	PSO	6.54E+01	1.42E+01
	BA	9.61E+01	3.25E+01
	CS	**3.38E+01**	6.97E+00
F_6	PSO	1.73E−02	2.05E−02
	BA	2.37E+02	8.13E+01
	CS	**8.61E-04**	3.85E−03

Taulukko 4: Numeeriset tulokset lähteen [17] mukaan

Lähteestä [10] on valittu vertailuun PSO-, ABC- ja CS-algoritmien tulokset. Testauksessa on käytetty 30 hakuagenttia ja 1500 iteraatiota jokaiselle algoritmille. Numeeriset tulokset on esitetty taulukossa 5 ja tuloksista on selkeyden vuoksi tummennettu paras ratkaisu. Saadut tulokset hajaantuvat enemmän eri algoritmien välille kuin taulukossa 4. Käkihaun tuottama parhaan ratkaisun keskiarvo on lähimpänä globaalia optimia vertailufunktioilla F_1, F_5 ja F_6. Partikkeliparvioptimoinnin tuottama parhaan ratkaisun keskiarvo on lähimpänä globaalia optimia vertailufunktioilla F_2 ja F_4. Mehiläisyhdyskuntaoptimoinnin parhaan ratkaisun keskiarvo on lähimpänä globaalia optimia vertailufunktiolla F_3. Näilläkin tuloksilla voidaan kuitenkin sanoa käkihaun olevan vertailtavista algoritmeista paras, mutta ei niin ylivoimaisesti kuin aiemmin taulukon 4 mukaan.

Saatuja tuloksia kahdesta esitellystä lähteestä voidaan vertailla myös keskenään, pitäen kuitenkin mielessä niissä käytetyt hieman erilaiset testaustavat esimerkiksi iteraatiomäärien osalta. Koska toisessa lähteessä käkihaku tuotti jokaisella vertailufunktiolla parhaan ratkaisun, on suhteellisen helppoa verrata sitä toisen lähteen tuloksiin. Vertailufunktioilla F_1, F_5 ja F_6 paras ratkaisu saadaan kummallakin testauksella käkihaulla. Jokaisessa näissä tarkempi kohdefunktion arvo saadaan lähteen [17] tuloksilla, jossa käytettiin 3000 iteraatiomäärää, mikä tuntuu loogiselta.

Vertailufunktioilla F_2, F_3 ja F_4 lähteessä [10] paras ratkaisu saadaan joko partikkeliparvioptimoinnilla tai mehiläisyhdyskuntaoptimoinnilla. Kun näitä verrataan samoilla vertailufunktioilla saatuihin käkihaun arvoihin, voidaan tehdä mielenkiin-

Vertailufunktio	Algoritmit	Keskiarvo	Keskihajonta
F_1	PSO	5.55E−07	1.51E−07
	ABC	9.83E−07	7.49E−08
	CS	**2.31E-08**	8.09E−09
F_2	PSO	**2.68E-03**	5.98E−04
	ABC	6.02E−02	6.87E−03
	CS	6.31E−02	2.99E−02
F_3	PSO	6.66E+00	5.78E+00
	ABC	**2.89E-05**	1.37E−05
	CS	3.92E−04	9.55E−05
F_4	PSO	**2.56E-04**	9.69E−04
	ABC	1.25E−03	5.27E−04
	CS	4.57E−04	1.13E−04
F_5	PSO	6.54E+01	8.75E+00
	ABC	1.95E+02	1.24E+01
	CS	**6.26E+01**	1.16E+01
F_6	PSO	1.52E−02	3.53E−03
	ABC	3.08E−02	2.06E−02
	CS	**8.84E-03**	5.09E−03

Taulukko 5: Numeeriset tulokset lähteen [10] mukaan

toinen havainto. Käkihaku onkin enää parempi vain vertailufunktiolla F_3, jossa sillä saadaan parempi ratkaisu kuin mehiläisyhdyskuntaoptimoinnilla. Kahden muun vertailufunktion kohdalla voidaan havaita, että partikkeliparvioptimoinnilla saadaan parempi ratkaisu kuin käkihaulla. Tämä voidaan selittää sillä, että partikkeliparvioptimointi toimii suhteellisen hyvin vielä pienissä optimointiongelmissa. Mikäli dimensiona olisi suurempi kuin 30, tilanne olisi luultavasti toinen.

Numeerisesti saatujen tulosten nojalla voidaan sanoa niiden olevan yhdenmukaisia kirjallisuuden havaintojen kanssa. Käkihaku on erittäin käyttökelpoinen algoritmi verrattuna muihin ja se löytää todennäköisimmin globaalin optimin. Pienissä ulottuvuuksissa partikkeliparvioptimointi ja mehiläisyhdyskuntaoptimointi antavat sille kuitenkin hyvän vastuksen. Lepakkoalgoritmi on hyvä, mutta se toimii paremmin suuremmissa optimointiongelmissa.

4.3 Luonnon inspiroimien algoritmien yleistä vertailua

Tutkielmassa esiteltyjen parviälyalgoritmien lisäksi on mielenkiintoista ottaa vertailuun vielä hieman suurempi luonnon inspiroimien algoritmien joukko. Seuraavaksi esitelty teoreettinen ja numeerinen vertailu on toteutettu kirjallisuuskatsauksena käyttäen lähteenä artikkelia *A Comparative Study of Common Nature-Inspired Algorithms for Continuous Function Optimization* [24]. Vertailuun on otettu mukaan 11 suosituinta ja yleisintä luonnon inspiroimaa algoritmia, jotka on valittu yli 120 metaheurististen algoritmien joukosta. Valitut algoritmit on lueteltu lyhenteineen

taulukossa 6.

Algoritmin nimi	Lyhenne
Partikkeliparvioptimointi	PSO
Mehiläisyhdyskuntaoptimointi	ABC
Lepakkoalgoritmi	BA
Käkihaku	CS
Geneettinen algoritmi	GA
Immuunialgoritmi	IA
Tulikärpäsalgoritmi	FA
Differentiaalievoluutio	DE
Gravitaatiohakualgoritmi	GSA
Harmaasusioptimointi	GWO
Harmoniahaku	HS

Taulukko 6: Algoritmien lyhenteet

Kuten on jo aikaisemmin todettu, vaikka luonnon inspiroimat algoritmit mallintavat erilaista populaatiokäyttäytymistä, ovat ne kaikki iteratiivisia menetelmiä. Tällöin niillä on väistämättäkin joitakin yhteisiä piirteitä. Yksi yhteinen piirre on satunnaisuus. Kaikki 11 algoritmia käyttävät satunnaisuutta muun muassa yksilöiden alustuksessa. Tämän lisäksi algoritmeihin on otettu mukaan myös muitakin satunnaistukseen liittyviä mekanismeja, mitkä voivat parantaa yksilöiden globaalia hakukykyä. Näitä mekanismeja ovat esimerkiksi mutaatio-operaattorit GA-, IA- ja DE-algoritmeissa, satunnaisparametrit PSO-, ABC-, BA-, FA-, DE-, GSA-, GWO- ja HS-algoritmissa sekä Lévy-kävely CS-algoritmissa. [24, s. 15]

Toinen ja kolmas yhteinen piirre koskee yksilöiden välistä vuorovaikutusta ja optimaalisuuden tavoittelua. Kaikissa 11 algoritmissa on havaittavissa tietojen jakamista toisille yksilöille joko suoraan tai epäsuorasti, mikä voi lisätä todennäköisyyttä löytää globaali optimi. Optimaalisuuden tavoittelu tarkoittaa sitä, että yksilöt etenevät koko ajan kohti optimaalista ratkaisua. Esimerkiksi GA-, IA- ja DE-algoritmeissa käytetään tähän tarkoitukseen risteytysoperaattoria. PSO- ja BA-algoritmeissa partikkelit ja lepakot käyttävät globaalia optimia sijainnin päivittämiseen. ABC-algoritmissa työ- ja tarkkailijamehiläiset käyttävät ravinnonlähteiden sopivuusarvoja uuden sijaintinsa päivittämiseen. FA-, GSA- ja GWO-algoritmeissa puolestaan sekoitetaan kahden tai useamman eri yksilön sijaintitietoja keskenään. HS-algoritmissa käytetään hyväksi harmoniamuistia. [24, s. 15]

Algoritmien väliset erot johtuvat esimerkiksi niiden oppimisstrategioista, topologisista rakenteista ja algoritmisten komponenttien vuorovaikutuksesta. Erilaiset oppimisstrategiat tarkoittavat algoritmien erilaisia tapoja päivittää ratkaisuaan. Esimerkiksi PSO-algoritmissa ratkaisua päivitetään partikkelin paikallisen optimin ja koko parven globaalin optimin mukaan, kun taas CS-algoritmissa kukin käki päivittää ensin oman ratkaisunsa satunnaisesti Lévy-kävelyn mukaan, jonka jälkeen kokonaisratkaisu päivitetään korvaamalla huonoimmat ratkaisut paremmalla ratkaisulla. [24, s. 17]

Topologisista rakenteista johtuvien erojen ansiosta algoritmit voidaan jakaa kahteen eri luokkaan: lokaaliseen ja globaaliseen naapurustotopologiaan. Lokaaliseen

naapurustotopologiaan kuuluvat GA-, DE-, ABC-, FA-, CS- ja IA-algoritmit, koska ne käyttävät ratkaisua päivittäessään hyväksi muutaman yksilön tietoja. Tällä tavoin parven yksilöt saavat tutkittua tehokkaasti lähialuetta, mutta globaalin optimin löytyminen voi olla hidasta. Muut eli PSO-, BA-, GWO-, GSA- ja HS-algoritmit kuuluvat globaaliseen naapurustotopologiaan, koska ne päivittävät ratkaisun ottaen huomioon kaikkien yksilöiden tiedot. Tämän etuna on nopea ja tehokas etsiminen, mutta samalla haittana liian aikaisin jumittuminen lokaaliin optimiin. [24, s. 18]

Algoritmisten komponenttien vuorovaikutukset tarkoittavat algoritmien osien välisiä vuorovaikutuksia. Usein nämä vaikuttavat algoritmien nopeuteen. Esimerkiksi PSO- ja BA-algoritmeissa välittyy jatkuvasti tietoa globaalista optimista, jolloin niillä on nopea konvergenssi. GA-, IA- ja DE-algoritmit taas välittävät tietoa valinta-, risteytys- ja mutaatio-operaattoreiden avulla, minkä takia niillä on hidas konvergenssi. [24, s. 18-19]

Algoritmien suorituskyvyn numeerinen vertailu on toteutettu 30 vertailufunktion avulla. Vertailufunktioiden mukana on sekä unimodaalisia ja multimodaalisia funktioita että hybridi- ja kokoonpanofunktioita. Algoritmit testataan kahdessa eri dimensiossa, 10 ja 50. Pieniulotteisessa avaruudessa iteraatioiden määrä on 1500 ja suuriulotteisessa 15 000. [24, s. 21] Yksinkertaisuuden vuoksi tästä tutkielmasta on jätetty pois lähteessä [24] esitetyt numeeriset tulokset ja keskitytty tulosten sanalliseen analysointiin.

Algoritmien suorituskykyä on verrattu kolmen eri analyysin avulla: tarkkuuden, vakauden ja parametrien herkkyyden vertailu, tehokkuuden vertailu ja ajoajan vertailu. Ensimmäisessä analyysissa vertaillaan algoritmien tarkkuutta ja vakautta sekä parametrien herkkyyttä. Pieniulotteisessa avaruudessa kaikki algoritmit pärjäävät vielä hyvin sekä tarkkuudessa että vakaudessa. Suuriulotteisessa avaruudessa kaikkien algoritmien tarkkuus ja vakaus huononee merkittävästi. Tällöin tarkkuudesta suhteellisen hyviä tuloksia saavat enää vain GSA-, DE- ja CS-algoritmit ja vakaudesta CS-, DE-, ABC- ja GSA-algoritmit. Tuloksista käy ilmi, että erityisesti DE- ja CS-algoritmit ovat tarkempia ja vakaampia kuin 9 muuta algoritmia. [24, s. 21-23]

Parametrien herkkyyden vertailussa saadaan tulokseksi, että DE-, CS-, HS-GSA-, GWO-, FA-, BA- ja IA-algoritmit ovat herkkiä parametriasetuksille suuriulotteisessa avaruudessa. Erityisen herkkiä näistä algoritmeista ovat DE- ja HS-algoritmit. Loput eli ABC-, PSO- ja GA-algoritmit ovat herkkiä parametriasetuksille sekä pieni- että suuriulotteisissa avaruuksissa. Näistä erityisen herkkä on PSO-algoritmi. [24, s. 24-25]

Toisessa ja kolmannessa analyysissa vertaillaan algoritmien tehokkuutta sekä ajoaikoja. Tulokseksi saadaan muun muassa, että FA- ja HS-algoritmeilla on huonoin optimointitehokkuus sekä pieni- että suuriulotteisissa avaruuksissa. PSO-, GSA- ja GWO-algoritmeilla taas on suurempi konvergenssinopeus sekä pieni- että suuriulotteisissa avaruuksissa. Ajoaikojen vertailussa saadaan tulokseksi, että DE- ja CS-algoritmit ovat nopeimpia kaikilla vertailufunktioilla sekä pieni- että suuriulotteisissa avaruuksissa. Hitaimmat algoritmit ovat FA- ja GSA-algoritmit. PSO-, GA-, BA-, GWO- ja HS-algoritmit ovat nopeita vain pieniulotteisessa avaruudessa. Tästä voidaan päätellä, että kaikki muut paitsi CS- ja DE-algoritmit sopivat hyvin vain pieniulotteisiin ongelmiin. [24, s. 25]

Analyysien pohjalta voidaan todeta, että ensinnäkin ne algoritmit, joilla on sel-

keä oppimisstrategia ovat suorituskyvyltään parempia kuin ne algoritmit, jotka perustuvat sattumanvaraisuuteen. Tämän johdosta CS- ja DE-algoritmeilla on selvästi parempi suorituskyky kuin muilla algoritmeilla. Ne tuottavat parhaimmat ratkaisut nopeimmin sekä pieni- että suuriulotteisissa avaruuksissa. [24, s. 26]

Vertailtaessa lähteestä [24] saatuja tuloksia aiemmin luvussa 4.2.2 saatuihin tuloksiin, voidaan todeta niiden olevan hyvin samansuuntaisia. Kummallakin testauksella käkihaku osoittautuu erittäin tehokkaaksi ja luotettavaksi algoritmiksi sekä pieni- että suuriulotteisissa avaruuksissa. Lisäksi muiden algoritmien tulokset ovat melko hyvin linjassa sen havainnon kanssa, että ne sopivat pääasiassa vain pieniulotteisiin ongelmiin. Ainoana vertailujen välisenä erona ovat partikkeliparvioptimoinnin ja mehiläisyhdyskuntaoptimoinnin sijoittuminen toisiinsa nähden. Aiemmin luvussa 4.2.2 saatujen tulosten nojalla voitiin partikkeliparvioptimoinnin sanoa olevan hieman tehokkaampi ja luotettavampi kuin mehiläisyhdyskuntaoptimoinnin, mutta tässä luvussa algoritmeista saadut tulokset ovat toistepäin. Ero ei kuitenkaan ole merkittävä ja voi hyvin johtua erilaisista testaustavoista ja parametriasetuksista.

5 Yhteenveto

Tutkielmassa käsiteltiin yleisesti optimointia, metaheuristiikkoja ja parviälyä sekä esiteltiin muutamia luonnon inspiroimia algoritmeja. Luonnon inspiroimat algoritmit ovat metaheuristisia optimointialgoritmeja, jotka ovat saaneet inspiraationsa luonnosta. Näistä algoritmeista valittiin erityisesti tarkasteluun eri eläinlajien inspiroimat parviälyalgoritmit. Tutkielman lopussa algoritmeja vertailtiin keskenään teoreettisesti ja numeerisesti.

Optimointia koskevassa luvussa määriteltiin optimointitehtäviin liittyviä käsitteitä koskien muun muassa lokaalia ja globaalia optimia. Lisäksi kerrattiin konveksisuuteen liittyviä käsitteitä ja tärkeitä ominaisuuksia. Optimoinnin ja konveksisuuden perusidea havainnollistettiin lineaarisen ja epälineaarisen optimointitehtävän esimerkkien avulla.

Kolmannessa luvussa käsiteltiin yleisesti heuristisia ja metaheuristisia algoritmeja. Heuristiikat ovat optimointialgoritmeja, jotka eivät välttämättä tuota parasta mahdollista ratkaisua, mutta ovat tehokkaita menetelmiä esimerkiksi suurten kombinatoristen optimointiongelmien ratkaisemisessa. Luvussa esiteltiin tunnettuina kombinatorisia optimointitehtävinä kauppamatkustajan ongelma ja selkäreppuongelma. Esimerkillä havainnollistettiin selkäreppuongelman ratkaiseminen heuristiikan avulla. Lisäksi luvussa esiteltiin metaheuristisina menetelminä tabuhaku, simuloitu jäähdytys ja geneettiset algoritmit.

Neljäs luku käsitteli parviälyä ja parviälyalgoritmeja. Parviälyalgoritmit ovat metaheuristisia optimointialgoritmeja, jotka ovat saaneet inspiraationsa luonnossa elävien eläinlajien parvikäyttäytymisestä. Tutkielmassa esiteltiin yhteensä kuusi parviälyalgoritmia: partikkeliparvioptimointi, muurahaisyhdyskuntaoptimointi, mehiläisyhdyskuntaoptimointi, lepakkoalgoritmi, käkihaku ja tulikärpäsalgoritmi. Lisäksi algoritmeja verrattiin keskenään teoreettisesti ja numeerisesti kirjallisuuskatsauksena. Lopuksi vertailuun otettiin mukaan myös muitakin luonnon inspiroimia algoritmeja. Vertailujen avulla saatiin tulokseksi käkihaun ja differentiaalievoluution olevan tehokkaimpia ja luotettavimpia algoritmeja verrattuna muihin, koska nämä tuottivat todennäköisimmin globaalin optimin.

Lähteet

[1] Ahammed J., Swathi A., Sanku D., Chakravarthy V.V.S.S.S., Ramesh H. : *Performance of Firefly Algorithmfor Null Positioning in Linear Arrays*. Proceedings of 2nd International Conference on Micro-Electronics, Electromagnetics and Telecommunications, pp.383-391. Springer Nature Singapore Pte Ltd, 2018.

[2] Dongshu Wang, Dapei Tan, Lei Liu : *Particle swarm optimization algorithm: an overview*. Springer-Verlag, Berlin, Heidelberg. 2017.

[3] Dorigo M., Maniezzo V., Colorni A. : *Positive feedback as a search strategy*. Report n. 91-016. Dipartimento di Elettronica, Politecnico di Milano, Milano, 1991.

[4] Gendreau M., Potvin J.-Y. : *Handbook of Metaheuristics*. Third edition. Springer, Switzerland, 2019.

[5] Haataja J., CSC - Tieteellinen laskenta OY : *Optimointitehtävien ratkaiseminen*. Picaset Oy, Helsinki. 2004.

[6] Haataja J., Heikonen J., Leino Y., Rahola J., Ruokolainen J., Savolainen V., CSC - Tieteellinen laskenta Oy : *Numeeriset menetelmät käytännössä*. Picaset Oy, Helsinki. 2002.

[7] Halme M. : *Geneettinen algoritmi ja sen sovellukset*. Kandidaatintutkielma. Tampereen yliopisto, 2021.

[8] Hartman E. : *Monitavoiteoptimoinnista ja evoluutioalgoritmeista – sovelluksena kauppamatkustaja- ja yhteysalusliikenneongelma*. Pro gradu -tutkielma, Turun yliopisto, 2011.

[9] Hietarinta L. : *Muurahaisyhdyskuntaoptimointi*. Pro gradu -tutkielma. Turun yliopisto, 2009.

[10] Hussain K., Salleh M. N. M., Prasetyo Y. A., Cheng S. : *Personal Best Cuckoo Search Algorithm for Global Optimization*. Artikkeli. International Journal on Advanced Science Engineering and Information Technology. Vol. 8 No. 4, pp. 1209-1217, 2018.

[11] IBM ILOG CPLEX Optimizer. Viitattu 26.10.2022. `https://www.ibm.com/analytics/cplex-optimizer`

[12] Kaaro J. : *Parvi päättää viisaasti*. Artikkeli. Tiede, 2012. Viitattu 29.9.2022. `https://www.tiede.fi/artikkeli/jutut/artikkelit/parvi_paattaa_viisaasti`

[13] Kirkpatrick S., Gelatt C.D., Vecchi M.P. : *Optimization by Simulated Annealing*. Science Vol. 220 No. 4598, pp. 671-680, 1983. Viitattu 19.10.2022. `http://www2.stat.duke.edu/~scs/Courses/Stat376/Papers/TemperAnneal/KirkpatrickAnnealScience1983.pdf`

[14] Mäkelä M. : *Konveksi analyysi ja optimointi*. Luentomoniste. Turun yliopisto, 2022.

[15] Mäkelä M. : *Matemaattinen optimointi I*. Luentomoniste. Turun yliopisto, 2018.

[16] Mäkelä M.: *Matemaattinen optimointi II*. Luentomoniste. Turun yliopisto, 2018.

[17] Mittal N., Singh U., Sohi S. B. : *Modified Grey Wolf Optimizer for Global Engineering Optimization*. Artikkeli. Applied Computational Intelligence and Soft Computing, 2016. Viitattu 24.11.2022.
https://doi.org/10.1155/2016/7950348

[18] Nenonen H. : *Kauppamatkustajan ongelman ratkaiseminen Kruskalin algoritmin avulla*. Pro gradu -tutkielma. Itä-Suomen yliopisto, 2019.

[19] Rahkonen J. : *Lintuparvien taidokas lentely kertoo parviälystä*. Artikkeli. Apu, 2019. Viitattu 31.8.2022.
https://www.apu.fi/artikkelit/lintuparvien-taidokas-lentely-kertoo-parvialysta

[20] Raiko V. : *Hiukkasparvioptimointialgoritmeista*. Pro gradu -tutkielma. Turun yliopisto, 2015.

[21] Suominen M. : *Parviälykkyys ja muurahaispohjaiset algoritmit*. Pro gradu -tutkielma. Itä-Suomen yliopisto, 2013.

[22] Tan Y. : *Swarm Intelligence - Volume 2: Innovation, new algorithms and methods*. The Institution of Engeneering and Technology, London/UK, 2018.

[23] Virolainen I. : *Parviälykkyys optimointiongelmien ratkaisemisessa*. Pro gradu -tutkielma. Tampereen yliopisto, 2015.

[24] Wang Z., Qin C., Wan B., Song W.W : *A Comparative Study of Common Nature-Inspired Algorithms for Continuous Function Optimization*. Artikkeli. Entropy. Vol. 23, No. 874, 2021. Viitattu 28.11.2022.
https://doi.org/10.3390/e23070874

[25] Yang X.-S. : *Cuckoo Search and Firefly Algorithm: Overview and Analysis*. Cuckoo Search and Firefly Algorithm. Springer, Switzerland, 2014.

[26] Yang X.-S : *Nature-Inspired Algorithms and Applied Optimization*. Springer, London/UK, 2018.

[27] Yang X.-S. : *Nature-Inspired Optimization Algorithms*. First edition. Elsevier, London/USA, 2014.

[28] Yang X.-S., Cui Z., Xiao R., Gandomi A. H., Karamanoglu M. : *Swarm Intelligence and Bio-Inspired Computation*. First edition. Elsevier, London/USA, 2013.

[29] Yang X.-S., Gandomi A. H : *Bat Algorithm: A Novel Approach for Global Engineering Optimization*. Engineering Optimization, Engineering Computations. Vol. 29 No. 5, pp. 464-483, 2012.

[30] Yang X.-S., He X. : *Bat Algorithm: Literature Review and Applications* Int. J. Bio-Inspired Computation. Vol. 5 No. 3, pp. 141-149, 2013.

[31] Yong E. : *The Real Wisdom of the Crowds*. Artikkeli. National Geographic, 2013. Viitattu 29.9.2022.
`https://www.nationalgeographic.com/science/article/the-real-wisdom-of-the-crowds`

Liitteet

Liite 1: CPLEX-koodit lineaarinen optimointitehtävä

LP-tiedosto:
maximize
200x + 350y
subject to
5x+5y<=300
x<=50
y<=35
0.6x+1.5y<=63
bounds
x>=0
y>=0
end

TXT-tiedosto:
display problem all
display solution variables *
display solution obj

CPLEX-ratkaisu NEOS-serverillä:
Dual simplex - Optimal: Objective = 1.6500000000e+04
Solution time = 0.00 sec. Iterations = 2 (0)
Deterministic time = 0.00 ticks (4.59 ticks/sec)
CPLEX> Maximize
obj1: 200 x + 350 y
Subject To
c1: 5 x + 5 y <= 300
c2: x <= 50
c3: y <= 35
c4: 0.6 x + 1.5 y <= 63
Bounds
All variables are >= 0.
CPLEX> Variable Name Solution Value
x 30.000000
y 30.000000
CPLEX> Dual simplex - Optimal: Objective = 1.6500000000e+04
CPLEX> CPLEX>

Liite 2: CPLEX-koodit selkäreppuongelma

LP-tiedosto:
maximize
4x1 + 8x2 +2x3 +3x4
subject to
10x1 + 12x2 + 8x3 + 6x4<=20
bounds
binary
x1 x2 x3 x4
end

TXT-tiedosto:
display problem all
display solution variables *
display solution obj

CPLEX-ratkaisu NEOS-serverillä:
MIP - Integer optimal solution: Objective = 1.1000000000e+01
Solution time = 0.00 sec. Iterations = 1 Nodes = 0
Deterministic time = 0.02 ticks (5.59 ticks/sec)
CPLEX> Maximize
obj1: 4 x1 + 8 x2 + 2 x3 + 3 x4
Subject To
c1: 10 x1 + 12 x2 + 8 x3 + 6 x4 <= 20
Bounds
0 <= x1 <= 1
0 <= x2 <= 1
0 <= x3 <= 1
0 <= x4 <= 1
Binaries
x1 x2 x3 x4
CPLEX> Variable Name Solution Value
x2 1.000000
x4 1.000000
All other variables matching '*' are 0.
CPLEX> MIP - Integer optimal solution: Objective = 1.1000000000e+01
CPLEX> CPLEX>