Muurahaisyhdyskuntaoptimointi

Lassi Hietarinta

Pro gradu -tutkielma
Helmikuu 2009

MATEMATIIKAN LAITOS
TURUN YLIOPISTO

TURUN YLIOPISTO
Matematiikan laitos

HIETARINTA, LASSI: Muurahaisyhdyskuntaoptimointi
Pro gradu -tutkielma, 104 s., 3 liites.
Sovellettu matematiikka
Helmikuu 2009

Muurahaisyhdyskuntaoptimointi jäljittelee nimensä mukaisesti muurahaisten, erityisesti ruokaa etsivien muurahaisten käyttäytymistä. Ruokaa etsivät muurahaiset jättävät käyttämilleen poluille feromoni-nimistä kemikaalia. Nämä feromonijäljet ovat hyvin tärkeitä muurahaisten kommunikoinnissa. Myös muurahaisyhdyskuntaoptimoinnin menetelmissä feromonijäljet ovat tärkeässä osassa. Niiden avulla menetelmät keskittyvät etsimään optimiratkaisua jo löydettyjen hyvien ratkaisujen lähettyviltä. Tämän tutkielman toisessa luvussa käsitellään muurahaisyhdyskuntaoptimoinnin perustana olevaa muurahaisten käyttäytymistä sekä muurahaisyhdyskuntaoptimoinnin menetelmien yleisiä piirteitä.

Ensimmäiset kombinatoristen tehtävien ratkaisuun tarkoitetut muurahaisyhdyskuntaoptimoinnin menetelmät kehitettiin 1990-luvun alussa. Siitä lähtien on yritetty kehittää myös jatkuvien tehtävien ratkaisuun soveltuvia muurahaisyhdyskuntaoptimoinnin menetelmiä. Ensimmäinen jatkuvien tehtävien ratkaisuun soveltuva muurahaisten käyttäytymiseen perustuva optimointimenetelmä onnistuttiin kehittämään vuonna 1995 ja nykyään näitä menetelmiä on jo monia erilaisia. Tämän tutkielman kolmannessa luvussa esitellään kombinatorisille ja viidennessä luvussa jatkuville tehtäville tarkoitettuja muurahaisyhdyskuntaoptimoinnin menetelmiä. Neljännessä luvussa esitetään niitä harvoja konvergenssitodistuksia, joita muurahaisyhdyskuntaoptimoinnin menetelmille on pystytty todistamaan.

Koska jatkuville tehtäville on nykyään olemassa monia erilaisia muurahaisten käyttäytymiseen perustuvia optimointimenetelmiä, ei voida helposti sanoa mikä menetelmistä on paras. Menetelmien paremmuusjärjestys riippuu optimoitavasta funktiosta ja menetelmissä käytettävien parametrien arvoista. Jatkuvien tehtävien optimointiin tarkoitettujen muurahaisyhdyskuntaoptimoinnin menetelmien toimivuutta on vertailtu tämän tutkielman luvussa kuusi.

Asiasanat: feromonijälki, jatkuva optimointitehtävä, muurahaisyhdyskuntaoptimointi.

Sisältö

1 Johdanto

Yleensä eteen tulevat ongelmat yritetään ratkaista parhaalla mahdollisella eli optimaalisella tavalla. Jotta arkipäivän ongelmia voitaisiin käsitellä matematiikan keinoin, pitää ongelmat jollain tavalla mallintaa matemaattisiksi optimointitehtäviksi. Tätä varten pitää valita esimerkiksi optimointitehtävän *kohdefunktio*, *muuttujat* ja arvot, joita muuttujat voivat saada. Optimointitehtävät ovat joko *maksimointi-* tai *minimointitehtäviä* riippuen siitä, halutaanko kohdefunktiolle mahdollisimman suuria vai pieniä arvoja. Muuttujien sallittujen arvojen perusteella optimointitehtävät voidaan jakaa *diskreetteihin* ja *jatkuviin tehtäviin*. Jos optimointitehtävän kaikki muuttujat kuvaavat esimerkiksi kappalemääriä, niin ne voivat saada vain ei-negatiivisia kokonaislukuarvoja eli muuttujien arvot pitää valita joukosta $\{0, 1, 2, ...\}$. Tällainen tehtävä kuuluu diskreetteihin optimointitehtäviin. Toinen esimerkki diskreeteistä optimointitehtävistä on tehtävä, jossa muuttujat voivat saada arvoja joukosta $\{0, 1\}$. Tällaista optimointitehtävää kutsutaan tarkemmin *binääriseksi optimointitehtäväksi*. Jos diskreetissä optimointitehtävässä kaikilla muuttujilla on äärellinen määrä sallittuja arvoja, kutsutaan tehtävää *kombinatoriseksi optimointitehtäväksi*. Binääriset optimointitehtävät kuuluvat siis kombinatorisiin optimointitehtäviin. Jos optimointitehtävän kaikki muuttujat kuvaavat esimerkiksi nestemääriä litroina, niin ne voivat saada ei-negatiivia reaaliarvoja eli muuttujien arvojen pitää vain olla suurempia kuin 0. Tällainen tehtävä on esimerkki jatkuvasta optimointitehtävästä. On myös olemassa optimoinitehtäviä, joissa osa muuttujista voi saada esimerkiksi vain tiettyjä kokonaislukuarvoja ja loput muuttujista voivat saada arvoja tietyiltä reaalilukuväleiltä. Tällaisia tehtäviä kutsutaan *sekalukuoptimointitehtäviksi*.

Kombinatoriset optimointitehtävät voitaisiin periaatteessa ratkaista käymällä läpi kaikki tehtävän sallitut ratkaisut ja valitsemalla niistä paras. Yleensä sallittuja ratkaisuja on kuitenkin niin paljon, että niiden kaikkien läpi käyminen veisi kohtuuttomasti aikaa. Jos optimointitehtävä ei ole kombinatorinen, ei kaikkia ratkaisuja voida edes periaattessa käydä läpi. Tällöin optimiratkaisun löytämiseksi pitää käyttää jotain *optimointialgoritmia*. Optimointialgoritmeja on kehitetty valtava määrä eikä niitä voida asettaa yleiseen paremmuusjärjestykseen vaan ratkaistavasta tehtävästä riippuu, mikä algoritmi toimii hyvin ja mikä huonosti. Optimointialgoritmeja voidaan luokitella monella eri tavalla. Ne voidaan esimerkiksi jakaa *tarkkoihin* ja *likimääräisiin*

algoritmeihin.

Tarkkojen algoritmien voidaan taata löytävän optimaalinen ratkaisu. Tämä on tietysti tavoiteltava ominaisuus, mutta optimointiongelman ollessa NP-vaikea (esimerkiksi [11] sisältää lisätietoa NP-vaikeista ongelmista) tarkat algoritmit vaativat pahimmassa tapauksessa eksponentiaalisen laskenta-ajan. Näin ollen NP-vaikeille tehtäville tarkat algoritmit ovat käytännössä käyttökelvottomia niiden vaatiman suuren laskenta-ajan takia.

Algoritmien tehokkuuden parantamiseksi pitää optimaalisuusvaatimuksesta luopua. Näin tehdään likimääräisissä algoritmeissa, joissa pyritään saamaan hyviä ratkaisuja polynomiaalisessa ajassa. Näitä algoritmeja kutsutaan usein *heuristiikoiksi.* Tarkoista algoritmeista saadaan likimääräisiä, kun ne keskeytetään esimerkiksi tietyn annetun ajan kuluttua. Likimääräiset algoritmit, joita ei saada tarkoista algoritmeista, voidaan jakaa kahteen luokkaan: *rakentaviin* ja *korjaaviin menetelmiin.*

Rakentavissa algoritmeissa lisätään alunperin tyhjään ratkaisuun iteratiivisesti uusia komponentteja ja näin rakennetaan lopullinen ratkaisu. Rakentavat algoritmit ovat tyypillisesti nopeimpia likimäärisiä algoritmeja, mutta niiden tuottamat ratkaisut ovat yleensä huonompia kuin korjaavien algoritmien tuottamat ratkaisut.

Korjaavissa menetelmissä aloitetaan jostain alkuratkaisusta ja sitä yritetään toistuvasti parantaa lokaaleilla muutoksilla. Käytännössä tämä suoritetaan määräämällä mahdollisille ratkaisuille *naapuriratkaisut.* Naapuriratkaisujen joukosta yritetään sitten löytää sen hetkistä ratkaisua parempi ratkaisu.

Rakentavissa ja korjaavissa menetelmissä on kuitenkin heikkoutensa. Joko ne tuottavat vain vähäisen määrän eri ratkuisuja tai ne voivat tuottaa ratkaisuina huonolaatuisia lokaaleja optimeja (toisin sanoen ratkaisuja, jotka ovat paikallisesti optimaalisia, mutta jäävät silti kauaksi tehtävän parhaasta ratkaisusta eli globaalista optimista, katso liite A). Metaheuristiikoissa yritetään välttää näitä ongelmia.

Edellä esitetyt jaottelut tehdään yleensä diskreettien optimointitehtävien ratkaisuun tarkoitetuille optimointialgoritmeille. Jatkuvien optimointitehtävien ratkaisuun tarkoitetut optimointialgoritmit jaetaan yleensä *lokaalin* ja *globaalin optimoinnin menetelmiin.* Metaheuristiikat kuulu-

vat globaalin optimoinnin menetelmiin. Lokaalin optimoinnin menetelmät päätyvät yleensä aloituspistettä lähimpänä olevaan lokaaliin optimiin (katso liite A). Lokaalin optimoinnin menetelmät voivat siis päätyä ratkaisuun, joka on optimointitehtävän huonolaatuinen lokaali optimi. Globaalin optimoinnin menetelmät yrittävät välttää näitä huonolaatuisia lokaaleja minimejä. Tämä vaatii tietysti enemmän laskenta-aikaa kuin se, että tyydyttäisiin ensimmäiseen löydettyyn lokaaliin optimiin. Joissain tapauksissa on kuitenkin tärkeää löytää globaali optimi. Tästä syystä globaalin optimoinnin menetelmiä onkin kehitetty paljon. Menetelmien tehokkuutta voidaan parantaa kehittelemällä sellaisia menetelmiä, joilla ratkaistaan vain tietyntyyppisiä tehtäviä. Globaalin optimoinnin menetelmät voidaan vielä jakaa *deterministisiin* ja *stokastisiin menetelmiin*. Deterministisissä menetelmissä käytetään yleensä hyväksi jotain erityisoletusta kohdefunktiosta. Ne on siis yleensä tarkoitettu tietyntyyppisten optimointitehtävien ratkaisuun. Kuten nimestä voi päätellä stokastisissa menetelmissä käytetään jotenkin hyväksi satunnaisuutta.

Toisin kuin heuristiikat, jotka ovat ongelmakohtaisia, metaheuristiikat ovat yleisiä algoritmien runkoja, joita voidaan suhteellisen pienillä muunnoksilla käyttää monen tyyppisten tehtävien ratkaisuun. Metaheuristiikat voidaankin ymmärtää yleisiksi heuristiikoiksi, jotka ohjaavat perustana olevaa heuristiikkaa kohti hakuavaruuden lupaavia alueita, joissa on hyviä ratkaisuja. Metaheuristiikat ovatkin parantaneet huomattavasti kykyä löytää hyviä ratkaisuja vaikeille optimointiongelmille kohtuullisessa ajassa.

Tässä tutkielmassa käsitellään *muurahaisyhdyskuntaoptimointia* (*ant colony optimization*), joka kuuluu metaheuristiikkojen laajaan joukkoon. Muita metaheuristiikkoja ovat muun muassa *simuloitu jäähdytys* [5, 19] ja *tabuhaku* [13, 14, 15]. Nimensä mukaisesti muurahaisyhdyskuntaoptimointi jäljittelee muurahaisten käyttäytymistä ja käyttää sitä hyväkseen optimoinnissa. Ensimmäisen muurahaisyhdyskuntaoptimoinnin menetelmän kehitti Marco Dorigo vuonna 1991 [10]. Tämä menetelmä on tarkoitettu kombinatoristen optimointitehtävien ratkaisemiseen. Samalla Dorigo loi niin sanotun muurahaisyhdyskuntaoptimoinnin rungon eli säännöt, jotka muurahaisyhdyskuntaoptimoinnin menetelmien pitää täyttää. Dorigon jälkeen monet muutkin kehittivät kombinatoristen optimointitehtävien ratkaisuun tarkoitettuja muurahaisyhdyskuntaoptimoinnin menetelmiä (esi-

merkiksi [4, 32]). Jatkuville ongelmille näiden menetelmien kehittäminen oli kuitenkin vaikeampaa. Ensimmäisen jatkuvien optimointitehtävien ratkaisuun tarkoitetun muurahaisten käyttäytymiseen perustuvan menetelmän loivat George Bilchev ja Ian C. Parmee vuonna 1995 [2]. Menetelmän luojien mielestä he olivat kehittäneet ensimmäisen jatkuvien optimointitehtävien ratkaisuun tarkoitetun muurahaisyhdyskuntaoptimoinnin menetelmän. Dorigo oli kuitenkin sitä mieltä, että tämä menetelmä ei ollut muurahaisyhdyskuntaoptimoinnin menetelmä, koska se ei aivan täyttänyt muurahaisyhdyskuntaoptimoinnin rungon edellyttämiä vaatimuksia. Yksi syy tähän oli se, että Bilchevin ja Parmeen menetelmässä muurahaisilla oli pesä, jota Dorigon säännöt eivät sallineet. Tämän jälkeen kehitettiin myös muita muurahaisten käyttäytymiseen perustuvia jatkuvien optimointitehtävien ratkaisuun tarkoitettuja menetelmiä, joista suurin osa ei täyttänyt Dorigon vaatimia ominaisuuksia (esimerkiksi [12, 26]). Dorigo itse kehitti Krzysztof Sochan kanssa uuden optimointimenetelmän vuonna 2005 [30]. Tämä oli yksi ensimmäisistä jatkuvien optimointitehtävien ratkaisuun tarkoitetuista menetelmistä, joka täytti Dorigon kehittämän muurahaisyhdyskuntaoptimoinnin rungon edellyttämät vaatimukset. Tämän jälkeenkin on kehitetty vain muutama Dorigon säännöt täysin täyttävä menetelmä, jolla voi ratkaista jatkuvia optimointitehtäviä (esimerkiksi [20, 21]).

Tämän tutkielman toisessa luvussa perehdytään hieman muurahaisten käyttäytymiseen ja esitellään muurahaisyhdyskuntaoptimoinnin perusperiaatteita. Kolmannessa luvussa esitellään tärkeimpiä kombinatoristen optimointitehtävien ratkaisemiseen tarkoitettuja muurahaisyhdyskuntaoptimoinnin menetelmiä. Neljäs luku käsittelee kolmannessa luvussa esiteltyjen menetelmien konvergenssiteoriaa. Viidennessä luvussa esitellään joitakin jatkuvien optimointitehtävien ratkaisemiseen tarkoitettuja muurahaisyhdyskuntaoptimoinnin menetelmiä. Samassa luvussa esitellään myös joitakin muurahaisten käyttäytymiseen perustuvia jatkuvien optimointitehtävien ratkaisuun tarkoitettuja menetelmiä, jotka eivät kuitenkaan täysin täytä muurahaisyhdyskuntaoptimoinnin menetelmille asetettuja vaatimuksia. Kuudennessa eli viimeisessä luvussa verrataan viidennessä luvussa esiteltyjen menetelmien toimivuutta optimoitaessa erilaisia testifunktioita.

2 Oikeat ja keinotekoiset muurahaiset

Muurahaisyhdyskuntaoptimointi jäljittelee muurahaisten käyttäytymistä ja käyttää sitä hyväkseen optimoinnissa. Tässä luvussa perehdytään muurahaisten käyttäytymiseen ja sen mallintamiseen.

2.1 Muurahaisten käyttäytyminen

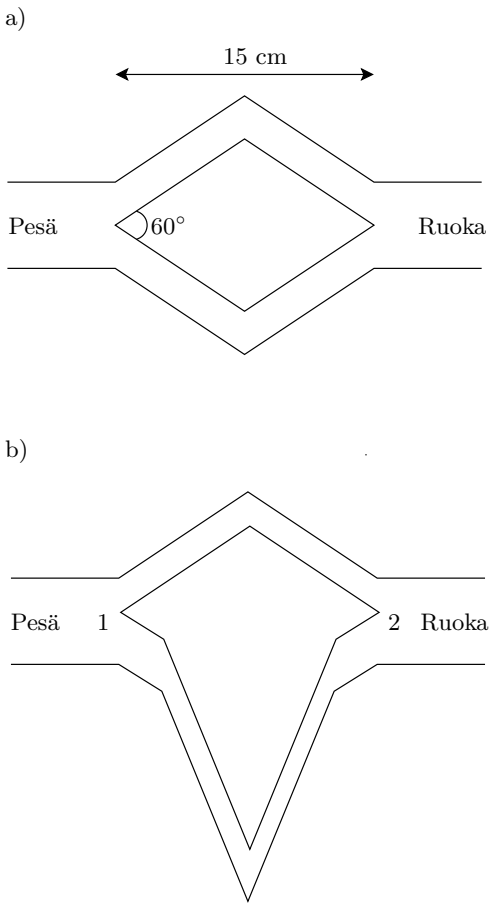
Ennen kuin itse muurahaisyhdyskuntaoptimoinnin metaheuristiikka esitellään, on hyvä tietää hieman sen biologisesta taustasta eli muurahaisten, erityisesti ruokaa etsivien muurahaisten käyttäytymisestä.

Monien muurahaislajien näkökyky on hyvin heikko ja eräät lajit ovat jopa täysin sokeita. Näin ollen niiden täytyy kommunikoida muiden aistien avulla. Kommunikointi perustuukin muurahaisten tuottamiin kemikaaleihin, joita kutsutaan *feromoneiksi*. Joillekin lajeille (esimerkiksi *Iridomyrmex humilis*) erityisen tärkeä on niin kutsuttu *polkuferomoni* (*trail pheromone*). Polkuferomonia muurahaiset käyttävät kulkemansa reitin merkitsemiseen. Aistimalla tätä feromonia, voivat ruokaa etsivät muurahaiset seurata muiden muurahaisten löytämää reittiä pesältä ruoan luo. Jos pesältä lähtee monta reittiä, joilla kaikilla on feromoneja, niin mitä enemmän feromonia reitillä on sitä todennäköisemmin muurahaiset valitsevat kyseisen reitin. Tämä muurahaisten kollektiivinen käyttäytyminen on ollut muurahaisyhdyskuntaoptimoinnin inspiraationlähteenä.

2.1.1 Kaksoissiltakokeet

Muurahaisten feromonien ohjaamaa käyttäytymistä on tutkittu erilaisilla kokeilla. Yksi tunnetuimmista tällaisista kokeista on Deneubourgin, Aronin, Gossin ja Pasteelsin [7, 17] esittelemä *kaksoissiltakoe* (*double bridge experiment*). Tässä kokeessa muurahaisilla on käytettävissään kaksi eri polkua pesältä ruoan luo. Koetta on suoritettu suhteen $r = \frac{l_p}{l_l}$ eri arvoilla. Tässä l_p on pidemmän polun pituus ja l_l lyhyemmän polun pituus.

Ensimmäisessä kokeessa $r = 1$ (katso kuva 2.1 a)) eli polut ovat yhtä pitkät. Alussa, kun muurahaiset eivät olleet vielä kulkeneet kummallakaan polulla, feromoneja ei ollut niissä lainkaan. Näin ollen muurahaiset valitsivat molemmat polut samalla todennäköisyydellä. Kuitenkin satunnaisen vaihtelun vuoksi yleensä toisen polun valitsi hieman suurempi määrä muurahaisia



Kuva 2.1: Kaksoissiltakoe, jossa a) polut ovat yhtä pitkät ($r = 1$) ja b) polut ovat eripituiset ($r > 1$).

kuin toisen. Koska muurahaiset jättivät kävellessään polulle feremoneja, niin feromonien määrä toisella polulla kasvoi suuremmaksi kuin toisella. Tämä taas johti siihen, että suurimmassa osassa kokeista ajan myötä lähes kaikki muurahaiset käyttivät samaa polkua ja vain muutamassa kokeessa molemmilla poluilla oli suunnilleen yhtä paljon liikennettä, vaikka polut olivatkin yhtä pitkiä.

Toisessa kokeessa $r = 2$ (katso kuva 2.1 b)) eli toinen poluista on kaksi kertaa niin pitkä kuin toinen. Samoin kuin ensimmäisessä kokeessa muurahaiset valitsivat alussa molemmat polut samalla todennäköisyydellä. Tällä kertaa lyhyemmän polun valinneet muurahaiset ehtivät ensin ruoan luo. Kun ne sitten palatessaan joutuivat taas valitsemaan kahden eri pituisen

polun väliltä, niin lyhyemmällä polulla oli jo feromoneja, mutta pidemmällä ei, koska aluksi pidemmän polun valinneet eivät olleet vielä ehtineet ruoan luo. Näin ollen pesälle palaavat muurahaiset valitsivat todennäköisemmin lyhyemmän polun ja lyhyemmälle polulle alkoi kerääntyä enemmän feromoneja kuin pitkälle. Kokeessa siis huomattiin, että ajan kuluessa suurin osa muurahaisista valitsi lyhyemmän polun. Toisaalta aina löytyi muutama muurahainen, joka valitsi pidemmän polun. Tämä voidaan tulkita eri polkujen tutkimiseksi.

On myös tehty koe, jossa muurahaisilla oli aluksi valittavanaan vain pitkä polku ja 30:n minuutin kuluttua kokeen alusta lisättiin lyhyempi polku. Lyhyemmän polun tullessa muurahaisille mahdolliseksi pidemmällä polulla oli jo feromoneja. Tämä johtaa siihen, että muurahaiset eivät kykenekään löytämään lyhyempää polkua. Jos feromonit haihtuisivat nopeammin, muurahaisilla olisi mahdollisuus unohtaa pidempi polku ja oppia optimaalinen polku.

2.1.2 Stokastinen malli

Artikkeleissa [7, 17] on myös esitelty yksinkertainen stokastinen malli, joka kuvaa muurahaisyhdyskunnan käyttäytymistä kaksoissiltakokeessa. Tässä mallissa ψ muurahaista sekunnissa ylittää polkujen haarautumiskohdan vakionopeudella v cm/s ja jättää yhden yksikön feromonia käyttämälleen polulle. Samoin kuin edellä l_p on pidemmän polun pituus (senttimetreinä) ja l_l lyhyemmän polun pituus (senttimetreinä). Nyt lyhyemmän polun kulkemiseen menee aikaa $t_l = \frac{l_l}{v}$ ja pidemmän polun kulkemiseen $t_p = \frac{l_p}{v} = \frac{l_p}{l_l} \cdot \frac{l_l}{v} = r \cdot t_l$. Feromonien määrää hetkellä t merkitään $\varphi_i^a(t)$, missä $i \in \{1, 2\}$ on polkujen haarautumiskohta ja $a \in \{l, p\}$ on polku (l merkitsee lyhyttä ja p pitkää polkua). Feromonien määrä $\varphi_i^a(t)$ on verrannollinen hetkeen t mennessä polkua a kulkeneiden muurahaisten määrään. Nyt saadaan todennäköisyydet, joilla haarautumiskohdassa i hetkellä t oleva muurahainen valitsee lyhyemmän polun ($p_i^l(t)$) ja pidemmän polun ($p_i^p(t)$):

$$\begin{aligned} p_i^l(t) &= \frac{[t_l+\varphi_i^l(t)]^\alpha}{[t_l+\varphi_i^l(t)]^\alpha+[t_l+\varphi_i^p(t)]^\alpha}, \\ p_i^p(t) &= \frac{[t_l+\varphi_i^p(t)]^\alpha}{[t_l+\varphi_i^l(t)]^\alpha+[t_l+\varphi_i^p(t)]^\alpha}. \end{aligned} \tag{2.1}$$

Näiden yhtälöiden funktionaaliset muodot sekä arvo α=2 on määrätty kokeellisesti [7].

Tässä mallissa feromonien haihtumista ei oteta huomioon, koska kokeellisesti on todettu, että feromonien haihtuminen on niin hidasta, ettei se käytännössä vaikuta muurahaisten käyttäytymiseen. Nyt stokastisen mallin kehittymistä voidaan kuvata seuraavilla differentiaaliyhtälöillä:

$$\begin{aligned} \frac{d\varphi_i^l}{dt} &= \psi p_j^l(t - t_l) + \psi p_i^l(t), \qquad (i = 1,\ j = 2 \text{ tai } i = 2,\ j = 1), \\ \frac{d\varphi_i^p}{dt} &= \psi p_j^p(t - r \cdot t_l) + \psi p_i^p(t), \quad (i = 1,\ j = 2 \text{ tai } i = 2,\ j = 1). \end{aligned} \tag{2.2}$$

Edellisten differentiaaliyhtälöiden määräämää dynaamista systeemiä on tutkittu Monte Carlo menetelmällä [24]. Tulokset ovat hyvin samantapaisia kuin saatiin kaksoissiltakokeessakin ja niinhän pitäisikin olla.

Tärkeää on huomata, että tässä mallissa muurahaiset jättävät feromoneja sekä eteenpäin (pesältä ruoan luo) että taaksepäin (ruoan luota pesälle) kulkiessaan. Tämä onkin osoittautunut välttämättömäksi lyhimmän polun löytämisen kannalta. Jos muurahaiset erittävät feromoneja vain eteenpäin tai vain taaksepäin kulkiessaan, ne eivät kykene löytämään lyhintä polkua.

2.2 Keinotekoiset muurahaiset

Keinotekoisten muurahaisten pitää pystyä aluksi samaan kuin oikeat muurahaiset eli löytämään kaksoissiltakokeessa lyhimmän polun pesältään ruoan luo käyttämällä kommunikointiin feromoneja.

2.2.1 Keinotekoiset muurahaiset ja kaksoissiltakoe

Ensin kaksoissiltakoe pitää esittää graafina (katso kuva 2.2 a)), jossa keinotekoiset muurahaiset liikkuvat. Oletetaan, että aika on diskreetti ja jokaisella aika-askeleella muurahaiset liikkuvat kohti naapurisolmua vakionopeudella yhden pituusyksikön yhdessä aikayksikössä. Oletetaan lisäksi, että muurahaiset liikkuvat lyhyemmän polun yhdessä aikayksikössä. Liikkuessaan muurahaiset jättävät yhden yksikön feromonia käyttämälleen kaarelle. Kun käytetään samoja merkintöjä kuin edellä, saadaan tässä mallis-

sa todennäköisyydet $p_i^l(t)$ ja $p_i^p(t)$ seuraavassa muodossa:

$$\begin{aligned} p_i^l(t) &= \frac{[\varphi_i^l(t)]^\alpha}{[\varphi_i^l(t)]^\alpha + [\varphi_i^p(t)]^\alpha}, \\ p_i^p(t) &= \frac{[\varphi_i^p(t)]^\alpha}{[\varphi_i^l(t)]^\alpha + [\varphi_i^p(t)]^\alpha}. \end{aligned} \tag{2.3}$$

Feromonien päivitys tapahtuu seuraavien kaavojen mukaisesti:

$$\begin{aligned} \varphi_i^l(t) &= \varphi_i^l(t-1) + p_i^l(t-1)m_i(t-1) + p_j^l(t-1)m_j(t-1), \\ \varphi_i^p(t) &= \varphi_i^p(t-1) + p_i^p(t-1)m_i(t-1) + p_j^p(t-r)m_j(t-r), \end{aligned} \tag{2.4}$$

missä $m_i(t)$ on muurahaisten lukumäärä solmussa i hetkellä t ja se saadaan muodossa

$$m_i(t) = p_j^l(t-1)m_j(t-1) + p_j^p(t-r)m_j(t-r). \tag{2.5}$$

Edellisissä yhtälöissä i=1, j=2 tai i=2, j=1.

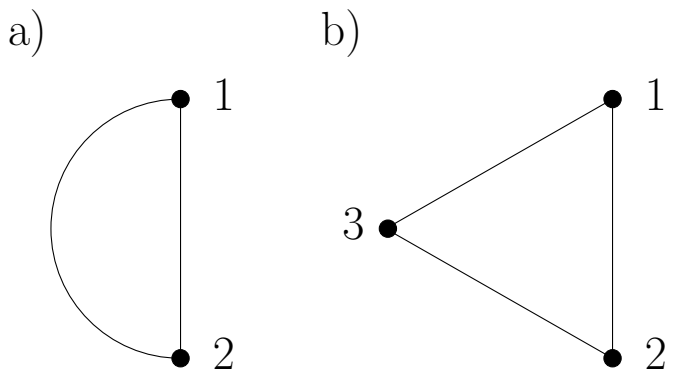
Tämä malli ei kuvaa yksittäisen muurahaisen käyttäytymistä, vaan koko systeemin keskimääräistä käyttäytymistä.

Kaksoissiltakoe voidaan esittää myös toisella tavalla graafina (katso kuva 2.2 b)). Tässä graafissa kaikki kaaret ovat yhtä pitkiä ja pidempi polku esitetään monena kaarena. Nyt feromonien päivitys tehdään yhden aikayksikön viiveellä. Tämän tapaisille graafeille on helpompi toteuttaa algoritmeja, koska graafeissa on enemmän solmuja.

2.2.2 Keinotekoiset muurahaiset ja lyhimmät tiet

Jotta keinotekoisista muurahaisista olisi hyötyä optimoinnissa, pitää niiden pystyä löytämään lyhin tie myös monimutkaisemmissa graafeissa (katso esimerkki kuvasta 2.3) kuin mitä oli kaksoissiltakokeessa. Graafista käytetään merkintää $G = (N, A)$, missä N on solmujen joukko ja A kaarien joukko. Jokaiseen kaareen on liitetty sen pituus (kustannus). Solmuja, joiden välistä lyhintä tietä etsitään, kutsutaan *lähteeksi* (*source*) ja *määränpääksi* (*destination*).

Jos keinotekoiset muurahaiset käyttäytyvät monimutkaisemmissa graafeissa samoin kuin kaksoissiltakokeessa, ne voivat luoda syklejä liikkuessaan lähteestä määränpäähän. Koska muurahaiset jättävät feromoneja kulkiessaan eteenpäin (lähteestä määränpäähän), sykleihin saattaa kertyä

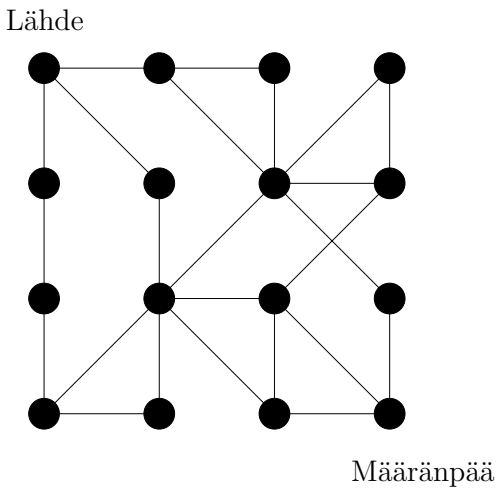


Kuva 2.2: Kaksoissiltakoe esitettynä graafina, jossa a) pidempi kaari on r kertaa niin pitkä lyhyt kaari ja b) kaikki kaaret ovat yhtä pitkiä, mutta pidemmällä polulla on r (tässä tapauksessa $r = 2$) kaarta, kun lyhyemmällä polulla on vain yksi kaari.

paljon feromoneja ja muurahaiset voivat jäädä kiertämään niihin. Vaikka muurahaiset pääsisivätkin jossain vaiheessa pois syklistä, feromoneja kerääntyy silti syklin sisältävälle polulle. Näin ollen lyhyillä poluilla ei enää olekaan suhteessa paljon feromoneja. Yksinkertaisin ratkaisu tähän olisi se, että eteenpäin liikkuvat muurahaiset eivät jättäisi feromoneja. Näin ei voida kuitenkaan tehdä, koska kuten jo edellä todettiin muurahaiset eivät löydä lyhintä tietä, jos ne jättävät feromoneja vain toiseen suuntaan liikkuessaan.

Ongelma voidaan ratkaista lisäämällä keinotekoisille muurahaisilla rajallinen muisti. Rajallisen muistinsa avulla muurahaiset pystyvät muistamaan kulkemansa polun ja polulla käytettyjen kaarien pituudet. Rajallisen muistin vuoksi muurahaisilla on kolme tärkeää ominaisuutta. Ensinnäkin ne pystyvät rakentamaan ratkaisun käyttämällä feromonien määrään perustuvia todennäköisyyksiä jättämättä kuitenkaan feromoneja kulkiessaan eteenpäin. Toiseksi ne pystyvät palaamaan takaisin lähteelle samaa polkua kuin tulivat määränpäähän eliminoiden kuitenkin polulta syklit. Tälle syklittömälle polulle muurahaiset jättävät samalla feromoneja. Kolmanneksi ne pystyvät arvioimaan polkunsa pituuden ja tämän avulla päättämään paljonko feromoneja polulleen jättävät.

Edellä mainittuja keinotekoisten muurahaisten ominaisuuksia on käytetty toteutettaessa algoritmia, jota kutsutaan *yksinkertaiseksi muurahaisyhdyskuntaoptimoinniksi* (*simple ant colony optimization*). Se on kuitenkin lähinnä vain hyvä työkalu muurahaisyhdyskuntaoptimoinnin perusmekanis-



Kuva 2.3: Esimerkki hieman monimutkaisemmasta graafista.

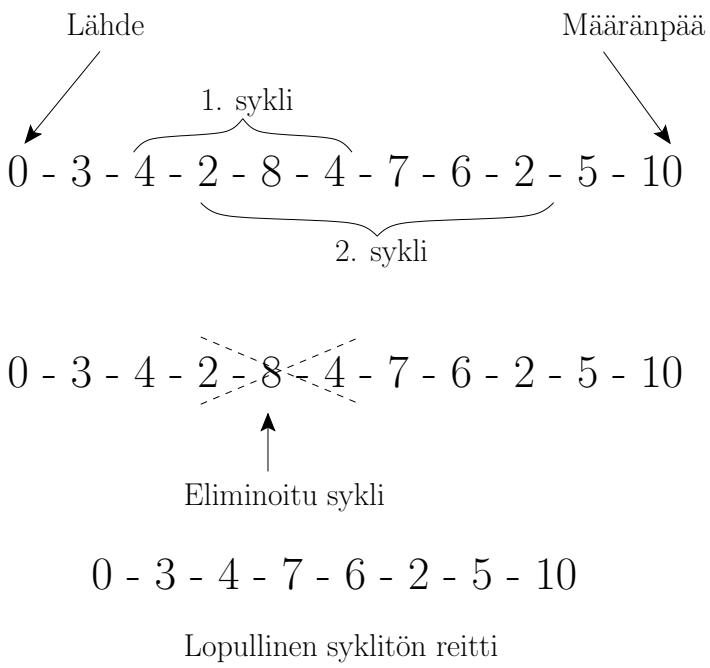
mien selittämiseksi.

Yksinkertaisessa muurahaisyhdyskuntaoptimoinnissa graafin $G = (N, A)$ jokaiseen kaareen (i, j) liitetään muuttuja τ_{ij}, joka kuvaa feromonien määrää kaarella (i, j). Muuttujan τ_{ij} arvo vähenee feromonien haihtumisen takia ja kasvaa aina, kun jokin muurahainen käyttää kaarta (i, j) kulkiessaan taaksepäin (eteenpäin liikkuvat muurahaiset eivät jätä feromoneja yksinkertaisessa muurahaisyhdyskuntaoptimoinnissa). Algoritmin suorituksen alussa asetetaan jokaiselle kaarelle $(i, j) \in A$ vakiomäärä feromoneja, esimerkiksi $\tau_{ij} = 1, \forall (i, j) \in A$.

Yksinkertaisessa muurahaisyhdyskuntaoptimoinnissa jokainen muurahainen rakentaa ratkaisun lähtien lähdesolmusta. Kun muurahainen k on solmussa i, se liikkuu seuraavaksi solmuun j todennäköisyydellä p_{ij}^k.

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \in N_i^k} \tau_{il}^\alpha}, & \text{jos } j \in N_i^k; \\ 0, & \text{jos } j \notin N_i^k; \end{cases} \tag{2.6}$$

missä N_i^k on solmun i naapurusto muurahaisen k ollessa solmussa i. Tässä tapauksessa naapurustoon N_i^k kuuluvat ne solmut, joista graafissa $G = (N, A)$ on kaari solmuun i, paitsi solmun i edeltäjä (se solmu, jossa muurahainen k on käynyt juuri ennen solmua i). Tämä naapuruston N_i^k määritelmä johtaa siihen, etteivät muurahaiset voi kulkea edestakaisin solmujen i ja j



Kuva 2.4: Esimerkki miten syklin eliminointi toimii.

väliä, mutta ne voivat kuitenkin muodostaa syklejä. Solmun i edeltäjä kuuluu naapurustoon kuitenkin siinä tapauksessa, että muuten olisi $N_i^k = \phi$ eli muurahainen on joutunut umpikujaan.

Muurahaiset liikkuvat solmusta toiseen käyttäen edellä esiteltyjä todennäköisyyksiä kunnes pääsevät määränpäähän. Muurahaiset kuitenkin tulevat määränpäähän eri aikoina, koska ne käyttävät eri pituisia polkuja.

Päästyään määränpäähän muurahaiset lähtevät takaisin lähteeseen samaa polkua kuin tulivat kuitenkin elimoinoiden ensin syklit, joita saattoi muodostua niiden liikkuessa lähteestä määränpäähän. Syklit eliminoidaan käymällä ensin polun solmuja läpi lähde-solmusta alkaen. Kun ollaan solmussa l, joka on i:s solmu lähde-solmusta laskien, aletaan käydä polkua läpi määränpää-solmusta alkaen. Solmuja käydään läpi kunnes päästään solmuun l, joka on nyt j:s solmu lähde-solmusta laskien. Aina on siis voimassa $j \geq i$. Jos $j > i$, niin ollaan löydetty sykli, ja polulta voidaan poistaa alipolku paikasta $i + 1$ paikkaan j. Näin menettelemällä syklit eliminoidaan samassa järjestyksessä kuin ne on muodostettu. Tästä johtuen pisimpiä syklejä ei välttämättä poisteta (katso kuva 2.4).

Kulkiessaan syklitöntä polkua takaisin lähteeseen muurahainen k jättää käyttämilleen kaarille (i, j) määrän $\Delta\tau^k$ feromonia. Feromonien määrä kaarella (i, j) kasvaa siis seuraavasti:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau^k. \tag{2.7}$$

Samalla todennäköisyys, että muurahaiset käyttävät myöhemminkin kaarta (i, j) kasvaa.

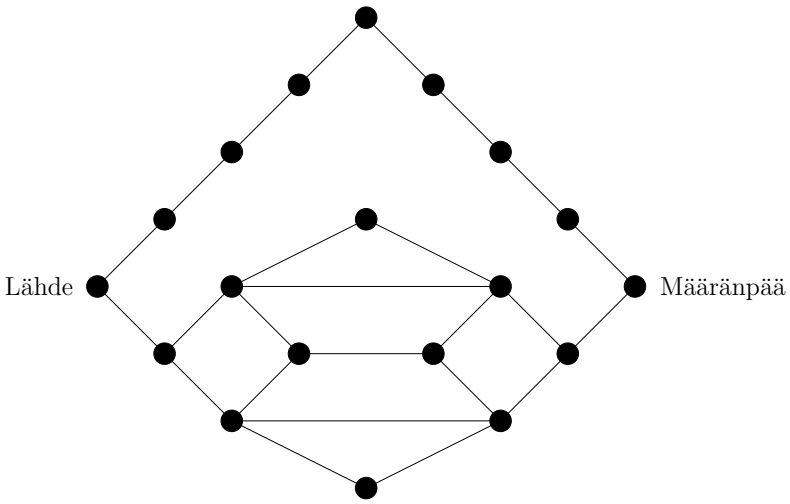
Kuinka sitten $\Delta\tau^k$ valitaan? Yksinkertaisinta on valita se samaksi vakioksi kaikille muurahaisille. Muurahaisten rajoitetun muistin takia on kuitenkin mahdollista valita $\Delta\tau^k$ muurahaisen k polun pituuden funktioksi. Yleisesti ottaen se valitaan ei-kasvavaksi polun pituuden funktioksi (esimerkiksi $\Delta\tau^k = \frac{1}{L^k}$, missä L^k on muurahaisen k käyttämän polun pituus).

Yksinkertaisessa muurahaisyhdyskuntaoptimoinnissa käytetään myös feromonien haihtumista. Edellä ollaan todettu, että oikeiden muurahaisten keskuudessa feromonien haihtumisella ei ole merkitystä, koska haihtuminen luonnossa on niin hidasta. Lyhimmän tien etsimisessä keinotekoisten muurahaisten avulla asia on kuitenkin toisin. Feromonien haihtuminen estää liian nopean konvergoimisen kohti ei-optimaalista ratkaisua ja antaa keinotekoisille muurahaisille mahdollisuuden tutkia erilaisia polkuja. Feromonien haihtuminen toteutetaan aina, kun jokainen muurahainen on liikkunnut seuraavan solmuun. Haihtuminen tapahtuu seuraavan kaavan mukaisesti:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}, \quad \forall (i, j) \in A, \tag{2.8}$$

missä $\rho \in (0, 1]$ on parametri. Feromonien haihtumisen jälkeen kaarille lisätään edellä esitetyn mukaisesti määrä $\Delta\tau^k$ feromoneja.

Yksinkertaisen muurahaisyhdyskuntaoptimoinnin toimivuutta on testattu erilaisilla graafeilla [11]. Testeissä on käytetty sekä tavallista kaksoissiltaa (kuva 2.2 b)) että *laajennettua kaksoissiltaa* (*extended double bridge*, kuva 2.5). Jos muurahainen laajennetussa kaksoissillassa lähtee lähteestä ylempään osaan graafia, se ei voi sen jälkeen enää valita kuin yhden polun, jonka pituus on 8 ja joka ei ole optimaalinen. Jos taas muurahainen valitsee alemman osan graafia, se voi löytää optimaalisen polun, jonka pituus on 5, mutta sen löytäminen vaatii monta oikeaa valintaa, koska alemmassa osassa on myös monta pidempää polkua.



Kuva 2.5: Laajennettu kaksoissilta. Ylemmässä osassa on vain yksi mahdollinen polku, jonka pituus on 8. Alemmassa osassa on kaksi optimaalista polkua, joiden pituus on 5, mutta siellä on myös monia muita pidempiä polkuja.

Tavallisella kaksoissillalla on tehty kokeita, joissa on testattu kannattaako päivityskaavassa (2.7) olevalle feromonien lisäykselle $\Delta\tau^k$ käyttää muotoa $\Delta\tau^k = vakio$ vai muotoa $\Delta\tau^k = \frac{1}{L^k}$, missä L^k on muurahaisen k käyttämän polun pituus [11]. Näissä kokeissa on saatu huomattavasti parempia tuloksia, kun on käytetty muotoa $\Delta\tau^k = \frac{1}{L^k}$. Samoissa kokeissa todettiin, että lyhimmän tien etsimiseen käytettyjen muurahaisten määrä vaikutti saadun ratkaisun laatuun. Muurahaisten määrä vaihteli kokeissa yhdestä 512:een. Mitä enemmän muurahaisia oli sitä pienempi osa kokeista päätyi kaksoissillan pidempään polkuun kunnes saavutettiin muurahaisten kriittinen määrä, josta lähtien kaikilla muurahaisten määrillä jokainen koe löysi kaksoissillan lyhyemmän polun. Kun kokeissa käytettiin feromonien lisäykselle muotoa $\Delta\tau^k = \frac{1}{L^k}$ kriittinen muurahaisten määrä oli 8, kun se käytettäessä muotoa $\Delta\tau^k = vakio$ oli 512. Muurahaisten määrää on siis turha kasvattaa kriittistä määrää suuremmaksi, koska tulokset eivät enää parane. Lisäksi kokeissa huomattiin, että käytettäessä kaavassa (2.6) parametrille α lähellä ykköstä olevia arvoja saatiin parhaimpia tuloksia.

Laajennetulla kaksoissillalla on tehty kokeita, joissa on testattu feromonien haihtumisen merkitystä [11]. Kokeissa on käytetty kaavassa (2.8) parametrille ρ arvoja 0; 0,01 ja 0,1. Testauksessa saatiin parhaat tulokset,

kun parametrille ρ käytettiin arvoa 0,01 ja selvästi huonoimmat, kun käytettiin arvoa 0.

3 Muurahaisyhdyskuntaoptimointi ja kauppamatkustajaongelma

Useimpia kombinatoriseen optimointiin tarkoitettuja muurahaisyhdyskuntaoptimoinnin menetelmiä on ensimmäisenä testattu kauppamatkustajaongelmalla. Tähän on muutamia syitä: kauppamatkustajaongelma on tärkeä NP-vaikea optimointiongelma, joka esiintyy monissa sovelluksissa; muurahaisyhdyskuntaoptimoinnin menetelmiä on helppo soveltaa kauppamatkustajaongelmaan ja optimointimenetelmän hyvää käyttäytymistä kauppamatkustajaongelmassa pidetään yleensä todisteena menetelmän käyttökelpoisuudesta. Näistä syistä muurahaisyhdyskuntaoptimoinnin menetelmät esitelläänkin kauppamatkustajaongelman avulla. Ennen varsinaisten menetelmien esittelyä esitellään siis kauppamatkustajaongelma.

Kaikki muurahaisyhdyskuntaoptimoinnin menetelmät noudattavat tiettyä niille kehitettyä runkoa. Tämän rungon pohjalta voidaan luoda myös uusia menetelmiä. Näin ollen ennen varsinaisten menetelmien esittelyä, esitellään tämä algoritmien runko.

3.1 Kauppamatkustajaongelma

Kauppamatkustajaongelmassa kotikaupungistaan lähtevä kauppamatkustaja haluaa löytää lyhimmän tien, kun hänen pitää matkallaan käydä tarkalleen kerran tietyissä kaupungeissa ja palata lopuksi kotikaupunkiinsa. Ongelma voidaan esittää graafina $G = (N, A)$, jossa N on kaupunkeja esittävien solmujen joukko ja A on 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. Epäsymmetrisessä kauppamatkustajaongelmassa ainakin yhdellä kaarella (i, j) on $d_{ij} \neq d_{ji}$. Symmetrisessä kauppamatkustajaongelmassa taas kaikilla kaarilla (i, j) on $d_{ij} = d_{ji}$. Dynaamisessa kauppamatkustajaongelmassa kaupunkeja voi tulla lisää tai poistua kesken ongelman ratkaisemisen. Kaikissa eri kauppamatkustajaongelman versioissa on tarkoituksena löytää solmujen {1, 2,..., n} ($n = |N|$) permutaatio π, jolle

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)} \tag{3.1}$$

on pienin. Funktiosta f huomataan, että kaupunkien absoluuttisella järjestyksellä ei ole merkitystä. Kun kaupunkeja on n kappaletta, on myös n kappaletta permutaatioita π, jotka antavat saman arvon funktiolle f. Esimerkiksi jos $n = 3$, niin permutaatiot (1, 2, 3), (3, 1, 2) ja (2, 3, 1) antavat saman arvon funktiolle f. Näin ollen kauppamatkustajaongelmaa ratkaistaessa ei ole merkitystä, mistä kaupungista ratkaiseminen aloitetaan.

3.2 Muurahaisyhdyskuntaoptimoinnin menetelmien runko

Muurahaisyhdyskuntaoptimoinnin menetelmissä keinotekoiset muurahaiset rakentavat ratkaisun lisäämällä stokastisesti uusia komponentteja osaratkaisuun. Esimerkiksi kauppamatkustajaongelmassa tämä tarkoittaa sitä, että muurahaiset lähtevät jostakin kaupungista (solmusta) ja valitsevat toistuvasti stokastisesti seuraavan kaupungin (solmun) kunnes ovat käyneet jokaisessa kaupungissa (solmussa). Keinotekoiset muurahaiset valitsevat seuraavaksi lisättävän komponentin *feromonijälkien* τ ja *heuristisen informaation* η perusteella. Sekä feromonijälki että heuristinen informaatio voidaan liittää joko komponentteihin (τ_i ja η_i) tai komponenttien väliseen yhteyksiin (τ_{ij} ja η_{ij}). Kauppamatkustajaongelman tapauksessa on todettu, että parhaat tulokset saadaan, kun τ_{ij} esittää kaupungissa j käynnin houkuttelevuutta heti kaupungin i jälkeen ja $\eta_{ij} = \frac{1}{d_{ij}}$.

Muurahaisyhdyskuntaoptimoinnin menetelmien runko voidaan esittää seuraavanlaisena pseudokoodina:

```
procedure Muurahaisyhdyskuntaoptimointi
  ToimintojenAjoitus
    RakennaMuurahaistenRatkaisut
    PäivitäFeromonit
    KeskitetytToiminnot    %vapaaehtoinen
  end ToimintojenAjoitus
end procedure
```

Muurahaisyhdyskuntaoptimoinnissa on siis kolme proseduuria `RakennaMuurahaistenRatkaisut`, `PäivitäFeromonit` ja `KeskitetytToiminnot`, jotka ovat vuorovaikutuksessa keskenään. `ToimintojenAjoitus` hoitaa näiden proseduurien ajoituksen. Se ei kuitenkaan määrää suoritetaanko proseduurit rinnakkain ja itsenäisesti

vai käytetäänkö niihin jonkinlaista synkronointia. Näin ollen proseduurit voidaan ajoittaa eri ongelmille eri tavalla.

Proseduuri `RakennaMuurahaistenRatkaisut` hoitaa ratkaisujen rakentamisen. Rakennettuaan ratkaisun keinotekoinen muurahainen laskee ratkaisun hyvyyden, jota sitten käytetään proseduurissa `PäivitäFeromonit` päättämään paljonko feromoneja lisätään. Proseduuri `PäivitäFeromonit` hoitaa sekä feromonien lisäämisen että vähentämisen (haihtuminen). Vapaaehtoinen proseduuri `KeskitetytToiminnot` sisältää toiminnot, joita yksittäinen muurahainen ei voi suorittaa. Tälläinen toiminto on esimerkiksi lokaali haku, jota voidaan käyttää keinotekoisten muurahaisten rakentamien ratkaisujen parantamiseen.

3.3 Muurahaisjärjestelmä

Muurahaisyhdyskuntaoptimoinnin algoritmeista vanhin on *Muurahaisjärjestelmä* (*ant system*). Siitä on kolme eri versiota: *muurahaistiheys* (*ant-density*), *muurahaisrunsaus* (*ant-quantity*) ja *muurahaiskierto* (*ant-cycle*). Kahdessa ensin mainitussa versiossa muurahaiset päivittävät feromoneja heti siirryttyään kaupungista toiseen. Viimeksi mainitussa versiossa feromonit päivitetään vasta, kun kaikki muurahaiset ovat rakentaneet ratkaisunsa. Nykyään muurahaisjärjestelmällä tarkoitetaan juuri muurahaiskiertoa, koska se tuottaa parempia ratkaisuja kuin kaksi muuta versiota.

Käytettäessä muurahaisyhdyskuntaoptimoinnin algoritmeja kauppamatkustajaongelmaan pitää jokaiselle kaarelle $(i, j) \in A$ alustaa feromonijälki $\tau_{ij} = \tau_0$. Tämä arvo ei saa olla liian pieni, jotta ensimmäisenä muodostetut ratkaisut eivät tule myöhemmin valituiksi liian suurella todennäköisyydellä. Toisaalta tämä arvo ei saa myöskään olla liian suuri, koska muuten kestää monta iteraatiota ennen kuin feromonijäljet eri kaarilla tulevat merkittävästi eri suuruisiksi. Muurahaisjärjestelmän tapauksessa hyväksi arvoksi on todettu $\tau_0 = \frac{m}{L^l}$ [11], missä m on muurahaisten määrä ja L^l on *lähimmän naapurin menetelmällä* saadun kierroksen pituus eli funktion (3.1) arvo, kun π on lähimmän naapurin menetelmällä saatu permutaatio. Lähimmän naapurin menetelmässä kaupungista i kuljetaan aina kaupunkiin j, joka on lähimpänä kaupunkia i ja jossa ei ole vielä käyty. Tätä menetelmää on helppo käyttää, mutta se ei yleensä tuota hyviä ratkaisuja, koska loppuvaiheessa kaupunkeja, joissa ei ole vielä käyty, on enää vähän ja niiden välimatkat saattavat olla

suuria. Jos tarvitaan jokin alkuratkaisu, sellainen voidaan kuitenkin hyvin rakentaa lähimmän naapurin menetelmällä.

3.3.1 Ratkaisujen rakentaminen

Muurahaisjärjestelmässä m muurahaista asetetaan aluksi satunnaisesti valittuihin kaupunkeihin, jonka jälkeen ne rakentavat ratkaisunsa. Kun muurahainen k on kaupungissa i, se siirtyy kaupunkiin j todennäköisyydellä

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \eta_{il}^\beta}, & \text{jos } j \in N_i^k; \\ 0, & \text{jos } j \notin N_i^k; \end{cases} \tag{3.2}$$

missä $\eta_{ij} = \frac{1}{d_{ij}}$, α ja β ovat kaksi parametriä, jotka määrittelevät feromonijäljen ja heuristisen informaation suhteellisen vaikutuksen sekä N_i^k on kaupungin i sallittu naapurusto muurahaisen k ollessa kaupungissa i eli tässä tapauksessa N_i^k koostuu kaikista kaupungeista, joissa muurahainen k ei vielä ole käynyt. Jos $\alpha = 0$, käytetään ratkaisun rakentamisessa vain heuristista informaatiota η_{ij}. Jos taas $\beta = 0$, käytetään ratkaisun rakentamisessa vain feromonijälkiä τ_{ij}.

Ratkaistaessa muurahaisjärjestelmällä kauppamatkustajaongelmaa parhaat tulokset on saatu [11], kun $\alpha = 1$ ja β on vaihdellut arvosta 2 arvoon 5. Myös muilla myöhemmin esiteltävillä menetelmillä, joissa käytetään parametrejä α ja β, parhaat tulokset kauppamatkustajaongemalle on saatu samoilla parametrien α ja β arvoilla. Myös muurahaisten määrä m vaikuttaa lopullisen ratkaisun laatuun. Kun kauppumatkustajaongelmaa ratkaistaan muurahaisjärjestelmällä tai muilla myöhemmin tässä luvussa esiteltävillä menetelmillä, kannattaa hyvien tulosten saamiseksi valita $m = n$ ellei kyseisen menetelmän kohdalla toisin mainita. Muurahaisten määrää ei siis kannata valita liian suureksi, kuten luvun 2 lopussa kaksoissillan tapauksessa todettiin. Muurahaisten määrän kasvattaminen suuremmaksi ei sinänsä heikennä saatujen tulosten laatua, mutta laskenta-aika kasvaa.

Muurahainen k säilyttää muistissaan M^k kaupungit, joissa se on käynyt sekä näiden kaupunkien järjestyksen. Muistiaan muurahainen käyttää sallitun naapuruston N_i^k määrittämiseen. Muistinsa avulla muurahainen myös laskee rakentamansa kierroksen K^k pituuden ja palaa takaisin käyttämäänsä polkua pitkin päivittäen samalla feromonijälkiä.

Ratkaisujen rakentamiseen on kaksi eri tapaa: rinnakkainen ja peräkkäinen. Jos ratkaisut rakennetaan rinnakkain, kaikki muurahaiset rakentavat ratkaisunsa yhtäaikaa. Jos taas ratkaisut rakennetaan peräkkäin, yksi muurahainen kerrallaan rakentaa oman ratkaisunsa. Jos käytetään muurahaiskiertoa, ei ole merkitystä käytetäänkö rinnakkaista vai peräkkäistä tapaa, koska feromonien päivitys tapahtuu vasta kaikkien muurahaisten rakennettua omat ratkaisunsa. Kahdessa muussa muurahaisjärjestelmän versiossa feromonien päivitys tapahtuu ratkaisun rakentamisen aikana, joten niissä ratkaisuihin vaikuttaa se, valitaanko rinnakkainen vai peräkkäinen ratkaisujen rakentamistapa.

3.3.2 Feromonijälkien päivittäminen

Kun muurahaiskierrossa kaikki muurahaiset ovat rakentaneet ratkaisunsa, päivitetään feromonijäljet. Ensin suoritetaan feromonien vähentäminen kaikilta kaarilta. Tämä feromonien haihtuminen toteutetaan kaavalla (2.8) eli

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \quad \forall (i,j) \in A, \tag{3.3}$$

missä $0 < \rho \leq 1$ on feromonien haihtumisnopeus. Parametriä ρ käytetään estämään feromonien rajoittamaton kertyminen kaarille. Sen avulla algoritmi voi myös unohtaa aiemmin tehdyt huonot valinnat. Hyvien tulosten saavuttamiseksi kauppamatkustajaongelmaa ratkaistaessa muurahaiskierrolla kannattaa käyttää arvoa $\rho = 0{,}5$ [11]. Feromonien haihtumisen jälkeen kaikki muurahaiset jättävät feromoneja niille kaarille, joita ovat käyttäneet kierroksellaan:

$$\tau_{ij} \leftarrow \tau_{ij} + \textstyle\sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \quad \forall (i,j) \in A, \tag{3.4}$$

missä $\Delta\tau_{ij}^{k}$ on feromonien määrä, jonka muurahainen k jättää käyttämälleen kaarelle. Muurahaiskierrossa määritellään

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q_1}{L^k}, & \text{jos kaari } (i,j) \text{ kuuluu kierrokseen } K^k; \\ 0, & \text{muuten,} \end{cases} \tag{3.5}$$

missä Q_1 on vakio ja L^k on muurahaisen k rakentaman kierroksen K^k pituus. Näin ollen mitä lyhyemmällä kierroksella kaari on ja mitä enemmän muurahaisia käyttää kaarta sitä enemmän feromonia kaarelle lisätään.

Myös muurahaistiheydessä ja muurahaisrunsaudessa feromonien haihtuminen tapahtuu sen jälkeen, kun kaikki muurahaiset ovat rakentaneet

ratkaisunsa ja se määritellään kuten muurahaiskierrossa eli kaavalla (3.3). Feromonien lisääminen kuitenkin tapahtuu näissä kahdessa versiossa heti, kun muurahainen on siirtynyt kaupungista toiseen. Lisäys tapahtuu kaavalla (3.4). Muurahaistiheydessä määritellään

$$\Delta\tau_{ij}^k = \begin{cases} Q_2, & \text{jos muurahainen } k \text{ käyttää kaarta } (i,j); \\ 0, & \text{muuten,} \end{cases} \tag{3.6}$$

missä Q_2 on vakio. Muurahaisrunsaudessa taas määritellään

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q_3}{d_{ij}}, & \text{jos muurahainen } k \text{ käyttää kaarta } (i,j); \\ 0, & \text{muuten,} \end{cases} \tag{3.7}$$

missä Q_3 on vakio ja d_{ij} on kaupunkien i ja j välinen etäisyys.

3.4 Elitistinen muurahaisjärjestelmä

Elitistisessä muurahaisjärjestelmässä (*elitist ant system*) painotetaan parasta tähän mennessä saatua ratkaisua (K^t) enemmän kuin muita. Käytettäessä tätä menetelmää kauppamatkustajaongelmaan kannattaa asettaa $\tau_0 = \frac{e+m}{\rho L^l}$ [11], missä m, ρ ja L^l ovat kuten aiemmin ja e on parametri, joka määrittää tähän mennessä parhaan ratkaisun painon. Tätä painoa käytetään feromonien päivityksessä, jotta feromonien lisäys parhaaseen tähän mennessä löydettyyn ratkaisuun kuuluvilla kaarilla on verrattaen suurempi kuin muihin ratkaisuihin kuuluvilla kaarilla. Ratkaisu rakennetaan samoin kuin muurahaisjärjestelmässä (kaava (3.2)).

3.4.1 Feromonijälkien päivittäminen

Myös elitistisessä muurahaisjärjestelmässä feromonien haihtuminen toteutetaan kaavalla (3.3). Kauppamatkustajaongelman tapauksessa kannattaa tälläkin kertaa käyttää arvoa $\rho = 0{,}5$ [11]. Feromonien lisäys tapahtuu seuraavan kaavan mukaisesti:

$$\tau_{ij} \leftarrow \tau_{ij} + \textstyle\sum_{k=1}^{m} \Delta\tau_{ij}^k + e\Delta\tau_{ij}^t, \quad \forall (i,j) \in A, \tag{3.8}$$

missä $\Delta\tau_{ij}^k$ on määritelty kaavalla (3.5) ja

$$\Delta\tau_{ij}^t = \begin{cases} \frac{Q_4}{L^t}, & \text{jos kaari } (i,j) \text{ kuuluu kierrokseen } K^t; \\ 0, & \text{muuten,} \end{cases} \tag{3.9}$$

missä Q_4 on vakio ja L^t on parhaan tähän mennessä löydetyn kierroksen pituus. Kauppamatkustajaongelmassa kannattaa asettaa $e = n$ [11]. Kokeellisesti on todettu [8, 9, 10], että sopivaa parametrin e arvoa käyttämällä elitistinen muurahaisjärjestelmä antaa parempia tuloksia kuin tavallinen muurahaisjärjestelmä.

3.5 Järjestykseen perustuva muurahaisjärjestelmä

Järjestykseen perustuvassa muurahaisjärjestelmässä (*rank-based ant system*) parhaan ratkaisun tähän mennessä rakentaneen muurahaisen lisäksi vain $w - 1$ iteraation parasta ratkaisua rakentannutta muurahaista jättää feromoneja. Tämä w onkin yksi menetelmän parametreista. Käytettäessä tätä menetelmää kauppamatkustajaongelmaan kannattaa asettaa $\tau_0 = \frac{0{,}5w(w-1)}{\rho L^l}$ [11], missä käytetään samoja merkintöjä kuin ennenkin sekä parametrin w arvoksi kannattaa asettaa 6. Myös tässä menetelmässä ratkaisu rakennetaan samoin kuin muurahaisjärjestelmässä (kaava (3.2)).

3.5.1 Feromonijälkien päivittäminen

Myös järjestykseen perustuvassa muurahaisjärjestelmässä feromonien haihtuminen toteutetaan kaavalla (3.3). Kauppamatkustajaongelman tapauksessa kannattaa tällä kertaa käyttää arvoa $\rho = 0{,}1$ [11]. Ennen feromonien lisäystä muurahaiset järjestetään kierroksen pituuden mukaan kasvavaan järjestykseen. Mitä pienempi muurahaisen järjestysluku r on sitä enemmän sen rakentamaa ratkaisua painotetaan. Jos muurahaisten rakentamat kierrokset ovat yhtä pitkät, niiden järjestys päätetään satunnaisesti. Jokaisella iteraatiolla vain $w - 1$ parasta muurahaista ja tähän mennessä parhaan ratkaisun rakentanut muurahainen jättävät feromoneja. Feromonien lisäys toteutetaan nyt seuraavalla kaavalla:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^r + w\Delta\tau_{ij}^t, \quad \forall(i,j) \in A, \tag{3.10}$$

missä $\Delta\tau_{ij}^r$ ja $\Delta\tau_{ij}^t$ saadaan vastaavasti kaavoista (3.5) ja (3.9). Kokeellisesti on todettu [4], että järjestykseen perustuva muurahaisjärjestelmä antaa hieman parempia ratkaisuja kuin elitistinen muurahaisjärjestelmä optimoitaessa eri kauppamatkustajaongelmia antaen molemmille menetelmille yhtä paljon laskenta-aikaa.

3.6 MAX-MIN-muurahaisjärjestelmä

MAX-MIN-muurahaisjärjestelmässä (*MAX-MIN ant system*) vain parhaan (joko parhaan tähän mennessä tai iteraation parhaan) ratkaisun löytänyt muurahainen jättää feromoneja. Tämä voi johtaa siihen, että kaikki muurahaiset rakentavat saman ratkaisun, koska feromoneja kerääntyy kaarille, joita parhaassa ratkaisussa käytetään. Tämän estämiseksi MAX-MIN-muurahaisjärjestelmässä feromonijälkien arvo on rajoitettu välille $[\tau_{min}, \tau_{max}]$. Kauppamatkustajaongelman tapauksessa hyviksi arvoiksi on todettu $\tau_{max} = \frac{1}{\rho L^t}$ ja $\tau_{min} = \tau_{max} \cdot \frac{1 - \sqrt[n]{0{,}05}}{(kesk-1) \cdot \sqrt[n]{0{,}05}}$, missä $kesk$ on niiden vaihtoehtoisten valintojen keskimääräinen määrä, jotka muurahaisella on joka askeleella rakentaessaan ratkaisua [32]. Lisäksi MAX-MIN-muurahaisjärjestelmässä asetetaan $\tau_0 = \tau_{max}$, missä aluksi $L^t = L^l$. Jos todennäköisyys jonkin ratkaisun valitsemiseksi tulee liian suureksi tai jos parasta tähän mennessä löydettyä ratkaisua ei ole tietyssä määrässä peräkkäisiä iteraatioita onnistuttu parantamaan, niin MAX-MIN-muurahaisjärjestelmässä kaikkien kaarien $(i, j) \in A$ feromonijäljet τ_{ij} alustetaan uudelleen arvoksi τ_{max}. Myös tässä menetelmässä ratkaisu rakennetaan samoin kuin muurahaisjärjestelmässä (kaava (3.2)).

3.6.1 Feromonijälkien päivittäminen

Myös MAX-MIN-muurahaisjärjestelmässä feromonien haihtuminen toteutetaan kaavalla (3.3). Kauppamatkustajaongelman tapauksessa kannattaa tällä kertaa käyttää arvoa $\rho = 0{,}02$ [11]. MAX-MIN-muurahaisjärjestelmän parametrejä on tutkittu enemmän artikkelissa [27]. Feromonien lisäys toteutetaan nyt kaavalla

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^b, \quad \forall (i, j) \in A, \tag{3.11}$$

missä $\Delta\tau_{ij}^b = \frac{1}{L^t}$ tai $\Delta\tau_{ij}^b = \frac{1}{L^e}$ ja L^e on iteraation parhaan kierroksen pituus. Mitä useammin käytetään parasta tähän mennessä löydettyä ratkaisua sitä nopeammin haku keskittyy lähelle ratkaisua K^t. Toisaalta mitä useammin käytetään iteraation parasta ratkaisua sitä suuremmalle alueelle haku ulottuu. Pienille kauppamatkustajaongelmille ($n \leq 200$) on todettu, että feromonijälkien päivityksessä kannattaa käyttää vain iteraatioiden parhaita ratkaisuja. Suuremmille kauppamatkustajaongelmille on taas todettu, että

parempia tuloksia saadaan, kun parasta tähän mennessä löydettyä ratkaisua käytetään sitä useammin mitä kauemmin algoritmia on suoritettu [11].

3.6.2 Feromonijälkien rajat

Feromonijälkien ala- ja ylärajasta (τ_{min} ja τ_{max}) seuraa, että todennäköisyydet p_{ij} on rajoitettu välille $[p_{min}, p_{max}]$, missä $0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1$. $p_{max} = 1$ vain, jos muurahaisella k on ainoastaan yksi mahdollisuus valita seuraava kaupunki eli $|N_i^k| = 1$, mutta silloin myös $p_{min} = 1$.

Voidaan osoittaa (katso Lemma 3.1.), että feromonijälkien yläraja τ_{max} ei koskaan ylitä arvoa $\frac{1}{\rho L^*}$, missä L^* on optimaalisen kierroksen pituus. Tästä johtuen parametrille τ_{max} käytetään arviota $\frac{1}{\rho L^t}$, joka päivitetään aina, kun löydetään uusi tähän mennessä paras ratkaisu. Kokeellisesti on osoitettu [31], että feromonijälkien alarajalla τ_{min} on ylärajaa τ_{max} tärkeämpi rooli, kun halutaan estää menetelmän jumittuminen epäoptimaaliseen ratkaisuun.

3.6.3 Feromonijälkien alustaminen ja uudelleenalustaminen

MAX-MIN-muurahaisjärjestelmässä feromonijäljet alustetaan arvoksi τ_{max}. Tähän yhdistettynä pieni haihtumisnopeus ρ takaa sen, että algoritmin suorituksen alussa todennäköisyydet p_{ij} eivät poikkea paljon toisistaan. Jos jossain vaiheessa algoritmin suoritusta jonkin ratkaisun valitsemisen todennäköisyys tulee paljon suuremmaksi kuin muiden, feromonijäljet τ_{ij} uudelleenalustetaan arvoksi τ_{max}. Tämä johtaa taas siihen, että kaikki todennäköisyydet p_{ij} ovat yhtä suuria. Uudelleenalustus tehdään myös, jos parasta tähän mennessä löydettyä ratkaisua ei ole tietyssä määrässä peräkkäisiä iteraatioita onnistuttu parantamaan. Uudelleenalustus johtaa siihen, että feromonien lisäämisessä voidaan käyttää myös parasta ratkaisua, joka on löydetty viimeisen uudelleenalustuksen jälkeen [31, 32].

3.7 Muurahaisyhdyskuntajärjestelmä

Muurahaisyhdyskuntajärjestelmässä (*ant colony system*) ei ratkaisuja rakennettaessa käytetäkään kaavaa (3.2) kuten kaikissa muissa tähän mennessä esitellyissä menetelmissä. Tämä menetelmä eroaa edellä esitellyistä menetelmistä myös siinä, että ratkaistaessa kauppamatkustajaongelmaa, jossa $n > 10$, ei kannatakaan valita $m = n$ vaan $m = 10$ [11].

3.7.1 Ratkaisujen rakentaminen

Kun muurahainen k on muurahaisyhdyskuntajärjestelmässä kaupungissa i, kaupunki j, johon se seuraavaksi siirtyy, valitaan seuraavan säännön mukaan:

$$j = \begin{cases} \text{argmax}_{l \in N_i^k}\{\tau_{il} \cdot \eta_{il}^{\beta}\}, & \text{jos } p \leq q_0; \\ J, & \text{jos } p > q_0; \end{cases} \tag{3.12}$$

missä p on tasaisesti välille [0,1] jakautunut satunnaismuuttuja, q_0 ($0 \leq q_0 \leq 1$) on parametri ja J on satunnaismuuttuja yhtälön (3.2) mukaisesta todennäköisyysjakaumasta (jossa $\alpha = 1$). Muurahainen valitsee siis todennäköisyydellä q_0 feromonijälkien ja heuristisen informaation kannalta parhaan mahdollisen kaupungin j.

3.7.2 Globaali feromonijälkien päivittäminen

Muurahaisyhdyskuntajärjestelmässä vain muurahainen, joka on löytänyt tähän mennessä parhaan ratkaisun, jättää feromoneja jokaisen iteraation jälkeen. Feromonien päivitys toteutetaan seuraavalla kaavalla:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}^{t}, \quad \forall (i,j) \in K^t, \tag{3.13}$$

missä ρ ($0 < \rho \leq 1$) on parametri ja $\Delta\tau_{ij}^{t} = \frac{1}{L^t}$. Kauppamatkustajaongelman tapauksessa kannattaa tällä menetelmällä käyttää arvoa $\rho = 0{,}1$ [11]. Kuten kaavasta (3.13) huomataan, feromonien haihtumista ei nyt tapahdu kaikilla kaarilla, mikä vähentää laskenta-aikaa. Tästä kaavasta nähdään myös, että uusi feromonijälki on vanhan feromonijäljen ja lisätyn feromonin määrän painotettu keskiarvo.

3.7.3 Lokaali feromonijälkien päivittäminen

Globaalin feromonijälkien päivittämisen lisäksi muurahaisyhdyskuntajärjestelmässä käytetään lokaalia feromonijälkien päivittämistä. Muurahaiset toteuttavat lokaalin feromonijälkien päivityksen samaan aikaan, kun rakentavat ratkaisua. Muurahaisen siirryttyä kaupungista i kaupunkiin j tapahtuu päivitys kaavalla

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0, \tag{3.14}$$

missä $0 < \xi < 1$ ja τ_0 ovat parametrejä. Muurahaisyhdyskuntamenetelmässä parametriä τ_0 käytetään siis sekä feromonijälkien alustukseen että niiden

päivittämiseen. Kauppamatkustajaongelmassa hyviksi arvoiksi on todettu ξ = 0,1 ja $\tau_0 = \frac{1}{nL^l}$ [11]. Lokaali feromonijälkien päivittäminen ei ainakaan lisää feromonien määrää kaarilla (i, j), joita muurahaiset käyttävät. Tämä estää menetelmää jumittumasta epäoptimaalisiin ratkaisuihin. Lokaalin feromonijälkien päivittämisen takia myös sillä, rakennetaanko ratkaisut rinnakkain vai peräkkäin, on merkitystä menetelmän käyttäytymisen kannalta. Kummankaan tavan paremmuudelle ei kuitenkaan ole kokeellisia todisteita.

Muurahaisyhdyskuntajärjestelmä perustuu menetelmään nimeltä *muurahais-Q* (*ant-Q*, [9]). Käytännössä näiden kahden menetelmän ainoa ero on kaavassa (3.14) parametrin τ_0 määrittelyssä. Muurahais-Q:ssa määritellään $\tau_0 = \gamma \cdot max_{j \in N_i^k}\{\tau_{ij}\}$, missä γ on parametri. On kuitenkin todettu, että muurahaisyhdyskuntajärjestelmän yksinkertaisempi päivityskaava antaa suunnilleen yhtä hyviä tuloksia kuin muurahais-Q:n päivityskaavalla saadut tulokset. Näin ollen muurahais-Q:n käytöstä on luovuttu.

Myös muurahaisyhdyskuntajärjestelmässä feromonijälkien arvoilla on ala- ja yläraja kuten MAX-MIN-muurahaisjärjestelmässä. Muurahaisyhdyskuntajärjestelmässä näitä rajoja ei kuitenkaan esitetä eksplisiittisesti. Päivityskaavoista (3.13) ja (3.14) seuraa, että feromonijäljet τ_{ij} eivät voi koskaan olla pienempiä kuin τ_0. Toisaalta kuten jo MAX-MIN-muurahaisjärjestelmän kohdalla todettiin, ei mikään τ_{ij} voi koskaan ylittää arvoa $\frac{1}{\rho L^*}$. Näin ollen $\tau_0 \leq \tau_{ij} \leq \frac{1}{\rho L^*}$. Kokeellisesti on todettu, että tässä esitellyistä menetelmistä MAX-MIN-muurahaisjärjestelmä ja muurahaisyhdyskuntajärjestelmä antavat parhaat tulokset [11].

3.7.4 Kandidaattilistat

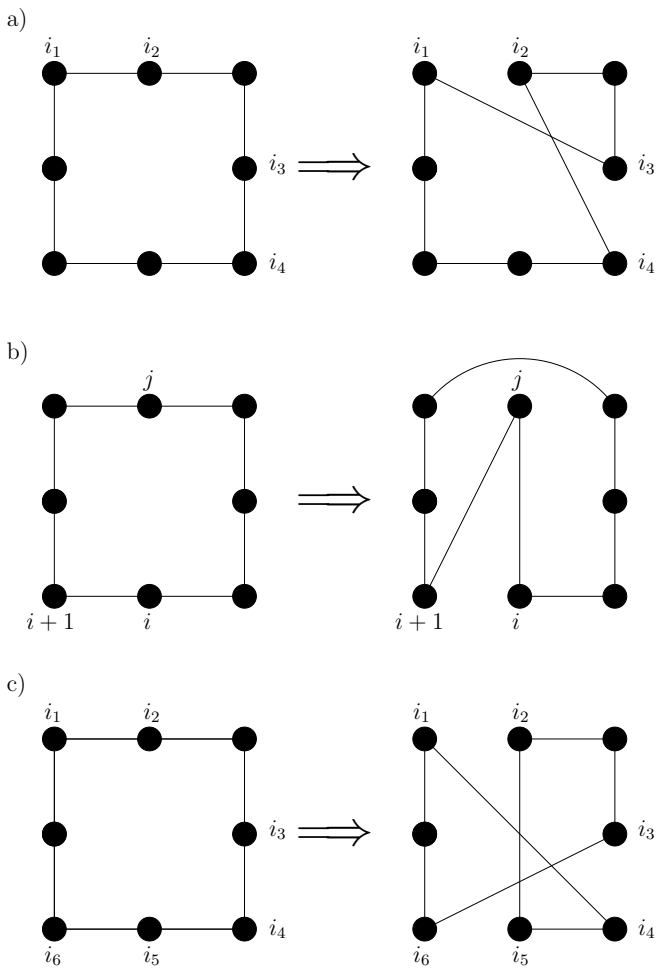
Muurahaisyhdyskuntajärjestelmä oli ensimmäinen muurahaisyhdyskuntaoptimoinnin algoritmi, jossa käytettiin *kandidaattilistoja*. Kandidaattilistat sisältävät tietyn määrän jonkin heuristisen informaation suhteen parhaita mahdollisia valintoja. Kauppamatkustajaongelman tapauksessa kandidaattilistat sisältävät jokaisessa kaupungissa i ne kaupungit j, joihin on lyhin matka. Muurahainen valitseekin seuraavan kaupungin kaupungeista, jotka ovat kandidaattilistalla eikä kaikista kaupungeista, jotka ovat sallittuja naapureita. Vain siinä tapauksessa, että muurahainen k on jo käynyt kaikissa kaupungeissa, jotka ovat sen nykyisen kaupungin i kandidaattilistalla, valit-

see muurahainen k jonkin kaupungin, joka ei ole kyseisellä kandidaattilistalla. Kauppamatkustajaongelman tapauksessa on todettu, että kandidaattilistat sekä parantavat lopullista löydettyä ratkaisua että nopeuttavat algoritmin suoritusta.

3.8 Lokaali haku

Kaikkiin edellä esiteltyihin menetelmiin voidaan yhdistää lokaali haku. Lokaalilla haulla yritetään löytää saadun ratkaisun jostakin naapurustosta parempia ratkaisuja. Kauppamatkustajaongelman tapauksessa usein käytettyjä lokaalin haun menetelmiä ovat 2-opt, 2,5-opt ja 3-opt (katso esimerkki näistä menetelmistä kuvasta 3.1). 2-opt-menetelmässä ratkaisun naapurustona käytetään kaikkia niitä ratkaisuja, jotka saadaan alkuperäisestä ratkaisusta vaihtamalla kaksi kaarta. 2,5-opt-menetelmässä tutkitaan 2-opt-menetelmän naapuruston lisäksi ratkaisuja, joissa kaupungin i ja sen seuraajan väliin lisätään jokin kaupunki. 3-opt-menetelmässä taas naapurustona käytetään kaikkia niitä ratkaisuja, jotka saadaan alkuperäisestä ratkaisusta vaihtamalla korkeintaan kolme kaarta. Yhdistämällä jokin näistä lokaalin haun menetelmistä muurahaisyhdyskuntaoptimoinnin menetelmään saadaan kauppamatkustajaongelmalle parempia tuloksia kuin käyttämällä samaa menetelmää ilman lokaalia hakua [11]. Toisaalta jos lokaalia hakua halutaan käyttää yksinään, ongelmaksi tulee hyvien aloitusratkaisujen löytäminen.

Kun lokaali haku yhdistetään muurahaisyhdyskuntaoptimoinnin menetelmään, pitää päättää, käytetäänkö nopeampaa ja vähemmän alkuperäistä ratkaisua parantavaa lokaalin haun menetelmää usein vai käytetäänkö hitaampaa, mutta enemmän alkuperäistä ratkaisua parantavaa lokaalin haun menetelmää hieman harvemmin. Pitää myös päättää, minkä muurahaisten ratkaisuja parannetaan lokaalilla haulla. MAX-MIN-muurahaisjärjestelmää on testattu [11], kun siihen on yhdistetty lokaali haku. Kun näissä testeissä on annettu eri lokaalin haun menetelmille yhtä paljon laskenta-aikaa, parhaat tulokset on saatu, kun on käytetty 3-opt-menetelmää ja huonoimpiin tuloksiin on päädytty 2-opt-menetelmällä. Näissä testeissä on myös huomattu, että parasta tähän mennessä löydettyä ratkaisua kannattaa käyttää feromonien päivityksessä useammin kuin käytettäessä MAX-MIN-muurahaisjärjestelmää ilman lokaalia hakua. Voi



Kuva 3.1: Esimerkkejä eri lokaalin haun menetelmistä. Vasemmalla on alkuperäinen ratkaisu ja oikealla a) 2-opt-menetelmällä löydetty ratkaisu, jossa kaaret (i_1, i_2) ja (i_3, i_4) on korvattu kaarilla (i_1, i_3) ja (i_2, i_4), b) 2,5-opt-menetelmällä löydetty ratkaisu, jossa kaupunkien i ja $i+1$ väliin on asetettu kaupunki j, c) 3-opt-menetelmällä löydetty ratkaisu, jossa kaaret (i_1, i_2), (i_3, i_4) ja (i_5, i_6) on korvattu kaarilla (i_1, i_4), (i_5, i_2) ja (i_3, i_6). Jokaisessa kohdassa ratkaisu siis hyväksytään, jos muodostettu ratkaisu on parempi kuin alkuperäinen.

myös olla, että edellä eri menetelmien kohdalla esitetyt hyvät parametrien arvot eivät lokaalia hakua käytettäessä olekaan enää hyviä [11].

Kun käytetään muurahaisyhdyskuntaoptimoinnin menetelmiä ilman lokaalia hakua, on heuristinen informaatio tärkeää hyvien ratkaisujen löytämiseksi. Testeissä on kuitenkin todettu, että kun lokaali haku lisätään

mukaan muurahaisyhdyskuntaoptimoinnin menetelmään, heuristinen informaatio menettää merkityksensä. Jossain tapauksissa on jopa saatu parempia tuloksia, kun heuristista informaatiota ei ole käytetty. Käytettäessä lokaalia hakua voidaan siis asettaa $\beta = 0$ ja saadaan silti hyviä tuloksia.

4 Konvergenssi muurahaisyhdyskuntaoptimoinnin menetelmissä

Lähes kaikki muurahaisyhdyskuntaoptimoinnin teoria on rakennettu kokeellisten tutkimusten perusteella kuten on käytännössä kaikkien muidenkin metaheuristiikkojen. Yksi syy siihen, että muurahaisyhdyskuntaoptimoinnille on vaikea matemaattisesti todistaa mitään teoreettisia ominaisuuksia, on se, että edellä luvussa 2.2 esitelty muurahaisyhdyskuntaoptimoinnin menetelmien runko on niin yleinen. Vaikka yleisyys onkin tavoiteltava ominaisuus, niin se tekee esimerkiksi menetelmien konvergenssin todistamisen mahdottomaksi. Jos kuitenkin muurahaisyhdyskuntaoptimoinnin menetelmässä ratkaisun rakentaminen ja feromonien päivitys noudattavat tiettyjä periaatteita, voidaan eräitä konvergenssilauseita todistaa. Voidaan todistaa, että jotkin muurahaisyhdyskuntaoptimoinnin menetelmät löytävät jossain vaiheessa jonkin optimaalisen ratkaisun (*arvokonvergenssi* (*convergence in value*)). Voidaan myös todistaa, että jotkin muurahaisyhdyskuntaoptimoinnin menetelmät generoivat jostain iteraatiokierroksesta lähtien aina saman optimaalisen ratkaisun (*ratkaisukonvergenssi* (*convergence in solution*)). Valitettavasti konvergenssinopeudesta ei näissäkään tapauksissa ole mitään tietoa.

4.1 Konvergenssiin vaadittavat ominaisuudet

Sekä arvo- että ratkaisukonvergenssitodistuksissa oletetaan, että ratkaisujen rakentamiseksi käytetään todennäköisyyksiä (vertaa (3.2))

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \in N_i^k} \tau_{il}^\alpha}, & \text{jos } j \in N_i^k; \\ 0, & \text{jos } j \notin N_i^k; \end{cases} \tag{4.1}$$

missä käytetään samoja merkintöjä kuin aikaisemmin.

Arvokonvergenssitodistuksessa oletetaan, että kaikilla kaarilla $(i,j) \in A$ feromonijäljet τ_{ij} alustetaan arvoksi τ_0 ja että feromonien haihtuminen toteutetaan kaavalla (3.3) eli

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij}, \quad \forall (i,j) \in A, \tag{4.2}$$

missä $0 < \rho \leq 1$. Feromonien lisäys taas hoidetaan kaavalla

$$\tau_{ij} \leftarrow \tau_{ij} + q_f(s^t), \quad \forall (i,j) \in s^t, \tag{4.3}$$

missä $q_f(s)$ on jokin fitnessfunktio, jonka arvo on sitä suurempi mitä parempi ratkaisu s on, ja s^t paras tähän mennessä saatu ratkaisu. Esimerkiksi kauppamatkustajaongelmassa voidaan valita $q_f(K^k) = \frac{1}{L^k}$, missä K^k ja L^k ovat kuten aiemmin. Feromonijäljillä τ_{ij} oletetaan vielä olevan muuttumaton alaraja τ_{min}, jolle on voimassa $\tau_{min} < q_f(s^*)$, missä s^* on optimaalinen ratkaisu. Ratkaisukonvergenssitodistuksessa käytetään feromonien päivityksen suhteen muuten samoja oletuksia kuin arvokonvergenssitodistuksessa paitsi, että feromonijälkien τ_{ij} alaraja $\tau_{min}(t)$ riippuu ajanhetkestä t. Todistuksissa käytetään myös merkintää θ, joka tarkoittaa nykyisen iteraation indeksiä.

4.2 Arvokonvergenssi

Ennen arvokonvergenssin todistamista menetelmille, jotka toteuttavat edellä esitetyt oletukset, todistetaan kaksi lemmaa, joita sitten käytetään avuksi arvokonvergenssitodistuksessa.

Lemma 4.1. *Kaikilla kaarilla $(i,j) \in A$ feromonijäljille τ_{ij} pätee*

$$\lim_{\theta \to \infty} \tau_{ij}(\theta) \leq \tau_{max} = \frac{q_f(s^*)}{\rho}.$$

Todistus: Suurin mahdollinen feromonien määrä, joka lisätään jokaiselle kaarelle $(i,j) \in A$ jokaisella iteraatiolla, on kaavan (4.3) mukaan $q_f(s^*)$. Näin ollen kaavojen (4.2) ja (4.3) perusteella suurin mahdollinen feromonien määrä ensimmäisen iteraation jälkeen kaarella (i,j) on $(1-\rho)\tau_0 + q_f(s^*)$ ja toisen iteraation jälkeen $(1-\rho)[(1-\rho)\tau_0 + q_f(s^*)] + q_f(s^*) = (1-\rho)^2\tau_0 + (1-\rho)q_f(s^*) + q_f(s^*)$ ja niin edelleen. Iteraation θ jälkeen feromonien määrä kaarella (i,j) on siis korkeintaan

$$\tau_{ij}^{max}(\theta) = (1-\rho)^\theta \tau_0 + \sum_{i=1}^{\theta} (1-\rho)^{\theta - i} q_f(s^*).$$

Koska $0 < \rho \leq 1$, niin saadaan

$$\lim_{\theta \to \infty} \tau_{ij}(\theta) \leq \tau_{max} = \lim_{\theta \to \infty} \tau_{ij}^{max}(\theta) = \frac{q_f(s^*)}{1-(1-\rho)} = \frac{q_f(s^*)}{\rho}. \quad \square$$

Lemma 4.2. *Kun optimaalinen ratkaisu s^* on löydetty, niin kaarille $(i,j) \in s^*$ pätee*

$$\lim_{\theta \to \infty} \tau_{ij}(\theta) = \tau_{max} = \frac{q_f(s^*)}{\rho}.$$

Todistus: Olkoon θ^* iteraatio, jolla optimaalinen ratkaisu s^* ensimmäisen kerran löydetään. Kun tarkastellaan iteraatiota $\theta \geq \theta^*$ ja kaaria $(i,j) \in s^*$, saadaan

$$\tau_{ij}(\theta) = (1-\rho)^{\theta-\theta^*}\tau_{ij}(\theta^*) + \sum_{i=\theta^*+1}^{\theta} (1-\rho)^{\theta-i} q_f(s^*).$$

Raja-arvona kaarilla $(i,j) \in s^*$ saadaan nyt

$$\tau_{max} = \lim_{\theta\to\infty} \tau_{ij}(\theta) = \frac{q_f(s^*)}{\rho}. \quad \square$$

Nyt siis kaikkien kaarien $(i,j) \in A$ feromonijäljillä τ_{ij} on yläraja $\frac{q_f(s^*)}{\rho}$, jota kaarien $(i,j) \in s^*$ feromonijäljet τ_{ij} lähestyvät asymptoottisesti.

Lause 4.3. *(Arvokonvergenssi) Olkoon $P^*(\theta)$ todennäköisyys, että algoritmi löytää optimaalisen ratkaisun ainakin kerran θ:n ensimmäisen iteraation aikana. Mielivaltaisen pienelle $\epsilon > 0$ ja riittävän suurelle θ pätee tällöin*

$$P^*(\theta) \geq 1 - \epsilon,$$

ja

$$\lim_{\theta\to\infty} P^*(\theta) = 1.$$

Todistus: Koska jokaisella kaarella $(i,j) \in A$ feromonijäljen arvoa rajoittavat alaraja τ_{min} ja yläraja τ_{max}, niin käytettäessä ratkaisujen rakentamiseen kaavaa (4.1) jokainen sallittu valinta tehdään vähintään todennäköisyydellä $p_{min} > 0$. Todennäköisyydelle p_{min} saadaan seuraava alaraja:

$$p_{min} \geq \hat{p}_{min} = \frac{\tau_{min}^{\alpha}}{(n-1)\tau_{max}^{\alpha} + \tau_{min}^{\alpha}},$$

missä n on graafissa G olevien solmujen määrä. Todennäköisyys $\hat{p}_{min}$ saavutettaisiin, jos yhdellä solmusta lähtevällä kaarella (i,j) on feromonijäljen τ_{ij} arvo τ_{min} ja kaikkiin loppuihin solmuihin (myös itseensä) johtavilla kaarilla (i,j) feromonijäljen τ_{ij} arvo on τ_{max}. Nyt kaikkien ratkaisujen, myös minkä tahansa optimaalisen ratkaisun s^*, rakentamisen todennäköisyys on vähintään $(\hat{p}_{min})^{N_c} > 0$, missä N_c on ratkaisuun kuuluvien solmujen enimmäismäärä. Koska riittää, että yksikin muurahainen löytää optimaalisen ratkaisun s^*, niin

$$P^*(\theta) \geq 1 - (1 - \hat{p}_{min}^{N_c})^{\theta}.$$

Kun nyt valitaan riittävän suuri θ, niin

$$P^*(\theta) \geq 1 - \epsilon.$$

Raja-arvona saadaan

$$\lim_{\theta \to \infty} P^*(\theta) \geq 1.$$

Mutta koska $P^*(\theta)$ tarkoittaa todennäköisyyttä, niin $0 \leq P^*(\theta) \leq 1$. Näin ollen

$$\lim_{\theta \to \infty} P^*(\theta) = 1. \quad \square$$

4.3 Ratkaisukonvergenssi

Ratkaisukonvergenssin todistuksessa avainasemassa on feromonijälkien τ_{ij} alarajan $\tau_{min}(\theta)$ arvon väheneminen kohti nollaa ajan kuluessa. Tämä alarajan arvo ei saa kuitenkaan vähetä liian nopeasti, jotta lopulta löydetään optimaalinen ratkaisu. Ennen ratkaisukonvergenssin todistamista osoitetaan, että alarajan $\tau_{min}(\theta)$ vähetessä sopivalla vauhdilla löydetään optimaalinen ratkaisu jossain vaiheessa todennäköisyydellä 1.

Lause 4.4. *Olkoon feromonijälkien alaraja*

$$\tau_{min}(\theta) = \tfrac{d}{\ln(\theta+1)}, \quad \forall \theta \geq 1,$$

missä d on vakio. Olkoon $P^*(\theta)$ todennäköisyys, että algoritmi löytää optimaalisen ratkaisun ainakin kerran θ:n ensimmäisen iteraation aikana. Tällöin pätee

$$\lim_{\theta \to \infty} P^*(\theta) = 1.$$

Todistus: Olkoon T_θ sellainen tapaus, että iteraatio θ on ensimmäinen iteraatio, jolla jokin optimaalinen ratkaisu löydetään. Nyt tapaus $\bigwedge_{\theta=1}^{\infty} \neg T_\theta$ tarkoittaa, että mitään optimaalista ratkaisua ei löydetä millään iteraatiolla. Olkoon s^* jokin mielivaltainen optimaalinen ratkaisu. Tällöin tapauksesta $\bigwedge_{\theta=1}^{\infty} \neg T_\theta$ seuraa, että ratkaisua s^* ei löydetä koskaan. Näin ollen

$$P\left(\bigwedge_{\theta=1}^{\infty} \neg T_\theta\right) \leq P(\text{ratkaisua } s^* \text{ ei löydetä koskaan}). \tag{4.4}$$

Kun käytetään samoja merkintöjä kuin lauseessa 4.3, saadaan

$$p_{min} \geq \hat{p}_{min}(\theta) = \frac{\tau_{min}^{\alpha}(\theta)}{(n-1)\tau_{max}^{\alpha} + \tau_{min}^{\alpha}(\theta)} \geq \frac{\tau_{min}^{\alpha}(\theta)}{n\tau_{max}^{\alpha}} = \hat{p}'_{min}(\theta).$$

Nyt alaraja sille, että muurahainen k rakentaa optimaalisen ratkaisun s^*, on $(\hat{p}'_{min}(\theta))^{N_c}$, missä N_c on ratkaisuun kuuluvien solmujen enimmäismäärä. Tämä alaraja on riippumaton siitä, mitä on tapahtunut ennen iteraatiota θ. Näin ollen saadaan

$$\begin{aligned} P(\text{ratkaisua } s^* \text{ ei löydetä koskaan}) &\leq \prod_{\theta=1}^{\infty}(1-(\hat{p}'_{min}(\theta))^{N_c}) \\ &= \prod_{\theta=1}^{\infty}\left(1-\left(\frac{\tau^{\alpha}_{min}(\theta)}{n\tau^{\alpha}_{max}}\right)^{N_c}\right). \end{aligned} \tag{4.5}$$

Tutkitaan nyt tämän tulon logaritmia. Lauseen oletuksen ja logaritmien laskusääntöjen mukaan saadaan

$$\begin{aligned} \ln\left[\prod_{\theta=1}^{\infty}\left(1-\left(\frac{\tau^{\alpha}_{min}(\theta)}{n\tau^{\alpha}_{max}}\right)^{N_c}\right)\right] &= \sum_{\theta=1}^{\infty}\ln\left[1-\left(\frac{\tau^{\alpha}_{min}(\theta)}{n\tau^{\alpha}_{max}}\right)^{N_c}\right] \\ &= \sum_{\theta=1}^{\infty}\ln\left[1-\left(\frac{\left(\frac{d}{\ln(\theta+1)}\right)^{\alpha}}{n\tau^{\alpha}_{max}}\right)^{N_c}\right]. \end{aligned} \tag{4.6}$$

Koska kaikille $x<1$ on voimassa $\ln(1-x)\leq -x$, saadaan

$$\begin{aligned} &\sum_{\theta=1}^{\infty}\ln\left[1-\left(\frac{\left(\frac{d}{\ln(\theta+1)}\right)^{\alpha}}{n\tau^{\alpha}_{max}}\right)^{N_c}\right] \\ &\leq -\left(\frac{d^{\alpha}}{n\tau^{\alpha}_{max}}\right)^{N_c}\sum_{\theta=1}^{\infty}\left(\frac{1}{\ln(\theta+1)}\right)^{\alpha N_c}. \end{aligned} \tag{4.7}$$

Toisaalta jokaiselle vakiolle $\delta>0$ ja riittävän suurelle x on voimassa $(\ln x)^i\leq \delta\cdot x$ ja näin ollen $\frac{\delta}{(\ln x)^i}\geq\frac{1}{x}$. Kun vielä muistetaan, että harmoninen sarja $\sum_{x=1}^{\infty}\frac{1}{x}$ hajaantuu eli $\sum_{x=1}^{\infty}\frac{1}{x}=\infty$, saadaan

$$-\left(\frac{d^{\alpha}}{n\tau^{\alpha}_{max}}\right)^{N_c}\sum_{\theta=1}^{\infty}\left(\frac{1}{\ln(\theta+1)}\right)^{\alpha N_c}=-\infty. \tag{4.8}$$

Yhdistämällä kaavat (4.6)–(4.8), saadaan

$$\ln\left[\prod_{\theta=1}^{\infty}\left(1-\left(\frac{\tau^{\alpha}_{min}(\theta)}{n\tau^{\alpha}_{max}}\right)^{N_c}\right)\right]\leq-\infty.$$

Logaritmifunktion käyttäytymisestä seuraa, että

$$\prod_{\theta=1}^{\infty}\left(1-\left(\frac{\tau^{\alpha}_{min}(\theta)}{n\tau^{\alpha}_{max}}\right)^{N_c}\right)=0.$$

Tällöim yhtälöistä (4.4) ja (4.5) saadaan

$$P\left(\bigwedge_{\theta=1}^{\infty}\neg T_{\theta}\right)\leq 0.$$

Kun vielä muistetaan, että $\lim_{\theta\to\infty} P^*(\theta) = 1 - P\left(\bigwedge_{\theta=1}^{\infty} \neg T_\theta\right)$, saadaan lopulta

$$\lim_{\theta\to\infty} P^*(\theta) = 1. \quad \square$$

Todistetaan vielä ennen ratkaisukonvergenssin todistamista yksi lemma, jota sitten hyödynnetään ratkaisukonvergenssin todistamisessa.

Lemma 4.5. *Olkoon feromonijälkien alaraja*

$$\tau_{min}(\theta) = \tfrac{d}{\ln(\theta+1)}, \quad \forall\theta \geq 1,$$

missä d on vakio. Kun optimaalinen ratkaisu s^* on löydetty, jokaiselle kaarelle $(i,j) \notin s^*$ pätee

$$\lim_{\theta\to\infty} \tau_{ij}(\theta) = 0.$$

Todistus: Olkoon θ^* taas ensimmäinen iteraatio, jolla optimaalinen ratkaisu s^* löydetään. Kaarille $(i,j) \notin s^*$ ei iteraation θ^* jälkeen enää lisätä feromoneja. Näin ollen näillä kaarilla feromonien määrä vähenee feromonien haihtumisen vuoksi. Kaarilla $(i,j) \notin s^*$ saadaan siis $\tau_{ij}(\theta^*+1) = \max\{\tau_{min}(\theta^*+1), (1-\rho)\tau_{ij}(\theta^*)\}$, $\tau_{ij}(\theta^*+2) = \max\{\tau_{min}(\theta^*+2), (1-\rho)^{\tau}_{ij}(\theta^*)\}$ ja niin edelleen. Kun $\theta \geq \theta^*$, saadaan siis $\tau_{ij}(\theta) = \max\{\tau_{min}(\theta), (1-\rho)^{\theta-\theta^*}\tau_{ij}(\theta^*)\}$. Kun muistetaan lemman oletus ja se, että $0 < \rho \leq 1$, saadaan

$$\begin{aligned}&\lim_{\theta\to\infty} \tau_{ij}(\theta) = \max\left\{\lim_{\theta\to\infty} \tfrac{d}{\ln(\theta+1)}, \lim_{\theta\to\infty}(1-\rho)^{\theta-\theta^*}\tau_{ij}(\theta^*)\right\}\\ &= \max\{0,0\} = 0.\end{aligned}$$

$\square$

Lause 4.6. *(Ratkaisukonvergenssi) Olkoon feromonijälkien alaraja*

$$\tau_{min}(\theta) = \tfrac{d}{\ln(\theta+1)}, \quad \forall\theta \geq 1,$$

missä d on vakio. Olkoon θ^* ensimmäinen iteraatio, jolla optimaalinen ratkaisu s^* löydetään, ja $P(s^*,\theta,k)$ todennäköisyys, että mielivaltainen muurahainen k rakentaa optimaalisen ratkaisun s^* iteraatiolla $\theta > \theta^*$. Tällöin pätee

$$\lim_{\theta\to\infty} P(s^*,\theta,k) = 1.$$

Todistus: Olkoon muurahainen k solmussa i ja kaari (i,j) kuuluu optimaaliseen ratkaisuun s^*. Todennäköisyydelle $p^*_{ij}(\theta)$, että muurahainen k tekee optimaalisen valinnan, saadaan alaraja $\hat{p}^*_{ij}(\theta)$ seuraavasti:

$$\hat{p}^*_{ij}(\theta) = \frac{(\tau^*_{ij}(\theta))^\alpha}{(\tau^*_{ij}(\theta))^\alpha + \sum_{(i,h)\notin s^*}(\tau^*_{ih}(\theta))^\alpha}.$$

Nyt lemmojen 4.2 ja 4.5 perusteella saadaan

$$\begin{aligned}\lim_{\theta\to\infty}\hat{p}_{ij}^{*}(\theta) &= \frac{\lim_{\theta\to\infty}(\tau_{ij}^{*}(\theta))^{\alpha}}{\lim_{\theta\to\infty}(\tau_{ij}^{*}(\theta))^{\alpha}+\sum_{(i,h)\notin s^{*}}\lim_{\theta\to\infty}(\tau_{ih}^{*}(\theta))^{\alpha}} \\ &= \frac{\tau_{max}^{\alpha}}{\tau_{max}^{\alpha}+\sum_{(i,h)\notin s^{*}}0^{\alpha}} = 1.\end{aligned}$$

Tämä tarkoittaa sitä, että jokainen muurahainen tekee jokaisessa solmussa lopulta optimaalisen valinnan todennäköisyydellä 1. Näin ollen

$$\lim_{\theta\to\infty} P(s^{*},\theta,k) = 1. \quad \square$$

Vaikka ratkaisukonvergenssi on periaatteessa vahvempi tulos kuin arvokonvergenssi, niin arvokonvergenssi on riittävä tulos, koska se takaa optimaalisen ratkaisun löytymisen. Valitettavasti edellä todistetut lauseet eivät kerro mitään siitä, kuinka nopeasti optimaalinen ratkaisu löydetään.

4.4 Konvergenssitodistukset ja muurahaisyhdyskuntaoptimoinnin menetelmien lisäominaisuudet

Monissa muurahaisyhdyskuntaoptimoinnin menetelmissä on sellaisia ominaisuuksia, joita edellä esitetyissä konvergenssitodistuksessa ei oletettu. Tärkeimmät näistä ominaisuuksista ovat lokaali haku ja heuristinen informaatio.

Tarkastellaan ensin lokaalia hakua. Lokaalissa haussa yritetään parantaa muurahaisen rakentamaa ratkaisua s. Jos löydetään ratkaisua s parempi ratkaisu s', sitä käytetään feromonijälkien päivittämiseen. Molemmat edellä esitetyt konvergenssitodistukset riippuvat vain tavasta, jolla ratkaisut rakennetaan, joten lokaali haku ei niihin vaikuta.

Heuristista informaatiota käytettäessä kaava (4.1) korvataan kaavalla (vertaa (3.2))

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum_{l\in N_i^k}\tau_{il}^{\alpha}\cdot\eta_{il}^{\beta}}, & \text{jos } j\in N_i^k; \\ 0, & \text{jos } j\notin N_i^k. \end{cases} \tag{4.9}$$

Heuristisen informaation käyttö vaikuttaa todennäköisyyden p_{min} alarajoihin. Jos kuitenkin $0 < \eta_{ij} < \infty$ jokaisella kaarella (i,j) ja $\beta < \infty$, ei heuristisen informaation käyttö vaikuta edellä esitettyihin konvergenssitodistuksiin. Näin ollen arvo- ja ratkaisukonvergenssi pysyvät voimassa, vaikka niiden oletuksiin lisättäisiinkin sekä lokaali haku että heuristinen informaatio.

4.5 Joidenkin muurahaisyhdyskuntaoptimoinnin menetelmien konvergenssi

Vaikka suoraan ei nähdä minkään aiemmin esitetyn muurahaisyhdyskuntaoptimoinnin menetelmän täyttävän arvo- tai ratkaisukonvergenssin todistuksissa vaadittuja oletuksia, niin lähemmän tarkastelun jälkeen huomataan joidenkin menetelmien kuitenkin täyttävän ne.

4.5.1 MAX-MIN-muurahaisjärjestelmä

MAX-MIN-muurahaisjärjestelmässä on oikeastaan vain kaksi pientä eroa arvokonvergenssin todistuksessa käytettyihin oletuksiin. Ensinnäkin MAX-MIN-muurahaisjärjestelmässä käytetään eksplisiittistä arvoa τ_{max}, kun taas arvokonvergenssin kohdalla tämä arvo oli implisiittinen. MAX-MIN-muurahaisjärjestelmässä käytetty eksplisiittinen arvo on kuitenkin arvio oletuksissa käytetylle implisiittiselle arvolle. Eksplisiittinen arvo on korkeintaan implisiittisen arvon suuruinen ja lähestyy tätä koko ajan. Tämä eksplisiittinen ylärajan arvo ei mitenkään vaikuta yhteenkään edellä esitettyyn todistukseen.

MAX-MIN-muurahaisjärjestelmässä voidaan käyttää feromonijälkien päivitykseen tähän mennessä parhaan ratkaisun sijaan iteraation parasta ratkaisua. Kuten aikaisemmin on todettu (luku 3.6) kannattaa algoritmin suorituksen edetessä käyttää yhä useammin päivitykseen tähän mennessä parasta ratkaisua, kunnes lopulta käytetään vain tähän mennessä parasta ratkaisua. Tämäkään eroavaisuus ei siis vaikuta arvokonvergenssiin. Näin ollen MAX-MIN-muurahaisjärjestelmälle arvokonvergenssi on voimassa. Jos lisäksi feromonijäljen alarajalle käytetään samaa kaavaa kuin lauseessa 4.6, saadaan myös ratkaisukonvergenssi voimaan MAX-MIN-muurahaisjärjestelmässä.

4.5.2 Muurahaisyhdyskuntajärjestelmä

Muurahaisyhdyskuntajärjestelmä eroaa kolmella tavalla arvokonvergenssin todistuksessa käytetyistä oletuksista. Ensinnäkin muurahaisyhdyskuntajärjestelmässä käytetään eri sääntöä ratkaisujen rakentamiseen (kaava (3.12)) kuin arvokonvergenssin todistuksessa on oletettu. Toiseksi muurahaisyhdyskuntajärjestelmässä ei feromonien haihtumista toteuteta kaikilla kaa-

rilla $(i, j) \in A$ vaan ainoastaan kaarilla, jotka kuuluvat tähän mennessä parhaaseen ratkaisuun. Kolmas ero on se, että muurahaisyhdyskuntajärjestelmässä käytetään lokaalia feromonijälkien päivittämistä, joka toteutetaan heti, kun muurahainen on liikkunut kaarella (i, j).

Esiteltäessä muurahaisyhdyskuntajärjestelmää todettiin, että siinä on feromonijäljillä τ_{ij} implisiittinen alaraja τ_0 ja myös implisiittinen yläraja. Oletetaan, että kaari (i, j) ei ole feromonijälkien ja heuristisen informaation kannalta paras mahdollinen valinta. Kun vielä muurahaisyhdyskuntajärjestelmässä käytetylle parametrille q_0 pätee $0 < q_0 < 1$, saadaan kaaren (i, j) valinnan todennäköisyydelle alaraja $(1 - q_0) \cdot \hat{p}_{min}$. Jos kaari (i, j) on feromonijälkien ja heuristisen informaation kannalta paras mahdollinen valinta, niin todennäköisyys, että se valitaan on q_0. Nyt kaikkien kaarien valitsemistodennäköisyys on suurempi kuin 0 ja arvokonvergenssin todistusta voidaan soveltaa muurahaisyhdyskuntajärjestelmään.

Ratkaisukonvergensissa vaadittiin, että feromonijäljillä τ_{ij} on ajasta t riippuva alaraja. Muurahaisyhdyskuntajärjestelmässä tämä alaraja ei kuitenkaan riipu ajasta t, vaan se on vakio τ_0. Näin ollen ratkaisukonvergenssia ei saada voimaan muurahaisyhdyskuntajärjestelmässä.

4.5.3 Muut muurahaisjärjestelmät

Edellä esitetyt konvergenssitodistukset eivät päde muurahaisjärjestelmälle, elitistiselle muurahaisjärjestelmälle eikä järjestykseen perustuvalle muurahaisjärjestelmälle. Tämä johtuu siitä, että näissä menetelmissä feromonijäljillä τ_{ij} ei ole mitään vakioalarajaa, vaan tämä alaraja voi pienentyä paljon nopeammin kuin ratkaisukonvergenssin todistuksessa on oletettu. Jos kyseisille menetelmille asetettaisiin sopivat feromonijälkien τ_{ij} vakioalarajat τ_{min}, saataisiin niille arvokonvergenssi voimaan. Jos taas kyseisille menetelmille asetettaisiin feromonijäljille τ_{ij} lauseessa 4.6 esitetyt ajasta t riippuvat alarajat, saataisiin niille ratkaisukonvergenssi voimaan.

5 Muurahaisyhdyskuntaoptimointi ja jatkuvat optimointitehtävät

Siitä asti, kun ensimmäiset muurahaisyhdyskuntaoptimoinnin menetelmät kombinatorisille ongelmille kehitettiin vuonna 1991, on myös jatkuvien ongelmien ratkaisuun soveltuvia muurahaisyhdyskuntaoptimoinnin menetelmiä yritetty kehittää. Tämä ei kuitenkaan ole ollut kovin yksinkertaista ja jatkuville ongelmille soveltuvia menetelmiä, jotka noudattavat täysin muurahaisyhdyskuntaoptimoinnin periaatteita ei olekaan olemassa kovin montaa. Tästä syystä tässä luvussa esitellään myös joitakin muurahaisten käyttäytymiseen perustuvia menetelmiä, jotka eivät oikeastaan ole muurahaisyhdyskuntaoptimoinnin menetelmiä. Osa tämän luvun menetelmistä esitellään rajoitteettomille optimointitehtäville, joiden optimoinnin laatuun (maksimointi tai minimointi) ei oteta kantaa. Optimointitehtävä on siis muotoa

$$\begin{aligned} &\max/\min \quad f(x) \\ &\text{s.e.} \qquad\quad x \in R^n. \end{aligned} \tag{5.1}$$

Loput tämän luvun menetelmistä esitellään optimointitehtäville, joiden optimoinnin laatuun ei myöskään oteta kantaa, mutta tehtävissä on laatikkorajoitteet. Tällainen optimointitehtävä on muotoa

$$\begin{aligned} &\max/\min \quad f(x) \\ &\text{s.e.} \qquad\quad x_i^a \leq x_i \leq x_i^y, \quad i = 1, 2, ..., n. \end{aligned} \tag{5.2}$$

Tehtävän dimensio on siis n ja ratkaisun komponentin i pitää saada arvo alarajan x_i^a ja ylärajan x_i^y väliltä. Joissakin menetelmissä tarvitaan ratkaistavalle optimointitehtävälle fitnessfunktiota, joka saa sitä suuremman arvon mitä parempi ratkaisu on. Tätä fitnessfunktiota ei määritellä tarkemmin, vaan se voidaan valita aina tilannekohtaisesti. Käytetään tässä luvussa fitnessfunktiosta merkintää q_f.

Tämän luvun menetelmät esitellään tehtäville, joissa on vain hyvin yksinkertaisia rajoitteita tai rajoitteita ei ole lainkaan. Luvun lopussa kerrotaankin lyhyesti, miten optimointitehtävien rajoitteita voidaan käsitellä, jotta tässä luvussa esiteltävillä menetelmillä voitaisiin ratkaista yleisiä rajoitteita sisältäviä optimointitehtäviä.

Tässä työssä ei käsitellä jatkuvien optimointitehtävien ratkaisuun tarkoitettujen muurahaisyhdyskuntaoptimoinnin menetelmien konvergenssi-

teoriaa. Tämä johtuu siitä, ettei näille menetelmille ole pystytty luomaan minkäänlaista konvergenssiteoriaa.

Tämä luvun perustana ovat suurimmalta osaltaan [1, 2, 6, 12, 20, 21, 22, 23, 25, 26, 28, 29, 30, 33, 35].

5.1 Jatkuva muurahaisyhdyskuntaoptimointi

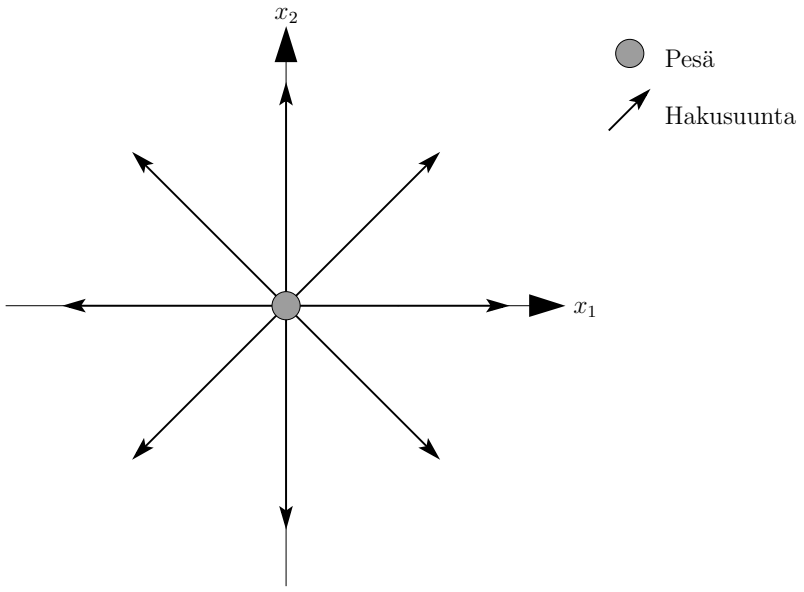
Jatkuva muurahaisyhdyskuntaoptimointi (*continuous ant colony optimization*) oli ensimmäinen jatkuvien optimointitehtävien ratkaisuun tarkoitettu muurahaisten käyttäytymiseen perustuva menetelmiä.

Jatkuvaa muurahaisyhdyskuntaoptimointia käytettäessä pitää ensin määrittää aloituspiste, josta optimointi aloitetaan. Tätä pistettä kutsutaan pesäksi. Pesältä muurahaiset lähtevät johonkin ennalta määrättyyn *hakusuuntaan* (katso kuva 5.1), joita on H kappaletta. Hakusuuntavektori i alkaa pesästä ja loppuu *määränpäähän* i. Näitä hakusuuntavektoreita ja määränpäitä päivitetään menetelmän suorituksen aikana. Jokaiselle hakusuunnalle i lasketaan määränpäässä i kohdefunktion arvo. Näiden kohdefunktioiden arvojen mukaan hakusuunnat järjestetään paremmuusjärjestykseen. Ennen itse optimoinnin aloittamista alustetaan vielä kaikille hakusuunnille feromonijäljeksi $\tau_i(0) = 1$. Optimointia suorittavat *lokaalit* ja *globaalit muurahaiset*. Muurahaisten yhteismäärä on m, joista globaaleja muurahaisia on G kappaletta ja lokaaleja muurahaisia L kappaletta.

5.1.1 Globaalit muurahaiset

Globaalit muurahaiset korvaavat G edellisen iteraatiokierroksen huonointa hakusuuntaa uusilla hakusuunnilla. Korvaukset toteutetaan kahdella eri menetelmällä: *satunnaiskululla* (*random walk*) ja *jälkien diffuusiolla* (*trail diffusion*). Nämä menetelmät muistuttavat geneettisten algoritmien menetelmiä [16, 18]. Globaaleista muurahaisista SK kappaletta käyttää satunnaiskulkua ja JD kappaletta jälkien diffuusiota. Näin ollen $G = SK+JD$.

Satunnaiskulussa valitaan ensin uudelle hakusuuntavektorille satunnaisesti jokin jo olemassa oleva hakusuuntavektori *vanhemmaksi*. Tämän vanhemman ensimmäinen komponentti otetaan uuden hakusuuntavektorin ensimmäiseksi komponentiksi. Toiseksi komponentiksi uuteen hakusuuntavektoriin otetaan todennäköisyydellä p_r jonkin satunnaisesti valitun toisen vanhemman toinen komponentti ja todennäköisyydellä $1 - p_r$ otetaan samal-



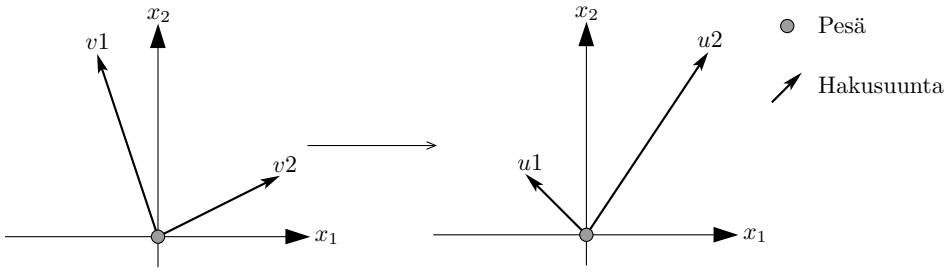
Kuva 5.1: Esimerkki alkuperäisistä hakusuunnista kaksiulotteisessa tapauksessa, jossa hakusuuntia on kahdeksan.

ta vanhemmalta toinen komponentti, jolta otettiin ensimmäinenkin komponentti. Näin jatketaan kunnes uusi hakusuuntavektori ollaan saatu valmiiksi (katso kuva 5.2). Jos $p_r = 1$, kaikki uuden hakusuuntavektorin komponentit ovat eri vanhemmilta. Jos taas $p_r = 0$, kaikki uuden hakusuuntavektorin komponentit ovat samalta vanhemmalta. Saatuun uuteen hakusuuntavektoriin tehdään vielä todennäköisyydellä p_m mutaatio. Mutaatiossa jokaiseen uuden hakusuuntavektorin komponenttiin joko lisätään tai vähennettään arvo

$$\Delta(t, R) = R(1 - p^{(1-\frac{t}{T})^b}), \tag{5.3}$$

missä R on maksimaalisen *mutaatioaskeleen* pituuden kertova parametri, p on satunnaisluku tasaisesta jakaumasta väliltä [0,1], t on tämän hetkinen iteraatiokierros, T on iteraatiokierrosten kokonaismäärä, joka on menetelmän parametri, ja $b > 0$ on parametri. Kaavasta (5.3) huomataan, että todennäköisyys, että mutaatioaskeleen pituus on lähellä nollaa kasvaa algoritmin suorituksen edetessä.

Jälkien diffuusiossa valitaan uudelle hakusuuntavektorille satunnaisesti kaksi vanhempaa jo olemassa olevista hakusuuntavektoreista. Käytetään uu-



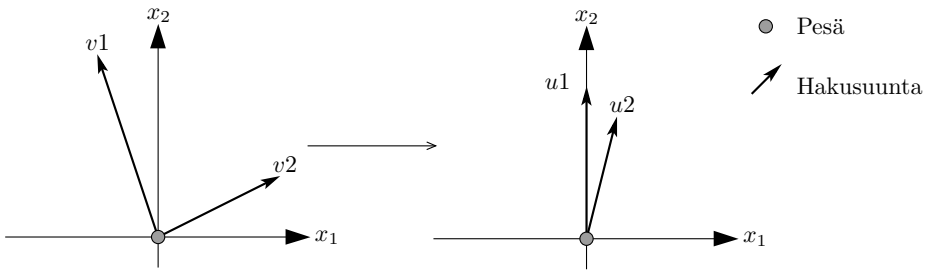
Kuva 5.2: Esimerkki satunnaiskulusta kaksiulotteisessa tapauksessa. Vasemmalla on esitetty uusien hakusuuntavektorien vanhemmat $v1$ ja $v2$. Oikealla on esitetty esimerkki satunnaiskululla ilman mutaatiota saaduista uusista hakusuuntavektoreista. Hakusuuntavektori $u1$ on saanut ensimmäisen komponenttinsa hakusuuntavektorilta $v1$ ja toisen komponenttinsa hakusuuntavektorilta $v2$. Hakusuuntavektori $u2$ on taas saanut ensimmäisen komponenttinsa hakusuuntavektorilta $v2$ ja toisen komponenttinsa hakusuuntavektorilta $v1$.

den hakusuuntavektorin komponentista i merkintää x_i^u, ensimmäisen vanhemman komponentista i merkintää x_i^{v1} ja toisen vanhemman komponentista i merkintää x_i^{v2}. Jälkien diffuusiossa saadaan joko $x_i^u = x_i^{v1}$, $x_i^u = x_i^{v2}$ tai $x_i^u = p \cdot x_i^{v1} + (1-p) \cdot x_i^{v2}$, missä p on satunnaisluku tasaisesta jakaumasta väliltä [0,1] (katso kuva 5.3). Todennäköisyys, että valitaan näistä viimeinen vaihtoehto on yhtä suuri kuin satunnaiskulussa käytetty mutaatiotodennäköisyys p_m. Ensimmäisen ja toisen vaihtoehdon valinnan todennäköisyydet ovat yhtä suuria eli molemmat ovat $\frac{1-p_m}{2}$. Näin ollen jos esimerkiksi $p_m = 0{,}5$, on sekä ensimmäisen että toisen vaihtoehdon valinnan todennäköisyys $\frac{1-0{,}5}{2} = 0{,}25$.

Tärkeä kysymys on, mitkä feromonijälkien arvot uusille hakusuunnille asetetaan. Jatkuvassa muurahaisyhdyskuntaoptimointissa uudeksi feromonijäljen $\tau_i(t)$ arvoksi asetetaan uuden hakusuunnan vanhempien feromonijälkien keskiarvo.

5.1.2 Lokaalit muurahaiset

Globaalien muurahaisten lopetettua, lokaalit muurahaiset aloittavat optimoinnin. Kaikki L lokaalia muurahaista valitsee eri hakusuunnan. Toden-



Kuva 5.3: Esimerkki jälkien diffuusiosta kaksiulotteisessa tapauksessa. Vasemmalla on esitetty uusien hakusuuntavektorien vanhemmat $v1$ ja $v2$. Oikealla on esitetty esimerkki jälkien diffuusiolla saaduista uusista hakusuuntavektoreista. Uusia hakusuuntavektoreita muodostettaessa on käytetty kaavaa $x_i^u = p \cdot x_i^{v1} + (1-p) \cdot x_i^{v2}$. Hakusuuntavektorin $u1$ ensimmäinen komponentti ollaan saatu käyttämällä arvoa $p = \frac{2}{3}$ ja toinen komponentti ollaan saatu käyttämällä arvoa $p = \frac{3}{4}$. Hakusuuntavektorin $u2$ molemmat komponentit ollaan saatu käyttämällä arvoa $p = \frac{1}{2}$.

näköisyys, että lokaali muurahainen valitsee hakusuunnan i on (vertaa (3.2))

$$p_i(t) = \frac{\tau_i(t)}{\sum_k \tau_k(t)}. \tag{5.4}$$

Valittuaan hakusuunnan i lokaali muurahainen liikkuu määränpäähän i. Määränpäästä i se ottaa vielä lyhyen askeleen s_i joko samaan suuntaan kuin edellinen hakusuunnan i valinnut lokaali muurahainen liikkui tai johonkin satunnaiseen suuntaan. Jos edellinen hakusuunnan i valinnut lokaali muurahainen onnistui parantamaan kohdefunktion arvoa hakusuunnassa i, niin valitaan sama suunta, johon edellinen hakusuunnan i valinnut lokaali muurahainen liikkui, muuten valitaan jokin satunnainen suunta. Otettuaan askeleen s_i lokaali muurahainen päätyy pisteeseen x_i'. Jos kohdefunktion arvo pisteessä x_i' on parempi kuin määränpäässä i, päivitetään määränpää i pisteeseen x_i', samalla hakusuuntavektori päivittyy. Myös feromonijälkeä $\tau_i(t)$ päivitetään. Feromonien lisäys $\Delta\tau_i$ on verrannollinen saatuun kohdefunktion arvon parannukseen. Jos taas kohdefunktion arvo pisteessä x_i' ei ole parempi kuin määränpäässä i, kasvatetaan hakusuunnan i *ikää* a_i. Askeleen s_i pituus riippuu hakusuunnan i iästä a_i. Mitä pienempi on hakusuunnan i ikä a_i sitä pidempi on askel s_i. Askeleen s_i pituus muuttuu lineaarisesti iän a_i funktiona.

Myös jatkuvassa muurahaisyhdyskuntaoptimointissa käytetään feromo-

nijälkien haihtumista. Feromonijälkien haihtuminen toteutetaan jokaisen iteraatiokierroksen jälkeen samalla kaavalla kuin esimerkiksi muurahaisjärjestelmässä (kaava (3.3)) eli

$$\tau_i(t+1) = (1-\rho)\tau_i(t), \quad \forall i, \tag{5.5}$$

missä siis $0 < \rho \leq 1$ on feromonien haihtumisnopeus. Jos jokin ennalta määrätty lopetusehto on feromonien haihtumisen jälkeen voimassa, lopetetaan algoritmin suoritus. Muussa tapauksessa globaalit muurahaiset aloittavat seuraavan iteraatiokierroksen suorittamisen.

Jatkuvan muurahaisyhdyskuntaoptimoinnin tehokkuutta voidaan parantaa esimerkiksi niin, että lokaalit muurahaiset eivät valitsekaan käytettävää hakusuuntaa kaikista mahdollisista hakusuunnista, vaan valintavaihtoehtoina on vain P kohdefunktion arvon perusteella parasta hakusuuntaa.

5.2 Jatkuva vuorovaikuttava muurahaisyhdyskunta

Oikeissa muurahaisyhdyskunnissa jokainen muurahainen voi kommunikoida minkä tahansa muun muurahaisen kanssa. Tätä ominaisuutta käytetään hyväksi menetelmässä, jota kutsutaan *jatkuvaksi vuorovaikuttavaksi muurahaisyhdyskunnaksi* (*Continuous interacting ant colony*). Tämä menetelmä perustuu kahden erilaisen *kommunikaatiokanavan* (*communication channel*) käyttöön. Nämä kommunikaatiokanavat ovat *feromoneihin perustuva kommunikaatiokanava* (*stigmergic communication channel*) ja *suora yksilöiden välinen kommunikaatiokanava* (*direct inter-individual communication channel*).

5.2.1 Feromoneihin perustuva kommunikaatiokanava

Myös jatkuvassa vuorovaikuttavassa muurahaisyhdyskunnassa muurahaiset käyttävät feromoneja eli ne kommunikoivat feromoneihin perustuvan kommunikaatiokanavan avulla. Muurahaiset jättävät feromoneja siihen hakuavaruuden pisteeseen, jossa ne ovat. Jätetty feromonien määrä on verrannollinen muurahaisen matkallaan löytämään kohdefunktion arvon parannukseen. Feromonijälkien, jotka tässä tapauksessa ovat feromonipisteitä, määrä ei siis pysy vakiona. Jokainen feromonipiste vetää kaikkia muurahaisia puoleensa. Feromonipisteen vetovoima riippuu sen ja muurahaisen välisestä etäisyydestä sekä siinä olevasta feromonien määrästä. Muurahainen j liikkuu

kohti feromonipistepilven *vetovoimakeskusta* (*gravity center*) V_j. Eri muurahaisille vetovoimakeskus sijaitsee eri paikassa. Vetovoimakeskuksen paikka muurahaiselle j lasketaan kaavasta

$$V_j = \sum_{i=1}^{\kappa} \frac{x_i \cdot \frac{\bar{\delta}}{2} \cdot e^{-\tau_i \cdot \delta_{ij}}}{\sum_{i=1}^{\kappa} \frac{\bar{\delta}}{2} \cdot e^{-\tau_i \cdot \delta_{ij}}}, \tag{5.6}$$

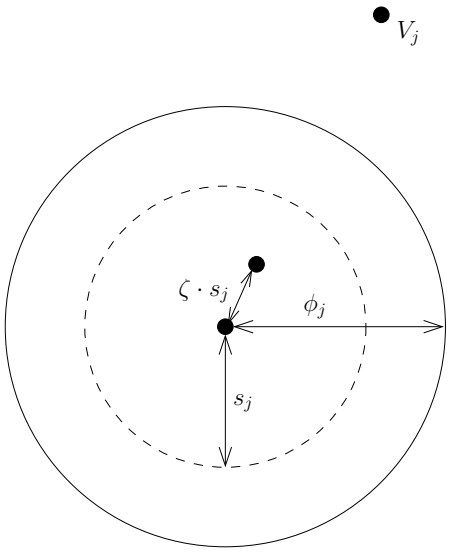
missä κ on feromonipisteiden lukumäärä, x_i feromonipisteen i paikka, $\bar{\delta}$ kahden muurahaisen keskimääräinen välimatka, τ_i feromonien määrä pisteessä x_i ja δ_{ij} muurahaisen j ja feromonipisteen i välinen etäisyys. Muurahainen j ei mene suoraan vetovoimakeskukseen V_j, vaan liikkuu tämän hetkisestä paikastaan tietyn matkan kohti vetovoimakeskusta V_j. Jokaisella muurahaisella on oma toimintasäteensä ϕ_j. Nämä toimintasäteet ϕ_j ovat normaalijakautuneet. Ennen liikkumistaan muurahainen j valitsee satunnaisen matkan s_j, joka on korkeintaan ϕ_j. Lopulta muurahainen j liikkuu kohti vetovoimakeskusta V_j matkan $\zeta \cdot s_j$, missä

$$\zeta = \begin{cases} \frac{\bar{\delta}}{\phi_j}, & \bar{\delta} \leq \phi_j, \\ 1, & \text{muuten.} \end{cases} \tag{5.7}$$

Kuvassa 5.3 on havainnollistettu sitä, miten muurahainen j liikkuu kohti vetovoimakeskusta V_j.

5.2.2 Suora yksilöiden välinen kommunikaatiokanava

Muurahainen voi ”lähettää” suoran yksilöiden välisen kommunikaatiokanavan välityksellä viestin toiselle muurahaiselle. Muurahaiset säilyttävät vastaanottamansa viestit viestipinossaan, josta ne jokaisen iteraation aikana ”lukevat” yhden satunnaisen viestin ellei niiden viestipino ole tyhjä. Viesti sisältää pisteen, jossa lähettävä muurahainen on sekä kohdefunktion arvon tässä pisteessä. Vastaanottava muurahainen vertaa kohdefunktion arvoa omassa pisteessään kohdefunktion arvoon lähettäjän pisteessä. Jos lähettäjän pisteessä kohdefunktion arvo on parempi kuin vastaanottajan pisteessä, siirtyy vastaanottaja satunnaiseen pisteeseen, joka on korkeintaan sen toimintasäteen ϕ_j etäisyydellä lähettäjän pisteestä. Jos taas vastaanottajan pisteessä kohdefunktion arvo on parempi kuin lähettäjän pisteessä, ”lähettää” vastaanottaja viestin, jossa on sen oma piste ja kohdefunktion arvo tässä pisteessä, toiselle satunnaisesti valitulle muurahaiselle. Tämän



Kuva 5.4: Esimerkki miten muurahainen j voi liikkua kohti vetovoimakeskusta V_j. Kuvassa keskellä oleva piste esittää muurahaisen j nykyistä paikkaa ja piste, joka on etäisyydellä $\zeta \cdot s_j$ muurahaisen j nykyisestä paikasta, esittää paikkaa, johon muurahainen j liikkuu feromoneihin perustuvan kommunikaatiokanavan ansiosta.

jälkeen muurahainen poistaa vanhan viestin viestipinostaan. On huomattava, että viestiä ei poisteta viestipinosta, jos lähettäjän pisteessä oli parempi kohdefunktion arvo kuin vastaanottajan pisteessä. Näin ollen viestien määrä systeemissä ei vähene vaan pysyy vakiona. Alussa lähetettävien viestien määrä v_0 onkin tärkeä parametri menetelmän tehokkuuden kannalta.

5.2.3 Lopullinen algoritmi

Jatkuvaa vuorovaikuttavaa muurahaisyhdyskunta -menetelmää käytettäessä pitää aluksi, kuten muissakin menetelmissä, alustaa parametrien arvot. Tämän jälkeen kaikki m muurahaista asetetaan normaalijakauman mukaan satunnaisesti hakuavaruuteen. Tästä syystä normaalijakauman parametrit μ ja σ ovat myös jatkuvassa vuorovaikuttavassa muurahaisyhdyskunta -menetelmässä tärkeitä parametreja. Kun muurahaiset on asetettu hakuavaruuteen, ne alkavat käyttää kommunikaatiokanavia. Muurahainen käyttää ensin feromoneihin perustuvaa kommunikaatiokanavaa ja sen jälkeen suoraa yksilöiden välistä kommunikaatiokanavaa, jonka jälkeen seuraava muurahainen tekee samoin. Kun kaikki muurahaiset ovat käyttäneet molem-

pia kommunikaatiokanavia, toteutetaan feromonien haihtuminen kaavalla (5.5). Jos feromonipisteen arvo menee tietyn parametrina annetun rajan ϵ_1 alle, niin kyseinen feromonipiste poistetaan. Feromonien haihtumisen jälkeen testataan, onko lopetusehto voimassa. Jos lopetusehto ei ole voimassa, aloitetaan uusi iteraatiokierros, jolloin muurahaiset alkavat taas käyttää kommunikaatiokanavia. Lopetusehto on voimassa, jos joko kohdefunktion arvo on parantunut edellisestä iteraatiosta alle parametrina annetun rajan ϵ_2 verran tai jos on jo suoritettu ennalta määrätty määrä iteraatioita. Tässä menetelmässä lopetusehto on siis määritelty aina samanlaiseksi, vaikka periaatteessa tässäkin menetelmässä voitaisiin tietysti käyttää muitakin lopetusehtoja.

5.3 API-menetelmä

API-menetelmä jäljittelee *Pachycondyla apicalis* muurahaisten käyttäytymistä, joiden mukaan menetelmä on saanut nimensäkin.

5.3.1 *Pachycondyla apicalis* muurahaisten käyttäytyminen

Pachycondyla apicalis muurahaiset lähtevät pesältä omille metsästyspaikoilleen. Jos muurahainen saa metsästyspaikaltaan saaliin, se palaa pesälle saaliin kanssa ja lähtee taas saalistamaan samalle metsästyspaikalle, josta oli saaliin saanut. Jos taas muurahainen ei saa saalista, se lähtee saalistamaan jollekin toiselle metsästyspaikalle. Muurahaisen löydettyä hyvän metsästyspaikan se voi johdattaa myös jonkin toisen muurahaisen tälle metsästyspaikalle. Kun läheltä pesää ei enää löydy saaliita, muurahaiset siirtävät pesänsä parempaan paikkaan.

Pachycondyla apicalis muurahaiset ovat siitä erikoisia muurahaisia, että ne eivät käytä feromoneja. Sen sijaan ne suunnistavat näköaistinsa avulla ja pystyvät muistamaan reitin muutamalle eri metsästyspaikalle maamerkkien avulla.

5.3.2 Yleinen API-menetelmä

API-menetelmää käytettäessä pitää ensin määrittää pesän paikka. Pesän sijainti valitaan satunnaisesti tasaisesta jakaumasta yli koko hakuavaruuden S. Pesää siirretään aina, kun on suoritettu T kappaletta iteraatioita.

Kun käytettäviä muurahaisia on m kappaletta, pesää siirretään siis aina $m \cdot T$ kohdefunktion arvon laskemisen jälkeen. Pesä siirretään aina pisteeseen, josta on löydetty siihen mennessä paras kohdefunktion arvo. Pesän paikan määräämisen jälkeen muurahaiset lähtevät pesältä suorittamaan optimointia. Aluksi muurahaiset luovat itselleen ν kappaletta metsästyspaikkoja. Muurahaiseen i liittyy luku $A_m(i)$, joka määrää kuinka kaukana pesästä muurahaisen i metsästyspaikat voivat sijaita. Jos $A_m(i) = 0$, ovat muurahaisen i metsästyspaikat pesällä. Jos taas $A_m(i) = 1$, voivat muurahaisen metsästyspaikat sijaita missä tahansa hakuavaruudessa S. API-menetelmässä määritellään

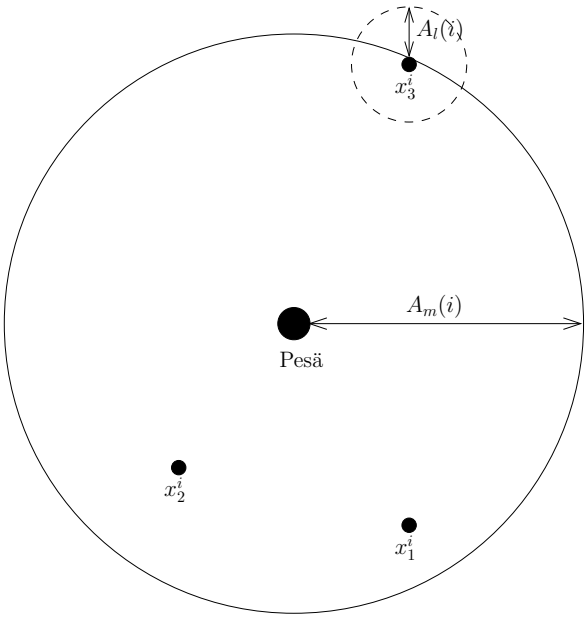
$$A_m(i) = 0{,}01^{1-\frac{i}{m}}. \tag{5.8}$$

Näin ollen $A_m(m) = 1$, joten muurahaisen m metsästyspaikat voivat sijaita missä tahansa hakuavaruudessa S. Muurahainen i valitsee itselleen ν metsästyspaikkaa satunnaisesti, kuitenkin niin, että metsästyspaikat sijaitsevat korkeintaan etäisyydellä $A_m(i)$ pesästä. Valintaa ei välttämättä tehdä tasaisesta jakaumasta, vaan valinnassa voidaan käyttää apuna jotain ennakkotietoa. Kuvassa 5.4 havainnollistetaan muurahaisen i metsästyspaikkojen luontia.

Luotuaan ν metsästyspaikkaa muurahainen i valitsee niistä satunnaisesti yhden ja liikkuu sinne. Metsästyspaikassaan muurahainen i alkaa lokaalisti hakea kohdefunktiolle parempaa arvoa. Olkoon metsästyspaikka pisteessä x_i^k. Pisteestä x_i^k muurahainen liikkuu satunnaisesti valittuun pisteeseen $x_i^{k'}$, joka kuitenkin sijaitsee korkeintaan etäisyydellä $A_l(i)$ pisteestä x_i^k. API-menetelmässä määritellään

$$A_l(i) = \frac{1}{10} A_m(i). \tag{5.9}$$

Tästä seuraa, että muurahainen i voi liikkua pisteeseen $x_i^{k'}$, jonka etäisyys pesästä on suurempi kuin $A_m(i)$ (katso kuva 5.5). Jos kohdefunktion arvo on parempi pisteessä $x_i^{k'}$ kuin pisteessä x_i^k, niin muurahaisen i lokaali haku on onnistunut. Tällöin se päivittää metsästyspaikan sijainnin pisteeseen $x_i^{k'}$ ja seuraavalla iteraatiokieroksella se palaa takaisin samaan metsästyspaikkaan. Jos taas kohdefunktion arvo on huonompi pisteessä $x_i^{k'}$ kuin pisteessä x_i^k, niin muurahaisen i lokaali haku on epäonnistunut. Tällöin muurahainen i valitsee seuraavalla iteraatiokierroksella metsästyspaikkansa satunnaisesti kaikkien omien metsäspaikkojensa joukosta. Kun muurahainen i



Kuva 5.5: Esimerkki muurahaisen i metsästyspaikoista, kun $\nu = 3$. Kuvassa $A_l(i) > \frac{1}{10}A_m(i)$, jotta kuvasta saisi paremmin selvää.

on epäonnistunut lokaalissa haussaan $M_l(i)$ kertaa samassa metsästyspaikassa, se unohtaa tämän metsästyspaikan ja luo sen tilalle uuden. Kun pesää siirretään, muurahaiset unohtavat kaikki vanhat metsästyspaikansa ja luovat tilalle ν uutta metsästyspaikkaa.

Jokaisen iteraatiokierroksen lopussa, kun kaikki m muurahaista ovat suorittaneet lokaalin hakunsa, valitaan satunnaisesti muurahaiset i ja j, joiden parhaat metsästyspaikat ovat vastaavasti pisteissä $x_i^{k^*}$ ja $x_j^{l^*}$. Jos kohdefunktion arvo on pisteessä $x_i^{k^*}$ parempi kuin pisteessä $x_j^{l^*}$, niin muurahainen j unohtaa parhaan metsästyspaikkansa ja korvaa sen muurahaisen i parhaalla metsästyspaikalla. Päinvastaisessa tilanteessa taas muurahainen i unohtaa parhaan metsästyspaikkansa ja korvaa sen muurahaisen j parhaalla metsästyspaikalla. Tämän seurauksena kahden muurahaisen parhaat metsästyspaikat asettuvat samaan pisteeseen. Jos jokin ennalta määrätty lopetusehto on tämän metsästyspaikkojen vaihdon jälkeen voimassa, lopetetaan algoritmin suoritus. Muussa tapauksessa muurahaiset aloittavat seuraavan iteraatiokierroksen suorittamisen.

5.4 Jatkuvien alueiden muurahaisyhdyskuntaoptimointi

Muurahaisyhdyskuntaoptimoinnin menetelmissä muurahaiset valitsevat rakentamansa ratkaisun komponentit satunnaisesti. *Jatkuvien alueiden muurahaisyhdyskuntaoptimoinnissa* (*ant colony optimization for continuous domain*) komponentin i valinta suoritetaan käyttäen normaalijakaumien painotettuja summia G^i. Näiden painotettujen summien tiheysfunktiot määritellään seuraavasti:

$$\sum_{l=1}^{k} \omega^l h_i^l(x) = \sum_{l=1}^{k} \omega^l \frac{1}{\sigma_i^l \sqrt{2\pi}} e^{-\frac{(x-\mu_i^l)^2}{2(\sigma_i^l)^2}}, \tag{5.10}$$

missä ω^l on normaalijakauman l paino, $(\sigma_i^l)^2$ on normaalijakauman l varianssi dimensiossa i ja μ_i^l normaalijakauman l odotusarvo dimensiossa i. Summassa käytetään siis k kappaletta normaalijakaumien tiheysfunktioita. Tässä menetelmässä k on yksi parametreista, jolle pitää päteä $k \geq n$, missä n ratkaistavan ongelman dimensio. Käytetään painotetussa summassa G^i olevasta yksittäisestä normaalijakaumasta, jonka tiheysfunktio on h_l^i, merkintää G_l^i

5.4.1 Feromonijälkien esittäminen

Jatkuvien alueiden muurahaisyhdyskuntaoptimointissa feromonijäljet esitetään normaalijakaumien painotettujen summien G^i avulla. Näin ollen jakaumissa G^i käytetyt vektorit ω=$(\omega^1, ..., \omega^k)$, σ_i=$(\sigma_i^1, ..., \sigma_i^k)$ ja μ_i=$(\mu_i^1, ..., \mu_i^k)$ ovat tärkeitä feromonijälkien esittämisen kannalta.

Jatkuvien alueiden muurahaisyhdyskuntaoptimointissa muistissa pitää säilyttää k kappaletta ratkaisuja. Käytetään nyt ratkaisuvektorista l merkintää x^l ja sen komponentista i merkintää x_i^l. Tässä menetelmässä normaalijakauman odotusarvo määritellään

$$\mu_i = (\mu_i^1, ..., \mu_i^k) = (x_i^1, ..., x_i^k). \tag{5.11}$$

Jokaiselle muistissa säilytettävälle ratkaisulle x^l annetaan järjestysnumero. Tämä järjestysnumero on sitä pienempi mitä paremman kohdefunktion arvon ratkaisu antaa. Parhaan kohdefunktion arvon antava ratkaisu saa siis järjestysnumeron 1 ja huonoimman kohdefunktion arvon antava ratkaisu saa

järjestysnumeron k. Järjestysnumeron l saaneen ratkaisun paino ω^l määritellään seuraavasti:

$$\omega^l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}}, \tag{5.12}$$

missä q on menetelmän parametri ja k on muistissa säilytettävien ratkaisujen määrä, joka siis myös on yksi menetelmän parameteista. Mitä pienempi parametri q on sitä enemmän painotetaan parhaita ratkaisuja.

Jatkuvien alueiden muurahaisyhdyskuntaoptimointissa hajonta σ_i^l määritellään seuraavasti:

$$\sigma_i^l = \xi \sum_{e=1}^{k} \frac{|x_i^e - x_i^l|}{k-1}, \tag{5.13}$$

missä $\xi > 0$ on parametri, joka tässä menetelmässä edustaa feromonien haihtumista. Mitä suurempi parametri ξ on sitä hitaammin algoritmi konvergoi. Toisin sanoen mitä pienempi parametri ξ on sitä todennäköisemmin muurahaiset valitsevat ratkaisunsa komponentit läheltä jo aikaisemmin löydettyä ratkaisua.

5.4.2 Lopullinen algoritmi

Jatkuvien alueiden muurahaisyhdyskuntaoptimointissa pitää aluksi luoda parametrin k määräämä määrä ratkaisuja, jotka säilytetään muistissa. Nämä alkuratkaisut valitaan satunnaisesti tasaisesta jakaumasta yli koko hakuavaruuden.

Alkuratkaisujen luomisen jälkeen kaikki m muurahaista rakentavat jokaisella iteraatiokierroksella ratkaisunsa n:n askeleen aikana. Askeeleen i aikana muurahainen valitsee ratkaisuunsa komponentin i. Tämän valinnan se tekee jakauman G^i mukaisesti. Jakauman G^i mukainen satunnaisluku saadaan generoitua kahdessa vaiheessa. Ensimmäisessä vaiheessa valitaan, mitä normaalijakaumaa G_l^i käytetään. Normaalijakauma G_l^i valitaan todennäköisyydellä

$$p_l = \frac{\omega^l}{\sum_{r=1}^{k} \omega^r}. \tag{5.14}$$

Toisessa vaiheessa otetaan valitun normaalijakauman G_l^i mukainen satunnaisluku. Tämä voidaan suorittaa käyttämällä satunnaislukugeneraattoria, joka generoi satunnaislukuja joko normaalijakaumasta tai tasaisesti jakaumasta, jotka sitten muunnetaan esimerkiksi Box–Muller-menetelmällä [3]

satunnaisluvuiksi normaalijakaumasta. Jokainen muurahainen tekee käytettävän normaalijakauman valinnan vain kerran iteraatiokierroksen aikana. Toisin sanoen, kun muurahainen on ensimmäisellä askeleella valinnut käytettävän normaalijakauman G_l^1 yhtälön (5.14) mukaisesti, niin se käyttää samalla iteraatiokierroksella myös normaalijakaumia $G_l^2, ..., G_l^n$. Näin ollen jokaisella iteraatiokierroksella ei tarvitse laskea koko varianssivektoria σ_i vaan vain tarvittavat komponentit σ_i^l.

Sen jälkeen kun muurahaiset ovat rakentaneet ratkaisunsa, pitää feromonijäljet päivittää. Koska tässä menetelmässä feromonijäljet esitetään normaalijakaumien painotettujen summien G^i avulla, niin jakaumat G^i pitää päivittää. Tämä taas tapahtuu päivittämällä muistissa pidettävät ratkaisut. Näin ollen feromonijälkien päivitys toteutetaan siten, että ensin kaikki muurahaisten kyseisellä iteraatiokierroksella rakentamat ratkaisut lisätään muistissa pidettävien ratkaisujen joukkoon. Nämä kaikki ratkaisut järjestetään paremmuusjärjestykseen niiden antaman kohdefunktion arvon mukaan. Lopulta muistissa pidettävien ratkaisujen joukosta poistetaan huonoimmat ratkaisut siten, että jäljelle jää k kappaletta ratkaisuja. Tämän jälkeen aloitetaan uusi iteraatiokierros, jolla muurahaiset rakentavat uusia ratkaisuja päivitettyjen jakaumien G^i avulla. Tätä jatketaan kunnes jokin ennalta määrätty lopetusehto tulee voimaan.

5.5 Muurahaisyhdyskuntasysteemin jatkuva versio

Myös *muurahaisyhdyskuntasysteemin jatkuvassa versiossa* (*continuous ant colony system*) käytetään feromonijälkien esittämiseen normaalijakaumia kuten tehtiin edellä esitetyssä jatkuvien alueiden muurahaisyhdyskuntaoptimoinnissa. Tässä edellä esitettyä menetelmää vanhemmassa menetelmässä käytetään kuitenkin dimensiossa i normaalijakaumien painotettujen summien G^i sijasta yhtä normaalijakaumaa (vertaa (5.10))

$$\tau_i(x) = e^{-\frac{(x-x_i^{min})^2}{2\sigma_i^2}}, \tag{5.15}$$

missä x_i^{min} on minimoitavalle kohdefunktiolle f tähän mennessä löydetyn pienimmän arvon antavan ratkaisun x^{min} $i : s$ komponentti ja σ_i^2 on normaalijakauman varianssi. Aluksi komponentit x_i^{min} valitaan tasaisesta jakaumasta ja varianssit σ_i^2 valitaan riittävän suuriksi, jotta todenäköisyys sille,

että muurahaiset valitsevat aluksi ratkaisuunsa komponentteja myös kauempaa ratkaisusta x^{min}, säilyy melko suurena.

Kun m muurahaista ovat rakentaneet ratkaisunsa, päivitetään feromonijäljet $\tau_i(x)$. Jos paras muurahaisten löytämä ratkaisu antaa kohdefunktiolle f pienemmän arvon kuin ratkaisu x^{min}, pitää ratkaisu x^{min} päivittää parhaaksi muurahaisten tällä iteraatiokierroksella löytämäksi ratkaisuksi. Jos taas muurahaiset eivät löydä iteraatiokierroksella parempaa ratkaisua kuin x^{min}, sitä ei päivitetä, mutta varianssit σ_i^2 päivitetään joka tapauksessa. Tässä menetelmässä varianssit lasketaan kaavasta

$$\sigma_i^2 = \frac{\sum_{j=1}^{m} \frac{1}{f_j - f_{min}} (x_i^j - x_i^{min})^2}{\sum_{j=1}^{m} \frac{1}{f_j - f_{min}}}, \tag{5.16}$$

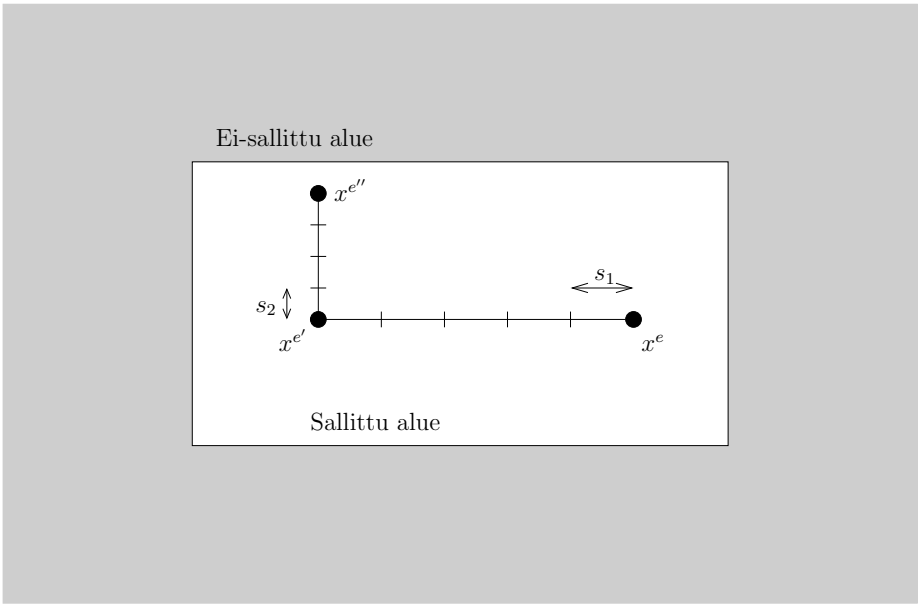
missä x_i^j on muurahaisen j kyseisellä iteraatiokierroksella rakentaman ratkaisun x^j $i : s$ komponentti, f_j on ratkaisun x^j antama arvo minimoitavalle kohdefunktiolle f ja f_{min} on ratkaisun x^{min} antama arvo minimoitavalle kohdefunktiolle f. Jos jokin ennalta määrätty lopetusehto on feromonijälkien $\tau_i(x)$ päivityksen jälkeen voimassa, lopetetaan algoritmin suoritus. Muussa tapauksessa muurahaiset aloittavat seuraavan iteraatiokierroksen suorittamisen valitsemalla ratkaisuunsa komponentit päivitettyjen normaalijakaumien $\tau_i(x)$ mukaisesti.

5.6 Muurahaisyhdyskuntaoptimoinnin suora sovellus jatkuville ongelmille

Muurahaisyhdyskuntaoptimoinnin suorassa sovelluksessa jatkuville ongelmille (*a direct application of ant colony optimization to function optimization problem in continuous domain*) käytetään feromonijälkien esittämiseen normaalijakaumia kuten tehtiin kahdessa edellä esitetyssä menetelmässäkin. Tässä menetelmässä käytetään dimensiossa i iteraatiokierroksella t yhtä normaalijakaumaa $G(\mu_i(t), \sigma_i(t)^2)$, jonka tiheysfunktio on (vertaa (5.10))

$$h_i(x,t) = \frac{1}{\sigma_i(t)\sqrt{2\pi}} e^{-\frac{(x-\mu_i(t))^2}{2\sigma_i(t)^2}}, \tag{5.17}$$

missä siis $\sigma_i(t)^2$ on normaalijakauman $G(\mu_i(t), \sigma_i(t)^2)$ varianssi ja $\mu_i(t)$ saman jakauman odotusarvo.



Kuva 5.6: Esimerkki muurahaisyhdyskuntaoptimoinnin suorassa sovelluksessa jatkuville ongelmille käytettävästä lokaalista hausta kaksiulotteisessa tapauksessa. Kuvassa lokaalia hakua käytetään iteraation parhaaseen ratkaisuun x^e. Kun lokaalia hakua käytetään tämän ratkaisun ensimmäiseen komponenttiin, löydetään paras ratkaisu pisteestä $x^{e'}$. Kun tämän jälkeen lokaalia hakua käytetään vielä ratkaisun toiseen komponenttiin, löydetään paras ratkaisu pisteestä $x^{e''}$, johon alkuperäinen ratkaisu x^e lopulta siirretään.

Algoritmin suorituksen alussa asetetaan

$$\begin{aligned} \mu_i(0) &= x_i^a + p(x_i^y - x_i^a); \\ \sigma_i(0) &= \tfrac{x_i^y - x_i^a}{2}, \end{aligned} \tag{5.18}$$

missä p on jokaiselle dimensiolle erikseen generoitava satunnaisluku tasaisesta jakaumasta väliltä $[0, 1]$. Algoritmin suorituksen edetessä variansseja $\sigma(t) = (\sigma_1(t), ..., \sigma_n(t))$ ja odotusarvoja $\mu(t) = (\mu_1(t), ..., \mu_n(t))$ päivitetään.

Kun muurahainen on generoinut itselleen ratkaisun $x^u = (x_1^u, ..., x_n^u)$ normaalijakaumien $G(\mu_i(t), \sigma_i(t)^2)$ mukaisesti, testataan ovatko kaikki ratkaisun komponentit x_i^u sallitulla alueella. Varsinaisen ratkaisun komponenteiksi x_i asetetaan

$$x_i = \begin{cases} x_i^a, & \text{jos } x_i^u < x_i^a, \\ x_i^u, & \text{jos } x_i^a \leq x_i^u \leq x_i^y, \\ x_i^y, & \text{jos } x_i^u > x_i^y. \end{cases} \tag{5.19}$$

Näin voidaan tietysti toimia myös muissa menetelmissä, jos generoidut ratkaisut eivät pysy sallitulla alueella. Kun iteraatiokierroksen t kaikki m muurahaista ovat rakentaneet ratkaisunsa, suoritetaan lokaali haku. Lokaali haku suoritetaan vain parhaalle tähän mennessä löydetylle ratkaisulle x^t ja iteraatiokierroksen parhaalle ratkaisulle x^e. Lokaali haku suoritetaan ratkaisun jokaiselle komponentille erikseen. Ensin ratkaisun komponenttia i kasvatetaan askeleen s_i verran. Tätä toistetaan kunnes askeleen aikana ratkaisu huononee tai kunnes ratkaisu ei ole enää sallittu. Tämän jälkeen suoritetaan ratkaisun komponentin i pienentäminen samalla tavalla. Askelpituus s_i määritellään seuraavasti:

$$s_i = p \cdot \sigma_i(t), \tag{5.20}$$

missä p on taas satunnaisluku tasaisesta jakaumasta väliltä $[0, 1]$. Jos lokaalilla haulla komponentin i suhteen on löydetty alkuperäistä ratkaisua parempi ratkaisu, päivitetään alkuperäistä ratkaisua. Tämän jälkeen suoritetaan lokaali haku komponentin $i + 1$ suhteen (katso kuva 5.6).

Lokaalin haun suorittamisen jälkeen suoritetaan feromonijälkien eli varianssien $\sigma(t) = (\sigma_1(t), ..., \sigma_n(t))$ ja odotusarvojen $\mu(t) = (\mu_1(t), ..., \mu_n(t))$ päivitys. Päivitykseen käytetään seuraavia kaavoja:

$$\begin{aligned} \mu(t+1) &= (1-\rho)\mu(t) + \rho(w^t x^t + w^e x^e + w^d x^d); \\ \sigma(t+1) &= (1-\rho)\sigma(t) + \rho|w^t x^t + w^e x^e + w^d x^d - \mu(t)|, \end{aligned} \tag{5.21}$$

missä $\rho \in [0, 1]$ on feromonien haihtumisnopeus sekä w^t, w^e ja w^d ovat vastaavasti parhaan tähän mennessä löydetyn ratkaisun x^t, iteraatiokierroksen parhaan ratkaisun x^e ja parhaan edellisen feromonijälkien uudelleenalustuksen jälkeen löydetyn ratkaisun x^d painot. Painot on esitetty taulukkossa 1, jossa k_u, k_1, k_2, k_3 ja k_4 ovat menetelmän parametreja. Näille parametreille pätee $k_u \leq k_1 \leq k_2 \leq k_3 \leq k_4$. Jos feromonijälkien päivittämisen jälkeen jokin ennalta määrätty lopetusehto on voimassa, lopetetaan algoritmin suoritus. Muussa tapauksessa, tarkistetaan pitääkö feromonijäljet uudelleenalustaa. Jos ei tarvitse, siirrytään seuraavalle iteraatiokierrokselle ja muurahaiset alkavat rakentaa uusia ratkaisuja päivitettyjen normaalijakaumien $G(\mu_i(t+1), \sigma_i(t+1)^2)$ mukaisesti.

Feromonijälkien uudelleenalustusta sekä painojen w^t, w^e ja w^d

	$k_u \leq kt < k_1$	$k_1 \leq kt < k_2$	$k_2 \leq kt < k_3$	$k_3 \leq kt < k_4$	$kt \geq k_4$
w^e	0	0	$\frac{1}{3}$	$\frac{2}{3}$	1
w^d	0	1	$\frac{2}{3}$	$\frac{1}{3}$	0
w^t	1	0	0	0	0

Taulukko 1: Muurahaisyhdyskuntaoptimoinnin suorassa sovelluksessa jatkuville ongelmille käytettävät ratkaisujen painot w^e, w^d ja w^t konvergenssitekijän kt funktiona.

määräämistä varten määritellään konvergenssitekijä kt:

$$kt = \frac{\sum_{i=1}^{n} \frac{2\sigma_i(t)}{x_i^y - x_i^a}}{n}. \tag{5.22}$$

Koska algoritmin suorituksen alussa asetettiin $\sigma_i(0) = \frac{x_i^y - x_i^a}{2}$, niin alussa konvergenssitekijä $kt = 1$. Kun algoritmi on lähellä lokaalia optimia (katso liite A), on kovergenssitekijä kt lähellä nollaa. Näin ollen tässä menetelmässä käytetään parametria k_u, joka määrittelee, koska feromonijälkien uudelleenalustus suoritetaan. Tämä tapahtuu, kun $kt < k_u$. Kun feromonijälkien uudelleenalustus suoritetaan iteraatiokierroksen t lopussa, niin kaavojen (5.21) asemasta käytetäänkin kaavoja

$$\begin{aligned} \mu(t+1) &= x^t + \rho \textstyle\sum_{j=0}^{n_u} w^j x^d(j); \\ \sigma(t+1) &= \frac{x_i^y - x_i^a}{2} + \rho \textstyle\sum_{j=0}^{n_u} w^j |x^d(j) - x^t|, \end{aligned} \tag{5.23}$$

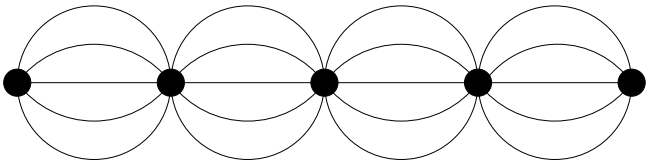
missä n_u on edeltävien feromonijälkien uudelleenalustusten määrä, w^j on ratkaisun $x^d(j)$ paino sekä $x^d(j)$ on j:nnen ja $(j+1)$:nnen feromonijälkien uudelleenalustuksen välisten iteraatiokierrosten aikana paras löydetty ratkaisu. Painot w^j määritellään seuraavasti:

$$w^j = \frac{q_f(x^d(j))}{\sum_{l=0}^{n_u} q_f(x^d(l))}, \tag{5.24}$$

missä q_f on siis jokin ratkaistavan optimointitehtävän fitnessfunktio.

5.7 Mukautuva muurahaisyhdyskunta-algoritmi

Mukautuvassa muurahaisyhdyskunta-algoritmissa (*the adaptive ant colony algorithm*) feromonijälkien haihtumisnopeus $\rho(t)$ ei ole vakio, vaan riippuu ajasta t. Aika t kuvaa tässä tapauksessa iteraatiokierrosta. Tämä



Kuva 5.7: Esimerkki mukautuvassa muurahaisyhdyskunta-algoritmissa käytettävästä graafista, jossa $n = 4$ ja $N = 5$.

menetelmä muistuttaa luvussa 3.7 esiteltyä muurahaisyhdyskuntajärjestelmää. Mukautuvassa muurahaisyhdyskunta-algoritmissa nimittäin jaetaan jokainen ratkaistavan tehtävän dimensio erillisiksi alueiksi, jonka jälkeen voidaan käyttää samanlaisia menetelmiä kuin esiteltiin luvussa 3 kauppamatkustajaongelman ratkaisemiseksi.

Ratkaistavan optimointitehtävän jokainen dimensio jaetaan N kappaleeseen yhtä pitkiä alueita, joiden pituus on

$$l_i = \frac{x_i^y - x_i^a}{N}, \quad i = 1, 2, ..., n. \tag{5.25}$$

Tämän jälkeen tehtävästä voidaan muodostaa graafi (katso kuva 5.7), jossa on $n + 1$ solmua ja jokaisesta solmusta i lähtee N kaarta solmuun $i + 1$ ja muita kaaria graafissa ei ole. Kaaret (i, j) edustavat alueita, joihin dimensio i on jaettu. Tässä tapauksessa merkintä (i, j) tarkoittaa solmusta i alkavaa j:nnettä kaarta eikä solmusta i alkavaa ja solmuun j päättyvää kaarta. Ala- ja ylärajoja (x_i^a ja x_i^y) päivitetään algoritmin suorittamisen edetessä, jolloin erotukset $x_i^y - x_i^a$ pienenevät. Tämän seurauksena myös alueiden pituudet l_i pienenevät ja algoritmin lopetusehtona käytetäänkin ehtoa

$$\max\{l_1, l_2, ..., l_n\} < \epsilon. \tag{5.26}$$

Ennen kuin muurahaiset alkavat rakentaa ratkaisujaan, alustetaan jokaisen kaaren (i, j) feromonijäljen τ_{ij} arvoksi τ_0. Muurahaisen rakentaessa ratkaisua se lähtee solmusta 1 ja päätyy solmuun $n+1$ käyden järjestyksessä graafin jokaisessa solmussa. Kun muurahainen on ratkaisua rakentaessaan solmussa i, se valitsee käytettävän kaaren (i, j) samoin kuin muurahaisyhdyskuntajärjestelmässä eli (kaava (3.12))

$$j = \begin{cases} \text{argmax}_{1 \leq s \leq N}\{\tau_{is}\}, & \text{jos } p \leq q_0; \\ J, & \text{jos } p > q_0; \end{cases} \tag{5.27}$$

missä p on tasaisesti välille [0,1] jakautunut satunnaismuuttuja, q_0 ($0 \leq q_0 \leq 1$) on menetelmän parametri ja J on satunnaismuuttuja todennäköisyysjakaumasta, jossa todennäköisyydet on määritelty seuraavasti (vertaa (5.4)):

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{s=1}^{N} \tau_{is}(t)}. \tag{5.28}$$

Aina muurahaisen liikuttua kaarta (i, j) pitkin solmusta i solmuun $i + 1$ suoritetaan lokaali feromonijälkien päivittäminen. Se suoritetaan seuraavan kaavan mukaisesti (vertaa (5.5)):

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_1, \tag{5.29}$$

missä $0 < \xi < 1$, on menetelmän parametri ja $\tau_1 = \min_{1 \leq s \leq N}\{\tau_{is}\}$.

Liikkuessaan kaarta (i, j) pitkin muurahainen valitsee ratkaisuunsa vasta alueen, jota se käyttää ratkaistavan tehtävän dimensiossa i. Vielä pitää päättää minkä arvon muuttuja x_i saa muurahaisen rakentamassa ratkaisussa. Valitaan

$$x_i = x_i^a + \frac{(2j - 1) \cdot l_i}{2}. \tag{5.30}$$

Kun kaikki m muurahaista ovat rakentaneet ratkaisunsa, lasketaan fitnessfunktion arvo kaikkilla näillä ratkaisuilla. Olkoon $q_f^e(t)$ iteraatiokierroksen t suurin fitnessfunktion arvo ja x^e ratkaisu, jolla tämä fitnessfunktion arvo saavutetaan. Globaali feromonijälkien päivittäminen suoritetaan vain, jos $q_f^e(t) \geq q_f^e(t - 1)$. Tämä päivitys toteutetaan kaavalla (vertaa (3.13))

$$\tau_{ij}(t + 1) = (1 - \rho(t))\tau_{ij}(t) + \rho(t)\Delta\tau_{ij}^e(t), \tag{5.31}$$

missä

$$\Delta\tau_{ij}^e(t) = \begin{cases} \frac{Q}{a + q_f^e(t)}, & \text{jos } (i, j) \in x^e; \\ 0, & \text{muuten;} \end{cases} \tag{5.32}$$

ja

$$\rho(t) = \max\{1 - \frac{\ln(t)}{\ln(t + b)}, \rho_{min}\}. \tag{5.33}$$

Näissä Q, a, b ja ρ_{min} ovat menetelmän parametreja. Kun parametrina annetulle feromonijälkien haihtumisnopeuden alarajalle asetetaan ehto $0 < \rho_{min} < 1$, niin myös $0 < \rho(t) < 1$.

Mukautuvassa muurahaisyhdyskunta-algoritmissa myös feromonijälkien arvoille asetetaan ala- ja ylärajat (τ_{min} ja τ_{max}) parametreina. Ennen seuraavan iteraatiokierroksen $t + 1$ alkua varmistetaan, että feromonijälkien arvot

pysyvät sallitulla välillä. Siis

$$\tau_{ij}(t+1) = \begin{cases} \tau_{min}, & \text{jos } \tau'_{ij}(t+1) < \tau_{min}; \\ \tau'_{ij}(t+1), & \text{jos } \tau'_{ij}(t+1) \in [\tau_{min}, \tau_{max}]; \\ \tau_{max}, & \text{jos } \tau'_{ij}(t+1) > \tau_{max}; \end{cases} \tag{5.34}$$

missä $\tau'_{ij}(t+1)$ on feromonijälkien päivityskaavojen avulla saatu iteraatiokierroksen $t+1$ feromonijäljen arvo kaarella (i,j).

Jos edellisen muuttujien ala- ja ylärajojen (x_i^y ja x_i^a) päivityksen jälkeen ollaan suoritettu T iteraatiokierrosta, niin ennen seuraavan iteraatiokierroksen alkua suoritetaan vielä ala- ja ylärajojen (x_i^a ja x_i^y) päivitys. Tätä varten ensin määritetään vektori $(m_1, m_2, ..., m_n)$, missä

$$m_i = \text{argmax}_{1 \leq s \leq N}\{\tau_{is}\}. \tag{5.35}$$

Tämän vektorin avulla tehdään päivitykset seuraavasti:

$$\begin{aligned} x_i^a &\leftarrow \max\{x_i^a, x_i^a + (m_i - \Delta)l_i\}, \\ x_i^y &\leftarrow \min\{x_i^y, x_i^a + (m_i + \Delta)l_i\}, \end{aligned} \tag{5.36}$$

missä Δ on menetelmän parametri. Jos nyt lopetusehto on voimassa eli $\max\{l_1, l_2, ..., l_n\} < \epsilon$, niin algoritmin suorittaminen lopetetaan ja ratkaisuksi saadaan

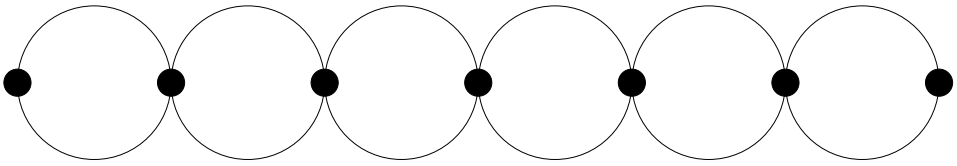
$$x_i^* = \frac{x_i^a + x_i^y}{2}, \quad i = 1, 2, ..., n. \tag{5.37}$$

Muuten aloitetaan taas uusi iteraatiokierros ja jatketaan kunnes lopetusehto on voimassa.

5.8 Binäärinen muurahaissysteemi

Nimensä mukaisesti *binäärisellä muurahaissysteemillä* (*binary ant system*) voidaan ratkaista optimointitehtäviä, joissa muuttujat ovat binäärisiä. Jotta tällä menetelmällä voidaan ratkaista optimointitehtäviä, joissa muuttujat ovat reaalisia, pitää reaaliset muuttujat muuntaa binäärisiksi. Näin ollen ratkaistaessa tällä menetelmällä reaalista ongelmaa pitää valita, kuinka monta binäärimuuttujaa on yhden reaalimuuttujan korvaavassa binäärijonossa. Jos tämän binäärijonon pituus on d ja jos alkuperäisiä reaalimuuttujia on v kappaletta, niin yhteensä binäärimuuttujia on $n = v \cdot d$ kappaletta.

Kuten edellä esitetyssä mukautuvassa muurahaisyhdyskunta-algoritmissa myös binäärisessä muurahaissysteemissä muodostetaan



Kuva 5.8: Esimerkki binäärisessä muurahaissysteemissä käytettävästä graafista, jossa $n = 6$ (eli esimerkiksi $d = 3$ ja $v = 2$). Graafi on samanlainen kuin edellä esitellyssä mukautuvassa muurahaisyhdyskunta-algoritmissa olisi, jos $n = 6$ ja $N = 2$

tehtävän ratkaisemiseksi graafi (katso kuva 5.8). Muodostettavassa graafissa on $n + 1$ solmua ja jokaisesta solmusta i lähtee 2 kaarta solmuun $i + 1$. Solmusta i lähteville kaarille käytetään merkintöjä $(i, 0)$ ja $(i, 1)$. Binäärisessä muurahaissysteemissä jokaisen kaaren (i, j) feromonijäljen τ_{ij} arvoksi alustetaan 0,5. Muurahaisen rakentaessa ratkaisua se lähtee solmusta 1 ja päätyy solmuun $n+1$ käyden järjestyksessä graafin jokaisessa solmussa. Kun muurahainen on ratkaisua rakentaessaan solmussa i, se valitsee käytettävän kaaren (i, j) todennäköisyydellä (vertaa (5.4))

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{s=0}^{1} \tau_{is}(t)}. \tag{5.38}$$

Myöhemmin todistetaan, että tässä menetelmässä todennäköisyydet p_{ij} ovat samat kuin feromonijälkien arvot τ_{ij}. Kun muurahainen valitsee kaaren (i, j), se tarkoittaa, että muurahaisen rakentamassa ratkaisussa binäärinen muuttuja i saa arvon j, missä siis j voi saada joko arvon 0 tai 1. Alkuperäisen reaalisen muuttujan k ovat nyt siis korvanneet binääriset muuttujat $(k - 1)d + 1, ..., kd$.

Kun kaikki m muurahaista ovat rakentaneet ratkaisunsa, päivitetään feromonijälkien arvot. Tämä päivitys tapahtuu seuraavan kaavan mukaisesti:

$$\tau_{ij}(t + 1) = (1 - \rho)\tau_{ij}(t) + \rho \sum_{x \in x^h, (i,j) \in x} w^x, \tag{5.39}$$

missä ρ on feromonien haihtumisnopeus, x^h on tehostettavien ratkaisujen joukko ja w^x on ratkaisun $x \in x^h$ paino. Binäärisessä muurahaissysteemissä parametrin ρ arvoksi alustetaan $\rho_0 \in (0, 1)$, mutta jokaisen feromonijälkien uudelleenalustuksen jälkeen asetetaan $\rho \leftarrow 0{,}9\rho$. Tehostettavien ratkaisujen

	$kt < k_1$	$k_1 \leq kt < k_2$	$k_2 \leq kt < k_3$	$k_3 \leq kt < k_4$	$k_4 \leq kt < k_5$
w^e	1	$\frac{2}{3}$	$\frac{1}{3}$	0	0
w^d	0	$\frac{1}{3}$	$\frac{2}{3}$	1	0
w^t	0	0	0	0	1

Taulukko 2: Binäärisessä muurahaissyystemissä käytettävät ratkaisujen painot w^e, w^d ja w^t konvergenssitekijän kt funktiona.

joukkoon x^h kuuluu paras tähän mennessä löydetty ratkaisu x^t, iteraatiokierroksen paras ratkaisu x^e ja paras edellisen feromonijälkien uudelleenalustuksen jälkeen löydetty ratkaisu x^d. Painot w^x toteuttavat seuraavat ehdot: $0 \leq w^x \leq 1$ ja $\sum_{x \in x^h} w^x = 1$. Tässä menetelmässä käytettävät painojen w^x arvot ovat taulukossa 2.

Feromonijälkien uudelleenalustukseen ja painojen w^x määritykseen liittyy *konvergenssitekijä* kt, joka määritellään seuraavasti:

$$kt = \frac{\sum_{i=1}^{n} |\tau_{i0} - \tau_{i1}|}{n}. \tag{5.40}$$

Koska kaikkien feromonijälkien τ_{ij} arvoksi alustetaan 0,5, niin konvergenssitekijän arvo algoritmin alussa on 0. Painojen w^x määrittämiseksi menetelmässä tarvitaan parametrit k_1, k_2, k_3, k_4 ja k_5. Näille parametreille on voimassa $0 \leq k_1 \leq k_2 \leq k_3 \leq k_4 \leq k_5 \leq 1$. Kun $kt > k_5$, menetelmässä suoritetaan feromonijälkien uudelleenalustus. Feromonijälkien uudelleenalustus suoritetaan seuraavasti:

$$\tau_{ij} = \begin{cases} \tau_H, & \text{jos } (i,j) \in x^t; \\ \tau_L, & \text{muuten;} \end{cases} \tag{5.41}$$

missä τ_H ja τ_L ovat menetelmän parametreja, joille pätee $0 < \tau_L < \tau_H < 1$ ja $\tau_L + \tau_H = 1$.

Seuraavaksi todistetaan, että binäärisessä muurahaissysteemissä käytettävät todennäköisyydet p_{ij} ovat itse asiassa samat kuin feromonijälkien arvot τ_{ij}.

Lause 5.1. *Binäärisessä muuraissysteemissä pätee*

$$p_{ij}(t) = \tau_{ij}(t); \quad i = 1, ..., n; j \in \{0, 1\}. \tag{5.42}$$

Todistus: Todistetaan ensin, että feromonijäljet $\tau_{ij}(t)$ voivat toimia todennäköisyyksinä eli $0 \leq \tau_{ij}(t) \leq 1$. Feromonijälkien päivityskaavan (5.39)

mukaan saadaan

$$\tau_{ij}(t) \geq (1-\rho)\tau_{ij}(t-1) \geq (1-\rho)^t \tau_{ij}(0) = 0{,}5 \cdot (1-\rho)^t.$$

Näin ollen $\tau_{ij}(t) > 0$, kun $t < \infty$. Toisaalta kun muistetaan, että $\sum_{x \in x^h} w^x = 1$, niin feromonijälkien päivityskaavan (5.39) mukaan pätee myös

$$\begin{aligned} \tau_{ij}(t) &\leq (1-\rho)\tau_{ij}(t-1) + \rho \leq (1-\rho)^t \tau_{ij}(0) + \textstyle\sum_{i=1}^{t} (1-\rho)^{i-1}\rho \\ &= (1-\rho)^t \tau_{ij}(0) + 1 - (1-\rho)^t = (1-\rho)^t(\tau_{ij}(0) - 1) + 1 \\ &= 1 - 0{,}5 \cdot (1-\rho)^t. \end{aligned}$$

Näin ollen $\tau_{ij}(t) < 1$, kun $t < \infty$. Saatiin siis todistettua, että $0 < \tau_{ij}(t) < 1$, kun $t < \infty$. Todistetaan vielä induktiolla, että aina pätee $\tau_{i0}(t) + \tau_{i1}(t) = 1$. Induktion lähtökohta on selvä: $\tau_{i0} + \tau_{i1} = 0{,}5 + 0{,}5 = 1$. Nyt induktio-oletuksena on, että $\tau_{i0}(t-1) + \tau_{i1}(t-1) = 1$. Koska aina jompi kumpi kaarista $(i, 0)$ tai $(i, 1)$ kuuluu ratkaisuun $x \in x^h$, niin feromonijälkien päivityskaavan (5.39) avulla saadaan

$$\begin{aligned} \tau_{i0}(t) + \tau_{i1}(t) &= (1-\rho)\tau_{i0}(t-1) + \rho \textstyle\sum_{x \in x^h, (i,0) \in x} w^x + (1-\rho)\tau_{i1}(t-1) \\ &\quad + \rho \textstyle\sum_{x \in x^h, (i,1) \in x} w^x \\ &= (1-\rho)(\tau_{i0}(t-1) + \tau_{i1}(t-1)) + \rho \textstyle\sum_{x \in x^h} w^x \\ &= 1 - \rho + \rho = 1. \end{aligned}$$

Nyt on siis todistettu, että aina $\tau_{i0}(t) + \tau_{i1}(t) = 1$. Näin ollen

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{s=0}^{1} \tau_{is}(t)} = \frac{\tau_{ij}(t)}{1} = \tau_{ij}(t). \quad \square$$

Binäärisessä muurahaissysteemissä m muurahaista ratkaisevat optimointitehtävää noudattaen edellä esiteltyjä sääntöjä kunnes jokin etukäteen määrätty lopetusehto tulee voimaan.

5.9 Pseudorinnakkainen muurahaisyhdyskuntaoptimointi

Pseudorinnakkaisessa muurahaisyhdyskuntaoptimoinnissa (*pseudo parallel ant colony optimization*) käytetään rinnakkain kahta muurahaispopulaatiota, jotka tietyin väliajoin kommunikoivat keskenään. Tässä menetelmässä muurahaiset valitsevat ratkaisuunsa komponentit algoritmin alussa määrättyjen arvojen joukosta. Kun muurahaiset ovat rakentaneet ratkaisunsa, yritetään näiden ratkaisujen tietystä ympäristöstä löytää parempia ratkaisuja.

Algoritmin alussa molempien populaatioiden muurahaiset rakentavat populaation käsittelemälle tehtävälle ratkaisun, jonka ne valitsevat tasaisesta jakaumasta yli koko hakuavaruuden. Molemmissa populaatioissa on m muurahaista, joten ratkaisuja rakennetaan yhteensä siis $2m$ kappaletta. Kun nämä alkuratkaisut on luotu, muodostetaan molemmille populaatioille oma graafinsa (katso kuva 5.9), jossa populaation muurahaiset tämän jälkeen liikkuvat. Graafin solmuksi j^i asetetaan kyseisen populaation muurahaisen i rakentaman ratkaisun komponentti j. Jokaiselle solmulle alustetaan feromonijäljen arvoksi τ_0. Muurahaiset siis valitsevat seuraavalla iteraatiokierroksella ratkaisunsa komponentiksi i jonkin arvon, jonka jokin muurahainen on valinnut myös edellisellä iteraatiokierroksella ratkaisunsa komponenteksi i.

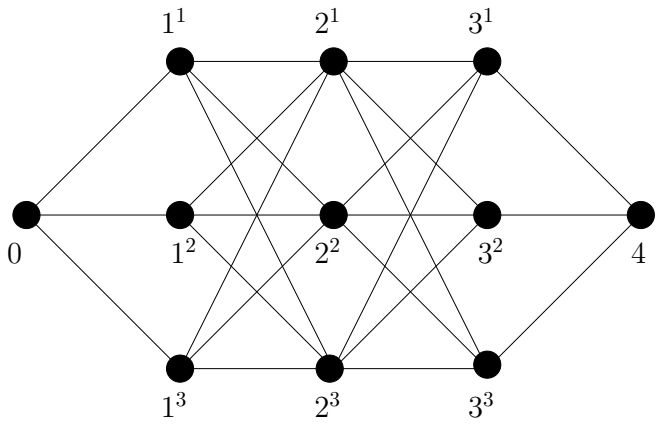
Kun molemmille populaatiolle on luotu graafit, lasketaan kunkin alkuratkaisun tuottama kohdefunktion arvo. Tämän perusteella ratkaisut järjestetään paremmuusjärjestykseen ja annetaan ratkaisuille järjestysnumerot, jolloin populaation paras ratkaisu saa järjestysnumeron 1 ja populaation huonoin ratkaisu vastaavasti järjestysnumeron m. Järjestysnumeroiden antamisen jälkeen päivitetään feromonijälkiä. Tämä tapahtuu hieman samalla tavalla kuin järjestykseen perustuvassa muurahaisjärjestelmässä (luku 2.5), eli

$$\tau_j^i(1) = \tau_0 + \alpha \cdot (w - r) \cdot \tau_0, \quad \text{jos } r < w, \tag{5.43}$$

missä $\tau_j^i(1)$ on muurahaisen i valitsemaa komponenttia j vastaavan solmun feromonijäljen arvo iteraatiokierroksella 1, $0 < \alpha < 1$ on menetelmän parametri, w on menetelmän parametri, joka määrittää feromonijälkien päivitykseen osallistuvien muurahaisten määrän ja r on muurahaisen rakentaman ratkaisun järjestysluku. Tähän feromonijälkien päivittämiseen osallistuu siis vain $w - 1$ parasta ratkaisua rakentanutta muurahaista.

Ennen kuin muurahaiset alkavat rakentaa uusia ratkaisuja pitää menetelmän parametri $0 \leq q_0 \leq 1$ olla määrätty. Aina kun muurahainen aloittaa ratkaisunsa rakentamisen, sille generoidaan satunnaisluku p tasaisesta jakaumasta väliltä $[0, 1]$. Jos $p > q_0$, muurahainen rakentaa saman ratkaisun kuin se rakensi edellisellä iteraatiokierroksella. Jos taas $p \leq q_0$, muurahainen liikkuu graafissa ja valitsee solmun j^i todennäköisyydellä $p_j^i(t)$. Tämä todennäköisyys määritellään seuraavasti (vertaa (5.4)):

$$p_j^i(t) = \frac{\tau_j^i(t)}{\sum_{l=1}^{m} \tau_j^l(t)}. \tag{5.44}$$



Kuva 5.9: Esimerkki pseudorinnakkaisessa muurahaisyhdyskuntaoptimoinnissa käytettävästä graafista, jossa $m = 3$ ja $n = 3$. Graafissa on ylimääräiset aloitus- ja lopetussolmu (solmut 0 ja 4), jotka voitaisiin myös jättää pois graafista.

Parametri q_0 siis määrittelee todennäköisyyden sille, että muurahainen rakentaa uuden ratkaisun.

Sen jälkeen kun populaation kaikki muurahaiset ovat rakentaneet ratkaisunsa, poistetaan h huonointa ratkaisua. Samalla populaation graafista poistetaan näitä huonoimpia ratkaisuja vastaavat solmut. Näiden ratkaisujen tilalle luodaan h uutta ratkaisua, jotka valitaan tasaisesta jakaumasta yli koko hakuavaruuden. Myös populaation graafiin luodaan näitä uusia ratkaisuja vastaavat solmut, joille feromonijälkien arvoiksi asetetaan τ_0. Kun uudet ratkaisut on luotu, pitää jäljelle jääneet ja uudet ratkaisut järjestää paremmusjärjestykseen ja antaa niille järjestysnumerot. Tämän jälkeen suoritetaan feromonijälkien päivitys seuraavan kaavan mukaisesti:

$$\tau_j^i(t+1) = \begin{cases} \rho \cdot \tau_j^i(t) + \alpha \cdot (w - r) \cdot \tau_0, & \text{jos } r < w; \\ \rho \cdot \tau_j^i(t), & \text{muuten,} \end{cases} \tag{5.45}$$

missä ρ on feromonijälkien haihtumisnopeus sekä α, w ja r ovat kuten kaavassa (5.43).

Feromonijälkien päivittämisen jälkeen tässä menetelmässä suoritetaan satunnaishaku. Tämän suorittamiseksi tarvitaan parametrina annettu vektori R, joka sisältää tehtävän jokaiselle dimensiolle oman hakusäteen (dimension j hakusäde on r_j). Satunnaishaussa poimitaan ratkaisun R-säteisestä ympäristöstä satunnaisesti jokin ratkaisu. Kun satunnaishakua käytetään

ratkaisuun $x = (x_1, ..., x_j, ..., x_n)$, saadaan satunnaishaulla ratkaisu, jonka komponentti j kuuluu välille $(x_j - r_j, x_j + r_j), j = 1, ..., n$. Pseudorinnakkaisessa muurahaisyhdyskuntaoptimointissa satunnaishakua käytetään jokaisella iteraatiokierroksella vain tämän kierroksen parhaaseen ratkaisuun. Satunnaishakua käytetään sh kertaa parhaaseen ratkaisuun, jonka jälkeen näitä satunnaishaulla saatuja ratkaisuja verrataan alkuperäiseen parhaaseen ratkaisuun. Jos paras satunnaishaulla saatu ratkaisu antaa paremman kohdefunktion arvon kuin alkuperäinen paras ratkaisu, korvataan graafissa alkuperäisen parhaan ratkaisun solmut satunnaishaulla saatua parasta ratkaisua vastaavilla solmuilla, mutta pidetään feromonijäljet ennallaan sekä suurennetaan hakusäteitä r_j. Jos taas paras satunnaishaulla saatu ratkaisu antaa huonomman kohdefunktion arvon kuin alkuperäinen paras ratkaisu, pienennetään vain hakusäteitä r_j. Tämä hakusäteiden muuttaminen parantaa menetelmän tarkkuutta. Kun satunnaishaku on suoritettu, siirrytään seuraavalle iteraatiokierrokselle ja muurahaiset alkavat rakentaa uusia ratkaisuja, ellei jokin ennalta määrätty lopetusehto ole voimassa.

5.9.1 Muurahaispopulaatioiden kommunikointi

Pseudorinnakkaisessa muurahaisyhdyskuntaoptimointissa optimointiongelma jaetaan kahteen osaan. Tämä jakaminen tapahtuu siten, että tehtävän muuttujat jaetaan kahteen suunnilleen yhtä suuren ryhmään. Ensimmäiselle muurahaispopulaatiolle ensimmäisen ryhmän muuttujat muodostavat *viritettävän vektorin* ja toisen ryhmän muuttujat muodostavat *muuntumattoman vektorin*. Toiselle muurahaispopulaatiolle taas ensimmäisen ryhmän muuttujat muodostavat muuntumattoman vektorin ja toisen ryhmän muuttujat muodostavat viritettävän vektorin. Muurahaispopulaatio optimoi omaa viritettävää vektoriaan samalla, kun muuntumaton vektori nimensä mukaisesti pysyy vakiona.

Tietyin väliajoin muurahaispopulaatiot kommunikoivat. Tämän seurauksena populaatioiden muuntumattomat vektorit voivat vaihtua toisen populaation parhaaksi viritettäväksi vektoriksi. Ennen tätä vaihtoa testataan populaation löytämällä parhaalla ratkaisulla, kannattaako vaihto. Jos populaation löytämä paras ratkaisu vielä paranee, kun sen muuntumaton vektori vaihdetaan toisen populaation parhaan ratkaisun viritettavään vektoriin, niin vaihto tehdään. Jos taas ratkaisun paremista ei tapahdu, vaihtoa ei tehdä.

Muurahaispopulaatioiden kommunikointi tapahtuu aina V iteraatiokierroksen välein. Yleisesti V voi olla vakio tai se voi muuttua algoritmin suorituksen edetessä. Jos V on vakio, tulee algoritmista yksinkertaisempi, mutta sopivan vakioarvon löytäminen saattaa olla vaikeaa. Pseudorinnakkaisessa muurahaisyhdyskuntaoptimointissa käytetäänkin kommunikaatiovälille V algoritmin edetessä muuttuvaa arvoa, joka ei välttämättä ole sama eri muurahaispopulaatioilla. Kommunikaatioväliä V päivitetään aina muurahaispopulaatioiden kommunikoidessa. Jos edellisen populaatioiden välisen kommunikoinnin jälkeen on löydetty uusi paras ratkaisu, kommunikaatioväliä V kasvatetaan:

$$V \leftarrow V + \left\lceil \frac{\theta_V}{t} \right\rceil, \tag{5.46}$$

missä $\theta_V > 0$ on menetelmän parametri, joka määrittää rajan kommunikaatiovälin V muutokselle ja t on iteraatiokierros. Jos edellisen populaatioiden välisen kommunikoinnin jälkeen paras löydetty ratkaisu ei ole muuttunut, mutta edellisestä ratkaisun parannuksesta on kulunut korkeintaan θ_T iteraatiokierrosta, niin kommunikaatioväli V pysyy ennallaan. Tässä käytettävä θ_T on menetelmän parametri. Nyt siis kommunikaatiovälin päivitys toteutetaan kaavalla

$$V \leftarrow V. \tag{5.47}$$

Jos taas edellisen populaatioiden välisen kommunikoinnin jälkeen paras löydetty ratkaisu ei ole muuttunut, mutta edellisestä löydetyn ratkaisun parannuksesta on kulunut enemmän kuin θ_T iteraatiokierrosta, niin kommunikaatioväliä V pienennetään:

$$V \leftarrow V - \lceil \gamma \cdot t \cdot \theta_V \rceil, \tag{5.48}$$

missä $\gamma > 0$ on menetelmän parametri sekä θ_V ja t kuten kaavassa (5.46).

5.10 Evolutiivinen muurahaisyhdyskunta-algoritmi

Evolutiivisessa muurahaisyhdyskunta-algoritmissa käytetään risteytys- ja mutaatio-operaatioita kuten muissakin evolutiivisissa algoritmeissa. Näitä operaatioita käytetään parantamaan muurahaisten rakentamia ratkaisuja.

Myös tässä menetelmässä pitää tehtävälle muodostaa aluksi graafi. Tätä varten m muurahaista rakentavat ensin ratkaisunsa, jotka ne valitsevat tasaisesta jakaumasta yli koko hakuavaruuden. Tämän jälkeen muodostetaan

graafi, jossa on $n+1$ solmua ja jokaisesta solmusta i lähtee m kappaletta kaaria solmuun $i+1$. Graafin solmut vastaavat tehtävän dimensioita ja kaaret muurahaisten rakentamien ratkaisujen komponentteja. Näin ollen jokaista muurahaisen rakentamaa ratkaisua vastaa graafissa tietty polku ensimmäisestä solmusta viimeiseen. Jokaiselle ratkaisulle lasketaan fitnessfunktion arvo ja graafissa ratkaisua vastaavan polun kaikille kaarille (i,j) alustetaan feromonijäljen $\tau_{ij}(1)$ arvoksi ratkaisun antama fitnessfunktion arvo. Tälläkin kertaa merkintä (i,j) tarkoittaa solmusta i alkavaa j:nnettä kaarta eikä solmusta i alkavaa ja solmuun j päättyvää kaarta.

Kun graafi on saatu muodostettua ja feromonijälkien arvot alustettua, aloittavat muurahaiset ensimmäisen iteraatiokierroksen. Ne lähtevät solmusta 1 ja kulkevat solmuun $n+1$. Ollessaan solmussa i muurahaiset valitsevat kaaren (i,j) todennäköisyydellä $p_{ij}(t)$, joka määritellään seuraavasti (vertaa (5.4)):

$$p_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{l=1}^{m} \tau_{il}(t)}. \tag{5.49}$$

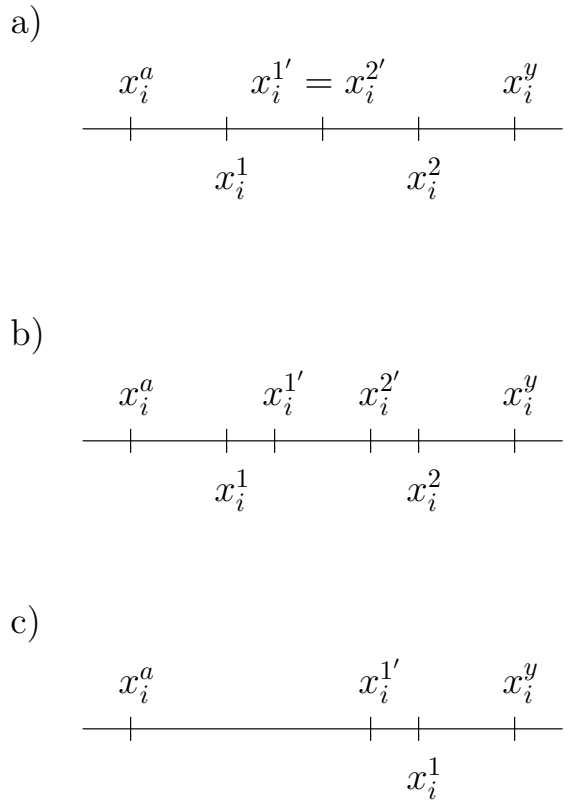
Jos muurahainen käyttää matkallaan solmusta 1 solmuun $n+1$ kaarta (i,j), tarkoittaa se, että muurahainen valitsee ratkaisunsa komponentiksi i saman arvon kuin, mikä oli edellisen iteraatiokierroksen ratkaisun j komponentti i.

Kaikkien muurahaisten rakennettua ratkaisunsa suoritetaan risteytys- ja mutaatio-operaatiot. Jokaiselle muurahaisten rakentamalle ratkaisulle yritetään suorittaa joko risteytys- tai mutaatio-operaatio. Se kumpaa operaatiota kunkin ratkaisun kullekin komponentille yritetään suorittaa valitaan satunnaisesti. Suoritetaanko operaatio lopulta riippuu varsinaisesta risteytystodennäköisyydestä p_r tai varsinaisesta mutaatiotodennäköisyydestä p_m. Itse risteytys- ja mutaatio-operaatiot esitellään myöhemmin.

Kun muurahaisten rakentamille ratkaisuille on suoritettu risteytys- ja mutaatio-operaatiot, lasketaan näiden uusien ratkaisujen antamat fitnessfunktion arvot. Käytetään ratkaisusta k merkintää x^k ja olkoon sen fitnessfunktion arvo q_f^k. Feromonijälkien $\tau_{ij}(t)$ päivityksessä tarvitaan näitä fitnessfunktioiden arvoja q_f^k. Päivitys suoritetaan kaikille kaarille (i,j) seuraavan kaavan mukaisesti:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + q_f^k, \quad \text{jos kaari } (i,j) \in x^k,\ k = 1, ..., m. \tag{5.50}$$

Aina suoritetun risteytysoperaation jälkeen pitää graafiin lisätä kaksi uutta kaarta ja suoritetun mutaatio-operaation jälkeen yksi uusi kaari. Käytet-



Kuva 5.10: Esimerkki evolutiivisessa muurahaisyhdyskunta-algoritmissa käytettävästä risteytys-operaatiosta, jossa a) $c_1 = c_2 = 0{,}5$, b) $c_1 = 0{,}75$ ja $c_2 = 0{,}25$ sekä esimerkki mutaatio-operaatiosta, jossa c) $x_i^a = 1$, $x_i^y = 5$, $x_i^1 = 4$, p = -0,5, λ = 0,01, $t = 100$ ja $q_f^1 = \ln 4$, joista näin ollen saadaan $r_i = d_i = 2$ ja $x_i^{1'} = 3{,}5$.

täessä päivityskaavaa (5.50) näihin uusiin kaariin on niillä $\tau_{ij}(t) = 0$. Uusien kaarien lisäyksen jälkeen kaikista solmuista, paitsi viimeisestä, lähtee vähintään m kaarta ja korkeintaan $2m$ kaarta. Ennen seuraavan iteraatiokierroksen alkua poistetaan kuitenkin ylimääräiset kaaret niin, että jokaisesta solmusta, paitsi viimeisestä, lähtee tasan m kaarta. Poistettaviksi joutuvat kaaret (i, j), joiden feromonijälkien $\tau_{ij}(t + 1)$ arvot on pienimmät. Tämän kaarien poiston jälkeen muurahaiset aloittavat rakentamaan seuraavan iteraatiokierroksen ratkaisujaan, ellei jokin ennalta määrätty lopetusehto ole voimassa.

5.10.1 Risteytys- ja mutaatio-operaatiot

Risteytysoperaation suorittamiseen tarvitaan kahden eri ratkaisun sama komponentti. Olkoon nämä komponentit x_i^1 ja x_i^2. Olkoon vastaavasti niillä ratkaisuilla, joihin komponentit x_i^1 ja x_i^2 kuuluvat, fitnessfunktion arvot q_f^1 ja q_f^2 sekä olkoon q_f^{max} fitnessfunktion arvo tämän hetkisellä parhaalla ratkaisulla. Varsinaisen risteytystodennäköisyyden p_r laskemiseksi tarvitaan risteytystodennäköisyys $0 \leq p_1 \leq 1$, joka on menetelmän parametri. Nyt p_r voidaan laskea kaavasta

$$p_r = p_1 \cdot \left(1 - \frac{q_f^1 + q_f^2}{2q_f^{max}}\right). \tag{5.51}$$

Näin ollen varsinainen risteytystodennäköisyys on sitä suurempi mitä huonompiin ratkaisuihin komponentit x_i^1 ja x_i^2 kuuluvat. Varsinaisen risteytystodennäköisyyden p_r laskemisen jälkeen pitää generoida satunnaisluku p tasaisesta jakaumasta väliltä $[0, 1]$. Risteytysoperaatio suoritetaan vain, jos $p < p_r$. Ennen itse risteytysoperaatiota pitää vielä generoida satunnaisluku c_1 tasaisesta jakaumasta väliltä $[-b, b]$, missä

$$b = 2 \cdot \left(1 - \frac{q_f^1 + q_f^2}{2q_f^{max}}\right). \tag{5.52}$$

Tämän jälkeen tarvitaan vielä luku c_2, joka saadaan kaavasta

$$c_1 + c_2 = 1. \tag{5.53}$$

Olkoon risteytysoperaatiosta saatavat uudet komponentit $x_i^{1'}$ ja $x_i^{2'}$. Nämä uudet komponentit saadaan seuraavista kaavoista:

$$\begin{aligned} x_i^{1'} &= c_1 x_i^1 + c_2 x_i^2, \\ x_i^{2'} &= c_2 x_i^1 + c_1 x_i^2. \end{aligned} \tag{5.54}$$

Tästä nähdään, että mitä huonompiin ratkaisuihin komponentit x_i^1 ja x_i^2 kuuluvat sitä suurempi hajonta uusilla komponenteilla $x_i^{1'}$ ja $x_i^{2'}$ on. Kuvissa 4.9 a) ja 4.9 b) on kaksi esimerkkiä risteytysoperaatiosta.

Mutaatio-operaation suorittamiseen tarvitaan vain yhden ratkaisun jokin komponentti. Olkoon tämä komponentti x_i^1. Käytetään muuten samoja merkintöjä kuin risteytysoperaation yhteydessä. Varsinaisen mutaatiotodennäköisyyden p_m laskemiseksi tarvitaan mutaatiotodennäköisyys $0 \leq p_2 \leq 1$,

joka on menetelmän parametri. Nyt p_m voidaan laskea kaavasta

$$p_m = p_2 \cdot \left(1 - \frac{q_f^1}{q_f^{max}}\right). \tag{5.55}$$

Näin ollen varsinainen mutaatiotodennäköisyys on sitä suurempi mitä huonompaan ratkaisuun komponentti x_i^1 kuuluu. Varsinaisen mutaatiotodennäköisyyden p_m laskemisen jälkeen pitää generoida satunnaisluku p tasaisesta jakaumasta väliltä $[0, 1]$. Mutaatio-operaatio suoritetaan vain, jos $p < p_m$. Vielä ennen mutaatio-operaation suorittamista pitää tehdä seuraavat määritelmät:

$$\begin{aligned} r_i &= \tfrac{x_i^y - x_i^a}{2}, \\ d_i &= |x_i^1 - r_i|. \end{aligned} \tag{5.56}$$

Nyt mutaatio-operaatiosta saatava uusi komponentti $x_i^{1'}$ voidaan laskea kaavasta

$$x_i^{1'} = x_i^1 + sign(x_i^1 - x_i^a - r_i) \cdot pe^{-\lambda t q_f^1} \cdot (d_i + r_i), \tag{5.57}$$

missä p on satunnaisluku tasaisesta jakaumasta väliltä $[-1, 1]$, 0,005 $\leq \lambda \leq$ 0,01 on menetelmän parametri ja t on iteraatiokierros. Kuvassa 5.9 c) on esimerkki mutaatio-operaatiosta.

5.11 Paranneltu feromonien kerääntymisjärjestelmä

Parannellussa feromonien kerääntymisjärjestelmässä (*enhanced aggregation pheromone system*) jokainen muurahainen muodostaa oman yksikkönsä. Jokaisella iteraatiokieroksella muurahainen rakentaa ratkaisunsa ja vertaa tätä yksikkönsä parhaaseen ratkaisuun eli parhaaseen ratkaisuun, jonka se itse on luonut aiemmilla iteraatiokierroksilla. Tarpeen mukaan tätä yksikön parasta ratkaisua sitten päivitetään.

Parannellussa feromonien kerääntymisjärjestelmässä ei käytetä graafia vaan feromonijäljet $\tau(t, x)$ ovat jakaantuneet koko hakuavaruuteen S. Algoritmin suorituksen alussa asetetaan $\tau(0, x) = c$, missä c on menetelmän parametri. Feromonijäljet ovat siis aluksi jakaantuneet tasaisesti koko hakuavaruuteen S. Aluksi kaikki m muurahaista rakentavat kaksi ratkaisua feromonijälkien $\tau(0, x) = c$ perusteella, tämän jälkeen jokaisella iteraatiokierroksella muurahaiset rakentavat vain yhden ratkaisun feromonijälkien

$\tau(t,x)$ perusteella. Todennäköisyys sille, että muurahainen iteraatiokierroksella t valitsee ratkaisun x on

$$p(t,x) = \frac{\tau(t,x)}{\int_S \tau(t,x)dx}. \tag{5.58}$$

Kun kaikki muurahaiset ovat algoritmin suorituksen alussa rakentaneet kaksi ratkaisuaan feromonijälkien $\tau(0,x) = c$ perusteella, näitä kahta ratkaisua verrataan toisiinsa ja jokaisen yksikön parempi ratkaisu säilytetään. Käytetään yksikön i paremmasta ratkaisusta merkintää $x^{i,0^*}$. Iteraatiokierroksella $t > 0$ yksikön i muurahainen luo vain yhden ratkaisun $x^{i,t}$. Tähän saatuun ratkaisuun tehdään todennäköisyydellä p_m jonkinlainen mutaatio, esimerkiksi evolutiivisen muurahaisyhdyskunta-algoritmin yhteydessä esitetty mutaatio (kaava (5.57)). Tätä ratkaisua verrataan sitten ratkaisuun $x^{i,t-1^*}$, parempi näistä ratkaisuista säilytetään seuraavalle iteraatiokierrokselle ja siitä käytetään merkintää x^{i,t^*}.

Iteraatiokierroksen t lopussa feromonijälkien $\tau(t,x)$ päivitykseen käytetään ratkaisuja x^{i,t^*}. Nämä ratkaisut järjestetään paremmuusjärjestykseen. Tässä menetelmässä paras ratkaisu ei saakaan järjestysnumeroa 1 vaan järjestysnumeron m. Vastaavasti huonoin ratkaisu saa nyt järjestysnumeron 1. Käytetään ratkaisun x^{i,t^*} järjestynumerosta merkintää $r(t,x^{i,t^*})$. Nyt ratkaisun x^{i,t^*} rakentanut muurahainen erittää feromoneja määrän $\Delta\tau'(t,x^{i,t^*},x)$, joka määritellään kaavalla

$$\Delta\tau'(t,x^{i,t^*},x) = C\frac{(r(t,x^{i,t^*}))^\alpha}{\sum_{j=1}^m j^\alpha} G(x,x^{i,t^*},\beta^2\Sigma_t), \tag{5.59}$$

missä C on yhdellä iteraatiokierroksella eritettävä feromonien kokonaismäärä, $\alpha > 0$ on ratkaisujen keskinäisen järjestyksen suhteellisen tärkeyden määrittävä parametri, $G(x,x^{i,t^*},\beta^2\Sigma_t)$ on monidimensioinen normaalijakauma, $\beta > 0$ on skaalausparametri ja Σ_t on kaikkien ratkaisujen muodostama kovarianssimatriisi iteraatiokierroksella t. Eritettävä feromonien kokonaismäärä iteraatiokierroksella t on

$$\Delta\tau(t,x) = \sum_{i=1}^m \Delta\tau'(t,x^{i,t^*},x). \tag{5.60}$$

Koko hakuavaruuteen eritetään siis feromoneja iteraatiokierroksella t yhteensä

$$\int_X \Delta\tau(t,x)dx = C. \tag{5.61}$$

Koska C on vakio, jokaisella iteraatiokierroksella eritetään yhtä paljon feromoneja. Lopullinen feromonijälkien $\tau(t,x)$ päivitys suoritetaan kaavalla

$$\tau(t+1,x) = \rho \cdot \tau(t,x) + \Delta\tau(t,x), \tag{5.62}$$

missä $0 \leq \rho < 1$ on feromonien haihtumisnopeuden määrävä parametri. Feromonijälkien $\tau(t,x)$ päivittämisen jälkeen aloitetaan uusi iteraatiokierros, ellei jokin ennalta määrätty lopetusehto ole voimassa.

5.11.1 Uusien ratkaisujen luominen

Suoraan kaavan (5.58) mukaisen satunnaisluvun generointi on vaikeaa. Tätä voidaan kuitenkin helpottaa tekemällä samankaltaisia toimenpiteitä kuin jatkuvien alueiden muurahaisyhdyskuntaoptimoinnissa (luku 5.4).

Aluksi kaava (5.62) pitää kirjoittaa muotoon

$$\tau(t+1,x) = \rho^{t+1} \cdot \tau(0,x) + \sum_{h=0}^{t} \rho^h \Delta\tau(t-h,x). \tag{5.63}$$

Tämän jälkeen kaavoista (5.58), (5.61) ja (5.63) saadaan

$$p(t+1,x) = \frac{\rho^{t+1}}{\sum_{k=0}^{t+1}\rho^k} \cdot \frac{\tau(0,x)}{C} + \sum_{h=0}^{t} \frac{\rho^h}{\sum_{k=0}^{t+1}\rho^k} \cdot \frac{\Delta\tau(t-h,x)}{C}. \tag{5.64}$$

Käytetään kaavassa (5.64) komponentista s merkintää f_s, joka siis määritellään se seuraavasti:

$$f_s(x) = \begin{cases} \frac{\Delta\tau(t-s,x)}{C}, & \text{jos } s \leq t; \\ \frac{\tau(0,x)}{C}, & \text{jos } s = t+1. \end{cases} \tag{5.65}$$

Jotta saadaan generoitua haluttu satunnaisluku, pitää kaavasta (5.64) valita komponentti s todennäköisyydellä

$$p_s = \frac{\rho^s}{\sum_{k=0}^{t+1}\rho^k}. \tag{5.66}$$

Jos valituksi tulee komponentti $f_{t+1}(x)$, on satunnaisluvun generointi helppoa, koska $\tau(0,x)$ on tasainen jakauma. Jos valituksi tulee jokin muu komponentti $f_s(x)$, on satunnaisluvun generointi hankalampaa. Nyt komponentti $f_s(x)$ on siis muotoa

$$\frac{\Delta\tau(t-s,x)}{C} = \sum_{i=1}^{m} \frac{(r(t-s,x^{i,t-s^*}))^\alpha}{\sum_{j=1}^{m} j^\alpha} N(x, x^{i,t-s^*}, \beta^2\Sigma_{t-s}). \tag{5.67}$$

Tästä kaavasta pitää vielä valita komponentti i samoin kuin edellä. Nyt komponentti i on $G(x, x^{i,t-s^*}, \beta^2\Sigma_{t-s})$. Komponentti i valitaan todennäköisyydellä p_i, joka saadaan kaavasta

$$p_i = \frac{(r(t-s, x^{i,t-s^*}))^\alpha}{\sum_{j=1}^{m} j^\alpha}. \tag{5.68}$$

Kun komponentti i on valittu, generoidaan satunnaisluku monidimensioisesta normaalijakaumasta $G(x, x^{i,t-s^*}, \beta^2\Sigma_{t-s})$. Näin ollaan saatu generoitua haluttu satunnaisluku.

Kun generoidaan kaavan (5.64) mukaisia satunnaislukuja edellä esitetyllä tavalla, tarvitsee muistissa säilyttää suuri määrä ratkaisuja $x^{i,t-h^*}$ ja kovarianssimatriiseja Σ_{t-h}. Koska kuitenkin $0 < \rho < 1$, niin $\rho^h \approx 0$ suurilla h. Tämän takia voidaan asettaa yläraja sille kuinka monta iteraatiokierrosta ratkaisuja $x^{i,t-h^*}$ ja kovarianssimatriiseja Σ_{t-h} säilytetään muistissa. Tässä menetelmässä voidaan käyttää parametria H, joka määrittää kyseisen ylärajan. Jos parametria H käytetään ja jos $t+1 > H$, pitää kaavaa (5.64) hieman muuttaa:

$$p(t+1, x) = \sum_{h=0}^{H} \frac{\rho^h}{\sum_{k=0}^{H} \rho^k} \cdot \frac{\Delta\tau(t-h, x)}{C}. \tag{5.69}$$

5.12 Ortogonaalin haun muurahaisyhdyskuntaoptimointi

Nimensä mukaisesti *ortogonaalin haun muurahaisyhdyskuntaoptimoinnissa* (*an orthogonal search embedded ant colony optimization*) käytetään muurahaisten rakentamien ratkaisujen parantamiseen ortogonaalia hakua. Näin ollen ennen itse menetelmän esittelyä pitää esitellä hieman ortogonaalia hakua.

5.12.1 Ortogonaalin haun esittely

Ortogonaalilla haulla etsitään ratkaisua kombinatoriselle ongelmalle. Kaikkia mahdollisia ratkaisuja ei ortogonaalissa haussa testata, vaan yritetään löytää mahdollisimman hyvä ratkaisu testaamalla vain tietty osa mahdollisista ratkaisuista. Testattavat ratkaisut esitetään *ortogonaalissa taulukossa* (*orthogonal array*), josta käytetään merkintää $L_n(S^m)$. Tämä merkintä tarkoittaa, että ortogonaalia hakua käytetään tehtävään, jossa on m muuttujaa, jokaisella muuttujalla on S mahdollista arvoa ja testattavia ratkaisuja on n

kappaletta. Ortogonaalissa taulukossa jokainen rivi tarkoittaa yhtä testattavaa ratkaisua, joten rivejä on n kappaletta. Ortogonaalin taulukon sarakkeessa i taas on muuttujan i arvo, joten sarakkeita on m kappaletta. Jokaisen muuttujan mahdolliset arvot pitää numeroida 1:stä S:ään. Ortogonaalissa taulukossa käytetään näitä muuttujan numeroita eikä muuttujien oikeita arvoja. Testattavien ratkaisujen joukossa ei ole kahta ratkaisua, joilla enemmän kuin yksi muuttuja saisi saman arvon. Esimerkiksi

$$L_4(2^3) = \begin{matrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1, \end{matrix} \tag{5.70}$$

$$L_9(3^4) = \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 3 & 3 & 3 \\ 2 & 1 & 2 & 3 \\ 2 & 2 & 3 & 1 \\ 2 & 3 & 1 & 2 \\ 3 & 1 & 3 & 2 \\ 3 & 2 & 1 & 3 \\ 3 & 3 & 2 & 1. \end{matrix} \tag{5.71}$$

Olkoon esimerkkinä seuraava kombinatorinen optimointiongelma:

$$\begin{array}{ll} \min & 5x_1 - 3x_2 - x_3 \\ \text{s.e.} & x_1 \in \{1, 2\}, \\ & x_2 \in \{3, 4\}, \\ & x_3 \in \{5, 6\}. \end{array} \tag{5.72}$$

Tehtävässä on siis 8 mahdollista ratkaisua ja optimaalinen ratkaisu on selvästi $(x_1, x_2, x_3) = (1, 4, 6)$. Käytetään muuttujista x_1, x_2 ja x_3 vastaavasti merkintöjä A, B ja C. Suoritetaan muuttujien arvojen numerointi niin, että aina pienempi mahdollinen arvo saa numeron 1 ja suurempi numeron 2. Nyt esimerkiksi merkintä $A_1B_1C_1$ tarkoittaa ratkaisua $(x_1, x_2, x_3) = (1, 3, 5)$. Kun tähän tehtävään käytetään ortogonaalia taulukkoa $L_4(2^3)$ (5.70), testattavat ratkaisut ovat $A_1B_1C_1$, $A_1B_2C_2$, $A_2B_1C_2$ ja $A_2B_2C_1$ eli $(x_1, x_2, x_3) = (1, 3, 5)$, $(x_1, x_2, x_3) = (1, 4, 6)$, $(x_1, x_2, x_3) = (2, 3, 6)$ ja

$(x_1, x_2, x_3) = (2, 4, 5)$. Tässä tapauksessa ortogonaalilla haulla siis löydettäisiin optimaalinen ratkaisu.

5.12.2 Itse menetelmä

Algoritmin suorituksen alussa pitää luoda N kappaletta hiloja. Hilalla H^i on seuraavat ominaisuudet: keskipiste $X^i = (x_1^i, ..., x_n^i)$, hakusäde $R^i = (r_1^i, ..., r_n^i)$, hilan fitnessfunktion arvo $q_f(X^i)$ ja feromonijäljen arvo $\tau_i(t)$. Hilojen keskipisteet $X^1, ..., X^N$ valitaan aluksi satunnaisesti tasaisesta jakaumasta yli koko hakuavaruuden. Nämä keskipisteet voivat siirtyä, jos löydetään parempi keskipiste $X^{i'}$, jossa $q_f(X^{i'}) > q_f(X^i)$. Hakusäteet $R_1, ..., R_N$ annetaan aluksi menetelmän parametreina, mutta ne voivat muuttua algoritmin suorituksen edetessä. Kaikkien feromonijälkien arvoiksi alustetaan $\tau_i(t) = \tau_0$, missä τ_0 on menetelmän parametri.

Jokaisella iteraatiokierroksella kaikki m muurahaista rakentaa ratkaisunsa valitsemalla ensin itselleen jonkin hilan H^i. Muurahainen valitsee hilan s seuraavan kaavan mukaisesti:

$$s = \begin{cases} \text{argmax}_{1 \leq i \leq N}\{\tau_i(t)\}, & \text{jos } p \leq q_0; \\ S, & \text{jos } p > q_0; \end{cases} \tag{5.73}$$

missä p on tasaisesti välille [0,1] jakautunut satunnaismuuttuja, $0 \leq q_0 \leq 1$ on menetelmän parametri ja S on satunnaismuuttuja. Todennäköisyys, että satunnaismuuttuja S saa arvon j on (vertaa (5.4))

$$p_j(t) = \frac{\tau_j(t)}{\sum_{i=1}^{N} \tau_i(t)}. \tag{5.74}$$

Näin ollen todennäköisyydellä q_0 muurahainen valitsee hilan H^i, jonka feromonijäljen $\tau_i(t)$ arvo on suurin.

Kun muurahainen on valinnut hilan H^i, suoritetaan ortogonaali haku kyseisen hilan parantamiseksi. Nyt siis yritetään löytää hilalle H^i uusi keskipiste, jossa fitnessfunktion arvo on suurempi kuin nykyisessä keskipisteessä $X^i = (x_1^i, ..., x_n^i)$. Tätä varten ortogonaalin haun suorittamiseksi tarvitaan uusia komponentteja $x_j^{i^k}$ uusien kandidaattiratkaisujen muodostamiseksi. Näitä uusia komponentteja $x_j^{i^k}$ luodaan l kappaletta jokaiseen dimensioon eli yhteensä $l \cdot n$ kappaletta. Luotavat komponentit saadaan kaavasta

$$x_j^{i^k} = x_j^i - r_j^i + \frac{2kr_j^i}{l-1}, \quad k = 0, ..., l-1. \tag{5.75}$$

Tämän jälkeen ortogonaalissa haussa käytetään ortogonaalia taulukkoa $L_{l^2}(l^n)$. Näin ollen saadaan l^2 kappaletta kandidaattiratkaisuja. Käytetään kandidaattiratkaisusta, jolla fitnessfunktion q_f arvo on suurin, merkintää $X^{i'}$. Jos $f(X^{i'}) > f(X^i)$, siirretään hilan H^i keskipiste pisteeseen $X^{i'}$. Jos taas $f(X^{i'}) \leq f(X^i)$, ei hilan H^i keskipistettä siirretä mihinkään. Ortogonaalin haun jälkeen hakusäteitä $R^i = (r_1^i, ..., r_n^i)$ päivitetään seuraavasti:

$$r_j^i \leftarrow \begin{cases} \frac{r_j^i}{\delta}, & \text{jos hilan } H^i \text{ keskipistettä siirrettiin;} \\ r_j^i \cdot \delta, & \text{jos hilan } H^i \text{ keskipistettä ei siirretty;} \end{cases} \tag{5.76}$$

missä $0 \leq \delta \leq 1$ on menetelmän parametri.

Kun kaikki m muurahaista ovat rakentaneet ratkaisunsa eli valinneet itselleen hilan H^i ja kun näille hiloille on suoritettu ortogonaali haku, poistetaan $(1 - \chi)N$ kappaletta hiloja. Menetelmän parametrille χ pitää olla voimassa $0 \leq \chi \leq 1$. Ne hilat H^i, joiden fitnessfunktioiden arvot $q_f(X^i)$ ovat pienimmät, poistetaan. Näin ollen χN kappaletta hiloja H^i, joiden fitnessfunktioiden arvot $q_f(X^i)$ ovat suurimmat, säilytetään. Poistettujen hilojen tilalle luodaan $(1 - \chi)N$ kappaletta uusia hiloja. Nämä uudet hilat luodaan samalla tavoin kuin algoritmin suorituksen alussa luotiin hiloja eli uusien hilojen H^i keskipisteet X^i valitaan satunnaisesti tasaisesta jakaumasta yli koko hakuavaruuden ja hakusäteet R^i annetaan menetelmän parametreina. Uusien hilojen H^i feromonijälkien arvoiksi asetetaan $\tau_i(t + 1) = \tau_0$. Jos säilytettävien hilojen joukossa on hiloja, joiden etäisyys toisistaan on pienempi kuin parametrina annettu ϵ_1, säilytetään näistä loppujen lopuksi vain yksi. Nytkin jokaisen poistetun hilan tilalle luodaan uusi hila edellä esitetyllä tavalla.

Hilojen poistamisen ja uudelleenluomisen jälkeen suoritetaan säilytettyjen hilojen H^i feromonijälkien arvojen $\tau_i(t)$ päivitys. Tätä varten säilytettävät hilat H^i järjestetään fitnessfunktioiden arvojen $q_f(X^i)$ mukaiseen järjestykseen. Suurimman fitnessfunktion arvon omaava hila saa järjestysnumeron 1, toiseksi suurimman fitnessfunktion arvon omaava hila saa järjestysnumeron 2 ja niin edelleen. Käytetään hilan H^i järjestysnumerosta merkintää j^i. Olkoon vielä m^i niiden muurahaisten lukumäärä, jotka ovat valinneet hilan H^i kyseisellä iteraatiokierroksella. Feromonijälkien $\tau_i(t)$ päivitys suoritetaan kaavalla

$$\tau_i(t + 1) = (1 - \rho)\tau_i(t) + \rho \cdot \tau_0 \cdot (\chi \cdot N - j^i + m^i), \tag{5.77}$$

missä $0 \leq \rho \leq 1$ on parametrina annettava feromonien haihtumisnopeus. Jos feromonijälkien päivityksen jälkeen jokin ennalta määrätty lopetusehto on voimassa, lopetetaan algoritmin suoritus. Muussa tapauksessa muurahaiset aloittavat seuraavan iteraatiokierroksen ratkaisujen rakentamisen.

5.13 Rajoitteisten tehtävien ratkaisu

Jos halutaan ratkaista rajoitteellista optimointitehtävää, niin monissa edellä esitellyissä menetelmissä saatetaan luoda ratkaisuja, jotka eivät pysy sallitulla alueella. Näitä ei-sallittuja ratkaisuja voidaan käsitellä monella eri tavalla. Yksinkertaisin tapa on poistaa ei-sallittu ratkaisu ja luoda sen tilalle uusi ratkaisu. Tätä voidaan jatkaa kunnes saadaan sallittu ratkaisu.

Ei-sallittu ratkaisu voidaan myös siirtää sallitulle alueelle. Jos tehtävässä on vain laatikkorajoitukset $x_i^a \leq x_i \leq x_i^y, i = 1, 2, ..., n$, voidaan tehdä kuten muurahaisyhdyskuntaoptimoinnin suorassa sovelluksessa jatkuville ongelmille luvussa 5.6 eli käyttää ei-sallitulle ratkaisulle x^- kaavaa (5.19). Jos tehtävässä on muitakin kuin laatikkorajoituksia, ei-sallitun ratkaisun siirtäminen sallitulle alueelle on hieman vaikeampaa. Siirtämiseen voidaan esimerkiksi käyttää menetelmää, jossa tarvitaan apuna jotakin sallittua ratkaisua x^+ (katso kuva 5.11). Menetelmässä etsitään ei-sallitun ratkaisun x^- ja sallitun ratkaisun x^+ väliseltä janalta piste, joka on yhtä kaukana molemmista ratkaisuista. Käytetään tästä pisteestä merkintää x^c. Tämän jälkeen testataan, onko x^c sallitulla alueella vai ei. Jos x^c on sallitulla alueella, asetetaan $x^+ \leftarrow x^c$. Jos taas x^c ei ole sallitulla alueella, asetetaan $x^- \leftarrow x^c$. Tätä ratkaisujen x^- ja x^+ välin puolittamista jatketaan kunnes löydetään sellainen piste x^{c*}, joka on sallittu ja riittävän lähellä sallitun alueen reunaa. Tässä tapauksessa "riittävän lähellä" voidaan määritellä esimerkiksi käytettävän menetelmän parametrina. Lopuksi asetetaan $x^- \leftarrow x^{c*}$ ja näin ollaan saatu siirrettyä ei-sallittu ratkaisu x^- sallitulle alueelle.

Tietysti on myös mahdollista käyttää jotakin *sakkofunktiomenetelmää* (*penalty function method*), jonka avulla rajoitteellinen optimointitehtävä muunnetaan rajoittamattomaksi optimointitehtäväksi. Tähän rajoittamattomaan optimointitehtävään voidaan sitten käyttää edellä esiteltyjä menetelmiä. Olkoon nyt tehtävänä ratkaista seuraava rajoitteellinen opti-

mointitehtävä:

$$\begin{array}{ll} \min & f(x) \\ \text{s.e.} & g_i(x) \leq 0 \quad i = 1, ..., m, \\ & h_i(x) = 0 \quad i = 1, ..., k. \end{array} \tag{5.78}$$

Yleisessä sakkofunktiomenetelmässä käytetään sakkofunktiota $\alpha(x)$:

$$\alpha(x) = \sum_{i=1}^{m} \varphi[g_i(x)] + \sum_{i=1}^{k} \Psi[h_i(x)], \tag{5.79}$$

missä funktio φ toteuttaa ehdot

$$\begin{cases} \varphi(y) = 0, & \text{jos } y \leq 0; \\ \varphi(y) > 0, & \text{jos } y > 0; \end{cases} \tag{5.80}$$

ja funktio Ψ toteuttaa ehdot

$$\begin{cases} \Psi(y) = 0, & \text{jos } y = 0; \\ \Psi(y) > 0, & \text{jos } y \neq 0. \end{cases} \tag{5.81}$$

Usein käytetään seuraavanlaisia funktioita:

$$\begin{array}{l} \varphi(y) = (\max\{0, y\})^{\beta}; \\ \Psi(y) = |y|^{\beta}, \end{array} \tag{5.82}$$

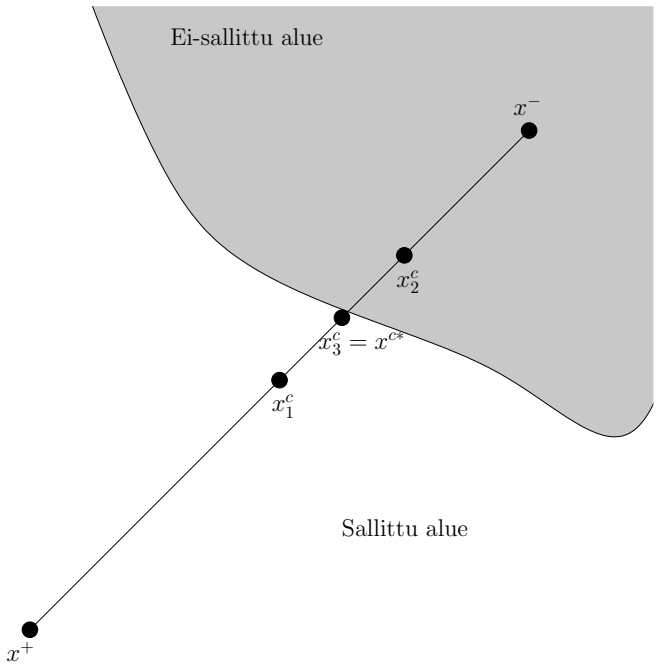
missä $\beta > 0$. Yleensä sakkofunktio onkin muotoa

$$\alpha(x) = \sum_{i=1}^{m} [\max\{0, g_i(x)\}]^2 + \sum_{i=1}^{k} [h_i(x)]^2. \tag{5.83}$$

Optimointitehtävässä käytetään lopulta kohdefunktiota $f(x) + \Omega\alpha(x)$ eli optimointitehtävä (5.78) on saatu muunnettua muotoon

$$\begin{array}{ll} \min & f(x) + \Omega\alpha(x) \\ \text{s.e.} & x \in R^n, \end{array} \tag{5.84}$$

missä $\Omega > 0$ on arvoltaan riittävän suuri *rangaistuskerroin*. Nyt tehtävään (5.84) voidaan käyttää edellä esitettyjä menetelmiä eikä niiden algoritmien suorituksen aikana voida luoda ei-sallittuja ratkaisuja, koska muunnetussa optimointitehtävässä ei ole rajoitteita. Pitää kuitenkin muistaa, että edelleen on mahdollista luoda alkuperäisen tehtävän kannalta ei-sallittuja ratkaisuja. Tämän takia onkin tärkeää valita käytettävä Ω riittävän suureksi, jotta mahdollisesti luodut alkuperäisen tehtävän kannalta ei-sallitut ratkaisut ovat



Kuva 5.11: Esimerkki ei-sallitun ratkaisun x^- siirtämisestä sallitulle alueelle sallitun ratkaisun x^+ avulla. Kuvassa merkintä x_i^c tarkoittaa pisteiden x^- ja x^+ välistä keskipistettä esitellyn menetelmän vaiheessa i.

muunnetun tehtävän kannalta riittävän huonoja verrattuna alkuperäisen tehtävän kannalta sallittuihin ratkaisuihin. Jos siis valitaan liian pieni Ω, voidaan muunnetun optimointitehtävän ratkaisuna saada ratkaisu, joka on alkuperäisen tehtävän kannalta ei-sallittu.

6 Muurahaisyhdyskuntaoptimoinnin menetelmien toimivuus

Tässä luvussa verrataan jatkuvien alueiden muurahaisyhdyskuntaoptimoinnin (katso luku 5.4) toimintaa muiden jatkuville ongelmille soveltuvien muurahaisyhdyskuntaoptimoinnin menetelmien toimintaan. Jatkuvien alueiden muurahaisyhdyskuntaoptimointiin (JAMYO) vertailtavien menetelmien tiedot on saatu kirjallisuudesta [1, 12, 20, 21, 23, 26, 28, 33, 34, 35]. Tässä työssä ei suoriteta vertailua pseudorinnakkaiseen muurahaisyhdyskuntaoptimointiin eikä evolutiiviseen muurahaisyhdyskunta-algoritmiin, koska näiden menetelmien osalta kirjallisuudesta ei löydy vertailun suorittamiseksi tarvittavaa informaatiota. JAMYO:n toimintaa on testattu Pentium 4 (3,2 GHz) tietokoneella ja Mathematica 6.0 ohjelmistolla. Tätä testausta varten tarvittava koodi funktiolle f_1, jossa $n_1 = 3$, $a_1 = -5$ ja $b_1 = 5$ (katso taulukot 3 ja 4) sekä menetelmään liittyvät parametrit ovat $m = 2$, $\xi = 0{,}85$, $q = 10^{-4}$ ja $k = 15$, on esitetty liitteessä B. Koodia täytyy tietyiltä osin hieman muuttaa, kun muita funktioita testataan tai parametrien arvoja muutetaan.

Yhtenä vertailukriteerinä tässä luvussa käytetään menetelmien onnistumisprosenttia optimoitaessa eri funktioita. Kaikissa vertailuissa onnistumisen määritelmä ei ole sama. Tämä johtuu siitä, että kaikkiin muihin menetelmiin paitsi JAMYO:in liittyvät arvot on saatu kirjallisuudesta ja kirjallisuuslähteissä onnistuminen määritellään eri tavalla eri menetelmien yhteydessä. Jotta tulokset olisivat vertailukelpoisia, pitää tietysti myös JAMYO:a testattaessa käyttää samoja määritelmiä kuin vertailtavien menetelmien testauksessa on käytetty. Samasta syystä vertailuissa on käytetty eri testifunktioita ja eri vertailukriteerejä.

Ennen eri menetelmien vertailua kuitenkin testataan JAMYO:n parametrien m, ξ, q ja k vaikutusta menetelmän toimintaan funktioita f_1 ja f_2 optimoitaessa.

6.1 Parametrien vaikutus JAMYO:n toimintaan

Kuten luvussa 5.4 mainittiin, on JAMYO:ssa neljä parametria: jokaisella iteraatiokierroksella käytettävien muurahaisten määrä m, feromonien haihtumisnopeus ξ, parhaiden ratkaisujen painotuksen määräävä parametri q sekä alussa luotavien ratkaisujen ja samalla muistissa koko ajan säilytettävien

funktio	kaava	lähteet
f_1	$\sum_{i=1}^{n_1} x_i^2$	[12, 26, 35]
f_2	$1 + \sum_{i=1}^{2} \frac{x_i^2}{4000} - \prod_{i=1}^{2} \cos\left(\frac{x_i}{\sqrt{i}}\right)$	[12]
f_3	$\sum_{i=1}^{n_3-1}(100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$	[1, 12, 26, 33]
f_4	$(1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_1^2)) \cdot (30 + (2x_1 - 3x_2)^2 \cdot (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$	[12]
f_5	$(x_1 - x_2)^2 + \left(\frac{x_1+x_2-10}{3}\right)^2$	[12]
f_6	$10 \cdot n_6 + \sum_{i=1}^{n_6}(x_i^2 - 10\cos(2\pi x_i))$	[26, 33, 35]
f_7	$(x_1^2 + x_2^2)^{0,25} \cdot \left(1 + (\sin(50 \cdot (x_1^2 + x_2^2)^{0,1}))^2\right)$	[26]
f_8	$10 - \sum_{i=1}^{3}(x_i - 5)^2$	[1]
f_9	$30 + \sum_{i=1}^{5} \lfloor x_i \rfloor$	[1]
f_{10}	$\frac{\sin\left(\sum_{i=1}^{5} \lvert x_i - 5 \rvert\right)}{\sum_{i=1}^{5} \lvert x_i - 5 \rvert}$	[1]
f_{11}	$\sum_{i=1}^{20} i x_i^2$	[33]
f_{12}	$\sum_{i=1}^{20} \left(\sum_{j=1}^{i} x_j\right)^2$	[33]
f_{13}	$\sum_{i=1}^{4} \lvert x_i \rvert + \prod_{i=1}^{4} \lvert x_i \rvert$	[35]
f_{14}	$\max_{1 \le i \le 4}\{\lvert x_i \rvert\}$	[35]
f_{15}	$10 - \sum_{i=1}^{4}(100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$	[1]

Taulukko 3: Menetelmien vertailuissa käytettävät funktiot. Joitakin funktioita käytetään useassa vertailussa, jolloin funktion dimensio ei välttämättä ole aina sama. Tämän takia esimerkiksi funktion f_1 dimensio on n_1. Kun tätä funktiota käytetään vartailussa, ilmoitetaan joka kerta erikseen minkä arvon n_1 saa.

funktio	optimoinnin laatu	hakuväli	optimiarvo	lokaali optimi
f_1	min	$[a_1; b_1]$	0	
f_2	min	[-5,12; 5,11]	0	
f_3	min	$[a_3; b_3]$	0	
f_4	min	[-2; 2]	3	
f_5	min	[-20; 20]	0	
f_6	min	$[a_6; b_6]$	0	x
f_7	min	[-100; 100]	0	x
f_8	max	[1; 10]	10	
f_9	max	[1; 10,1]	80	x
f_{10}	max	[1; 10]	1	
f_{11}	min	[-3,12; 7,12)	0	
f_{12}	min	[-44; 84)	0	
f_{13}	min	[-10; 10]	0	
f_{14}	min	[-100; 100]	0	x
f_{15}	max	[1; 10]	10	

Taulukko 4: Tietoja menetelmien vertailuissa käytettävistä funktioista. Sarakkeessa *optimoinnin laatu* ilmoitetaan, onko kyseessä minimoitava vai maksimoitava funktio. Sarakkeessa *hakuväli* ilmoitetaan miltä väliltä kyseisen funktion optimia haetaan jokaisessa dimensiossa. Sarakkeessa *optimiarvo* ilmoitetaan, minkä arvon kyseinen funktio saa optimipisteessä tai -pisteissä. Funktion kohdalla on viimeisessä sarakkeessa *lokaali optimi* merkintä x, jos funktiolla on vertailuissa käytetyillä hakuväleillä ainakin yksi lokaali optimi, joka ei ole samalla globaali optimi. Jos sitä vastoin funktion kohdalla ei sarakkeessa *lokaali optimi* ole merkintää x, niin funktioilla ei ole vertailuissa käytetyillä hakuväleillä yhtään lokaalia optimia, joka ei olisi samalla globaali optimi.

ratkaisujen määrä k. Artikkelissa [30] on optimoitu tehtäviä, joiden dimensio 10. Kyseisessä artikkelissa on todettu, että parametreille hyvät arvot ovat $m = 2$, $\xi = 0{,}85$, $q = 10^{-4}$ ja $k = 50$. Mutta kuten edellä kappaleessa 5.4 todettiin, pitää olla $k \geq n$. Näin ollen voidaan olettaa, että kannattaa asettaa $k = 5n$, missä n optimoitavan tehtävän dimensio. Kun nyt testataan parametrien vaikutusta JAMYO:n toimintaan, käytetäänkin näitä arvoja ($m = 2$, $\xi = 0{,}85$, $q = 10^{-4}$ ja $k = 5n$) parametrien perusarvoina, joista sitten muutetaan yhtä kerrallaan muiden parametrien pysyessä perusarvossaan. Testiajoissa käytetään seuraavia parametrien arvoja: $m = 1, 2, 3, 5, 8$, $\xi = 0{,}65;\ 0{,}75;\ 0{,}85;\ 0{,}95;\ 1{,}05$, $q = 10^{-6}, 10^{-5}, 10^{-4}, 10^{-2}, 2 \cdot 10^{-1}$ ja $k = 2n, 3n, 5n, 10n, 15n$.

Tässä luvussa esitetyissä taulukoissa sarake *onn.* $-$ % tarkoittaa menetelmän onnistumisprosenttia. Onnistumiseksi lasketaan nyt se, jos menetelmä löytää ratkaisun, jolla kohdefunktion arvon ja kohdefunktion optimiarvon ero on korkeintaan 10^{-4}. Kun menetelmä on onnistunut eli se on löytänyt tarpeeksi hyvän ratkaisun, on testiajo lopetettu eikä ratkaisua ole enää yritetty parantaa. Testiajoja on tehty jokaiselle parametrikombinaatiolle 100 kappaletta ja jokaisessa testiajossa kohdefunktion arvon laskuja on suoritettu korkeintaan 10000 kappaletta. Jos siis 10000 kohdefunktion arvon laskun jälkeen ei ole löydetty ratkaisua, jolla kohdefunktion arvon ja kohdefunktion optimiarvon ero olisi korkeintaan 10^{-4}, niin menetelmän katsotaan epäonnistuneen optimoinnissa. Sarake *alka* taas tarkoittaa kohdefunktion arvon laskujen määrän keskiarvoa onnistuneissa testiajoissa.

Ensin testataan parametrien vaikutusta JAMYO:n toimintaan optimoitaessa funktiota f_1, missä $n_1 = 3$, $a_1 = -5$ ja $b_1 = 5$. Näin ollen parametrien perusarvot ovat $m = 2$, $\xi = 0{,}85$, $q = 10^{-4}$ ja $k = 15$. Taulukosta 5 huomataan, että jokaisella iteraatiokierroksella käytettyjen muurahaisten määrä m ei paljoakaan vaikuta JAMYO:n toimintaan ainakaan tämän funktion tapauksessa. Kun muurahaisten määrää m kasvatetaan yhdestä kahteen, laskee kohdefunktion arvon laskujen määrän keskiarvo hieman. Samoin käy, kun muurahaisten määrää m kasvatetaan kahdesta kolmeen. Muurahaisten määrän m kasvaessa kolmesta viiteen nousee kohdefunktion arvon laskujen määrän keskiarvo hieman. Kun taas muurahaisten määrää m kasvatetaan viidestä kahdeksaan, laskee kohdefunktion arvon määrän laskujen keskiarvo. Muutokset kohdefunktion arvon laskujen määrän keskiarvossa

eivät kuitenkaan ole merkittäviä: Huonoin ($m = 1$) on noin 1,4-kertainen parhaaseen ($m = 3$) verrattuna. Myöskään onnistumisprosentti ei paljoa muutu: Se on kaikissa tapauksissa joko 99 tai 100.

Feromonien haihtumisnopeudella ξ on huomattava vaikutus JAMYO:n toimintaan optimoitaessa funktiota f_1, kuten taulukosta 5 huomataan. Kun $\xi = 0{,}65$, on kohdefunktion arvon laskujen määrän keskiarvo selvästi pienempi kuin suuremmilla parametrin ξ arvoilla (0,75, 0,85, 0,95 ja 1,05): Toiseksi paras ($\xi = 0{,}95$) arvo on noin 2,3-kertainen verrattuna parhaaseen. Näiden suurempien arvojen välillä kohdefunktion arvon laskujen määrän keskiarvo ei poikkea kovinkaan huomattavasti: Huonoin ($\xi = 0{,}85$) arvo on noin 1,5-kertainen verrattuna toiseksi parhaaseen. Vaikka parhaat tulokset kohdefunktion arvon laskujen määrän perusteella saadaankin, kun $\xi = 0{,}65$, niin tätä arvoa parametrille ξ ei voida pitää hyvänä, koska tällä arvolla menetelmän onnistumisprosentti on vain 11. Kun $\xi = 0{,}75$, on menetelmän onnistumisprosentti 39 ja muilla parametrin ξ arvoilla 100. Näin ollen ainakin tällä funktiolla kannattaa valita parametrin ξ arvoksi vähintään 0,85.

Parametrien q ja k muuttamisella ei ole huomattavaa vaikutusta JAMYO:n toimintaan optimoitaessa funktiota f_1. Kaikilla testattavilla parametrien q ja k arvoilla menetelmän onnistumisprosentti on 100. Kohdefunktion arvon laskujen määrän keskiarvokaan ei muutu paljoa, kun parametria q muutetaan: Huonoin ($q = 2 \cdot 10^{-1}$) arvo on vajaa 1,2-kertainen verrattuna parhaaseen ($q = 10^{-6}$) arvoon. Myös parametrin k arvon muutos vaikuttaa kohdefunktion arvon laskujen määrän keskiarvoon vain vähän: Huonoin ($k = 15$) arvo on noin 1,4-kertainen verrattuna parhaaseen ($k = 30$) arvoon. Parametrin k arvon suurentaminen kuitenkin lisää algoritmin suorituksen aikana tarvittavaa muistin määrää ja laskenta-aikaa. Näin ollen parametria k ei kannata valita kovin suureksi. Parametrien vaikutusta testattaessa laskenta-ajat ovat molempien testifunktioiden kohdalla kuitenkin niin lyhyitä, että parametrin k valinnalla ei ole käytännössä näissä tapauksissa suurta merkitystä.

Toisena testataan parametrien vaikutusta JAMYO:n toimintaan optimoitaessa funktiota f_2. Tässä tapauksessa parametrien perusarvot ovat $m = 2$, $\xi = 0{,}85$, $q = 10^{-4}$ ja $k = 10$. Tätä funktiota optimoitaessa jokaisella iteraatiokierroksella käytettyjen muurahaisten määrä m vaikuttaa menetelmän toimintaan enemmän kuin funktion f_1 tapauksessa. Huonoim-

	$onn.-\%(f_1)$	$alka(f_1)$	$onn.-\%(f_2)$	$alka(f_2)$
$m=1$	99	302	15	837
$m=2$	**100**	251	12	**414**
$m=3$	**100**	**222**	**22**	456
$m=5$	**100**	276	15	769
$m=8$	99	253	11	537
$\xi=0{,}65$	11	**73**	11	**151**
$\xi=0{,}75$	39	248	**20**	823
$\xi=0{,}85$	**100**	251	12	414
$\xi=0{,}95$	**100**	166	8	1107
$\xi=1{,}05$	**100**	169	13	1715
$q=10^{-6}$	**100**	**225**	14	**346**
$q=10^{-5}$	**100**	232	11	971
$q=10^{-4}$	**100**	251	12	414
$q=10^{-2}$	**100**	238	**17**	572
$q=2\cdot10^{-1}$	**100**	261	8	842
$k=2n$	**100**	185	16	527
$k=3n$	**100**	226	14	951
$k=5n$	**100**	251	12	**414**
$k=10n$	**100**	**180**	17	2098
$k=15n$	**100**	207	**18**	1194

Taulukko 5: Parametrien vaikutus JAMYO:n toimintaan funktioita f_1 ja f_2 optimoitaessa. Vasemman puoleiset sarakkeet liittyvät funktioon f_1 ja oikean puoleiset funktioon f_2. Testattaessa funktiota f_1, saa parametri k arvot 6, 9, 15, 30 ja 45 ($n=3$). Testattaessa funktiota f_2, saa parametri k arvot 4, 6, 10, 20 ja 30 ($n=2$). Taulukossa on lihavoitu jokaisessa sarakkeessa jokaisen testattavan parametrin suhteen parhaat arvot.

massa tapauksessa ($m = 1$) on kohdefunktion arvon laskujen määrän keskiarvo yli 2-kertainen verrattuna parhaaseen tapaukseen ($m = 2$). Onnistumisprosentti on kaikilla parametrin m arvoilla huono, kuten taulukosta 5 huomataan.

Funktion f_2 tapauksessa feromonien haihtumisnopeuden ξ vaikutus onnistumisprosenttiin ei ole niin selvä kuin funktion f_1 tapauksessa. Onnistumisprosentti funktiota f_2 optimoitaessa on alhainen kaikilla testattavilla parametrin ξ arvoilla eikä siis tällä kertaa parametrin ξ arvon kasvattaminen auta, kuten auttoi funktion f_1 tapauksessa. Paras onnistumisprosentti on nyt 20 ($\xi = 0{,}75$) ja huonoin 8 ($\xi = 0{,}95$). Tällä kertaa parametrin ξ arvon vaikutus kohdefunktion arvon laskujen määrän keskiarvoon on vielä suurempi kuin funktiota f_1 optimoitaessa: Huonoin tulos ($\xi = 1{,}05$) on yli 11-kertainen varrattuna parhaaseen tulokseen ($\xi = 0{,}65$).

Kun parametrin q arvoa muutetaan optimoitaessa funktiota f_2, saadaan kohdefunktion arvon laskujen määrän keskiarvon perusteella huonoin tulos, kun $q = 10^{-5}$. Tämä huonoin tulos on noin 2,8-kertainen verrattuna parhaaseen tulokseen, joka saadaan, kun $q = 10^{-6}$. Myös kaikilla testattavilla parametrin q arvoilla onnistumisprosentti on alhainen. Paras onnistumisprosentti (17) saavutetaan, kun $q = 10^{-2}$ ja huonoin onnistumisprosentti (8) on silloin, kun $q = 2 \cdot 10^{-1}$.

Testattaessa parametrin k vaikutusta JAMYO:n toimintaan optimoitaessa funktiota f_2 huomataan, että myös kaikilla parametrin k arvoilla menetelmän onnistumisprosentti on huono. Parhaimmillaan onnistumisprosentti on 18 ($k = 30$) ja huonoimmillaan 12 ($k = 10$). Parametrin k arvoa muutettaessa huonoin tulos ($k = 20$) kohdefunktion arvon laskujen määrän perusteella on yli 5-kertainen parhaaseen tulokseen ($k = 10$) verrattuna.

6.2 Muurahaisyhdyskuntaoptimoinnin menetelmien toiminnan vertailu

Seuraavaksi JAMYO:n toimintaa verrataan eri funktioiden avulla muiden luvussa 5 esiteltyjen menetelmien toimintaan. Kaikissa vertailussa käytetään JAMYO:n parametreille edellä esiteltyjä perusarvoja ($m = 2$, $\xi = 0{,}85$, $q = 10^{-4}$ ja $k = 5n$). Testifunktioissa on vain laatikkorajoitteita, joten jos JAMYO luo ratkaisun, joka ei ole sallitulla alueella, se siirretään sallitun alueen rajalle (katso kaava (5.19)).

Ensin JAMYO:n toimintaa verrataan funktioiden f_1, f_3, f_4 ja f_5 avulla jatkuvaan muurahaisyhdyskuntaoptimointiin (JMYO, katso luku 5.1), jatkuvaan vuorovaikuttavaan muurahaisyhdyskuntaan (JVMY, katso luku 5.2), muurahaisyhdyskuntaoptimoinnin suoraan sovellukseen jatkuville ongelmille (MYOSS, katso luku 5.6) ja binääriseen muurahaissysteemiin (BMS, katso luku 5.8). Näissä vertailuissa funktiossa f_1 käytetään arvoja $n_1 = 6$, $a_1 =$ -5,12 sekä $b_1 =$ 5,12 ja funktiossa f_3 arvoja $n_3 = 2$, $a_1 = -5$ sekä $b_1 = 10$. Tällä kertaa onnistumiseksi lasketaan se, jos menetelmä löytää ratkaisun, jolla kohdefunktion arvon ja kohdefunktion optimiarvon ero on korkeintaan $10^{-4} + 10^{-4} \cdot f^*$, missä f^* on kohdefunktion optimiarvo. Jokaisella funktiolla tehdään 100 toisistaan riippumatonta testiajoa.

Vertailua varten tarvitaan myös muiden vertailtavien menetelmien kuin JAMYO:n parametrien arvot. JMYO:n kaikkien parametrien, paitsi feromonien haihtumisnopeuden ($\rho = 0{,}9$), arvot muuttuvat optimoitavan funktion muuttuessa [25]. Funktiota f_1 optimoitaessa käytetään seuraavia parametrien arvoja: $m = 100$, $H = 200$, $p_m = 0{,}5$, $p_r = 1$ ja $b = 10$. Funktiolle f_3 käytetään arvoja $m = 150$, $H = 200$, $p_m = 0{,}5$, $p_r = 0{,}8$ ja $b = 10$. Funktiolle f_4 parametrien arvot ovat $m = 150$, $H = 200$, $p_m = 0{,}6$, $p_r = 0{,}2$ ja $b = 10^{-4}$. Funktiolle f_5 menetelmän parametrit ovat $m = 40$, $H = 60$, $p_m = 0{,}5$, $p_r =$ 0,3 ja $b = 10$. JVMY:ssa parametrien arvot ovat $m = 1000 \cdot (1 - e^{-\frac{n}{10}}) + 5$ (missä n on optimoitavan funktion dimensio), $\rho = 0{,}1$, $\sigma = 0{,}9$ ja $v_0 = \frac{2}{3} \cdot m$ [12]. MYOSS:ssa käytettävät parametrien arvot ovat $m = 3$, $\rho = 0{,}85$, k_1 = 0,05, $k_2 = 0{,}1$, $k_3 = 0{,}5$ ja $k_4 = 1$ [20]. BMS:ssä käytetään seuraavia parametrien arvoja: $m = 5$, $d = 12$, $\tau_H = 0{,}65$, $\tau_L = 0{,}35$, $\rho_0 = 0{,}3$, $k_1 =$ 0,2, $k_2 = 0{,}3$, $k_3 = 0{,}4$, $k_4 = 0{,}5$ ja $k_5 = 0{,}9$ [21].

JMYO:n testaamiseen on käytetty Pentium (200 MHz) tietokonetta ja Fortran 77 ohjelmointikieltä [28]. JVMY:n, MYOSS:n ja BMS:n osalta kirjallisuudesta ei löydy vastaavaa testausinformaatiota.

Taulukosta 6 nähdään, että kohdefunktion arvon laskujen määrän keskiarvon suhteen JAMYO on verrattavista menetelmistä kolmanneksi paras optimoitaessa funktiota f_1, neljänneksi paras (toiseksi huonoin) optimoitaessa funktiota f_3, paras optimoitaessa funktiota f_4 ja toiseksi paras optimoitaessa funktiota f_5. Funktion f_4 kohdalla pitää kuitenkin huomioida, että JAMYO:n onnistumisprosentti on vain 70, kun kolmen muun menetelmän onnistumisprosentti on 100. Taulukosta 6 nähdään myös, että funktioita

		f_1	f_3	f_4	f_5
JAMYO	$onn.-\%$	**100**	**100**	70	**100**
	$alka$	468	11449	**166**	273
JMYO	$onn.-\%$	**100**	**100**	**100**	**100**
	$alka$	22050	6842	5330	1688
JVMY	$onn.-\%$	**100**	**100**	56	20
	$alka$	50000	11797	23391	11751
MYOSS	$onn.-\%$	**100**	**100**	**100**	**100**
	$alka$	265	**1947**	230	**169**
BMS	$onn.-\%$	**100**	**100**	**100**	**100**
	$alka$	**74**	5505	1256	2723

Taulukko 6: Ensimmäisen vertailun tulokset. Taulukossa on lihavoitu jokaisen funktion suhteen parhaat arvot.

f_1, f_4 ja f_5 optimoitaessa JVMY toimii vertailtavista menetelmistä selvästi huonoimmin. Vain funktiota f_3 optimoitaessa JAMYO:n toiminta on suunnilleen yhtä huonoa kuin JVMY:n. Vertailussa MYOSS on paras funktioiden f_3 sekä f_5 optimoinnissa ja BMS funktion f_1 optimoinnissa. Funktion f_4 kohdalla parhaan menetelmän valinta ei ole niin yksinkertaista, koska kohdefunktion arvon laskujen määrän keskiarvon suhteen paras menetelmä ei ole paras onnistumisprosentin suhteen.

Toisena JAMYO:n toimintaa verrataan funktioiden f_1, f_3, f_6 ja f_7 avulla API-menetelmään (katso luku 5.3) ja muurahaisyhdyskuntasysteemin jatkuvaan versioon (MYSJV, katso luku 5.5). Näissä vertailuissa funktiossa f_1 käytetään arvoja $n_1 = 3$, a_1 = -5,12 sekä b_1 = 5,12, funktiossa f_3 arvoja $n_3 = 2$, a_3 = -2,048 sekä b_3 = 2,047 ja funktiossa f_6 arvoja $n_6 = 5$, a_6 = -5,12 sekä b_6 = 5,12. Tällä kertaa yhdessä testiajossa suoritetaan 10000 kohdefunktion arvon laskua, jonka jälkeen menetelmän suoritus lopetetaan ja parhaan ratkaisun antama kohdefunktion arvo säilytetään muistissa. Jokaisella funktiolla tehdään 50 toisistaan riippumatonta testiajoa. Näiden testiajojen parhaiden ratkaisujen antamien kohdefunktioiden arvojen keskiarvot eri menetelmillä on esitetty taulukossa 7 ($kfka$).

Vertailuarvot API-menetelmästä on saatu, kun on käytetty parametreille seuraavia arvoja: $m = 20$ ja $\nu = 2$ [26]. MYSJV:ssa on vain yksi parametri

		f_1	f_3	f_6	f_7
JAMYO	$kfka$	$3{,}0 \cdot 10^{-88}$	0,49010	14,9296	0,70572
API	$kfka$	0,00	0,00	5,26	0,15
MYSJV	$kfka$	$3{,}6 \cdot 10^{-37}$	$1{,}6 \cdot 10^{-33}$	**4,9**	**0,0025**

Taulukko 7: Toisen vertailun tulokset. Taulukossa on lihavoitu jokaisen funktion suhteen parhaat arvot.

ja sille on käytetty arvoa $m = 100$ [28].

API-menetelmän ja MYSJV:n osalta kirjallisuudesta ei löydy informaatiota siitä minkälaisella tietokoneella tai millä ohjelmointikielellä menetelmiä on testattu.

Taulukosta 7 nähdään, että vertailtavista menetelmistä JAMYO on suunnilleen yhtä hyvä kuin muut funktiota f_1 optimoitaessa ja huonoin muita funktioita (f_3, f_6 ja f_7) optimoitaessa. Näiden muiden funktioiden optimoinnissa parhaiten toimii MYSJV. API-menetelmästä ei ole kirjallisuudessa [26] tarkempia arvoja kuin taulukossa 7 on ilmoitettu.

Kolmantena JAMYO:n toimintaa verrataan funktioiden f_8, f_9, f_{10} ja f_{15} avulla mukautuvaan muurahaisyhdyskunta-algoritmiin (MMYA, katso luku 5.7). Nyt onnistumiseksi lasketaan se, jos menetelmä löytää ratkaisun, jolla kohdefunktion arvo on riittävän suuri. Funktion f_8 tapauksessa kohdefunktion arvon pitää olla vähintään 78, funktion f_9 tapauksessa vähintään 9,8, funktion f_{10} tapauksessa vähintään 0,999 ja funktion f_{15} tapauksessa vähintään 9,99. Tällä kertaa yhdessä testiajossa suoritetaan korkeintaan 10000 kohdefunktion arvon laskua. Jokaisella funktiolla tehdään 10 toisistaan riippumatonta testiajoa.

Vertailuarvot MMYA:sta on saatu, kun on käytetty parametreille seuraavia arvoja: $m = 25$, $q_0 = 0{,}7$, $\xi = 0{,}2$, $\rho_{min} = 0{,}1$, $Q = 10$, $a = 10$, $b = 250$, $\epsilon = 0{,}001$ ja $T = 100$ [1].

MMYA:n testaamiseen on käytetty Pentium 4 (1,5 GHz) tietokonetta ja MATLAB 6.5 ohjelmistoa [1].

Taulukosta 8 nähdään selvästi, että jokaista vertailtavaa funktiota optimoitaessa JAMYO toimii huonommin kuin MMYA. Parhaaseen tulokseen JAMYO pääsee funktiota f_8 optimoitaessa. Tällöin onnistumisprosentti molemmilla menetelmillä on 100 ja kohdefunktion arvon laskujen määrän

		f_8	f_9	f_{10}	f_{15}
JAMYO	*onn.* – %	**100**	90	0	30
	alka	224	33	-	101
MMYA	*onn.* – %	**100**	**100**	**80**	**80**
	alka	**126**	**17**	**110**	**3**

Taulukko 8: Kolmannen vertailun tulokset. Taulukossa on lihavoitu jokaisen funktion suhteen parhaat arvot.

		f_3	f_6	f_{11}	f_{12}
JAMYO	*onn.* – %	0	0	**100**	70
	alka	-	-	**2788**	**38511**
PFKJ	*onn.* – %	**100**	**100**	**100**	**100**
	alka	**61769**	**239365**	35928	43044

Taulukko 9: Neljännen vertailun tulokset. Taulukossa on lihavoitu jokaisen funktion suhteen parhaat arvot.

keskiarvo on JAMYO:ssa vajaa 1,8-kertainen verrattuna MMYA:in.

Seuraavaksi JAMYO:n toimintaa verrataan funktioiden f_3, f_6, f_{11} ja f_{12} avulla paranneltuun feromonien kerääntymisjärjestelmään (PFKJ, katso luku 5.11). Näissä vertailuissa funktiossa f_3 käytetään arvoja $n_3 = 20$, a_3 = -2,048 sekä b_3 = 2,048 ja funktiossa f_6 arvoja $n_6 = 20$, a_6 = -3,12 sekä b_6 = 7,12. Tällä kertaa onnistumiseksi lasketaan se, jos menetelmä löytää ratkaisun, jolla kohdefunktion arvon ja kohdefunktion optimiarvon ero on korkeintaan $2 \cdot 10^{-5}$. Jokaisella funktiolla tehdään 20 toisistaan riippumatonta testiajoa ja yhdessä testiajossa suoritetaan korkeintaan 500000 kohdefunktion arvon laskua.

Vertailuarvot PFKJ:stä on saatu, kun on käytetty parametreille seuraavia arvoja: $m = 120$, $H = 100$, $\alpha = 32$, ρ = 0,2, ρ_{min} = 0,1 ja $\beta = 1$ paitsi funktiota f_6 optimoitaessa, jolloin β = 0,6 [33].

PFKJ:n testaamiseen on käytetty Pentium 4 (2,8 GHz) tietokonetta ja Java ohjelmointikieltä [33].

Taulukosta 9 huomataan, että kahdesta vertailtavasta menetelmästä PFKJ toimii huomattavasti paremmin funktioden f_3 ja f_6 optimoinnissa. JAMYO ei onnistu näiden funktioiden optimoinnissa kertaakaan. Näissä

20 testiajossa JAMYO saavuttaa funktion f_3 tapauksessa kohdefunktiolle keskiarvon 16,98 ja funktion f_6 tapauksessa keskiarvon 74,50. Näin ollen tämä menetelmä jää keskimäärin erittäin kauas optimista funktioita f_3 ja f_6 optimoitaessa. Funktiota f_{11} optimoitaessa JAMYO toimii sen sijaan huomattavasti paremmin kuin PFKJ. Myös funktion f_{12} tapauksessa JAMYO saavuttaa hieman paremman tuloksen kohdefunktion arvon laskujen määrän keskiarvon kannalta kuin PFKJ. Pitää kuitenkin huomata, että JAMYO:n onnistumisprosentti funktiota f_{12} optimoitaessa on vain 70, kun taas PFKJ:n onnistumisprosentti on 100.

Lopuksi JAMYO:n toimintaa verrataan funktioiden f_1, f_6, f_{13} ja f_{14} avulla ortogonaalin haun muurahaisyhdyskuntaoptimointiin (OHMYO, katso luku 5.12). Näissä vertailuissa funktiossa f_1 käytetään arvoja $n_1 = 4$, $a_1 = -100$ sekä $b_1 = 100$ ja funktiossa f_6 arvoja $n_6 = 4$, a_6 = -5,12 sekä b_6 = 5,12. Tällä kertaa yhdessä testiajossa suoritetaan 100000 kohdefunktion arvon laskua, jonka jälkeen menetelmän suoritus lopetetaan ja parhaan ratkaisun antama kohdefunktion arvo säilytetään muistissa. Jokaisella funktiolla tehdään 50 toisistaan riippumatonta testiajoa. Taulukoissa 10 ja 11 on esitetty näiden testiajojen parhaiden ratkaisujen antamien kohdefunktioiden arvojen keskiarvot ($kfka$) sekä paras näillä testiajoilla saavutettu kohdefunktion arvo (pkf).

Vertailuarvot OHMYO:sta on saatu, kun on käytetty parametreille seuraavia arvoja: $m = 100$, $N = 30$ (funktiot f_1, f_{13} ja f_{14}) tai 200 (funktio f_6), τ_0 = 0,001, δ = 0,9, χ = 0,1, q_0 = 0,3, ρ = 0,2, ja hakusäteiden r_j^i arvoksi asetetaan aluksi hakuvälin kymmesosa [35].

OHMYO:n osalta kirjallisuudesta ei löydy informaatiota siitä minkälaisella tietokoneella tai millä ohjelmointikielellä menetelmää on testattu.

Taulukoista 10 ja 11 huomataan, että kahdesta vertailtavasta menetelmästä OHMYO on kohdefunktion keskiarvon perusteella parempi jokaista vertailussa käytettävää funktiota optimoitaessa. Parhaan löydetyn kohdefunktion arvon perusteella JAMYO on kuitenkin parempi ainakin funktioiden f_1 ja f_{13} optimoinnissa. Funktion f_6 osalta ei voida varmasti sanoa kumpi vertailtavista menetelmistä on parempi parhaan löydetyn kohdefunktion arvon perusteella, koska ei tiedetä kuinka tarkka ortogonaalin haun muurahaisyhdyskuntaoptimoinnissa saatu arvo 0 on. Funktion f_{14} osalta on

		f_1	f_6
JAMYO	$kfka$	$2{,}3{\cdot}10^{-11}$	6,3478
	pkf	$\mathbf{3{,}2{\cdot}10^{-203}}$	$1{,}3{\cdot}10^{-202}$
OHMYO	$kfka$	$\mathbf{1{,}8{\cdot}10^{-88}}$	**0,1052**
	pkf	$9{,}0{\cdot}10^{-95}$	0

Taulukko 10: Viidennen vertailun tulokset funktioiden f_1 ja f_6 osalta. Taulukossa on lihavoitu jokaisen funktion suhteen parhaat arvot.

		f_{13}	f_{14}
JAMYO	$kfka$	0,2000	1,7381
	pkf	$\mathbf{1{,}3{\cdot}10^{-101}}$	$1{,}4{\cdot}10^{-6}$
OHMYO	$kfka$	$\mathbf{1{,}1{\cdot}10^{-61}}$	$\mathbf{2{,}6{\cdot}10^{-69}}$
	pkf	$2{,}6{\cdot}10^{-69}$	$\mathbf{2{,}7{\cdot}10^{-76}}$

Taulukko 11: Viidennen vertailun tulokset funktioiden f_{13} ja f_{14} osalta. Taulukossa on lihavoitu jokaisen funktion suhteen parhaat arvot.

kuitenkin selvää, että OHMYO on parempi vertailtavista menetelmistä myös parhaan löydetyn kohdefunktion arvon perusteella.

6.3 Yhteenveto

Edellä esitettyjen vertailujen perusteella näyttää siltä, että JAMYO toimii melko hyvin yksinkertaisempien, esimerkiksi konveksien funktioiden (katso liite A) optimoinnissa. Tällä menetelmällä näyttää kuitenkin olevan ongelmia löytää globaali optimi optimoitaessa niitä funktioita, joilla on olemassa vähintään yksi lokaali optimi, joka ei samalla ole globaali optimi.

JVMY vaikuttaa toimivan tässä luvussa vertailluista menetelmistä huonoiten funktiosta riippumatta. Myös JMYO:n toiminta vertailuissa on sen verran huonoa, että ainakaan tämän perusteella sen käyttöä ei voi suositella.

MYOSS näyttää toimivan verrattaen hyvin optimoitavasta funktiosta riippumatta. Sitä vastoin BMS:n toimivuus näyttää riippuvan suuresti optimoitavasta funktiosta. Vertailuissa tämä menetelmä toimi erittäin hyvin funktiota f_1 optimoitaessa, mutta muiden testifunktioiden optimoinnissa

menetelmä toimi melko huonosti.

API-menetelmä toimii vertailuissa jokaisen funktion optimoinnissa kohtuullisesti, mutta näyttää kuitenkin siltä, ettei se toimi parhaiden tässä luvussa vertailtujen menetelmien veroisesti. MYSJV näyttää vertailujen perusteella toimivan vielä hieman API-menetelmää paremmin optimoitavasta funktiosta riippumatta.

Vertailuista voidaan päätellä, että MMYA toimii funktiosta riippumatta erinomaisesti. Toisaalta nämä erittäin hyvät tulokset johtuvat osittain tämän menetelmän vertailuissa käytettävästä onnistumisen määritelmästä, joka takaa menetelmälle onnistumisen helpommin kuin muiden menetelmien vertailuissa käytetyt onnistumisen määritelmät.

Vertailujen perusteella PFKJ näyttää toimivan verrattaen melko hyvin optimoitaessa niitä funktioita, joilla on lokaaleja optimeja, jotka eivät samalla ole globaaleja optimeja. Yksinkertaisempien funktioiden optimoinnissa tämä menetelmä ei näytä toimivan kovinkaan hyvin.

OHMYO näyttää vertailujen perusteella toimivan hyvin optimoitavasta funktiosta riippumatta. Tämä menetelmä näyttäisikin olevan yksi parhaista tässä luvussa vertailluista menetelmistä.

Koska eri vertailuissa käytettävät kriteerit ovat erilaisia ja optimoitavat funktiot eivät ole samoja, ei varmoja johtopäätöksiä menetelmien paremmuudesta voi tehdä. Näiden vertailujen perusteella optimointiin voi kuitenkin suositella MYOSS:a, MMYA:a tai OHMYO:a, jos optimoitavasta funktiosta ei ole mitään ennakkotietoa.

A Yleisiä määritelmiä

1) **Globaali optimi**: Olkoon optimointitehtävän sallittu alue $S \subset \mathbf{R}^n$. Tällöin piste x^* on tehtävän globaali minimi (minimointitehtävän globaali optimi), jos

$$f(x^*) \leq f(x) \quad \forall x \in S.$$

Vastaavasti piste x^* on tehtävän globaali maksimi (maksimointitehtävän globaali optimi), jos

$$f(x^*) \geq f(x) \quad \forall x \in S.$$

2) **Lokaali optimi**: Olkoon optimointitehtävän sallittu alue $S \subset \mathbf{R}^n$. Tällöin piste x^* on tehtävän lokaali minimi (minimointitehtävän lokaali optimi), jos on olemassa $\delta > 0$ siten, että

$$f(x^*) \leq f(x) \quad \forall x \in S, \text{ joilla } ||x - x^*|| \leq \delta.$$

Vastaavasti piste x^* on tehtävän lokaali maksimi (maksimointitehtävän lokaali optimi), jos on olemassa $\delta > 0$ siten, että

$$f(x^*) \geq f(x) \quad \forall x \in S, \text{ joilla } ||x - x^*|| \leq \delta.$$

3) **Konveksi funktio**: Olkoon $f : S \to \mathbf{R}$, $S \subset \mathbf{R}^n$ ja $S \neq \phi$. Funktio f on konveksi funktio joukossa S, jos

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y) \quad \forall x, y \in S, \lambda \in (0, 1).$$

B Testaukseen tarvittava koodi

Tässä liitteessä esitellään JAMYO:n testausta varten tarvittava koodi funktiolle f_1, jossa $n_1 = 3$, $a_1 = -5$ ja $b_1 = 5$ (katso taulukot 3 ja 4) sekä menetelmään liittyvät parametrit ovat $m = 2$, $\xi = 0{,}85$, $q = 10^{-4}$ ja $k = 15$. Koodia täytyy tietyiltä osin hieman muuttaa, kun muita funktioita testataan tai parametrien arvoja muutetaan. Koodin antama tulos on muodossa {{menetelmän löytämä paras ratkaisu, kohdefunktion arvo saadulla parhaalla ratkaisulla}, kohdefunktion arvojen laskun lukumäärä}. Ohjelmointi on suoritettu Mathematican versiolla 6.0.

```
f[x_] := Sum[x[[i]]^2, {i,1, 3}];
om[lista_] :=
```

```
  Module[{t}, t = Table[lista[[j]][[-2]],
  {j, 1, Length[lista]}]; Accumulate[t]/Total[t]];
valinta[sl_] :=
  Flatten[Module[{u, val, i}, u = Sort[Append[om[ratk], sl]];
    i = Flatten[Position[u, sl]]; val = ratk[[i]]; val]];
var[l_, i_, lista_] := 0.85*Sum[(Abs[lista[[k]][[i]]-
  lista[[l]][[i]]])/(Length[lista[[l]]]-4) + 10^(-100),
  {k,1,Length[lista[[l]]]-2}];
ratk = RandomReal[{-5, 5}, {15, 3}];
For[j = 1, j < Length[ratk] + 1, j++,
 AppendTo[ratk[[j]], f[ratk[[j]]]]];
ratk = Sort[ratk, #1[[-1]] < #2[[-1]] &];
Sort[ratk, #1[[-1]] < #2[[-1]] &];
For[j = 1, j < Length[ratk] + 1, j++,
 AppendTo[ratk[[j]],
  1/(10^(-4)*15*Sqrt[2*Pi])
   e^(-((j - 1)^2/(2*(10^(-4))^2*15^2)))]];
For[j = 1,
 j < Length[ratk] + 1, j++, AppendTo[ratk[[j]], j]];
menetelmä[m_, n_] :=
  Module[{uusratk, valitut, apu, apu2, apu3, apu4, t = 0},
   While[Abs[ratk[[1]][[-3]]] > 10^(-4) && t < 10000,
    uusratk = {};
    apu = {};
    apu3 = Flatten[Table[{{i, -1}, {i, -2}},
      {i, 1, Length[ratk]}], 1];
    apu4 = Table[{-i}, {i, 1, m}];
    valitut = Table[valinta[RandomReal[]], {m}];
    For[j = 1, j < m + 1, j++, apu = {};
     For[l = 1, l < n + 1, l++,
      apu2 = RandomReal[
        NormalDistribution[valitut[[j]][[l]],
         var[valitut[[j]][[-1]], l, ratk]]];
      Which[apu2 < -5, apu2 = -5, apu2 > 5, apu2 = 5];
      AppendTo[apu, apu2]]; AppendTo[uusratk, apu]];
```

```
For[j = 1, j < Length[uusratk] + 1, j++,
 AppendTo[uusratk[[j]], f[uusratk[[j]]]]];
ratk = Delete[ratk, apu3];
For[j = 1, j < m + 1, j++, AppendTo[ratk, uusratk[[j]]]];
ratk = Sort[ratk, #1[[-1]] < #2[[-1]] &];
ratk = Delete[ratk, apu4];
For[j = 1, j < Length[ratk] + 1, j++,
 AppendTo[ratk[[j]],
  1/(10^-4*15*Sqrt[2*Pi])
   E^(-((j - 1)^2/(2*(10^(-4))^2*15^2)))]];
For[j = 1, j < Length[ratk] + 1, j++,
AppendTo[ratk[[j]], j]];
t = t + m]; {Delete[ratk[[1]], {{-1}, {-2}}], t}];
```

Kirjallisuutta

[1] Baojiang Z., Shiyong L.: Ant colony optimization algorithm and its application to Neuro-Fuzzy controller design. *Journal of Systems Engineering and Electronics*, 18 (2007), 603–610.

[2] Bilchev G., Parmee I. C.: The ant colony metaphor for searching continuous design spaces. *Proceedings of the AISB Workshop on Evolutionary Computation, Evolutionary Computing, University of Sheffield, UK, April 3–4, 1995*, ed. by T. C. Fogarty, 25–39. Springer-Verlag, Berlin 1995.

[3] Box G. E. P., Muller M. E.: A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29 (1958), 610–611.

[4] Bullnheimer B., Hartl R. F., Strauss C.: A new rank-based version of the ant system: A computational study. *Central European Journal for Operations Research and Economics*, 7 (1999), 25–38.

[5] Cerný V.: A thermodynamical approach to the traveling salesman problem. *Journal of Optimization Theory and Applications* 45 (1985), 41–51.

[6] Chen L., Shen J., Qin L., Fan J.: A method for solving optimization problem in continuous space using improved ant colony algorithm. *Data Mining and Knowledge Management, Chinese Academy of Sciences Symposium, CASDMKM 2004, Beijing, China, July 2004, Revised Papers*, ed. by Y. Shi, W. Xu, Z. Chen, 61–70. Springer, Berlin, 2004.

[7] Deneubourg J.-L., Aron S., Goss S., Pasteels J.-M.: The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behavior* 3 (1990), 159–168.

[8] Dorigo M.: *Optimization, learning and natural algorithms*. PhD Thesis, Dipartimento di Elettronica, Politecnico di Milano, Milano, 1992.

[9] Dorigo M., Gambardella L. M.: A study of some properties of Ant-Q. *Proceedings of PPSN-IV, Fourth International Conference on Parallel Problem Solving from Nature*, ed. by H. Voigt, W. Ebeling, I. Rechenberg, H. Schwefel, 656–665. Springer, Berlin, 1996.

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

[11] Dorigo M., Stützle T.: *Ant colony optimization.* The MIT Press, Cambridge, 2004.

[12] Dréo J., Siarry P.: Continuous interacting ant colony algorithm based on dense heterarchy. *Future Generation Computer Systems*, 20 (2004), 841–856.

[13] Glover F.: Tabu search—Part I. *ORSA Journal on Computing* 1 (1989), 190–206.

[14] Glover F.: Tabu search—Part II. *ORSA Journal on Computing* 2 (1990), 4–32.

[15] Glover F., Laguna M.: *Tabu search.* Kluwer Academic Publishers, Boston, 1997.

[16] Goldberg D. E.: *Genetic algorithms in search, optimization and machine learning.* Addison–Wesley, Reading (MA), 1989.

[17] Goss S., Aron S., Deneubourg J.-L., Pasteels J.-M.: Self-organized shortcuts in the Argentine ant. *Naturwissenschaften* 76 (1989), 579–581.

[18] Holland J.: *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[19] Kirkpatrick S., Gelatt C. D. Jr., Vecchi M. P.: Optimization by simulated annealing. *Science* 220 (1983), 671–680.

[20] Kong M., Tian P.: A direct application of ant colony optimization to function optimization problem in continuous domain. *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4–7, 2006, Proceedings*, ed. by M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle, 324–331. Springer, Berlin, 2006.

[21] Kong M., Tian P.: Introducing a binary ant colony optimization. *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4–7, 2006, Proceedings*, ed. by M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle, 444–451. Springer, Berlin, 2006.

[22] Leguizamón G., Coello Coello C. A.: Boundary search for constrained numerical optimization problems in ACO algorithms. *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4–7, 2006, Proceedings*, ed. by M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle, 108–119. Springer, Berlin, 2006.

[23] Lin Y., Cai H. C., Xiao J., Zhang J.: Pseudo parallel ant colony optimization for continuous functions. *Advances in Natural Computation, 3rd International Conference, ICNC 2007, Xi´an, China, September 2007, Proceedings*, 494–500, Springer, Berlin, 2007.

[24] Liu J. S.: *Monte Carlo strategies in scientific computing*. Springer-Verlag, New York, 2001.

[25] Mathur M., Karale S. B., Priye S., Jayaraman V. K., Kulkarni B. D.: Ant colony approach to continuous function optimization. *Ind. Eng. Chem. Res.*, 39 (2000), 3814–3822.

[26] Monmarché N., Venturini G., Slimane M.: On how Pachycondyla apicalis ants suggest a new search algorithm. *Future Generation Computer Systems*, 16 (2000), 937–946.

[27] Pellegrini P., Favaretto D., Moretti E.: On MAX-MIN ant system's parameters. *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4–7, 2006, Proceedings*, ed. by M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle, 203–214. Springer, Berlin, 2006.

[28] Pourtakdoust S. H., Nobahari H.: An extension of ant colony system to continuous optimization problems. *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004, Brussels, Belgium, September 5–8, 2004, Proceedings*, ed. by M. Dorigo, M. Birat-

tari, C. Blum, L. M. Gambardella, F. Mondada, T. Stützle, 294–301. Springer, Berlin, 2004.

[29] Pursiheimo U.: *Optimointialgoritmit*, Opintomoniste, Turun yliopisto, 2006.

[30] Socha K., Dorigo M.: Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185 (2008), 1155–1173.

[31] Stützle T.: *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. Infix, Sankt Augustin, 1999.

[32] Stützle T., Hoos H. H.: MAX-MIN ant system. *Future Generation Computer Systems*, 16 (2000), 889–914.

[33] Tsutsui S.: An enhanced aggregation pheromone system for real-parameter optimization in the ACO metaphor. *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4–7, 2006, Proceedings*, ed. by M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle, 60–71. Springer, Berlin, 2006.

[34] Yao X., Liu Y.,Lin G.: Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3 (1999), 82–102.

[35] Zhang J., Chen W.-N., Tan X.: An orthogonal search embedded ant colony optimization approach to continuous function optimization. *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4–7, 2006, Proceedings*, ed. by M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle, 372–379. Springer, Berlin, 2006.