Simuloitu jäähdytys

Oona Liuhonen

LuK-tutkielma
Marraskuu 2024

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

Oona Liuhonen: Simuloitu jäähdytys
LuK-tutkielma, 23 s., 3 liites.
Sovellettu matematiikka
Marraskuu 2024

Tässä tutkielmassa esitetään simuloiduksi jäähdytykseksi kutsutun optimointimenetelmän teoreettinen tausta sekä menetelmän yleinen toimintaperiaate. Simuloitu jäähdytys on metaheuristiikka. Tämä tarkoittaa sellaista heuristista menetelmää, jossa ohjeet menetelmän implementointiin on annettu niin yleisesti, että sitä voidaan soveltaa useiden erilaisten tehtävien ratkaisemiseen. Lisäksi simuloitu jäähdytys perustuu reaalimaailman ilmiöön, jossa kuuman metallipalan jäähdytysnopeus vaikuttaa jäähtyneen metallipalan taipuvuuteen ja haurauteen.

Työn alussa perehdytään ensin lyhyesti metaheuristiikkoihin ja kuvaillaan tarkemmin simuloidun jäähdytyksen yhteyttä metallin lämpökäsittelyyn. Lisäksi tarkastellaan lyhyesti Metropolisin algoritmia, joka on vahvasti simuloidun jäähdytyksen kehittämiseen vaikuttanut menetelmä. Simuloidun jäähdytyksen toiminnasta esitetään tarkemmin siinä käytettävät parametrit, jonka jälkeen menetelmän algoritmi esitetään sekä sanallisesti että pseudokoodina. Lisäksi menetelmän konvergenssituloksia kommentoidaan lyhyesti.

Simuloidun jäähdytyksen toimintaa on työssä havainnollistettu implementoimalla menetelmä toimivaksi Java-ohjelmaksi ja testaamalla sitä sudoku-tehtävien ratkaisemiseen. Lisäksi tutkielmassa tarkastellaan Java-ohjelmalla sudoku-tehtäville saatuja numeerisia tuloksia. Simuloidun jäähdytyksen implementointi sudokun tapauksessa esitetään mallintamalla sudoku optimointitehtävänä, ja tulkitsemalla mitä menetelmän eri parametrit tarkoittavat sudokun tapauksessa. Tämän perusteella on luotu Java-ohjelma, jonka avulla on testattu menetelmän toimintaa eri parametrivalinnoilla. Lopuksi pohditaan, mitkä eri asiat ovat voineet vaikuttaa saatuihin numeerisiin tuloksiin, ja päätellään, millaisia kokeita ohjelmalle kannattaisi suorittaa jatkossa.

Asiasanat: simuloitu jäähdytys, metaheuristiikka, Boltzmannin jakauma, sudoku.

Sisällys

1 Johdanto

Parhaan mahdollisen ratkaisun löytäminen nopeasti ja luotettavasti on tärkeää monessa eri tilanteessa. Esimerkiksi ravintola voisi haluta järjestää aukioloaikansa ja työntekijöidensä työtunnit vastaamaan mahdollisimman hyvin päivän ja viikon aikana saapuvien asiakkaiden määriä. Tietokonesirujen valmistajat voivat puolestaan olla kiinnostuneita säästämään materiaalikustannuksissa ja haluavat täten sijoittaa piirilevyn komponentit mahdollisimman tehokkaasti. Matemaattinen optimointi tarjoaa työkaluja ja keinoja tämänkaltaisten ongelmien ratkaisemiseen. Tärkeimmät näistä työkaluista ovat tavat mallintaa tarkasteltava ongelma matemaattisena optimointitehtävänä, sekä kaikki erilaisten optimointitehtävien ratkaisemiseen kehitetyt menetelmät.

Mallinnettaessa ongelmaa matemaattiseksi optimointitehtäväksi tuntemattomat eli selvitettävät asiat esitetään päätösmuuttujilla. Tavoitteena on löytää päätösmuuttujille mahdollisimman hyvät arvot. Jotta erilaiset päätösmuuttujien arvot voidaan laittaa paremmuusjärjestykseen, niin optimointitehtävään tarvitaan kohdefunktio. Kohdefunktiota joko minimoidaan tai maksimoidaan, ja täten sen avulla etsitään tehtävälle paras ratkaisu eli globaali optimi. Malliin voidaan myös sisällyttää rajoitteita, joiden avulla voidaan vaatia ratkaisuilta tiettyjä ominaisuuksia. Näin voidaan rajoittua esimerkiksi vain positiivisiin päätösmuuttujien arvoihin, jos tarkastelussa oleva muuttuja kuvaa vaikka ostetun teräksen määrää.

Optimointitehtävä voidaan luokitella sen mukaan, millaisia tehtävän rajoitteet, päätösmuuttujat sekä kohdefunktio ovat, ja tätä kautta tehtävän ratkaisemiseen voidaan helpommin valita sopiva optimointialgoritmi. Mahdollisimman tarkka optimointitehtävän luokittelu on tehokkaan optimointimenetelmän valinnassa erittäin tärkeää, sillä monet menetelmät hyödyntävät toiminnassaan tietynlaista tehtävän rakennetta ja vaativat näin ollen toimiakseen tiettyjä ominaisuuksia tehtävässä esiintyviltä funktioilta ja päätösmuuttujilta. Luokittelu on myös tärkeää siksi, että vaikka on olemassa yleismenetelmiä, jotka toimivat lähes millaiselle tehtävälle tahansa, ne ovat yleensä melko huonoja. Luokittelussa huomioidaan tyypillisesti esimerkiksi se, että ovatko päätösmuuttujat jatkuvia vai diskreettejä, onko kohdefunktio lineaarinen, konveksi vai epäkonveksi ja onko optimointitehtävässä rajoitteita, ja jos on, millaisia nämä rajoitteet ovat. Optimointitehtävässä voi myös olla samanaikaisesti sekä jatkuvia että diskreettejä päätösmuuttujia, jolloin tehtävää kutsutaan sekalukutehtäväksi.

Yksi erittäin tutkittu optimointitehtäväluokka on lineaariset optimointitehtävät. Jotta tehtävä voi kuulua tähän luokkaan, sen päätösmuuttujien pitää olla positiivisia ja jatkuvia, ja sen kohdefunktion ja kaikkien rajoitefunktioiden oltava lineaarisia. Lineaarisille tehtäville erittäin tehokas ratkaisutapa on Simplex-menetelmä, joka löytää luotettavasti tehtävän parhaan mahdollisen ratkaisun [12]. Simplex-menetelmä on esimerkki niin sanotusta tarkasta algoritmista, jolla voidaan varmistua siitä, että tehtävälle on löydetty paras mahdollinen ratkaisu eli globaali optimi.

On melko yleistä, että optimointitehtävälle ei tunneta tarkkaa algoritmia, joka mahdollistaisi optimin löytämisen polynomiaalisessa ajassa [4]. Esimerkiksi NP-vaikeille tehtäville tällaista algoritmia ei tunneta, ja lisäksi kysymys siitä, onko kyseisille tehtäville tällaista algoritmia edes olemassa, on edelleen avoin [4]. Esimerk-

kejä NP-vaikeista tehtävistä ovat kauppamatkustajan ongelma, kvadraattinen sijoitustehtävä (*quadratic assignment problem*) ja joukon peitto-ongelma (*set covering problem*) [18].

Algoritmin toimiminen polynomiaalisessa ajassa nähdään usein rajapyykkinä sille, voiko sitä käytännössä soveltaa ongelmien ratkaisemiseen. Syytä tähän usein havainnollistetaan esimerkillä siitä, kuinka kauan aikaa eksponentiaalisessa ajassa toimivat algoritmit voivat vaatia. Oletetaan esimerkiksi, että käytössä on tietokone, joka pystyy suorittamaan sata miljardia laskutoimitusta sekunnissa, ja että algoritmin vaatimien laskutoimitusten määrä voidaan määrittää kaavalla 3^n, jossa n on optimointitehtävän päätösmuuttujen lukumäärään. Jos $n = 20$ tehtävän laskemiseen kuluu alle sekunti, ja jos $n = 30$ siihen kuluu noin 34 minuuttia ja 19 sekuntia. Kuitenkin jos $n = 50$ aikaa kuluu jopa 227644 vuotta. Näin ollen jo melko pientä tehtävää on käytännössä mahdotonta ratkaista kyseisellä algoritmilla.

Tästä syystä käytetään usein heuristiikkoja, joissa ei tavoitella parhaan ratkaisun varmistamista, vaan sen sijaan pyritään löytämään jokin riittävän hyvä ratkaisu nopeasti ja säästämään laskenta-ajassa. Tällainen riittävän hyvä ratkaisu voi olla esimerkiksi lokaali eli paikallinen optimi. Heuristiikat ovat usein erittäin tehtäväkohtaisia, vaikka monen heuristiikan voi ajatella perustuvan samoihin lähestymistapoihin. Yleisesti ottaen heuristiikoissa sovelletaan jotain nopeaa ja jollain tavalla systemaattista tapaa arvioida kohdefunktion arvoa, ja ratkaisua pyritään parantamaan askeleittain. Esimerkiksi jos kohdefunktio mittaa kahden kaupungin välillä ajamiseen kuluvaa aikaa, niin aikaa voidaan arvioida kaupunkien välisellä etäisyydellä. Kuitenkin heuristiikat ovat yleensä keskenään hyvinkin erilaisia, ei ainoastaan sen suhteen, millaisia tehtäviä voidaan ratkaista, mutta myös käytettävän kohdefunktion arviointitavan ja saadun ratkaisun tarkkuuden suhteen. Judea Pearl tarjosi havainnollistavan esimerkin siitä, miten monen arjen päätöksen voi ajatella olevan heuristinen: verkkomelonin kypsyyttä voi testata painamalla kohtaa, josta meloni on ollut kiinnittynyt kasviin, ja sen jälkeen haistamalla kyseistä kohtaa. [15]

Metaheuristiikat ovat tehtävästä riippumattomia optimointialgoritmin kehyksiä, jotka eivät itsessään tarjoa valmista optimointialgoritmia, vaan sen sijaan tarjoavat ohjeita tällaisen algoritmin kehittämiseen. Metaheuristiikan tarkemmat yksityiskohdat pitää suunnitella aina ratkaistavan tehtävän kohdalla erikseen, ja usein myös menetelmän implementointi on toteutettava itse. Metaheuristiikan toteutuksen suunnittelijan on muun muassa päätettävä, miten eri parametrit tulkitaan, miten erilaiset keskeiset toimenpiteet metaheuristiikassa suoritetaan ja miten ratkaisut esitetään. Metaheuristiikat, kuten simuloitu jäähdytys, usein soveltavat samanaikaisesti sekä iteratiivisia lokaalin haun menetelmiä että globaalin haun strategiaa lokaalien ääriarvopisteiden välttämiseen [14]. Muita yleisiä metaheuristiikkoja ovat tabuhaku, VNS (*Variable Neighborhood Search*) ja geneettiset algoritmit [14, 18].

Eräs tärkeä menetelmäluokka on niin sanotut hybridimetaheuristiikat, jotka eivät seuraa vain yhtä tiettyä metaheuristista menetelmää, vaan ennemminkin yhdistelevät osia eri metaheuristiikoista tai muista optimointimenetelmistä. Itse asiassa on melko yleistä, että kun metaheuristiikkoja hyödynnetään käytännön ongelmien ratkaisemiseen, käytössä on hybridimetaheuristiikka. Syy tähän on se, että monet näistä hybridimenetelmistä ovat usein tehokkaampia kuin tavalliset metaheuristiikat, koska esimerkiksi tehtävän kohdefunktiosta tai sallitusta alueesta riippuen jot-

kin menetelmät suoriutuvat riittävän hyvän ratkaisun löytämisestä nopeammin kuin toiset. Metaheuristiikkojen soveltaminen voikin joskus olla enemmän taidetta kuin tiedettä, ja hybridiversiot tarjoavat usein mahdollisuuksia löytää ratkaisuja entistä tehokkaammin. [16]

Tässä työssä käsitellään simuloitua jäähdytystä, joka on yksi yleisimmistä sekä tutkituimmista metaheuristiikoista ja jonka tutkimukseen liittyy paljon sekä kokeellista että teoreettista tietoa [14]. Simuloituun jäähdytykseen perustuva metaheuristiikka on esitelty ensimmäisen kerran IBM:n tutkijoiden Kirkpatrick, Gelatt ja Vecchi työssä [8] vuonna 1983. Menetelmää on vuosien varrella sovellettu esimerkiksi amerikkalaisen pesäpalloturnauksen aikataulun suunnittelussa (*traveling tournament problem*) [2], selkäreppuongelman ratkaisemisessa [1] ja kauppamatkustajan ongelman reitin etsimisessä (*traveling salesman problem*) [18].

Monien muiden metaheuristiikkojen tavoin simuloitu jäähdytys perustuu jonkin reaalimaailman ilmiön jäljittelyyn matemaattisesti. Simuloidun jäähdytyksen tapauksessa jäljiteltävä ilmiö on metallin lämpökäsittely, erityisesti päästäminen. Lämpökäsittelyssä kuuman metallipalan jäähdytysnopeus vaikuttaa paljon metallipalan taipuvuuteen ja haurauteen palan jäähdyttyä. Korkeassa lämpötilassa hiukkaset liikkuvat vapaammin eri energiatasojen välillä, ja lämpötilan laskiessa hiukkasten liike vähenee ja sijainti vakiintuu. Palan nopea jäähdyttäminen voi merkitä sitä, että kappale vakiintuu johonkin korkeassa lämpötilassa saavutettuun epävakaaseen energiatasoon luoden hauraan lopputuloksen. Tätä kutsutaan karkaisuksi. Sen sijaan päästämisessä lämpötilan annetaan laskea riittävän hitaasti, jolloin mahdollisten energiatasojen joukko konvergoi kohti globaalia minimiä luoden vakaamman ja vähemmän hauraan lopputuloksen.

Työssä esitellään ensin lyhyesti simuloidun jäähdytyksen teoreettinen tausta. Tiivistettynä simuloidussa jäähdytyksessä jäljitellään matemaattisesti metallin lämpökäsittelyä siinä mielessä, miten lämpötilan nostaminen ja laskeminen vaikuttaa todennäköisyyteen olla eri energiatasoissa tai simuloidun jäähdytyksen tapauksessa siirtyä ratkaisuavaruudessa eri iteraatiopisteestä toiseen. Tämän jälkeen perehdytään tarkemmin menetelmän yleiseen algoritmiin. Ideana menetelmässä on vertailla iteratiivisesti kohdefunktion arvoa eri pisteissä lämpötilaparametria hyödyntäen. Algoritmilla on muistissa jokin iteraatiopiste, jonka kohdefunktion arvoa verrataan toisen pisteen, niin sanotun naapuripisteen, kohdefunktion arvoon. Jos naapuripisteen kohdefunktion arvo on parempi kuin nykyisen iteraatiopisteen, niin se hyväksytään uudeksi iteraatiopisteeksi, ja jos se on huonompi, niin se hyväksytään tietyllä lämpötilaparametrin arvosta riippuvalla todennäköisyydellä. Asian voi tiivistää niin, että huonommat naapuripisteet hyväksytään todennäköisemmin, jos lämpötila on korkea tai jos piste on vain lievästi nykyistä iteraatiopistettä huonompi. Simuloidun jäähdytyksen aikana lämpötilaa lasketaan pikkuhiljaa ja algoritmi etenee kohti jotakin *ääriarvoa* eli lokaalia tai globaalia optimia. Todennäköisyys paeta ääriarvosta hyväksymällä jokin nykyistä iteraatiopistettä huonompi piste muuttuu koko ajan pienemmäksi ja pienemmäksi. Menetelmän suoritus lopetetaan, kun jokin sen lopetusehdoista täyttyy, mikä usein tapahtuu, kun algoritmi on saavuttanut jonkin ääriarvon ja ei enää kykene siirtymään kohdefunktion arvoa huonontaviin pisteisiin. Työssä käydään myös läpi, miten simuloitua jäähdytystä voidaan käyttää sudokun ratkaisemiseen, ja lopuksi esitellään sudoku-tehtäville saatuja numeerisia tuloksia.

Työn rakenne on seuraava. Luvussa 2 tutustutaan simuloituun jäähdytykseen. Ensin aliluvussa 2.1 käydään läpi simuloidun jäähdytyksen teoreettista taustaa Boltzmannin jakauman ja Metropolisin algoritmin kautta. Aliluvussa 2.2 kuvaillaan simuloidun jäähdytyksen perusidea ja toiminta sekä sanallisesti että havainnollistamalla sitä pseudokoodilla. Aliluvussa 2.3 esitetään joitakin simuloidun jäähdytyksen ominaisuuksia, ja lopuksi aliluvussa 2.4 tarjotaan ohjeita menetelmän implementointiin. Luvussa 3 siirrytään tarkastelemaan sudoku-tehtävien ratkaisemista simuloidulla jäähdytyksellä. Aliluvussa 3.1 käydään läpi menetelmän implementoinnissa tehdyt valinnat ja tulkinnat, ja taas aliluvussa 3.2 esitetään parametrivalinnat, joilla sudokujen numeeriset tulokset on saatu. Aliluvussa 3.3 tarkastellaan saatuja numeerisia tuloksia ja pohditaan hieman, mitä johtopäätöksiä saaduista tuloksista voi tehdä. Lopuksi luvussa 4 on työn yhteenveto.

2 Simuloitu jäähdytys

2.1 Teoreettinen tausta

Simuloidun jäähdytyksen taustalla on Metropolisin algoritmi. Siinä missä simuloidulla jäähdytyksellä jäljitellään matemaattisesti metallin päästämistä, niin Metropolisin algoritmissa ennemminkin jäljitellään termodynaamista järjestelmää jossakin tietyssä lämpötilassa [18]. Simuloidussa jäähdytyksessä käytetään Metropolisin algoritmin valintakriteeriä, joka taas soveltaa tilastollisessa mekaniikassa tärkeää Boltzmannin jakaumaa [10, 18]. Boltzmannin jakauman tai Metropolisin algoritmin tunteminen ei ole välttämätöntä simuloidun jäähdytyksen ymmärtämiseen, mutta ne ovat kuitenkin tärkeitä taustatietoja.

2.1.1 Boltzmannin jakauma

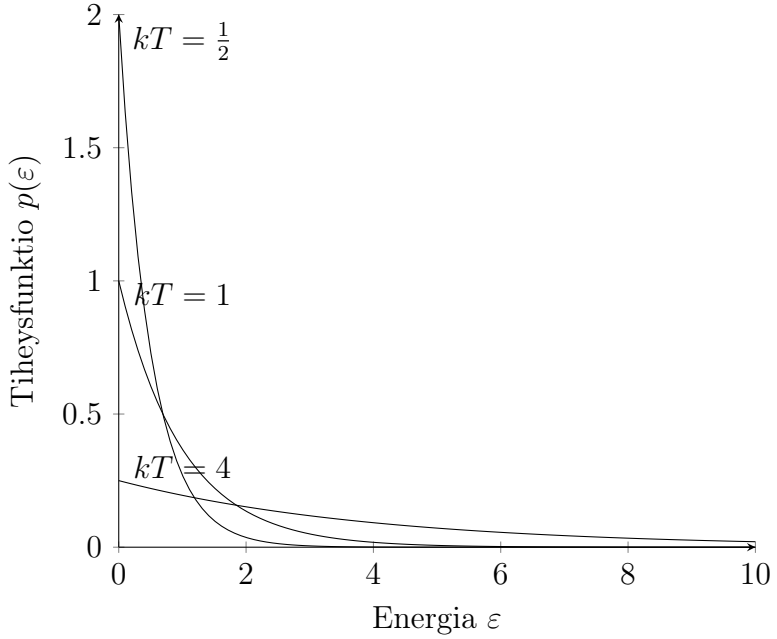
Boltzmannin jakauma on nimetty Ludwig Boltzmannin mukaan, joka alun perin formuloi jakauman vuonna 1868 [3], ja jakaumaa käytetään erityisesti tilastollisessa mekaniikassa ja termodynamiikassa. Boltzmannin jakaumassa todennäköisyys, että järjestelmä on jossain tilassa, voidaan määrittää tilan energian ja lämpötilan funktiona. Eli tarkemmin ilmaistuna

$$p_i = \frac{e^{\frac{-\varepsilon_i}{kT}}}{\sum_i e^{\frac{-\varepsilon_i}{kT}}},$$

missä p_i viittaa todennäköisyyteen olla tilassa i, ε_i on energia tilassa i, k on Boltzmannin vakio ja T on lämpötila [9]. Saman voi ilmaista myös yksinkertaisemmin käyttämällä verrannollisuussymbolia $\propto$, jolloin voidaan kirjoittaa

$$p_i \propto e^{\frac{-\varepsilon_i}{kT}}.$$

Havainnollistetaan seuraavaksi Boltzmannin jakauman käyttäytymistä. Tätä varten luodaan jakauman pohjalta jatkuva tiheysfunktio, jossa lämpötila T on oletettu vakioksi. Jos tilojen lukumäärä on tarpeeksi suuri, niin tarkastelun vaihtaminen



Kuva 1: Tiheysfunktio p energian ε kuvaajana eri lämpötilan T arvoilla.

diskreetistä jatkuvaksi ei ole ongelma. Tiheysfunktioksi saadaan

$$p(\varepsilon) = \frac{1}{Q} e^{\frac{-\varepsilon}{kT}}, \tag{1}$$

jossa $Q > 0$ on yhtälön standardointiin käytetty vakio. Ratkaisemalla yhtälö

$$1 = \int_0^\infty \frac{1}{Q} e^{\frac{-\varepsilon}{kT}} \, d\varepsilon$$

vakion Q arvoksi saadaan kT. Tällöin funktio (1) voidaan kirjoittaa muodossa

$$p(\varepsilon) = \frac{e^{\frac{-\varepsilon}{kT}}}{kT}, \tag{2}$$

jossa $T > 0$. Kuvaan 1 on piirretty funktion (2) käyriä parametrin T eri arvoilla. Kuvasta voi huomata, miten todennäköisyys olla energiaa ε vastaavassa tilassa kasautuu sitä lähemmäs nollaa, mitä pienempi lämpötila T on. Huomion arvoista on myös se, että kaikki esitetyt parametrin T arvot ovat kuvassa 1 melko suuria, koska Boltzmannin vakion arvo on hyvin pieni (k on noin $1.380649 \cdot 10^{-23} \frac{J}{K}$).

2.1.2 Metropolisin algoritmi

Metropolisin algoritmi on menetelmä, jonka avulla voidaan generoida joukko toisistaan riippuvia lukuja tai pisteitä jostain tietystä jakaumasta. Menetelmä on esitelty alun perin vuonna 1953 tiedejulkaisussa [10], jossa käsiteltiin kaasun tilayhtälöiden kaltaisten ominaisuuksien arviointia muunnellun Monte Carlo -integraation avulla. Tässä alkuperäisessä versiossa pistejoukko generoitiin Boltzmannin jakaumasta

eikä mielivaltaisesti valitusta tiheys- tai pistetodennäköisyysfunktiosta. Boltzmannin jakaumaan rajoittumaton versio Metropolisin algoritmista julkaistiin Hastingsin artikkelissa [6] vuonna 1970. Hastingsin version toiminta perustuu äärellisiin Markovin ketjuihin, ja sen avulla voidaan diskreeteissä tapauksissa generoida yleisemmin luku- tai pistejoukkoja, esimerkiksi jonkin todennäköisyysjakauman pohjalta tai Markovin ketjuna havainnollistettavan tilanteen perusteella. Hastingsin artikkelissa myös käsiteltiin tapoja arvioida muun muassa luku- tai vektorijoukon odotusarvoa ja varianssia. Metropolisin algoritmia kutsutaankin usein Metropolisin ja Hastingsin algoritmiksi.

Seuraavaksi esitetään, miten alkuperäisessä Metropolisin algoritmissa generoidaan pistejoukkoja Boltzmannin jakaumasta. Esitystapa pohjautuu tiedejulkaisun *The Journal of Chemical Physics* artikkeliin [10]. Oletetaan, että järjestelmänä on 2-ulotteinen neliömäinen taso, jossa on n kappaletta 2-ulotteisia pisteitä, ja että taso on jaksollisesti aseteltu toisten identtisten neliötasojen viereen. Merkitään tason pisteitä $\mathbf{v}^1,\mathbf{v}^2,\ldots,\mathbf{v}^n$, jossa jokainen $\mathbf{v}^i = (v_x^i,v_y^i)^{\mathrm{T}}$, ja oletetaan että kunkin pisteen x- ja y-komponenttien arvo tunnetaan. Tällöin voidaan määrittää järjestelmän potentiaalienergia E_p kaavasta

$$E_p = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} V(d_{ij}), \tag{3}$$

jossa V määrittää kahden hiukkasen (eli pisteen) välisen potentiaalienergian niiden etäisyyden perusteella ja d_{ij} on pisteiden $\mathbf{v}^i$ ja $\mathbf{v}^j$ välinen minimietäisyys. Artikkelissa [10] edellä mainittuihin pisteisiin viitattiin hiukkasina, joka johtuu siitä, että iso osa artikkelin teoreettisesta taustasta on peräisin termodynamiikasta. Tämä myös selittää sen, mihin potentiaalienergian kaava (3) perustuu.

Algoritmi alustetaan asettamalla kullekin joukon $\{\mathbf{v}^1,\mathbf{v}^2,...,\mathbf{v}^n\}$ pisteelle jokin sijainti tarkasteltavalta tasolta. Tämän jälkeen pisteet käydään yksi kerrallaan järjestyksessä läpi, ja niille asetetaan uusi sijainti säännöillä

$$v_x^i \to v_x^i + \alpha\xi_1 \tag{4}$$

$$v_y^i \to v_y^i + \alpha\xi_2. \tag{5}$$

Yllä olevissa kaavoissa α on suurin sallittu askelpituus ja ξ_1 sekä ξ_2 ovat tasaisesti jakautuneita satunnaislukuja väliltä $[-1,1]$. Jos piste sijoitetaan kohtaan, joka on neliötason ulkopuolella, niin tasojen jaksollisuuden myötä sen voidaan olettaa siirtyvän viereiselle tasolle. Käytännössä tämä tarkoittaa sitä, että esimerkiksi jos pisteelle asetetaan uusi sijainti, joka menee neliötason ulkopuolelle tason oikealta puolelta, niin piste palaa takaisin neliötasolle sen vasemmalta puolelta. Kun yksittäinen piste on siirretty uuteen sijaintiin, määritellään siirron aiheuttama potentiaalienergian muutos ΔE_p. Jos $\Delta E_p < 0$, niin pisteen siirto hyväksytään, ja jos taas $\Delta E_p > 0$, niin siirto hyväksytään todennäköisyydellä $\exp\frac{-\Delta E_p}{kT}$, eli käyttäen Boltzmannin jakaumaa. Jos siirtoa ei hyväksytä, niin piste palautetaan takaisin alkuperäiselle paikalleen. Tätä siirtojen hyväksymiseen tai hylkäämiseen käytettävää sääntöä kutsutaan *Metropolisin valintakriteeriksi* [18]. Tämän jälkeen valitaan järjestyksessä seuraava piste, ja siirretään se uuteen sijaintiin kaavoja (4) ja (5) käyttäen. Kun

kaikki pisteet on käyty kertaalleen läpi, niin yksi kierros algoritmia on suoritettu, ja algoritmin suoritusta voidaan jatkaa palaamalla alkuun.

Suorittamalla edellä kuvattua menetelmää riittävän monta kierrosta, joukon $\{\mathbf{v}^1,\mathbf{v}^2,...,\mathbf{v}^n\}$ pisteet jäljittelevät Boltzmannin todennäköisyysjakaumaa $\exp\frac{-\Delta E_p}{kT}$. Artikkelissa [10] ei kuitenkaan tarjota tarkkaa ohjeistusta sille, kuinka monta kierrosta menetelmää tulisi suorittaa. Kierrosten lukumäärää voi kuitenkin arvioida, jolloin on otettava huomioon, mihin tarkoitukseen menetelmää sovelletaan.

Kutsutaan pistejoukon $\{\mathbf{v}^1,\mathbf{v}^2,...,\mathbf{v}^n\}$ pisteiden sijaintien yhdistelmää *tilaksi*. Tilaa voidaan merkitä $n \times 2$ –matriisina

$$\begin{pmatrix} v_x^1 & v_y^1 \\ v_x^2 & v_y^2 \\ \vdots & \vdots \\ v_x^n & v_y^n \end{pmatrix},$$

ja olkoon kaikkien mahdollisten tilojen joukko S. Eräs tärkeä ominaisuus metropolisin algoritmille on se, että kustakin joukon S tilasta voidaan algoritmin suorittamisen myötä päästä mihin tahansa toiseen joukon S tilaan. Ei ole välttämätöntä, että yhdestä tilasta on mahdollista siirtyä toiseen tilaan soveltamalla kaavoja (4) ja (5) vain kerran, vaan että tilojen välillä siirtyminen on mahdollista, jos pisteitä siirretään jokin riittävä äärellinen määrä kertoja. Yllä esitetyn algoritmin toiminnassa on melko ilmeistä, että siirtyminen yhdestä tilasta toiseen on harvoin mahdollista yhdessä kierroksessa ja että peräkkäisten kierrosten tilat korreloivat vahvasti keskenään.

2.2 Perusidea ja algoritmi

Tässä aliluvussa esitetään simuloidun jäähdytyksen algoritmi sekä sanallisesti että pseudokoodina. Tätä varten määritellään ensin *diskreetti ratkaisuavaruus* Ω, joka on joukon $\mathbb{Z}^n$ osajoukko. Yleisesti ottaen simuloidussa jäähdytyksessä ratkaisuavaruuden Ω ei tarvitse olla diskreetti, vaan se voi olla myös jatkuva ja koostua pisteistä, jotka kuuluvat joukon $\mathbb{R}^n$ konveksiin osajoukkoon. Jatkuvan tapauksen algoritmi eroaa kuitenkin hieman diskreetistä tapauksesta. Lisäksi määritellään kohdefunktio $f : \Omega \to \mathbb{R}$, jolloin ratkaistava optimointitehtävä voidaan esittää muodossa

$$\begin{aligned} \min \quad & f(\boldsymbol{\omega}) \\ \text{s.t.} \quad & \boldsymbol{\omega} \in \Omega. \end{aligned}$$

Simuloitua jäähdytystä voi soveltaa myös monitavoitteisiin optimointitehtäviin, mutta tässä tutkielmassa keskitytään vain yksitavoitteisiin ongelmiin.

Simuloidun jäähdytyksen suorittamista varten tarvitaan myös relaatio $N : \Omega \to \Omega$, jota kutsutaan *naapurifunktioksi*. Naapurifunktion tehtävänä on kuvata ratkaisuavaruuden Ω piste joksikin toiseksi ratkaisuavaruuden pisteeksi. Vaikka relaatiota $N(\boldsymbol{\omega})$ kutsutaan naapurifunktioksi, niin se ei oikeastaan ole funktio. Tämä johtuu siitä, että naapurifunktion toiminta perustuu usein todennäköisyyksiin, ja siitä, ettei se ole bijektio $\Omega \to \Omega$. Esimerkiksi $N(\boldsymbol{\omega}')$ voisi antaa tulokseksi $\boldsymbol{\omega}^1$ todennäköisyydellä $p \in (0,1)$ ja $\boldsymbol{\omega}^2$ todennäköisyydellä $1 - p$, missä $\boldsymbol{\omega}^1 \neq \boldsymbol{\omega}^2$.

Simuloidussa jäähdytyksessä tarvitaan myös lämpötilaparametriä $t_k \in \mathbb{R}$, jossa $k \in \{0, 1, 2, \ldots\}$ on algoritmin ulompi iteraatiokierrosindeksi. Lämpötilajono $\{t_k\}$ määritellään usein käyttäen valittua alkulämpötilaa t_0 ja jäähdytysfunktiota $\alpha : \mathbb{R} \to \mathbb{R}$ kaavalla $\alpha(t_k) = t_{k+1}$. Lisäksi usein määrätään kussakin lämpötilassa suoritettavien iteraatiokierrosten määrä M_k [13]. Menetelmän toiminnan kannalta oleellista on, että $t_k > 0$ kaikilla $k = \{0,1,2,\ldots\}$, ja että $\lim_{k\to\infty} t_k = 0$.

Seuraavaksi esitetään simuloidun jäähdytyksen toimintaa yleisesti, ja valittu lähestymistapa noudattaa lähteessä [14] käytettyä esitystapaa. Menetelmän alussa asetetaan ensin lähtöpisteeksi $\boldsymbol{\omega}$ jokin piste joukosta Ω ja määritellään lähtölämpötila t_0. Lisäksi valitaan jäähdytysfunktio α, jonka avulla saadaan muodostettua lämpötilajono $\{t_k\}$, sekä asetetaan kussakin lämpötilassa t_k suoritettavien iteraatiokierrosten lukumäärä M_k.

Oletetaan, että algoritmissa lämpötila on t_k ja että nykyinen iteraatiopiste on $\boldsymbol{\omega}$. Seuraavaksi tarkastellaan, miten yksittäinen iteraatiokierros kyseisessä lämpötilassa suoritetaan. Ensin määrätään ratkaisuehdokas $\boldsymbol{\omega}' \in N(\boldsymbol{\omega})$, eli ehdokas seuraavaksi iteraatiopisteeksi nykyisen iteraatiopisteen naapuripisteiden joukosta. Tämän jälkeen verrataan iteraatiopisteen $\boldsymbol{\omega}$ ja ratkaisuehdokkaan $\boldsymbol{\omega}'$ kohdefunktion arvoja. Jos $f(\boldsymbol{\omega}') < f(\boldsymbol{\omega})$, eli kohdefunktion arvo pienenee ratkaisuehdokkaassa, niin asetetaan ratkaisuehdokas $\boldsymbol{\omega}'$ seuraavan iteraatiokierroksen iteraatiopisteeksi. Jos taas kohdefunktion arvo kasvaa ratkaisuehdokkaassa (eli $f(\boldsymbol{\omega}') > f(\boldsymbol{\omega})$), niin määritellään seuraavan iteraatiokierroksen iteraatiopiste $\boldsymbol{\omega}$ kaavasta

$$\boldsymbol{\omega} = \begin{cases} \boldsymbol{\omega}', & \text{jos } p < \exp\left(\frac{-(f(\boldsymbol{\omega}')-f(\boldsymbol{\omega}))}{t_k}\right) \\ \boldsymbol{\omega}, & \text{muulloin,} \end{cases} \tag{6}$$

jossa p on tasaisesti jakautunut satunnaisluku väliltä $[0, 1]$, joka generoidaan ennen kuin kaavaa (6) käytetään. Kaavasta (6) nähdään, että kun lämpötila t_k on korkea, niin todennäköisyys hyväksyä kohdefunktion arvoa huonontava ratkaisuehdokas $\boldsymbol{\omega}'$ on myös suuri, ja kun t_k on lähellä nollaa, ratkaisuehdokkaan hyväksymistodennäköisyys on pieni. Lämpötilassa t_k iteraatiokierroksia suoritetaan yllä mainitulla tavalla M_k kertaa, jonka jälkeen siirrytään uuteen lämpötilaan t_{k+1}. Algoritmin suoritusta jatketaan, kunnes jokin sen lopetusehto saavutetaan. Lopetusehtoja voivat olla esimerkiksi se, että globaali optimi on löytynyt tai että algoritmia on suoritettu riittävän monta iteraatiokierrosta.

Lisäksi simuloidussa jäähdytyksessä on usein tarpeen pitää muistissa paras tähän mennessä saavutettu iteraatiopiste. Erityisen tärkeää tämä on silloin, kun simuloitua jäähdytystä sovelletaan sellaisen ongelman ratkaisemiseen, jossa ei voida helposti tarkistaa onko jokin piste $\boldsymbol{\omega}$ tehtävän globaali optimi. Toisaalta esimerkiksi sudokun ratkaisun voi aina tarkistaa melko helposti, jolloin ei ole tarpeen säilyttää muistissa oikeaa ratkaisua edeltänyttä parasta väärää ratkaisua.

Edellä on nähty, että simuloidussa jäähdytyksessä on käytössä Metropolisin valintakriteeri, ja kaavasta (6) huomaa helposti, että siinä sovelletaan yksinkertaistettua versiota Boltzmannin jakaumasta. Jos muistetaan, että Metropolisin algoritmilla voidaan jäljitellä Boltzmannin jakaumaa, niin tarkastelemalla kuvaa 1 voidaan huomata, että simuloidun jäähdytyksen suorituksen aikana siirrytään litteämmistä korkean lämpötilan jakaumista nollan lähelle kasautuneisiin matalan lämpötilan jakaumiin. Mahdollisimman tehokkaasti tämä tapahtuu siten, että algoritmin alussa

on riittävän korkea lämpötila, koska tällöin algoritmi voi liikkua vapaasti erilaisten pisteiden välillä ja saavuttaa nopeasti kyseistä lämpötilaa vastaavan todennäköisyysjakauman. Lisäksi lämpötilaa tulisi laskea riittävän hitaasti, jotta edellisessä lämpötilassa saavutettu todennäköisyysjakauma on riittävän hyvä arvio seuraavalle. [14]

Lopuksi esitellään simuloidun jäähdytyksen pseudokoodi pelkistetyssä muodossa, ja esitystapa perustuu edelleen pääosin lähteeseen [14]. Alla esitettyä pseudokoodia on kuitenkin muokattu hieman, sillä siinä on mukana nykyisen parhaan ratkaisun $\boldsymbol{\omega}^*$ muistissa säilyttäminen. Parhaan ratkaisun säilyttäminen ei kuitenkaan vaikuta algoritmin varsinaiseen toimintaan.

Algoritmi 1. Simuloitu jäähdytys.

Alusta lähtöpiste $\boldsymbol{\omega}$.
Aseta nykyinen paras ratkaisu $\boldsymbol{\omega}^* \leftarrow \boldsymbol{\omega}$.
Aseta $k \leftarrow 0$.
Alusta kaikki lämpötilat t_0,t_1,t_2,...
Alusta kaikki iteraatiokierrosten lukumäärät M_0,M_1,M_2,...
while lopetusehto ei ole voimassa **do**
    Aseta $m \leftarrow 0$.
    while $m < M_k$ **do**
        Luo ratkaisuehdokas $\boldsymbol{\omega}' \in N(\boldsymbol{\omega})$.
        Laske muutos $\Delta_{\boldsymbol{\omega},\boldsymbol{\omega}'} = f(\boldsymbol{\omega}') - f(\boldsymbol{\omega})$.
        if $\Delta_{\boldsymbol{\omega},\boldsymbol{\omega}'} \leq 0$ **then**
            Aseta $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}'$.
        else
            Generoi satunnaisluku p jakaumasta $U(0,1)$.
            if $p < \exp\left(\frac{-\Delta_{\boldsymbol{\omega},\boldsymbol{\omega}'}}{t_k}\right)$ **then**
                Aseta $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}'$.
            end if
        end if
        if $f(\boldsymbol{\omega}^*) > f(\boldsymbol{\omega}')$ **then**
            Aseta $\boldsymbol{\omega}^* \leftarrow \boldsymbol{\omega}'$.
        end if
        Aseta $m \leftarrow m + 1$.
    end while
    Aseta $k \leftarrow k + 1$.
end while

2.3 Ominaisuudet ja heikkoudet

Simuloidun jäähdytyksen toiminnasta on sekä teoreettista että kokeellista tietoa. Teoreettisesti on esimerkiksi voitu todistaa menetelmän asymptoottinen konvergenssi kohti globaalia minimiä [11, 17]. Menetelmän toimintaa äärellisellä määrällä iteraatioita on myös tutkittu teoreettisesti, mutta konvergenssia ei olla voitu todistaa ilman, että lämpötilan laskeminen tapahtuu niin hitaasti, että algoritmista tulee käyttökelvoton [14]. Algoritmin suoriutumista äärellisessä ajassa voidaan kui-

tenkin arvioida, esimerkiksi algoritmin informaatioteoreettisen entropian avulla [5]. Kokeellisesti on nähty, että simuloitu jäähdytys saavuttaa yleensä hyvän tuloksen myös käytännössä, eli siis globaalin tai lokaalin minimin [18].

Simuloidun jäähdytyksen suosioon on vaikuttanut menetelmän implementoinnin helppous ja algoritmin yleisluontoisuus. Simuloitua jäähdytystä voi soveltaa pitkälti kaikkiin optimointitehtäviin, joihin iteratiiviset optimointimenetelmät sopivat ja joissa voidaan laskea nopeasti kohdefunktion arvon muutos siirryttäessä uuteen iteraatiopisteeseen. Menetelmä on myös melko joustava, sillä algoritmin koodia on usein helppo muokata myös jälkikäteen. [18]

Menetelmän yleisluontoisuuden kääntöpuoli on toisaalta se, ettei simuloidun jäähdytyksen toiminnan tehokkuutta voida taata kaiken tyyppisille optimointitehtäville. Menetelmää implementoitaessa puolestaan kokemus erilaisista optimointitehtävistä ja heuristisista menetelmistä on tyypillisesti eduksi. Lisäksi menetelmän käyttämiseen ja parametrien valintaan on olemassa hyviä ja melko yleisiä ohjeita, joita esitellään luvussa 2.4. Nämä ohjeet pohjautuvat kuitenkin usein kokeelliseen tietoon. Tämän takia optimointitehtävään hyödynnettävä tieto voi joskus olla peräisin sellaisesta tehtävästä, joka eroaa merkittävästi tarkasteltavasta tehtävästä. Simuloidussa jäähdytyksessä on myös paljon parametrejä, ja niiden hienosäätäminen voi olla erityisen hankalaa sekä aikaa vievää edellä mainittujen syiden takia. Joissain tapauksissa laskentaan vaadittava aika voi myös olla liian pitkä, mutta tähän voi yleensä vaikuttaa lopetusehdolla. Usein algoritmin suoritus voidaan esimerkiksi lopettaa, kun algoritmia on suoritettu jokin tietty aika tai kun iteraatiokierroksia on suoritettu jokin tietty määrä. Näin voidaan välttää turhien iteraatiokierrosten suorittaminen, kun algoritmi on jumittunut johonkin lokaaliin optimiin. [18]

Simuloidussa jäähdytyksessä, kuten Metropolisin algoritmissakin, monet teoreettiset tulokset algoritmin konvergoinnista perustuvat Markovin ketjuihin, joilla voidaan mallintaa todennäköisyyttä siirtyä jostain pisteestä toisiin pisteisiin [6, 14]. Konvergenssin todistamiseen vaadittuja oletuksia ovat esimerkiksi redusoimattomuus, jaksottomuus ja symmetrisyys [14], ja simuloidun jäähdytyksen tapauksessa nämä vaatimukset vaikuttavat lähinnä naapurifunktioon N. Markovin ketjuissa *redusoimattomuus* tarkoittaa sitä, että jokaisesta pisteestä $\boldsymbol{\omega}$ on mahdollista siirtyä mihin tahansa muuhun ratkaisuavaruuden Ω pisteeseen, jos suoritetaan riittävän monta iteraatiokierrosta. *Jaksottomuus* tarkoittaa taas sitä, ettei yksikään ratkaisuavaruuden Ω piste ole jaksollinen. Piste $\boldsymbol{\omega}$ on *jaksollinen*, jos jossain pisteessä käytyään algoritmi palaa samaan pisteeseen cd iteraatiokierroksen jälkeen, jossa c ja d ovat positiivisia kokonaislukuja ja lisäksi $d > 1$ on vakio. Esimerkiksi jos piste $\boldsymbol{\omega}$ on jaksollinen ja $d = 2$, niin tällöin algoritmi palaa pisteeseen $\boldsymbol{\omega}$ jonkin joukkoon $\{2,4,6,\ldots\}$ kuuluvan äärellisen askelmäärän jälkeen. *Symmetrisyys* tarkoittaa sitä, että todennäköisyys siirtyä pisteestä $\boldsymbol{\omega}'$ pisteeseen $\boldsymbol{\omega}''$ on yhtä suuri kuin todennäköisyys siirtyä pisteestä $\boldsymbol{\omega}''$ pisteeseen $\boldsymbol{\omega}'$. Simuloidussa jäähdytyksessä oletus symmetrisyydestä kohdistuu pelkästään naapureiden generointiin eikä ota huomioon kohdefunktion f arvon vaikutusta todennäköisyyteen hyväksyä eri siirrot. On kuitenkin hyvä huomioida, että eri konvergenssitodistuksissa sovelletaan eri oletuksia, ja esimerkiksi symmetrisyyden vaatimusta on joissain todistuksissa höllennetty [17]. Simuloitua jäähdytystä soveltaessa ei siis ole välttämätöntä, että symmetrisyys toteutuu aina täydellisesti, mutta edellä mainitut ominaisuudet on kuitenkin hyvä

ottaa huomioon naapurifunktion N valinnassa.

2.4 Implementointi ja parametrien valinta

Tässä luvussa käsitellään erilaisia ohjeistuksia siitä, miten simuloitu jäähdytys kannattaa implementoida tietokoneohjelmaksi. Ohjeistukset pohjautuvat lähteeseen [14] ja kirjan [18] lukuun 2. Simuloidun jäähdytyksen implementoinnissa tehdyt valinnat voidaan jakaa ensinnäkin ongelmakohtaisiin ja yleisiin valintoihin. Ongelmakohtaisiin valintoihin kuuluu kohdefunktio f, naapurifunktio N ja ratkaisuavaruus Ω. Joissain optimointitehtävissä voi vaikuttaa omituiselta ajatella kohdefunktiota f tai ratkaisuavaruutta Ω valintoina, koska ne seuraavat suoraan ratkaistavasta optimointitehtävästä. Kuitenkin jos simuloitua jäähdytystä sovelletaan esimerkiksi jonkin reaalimaailman ongelman ratkaisemiseen, niin ei välttämättä ole aina itsestään selvää, millaisella kohdefunktiolla ratkaisun laatua olisi hyvä mitata. Yleisiä tehtävästä riippumattomia valintoja ovat satunnaislukujen generointi, lämpötilan t_k päivitys (eli jäähdytysfunktio α ja siinä esiintyvät parametrit), iteraatiokierrosten lukumäärät M_k ja alkulämpötila t_0. Yleisesti ottaen, jos menetelmä on toteutettu hyvin, algoritmin antamissa tuloksissa ei ole hirveästi vaihtelua eri ajokertojen välillä.

Simuloidussa jäähdytyksessä olennaista on se, että muutos kohdefunktion arvossa voidaan laskea nopeasti. Joskus riittää ensin määrätä kohdefunktion arvo tarkasteltavissa pisteissä $\boldsymbol{\omega}^1$ ja $\boldsymbol{\omega}^2$, ja laskea muutos tätä kautta. Toisaalta jos tarkasteltavat pisteet $\boldsymbol{\omega}^1$ ja $\boldsymbol{\omega}^2$ eroavat toisistaan vain yhdessä komponentissa, niin olemassa voi olla nopeampia tapoja laskea kohdefunktion muutos kuin selvittämällä ensin arvot $f(\boldsymbol{\omega}^1)$ ja $f(\boldsymbol{\omega}^2)$. Joissain tehtävissä muutosta voi olla mahdollista jopa arvioida stokastisesti.

Rajoitteet ovat oleellinen osa optimointitehtäviä, joten niiden sisällyttäminen algoritmiin on tärkeää. Yksi tapa huomioida rajoite on jättää ratkaisuavaruuden Ω ulkopuolelle kaikki ne pisteet, jotka eivät toteuta rajoitteen ehtoa. Toisaalta joskus ratkaistava rajoitteellinen optimointitehtävä on sen verran monimutkainen, että rajoitteiden sisällyttäminen suoraan ratkaisuavaruuteen aiheuttaa hankaluuksia. Esimerkiksi tehtävässä

$$\begin{aligned} \min \quad & f(x,y) \\ \text{s.t.} \quad & x^2+y^2 \geq 4 \\ & -2 \leq x \leq 2 \\ & -2 \leq y \leq 2 \end{aligned} \tag{7}$$

jokaisen kolmen rajoitteen sisällyttäminen ratkaisuavaruuteen Ω edellä mainitulla tavalla voisi rajoittaa ratkaisuavaruutta liikaa, ja täten vaikeuttaa siirtymistä eri pisteiden välillä. Sen sijaan tehtävässä (7) voisi sisällyttää viimeiset kaksi ehtoa ratkaisuavaruuteen Ω ja ottaa ensimmäinen ehto huomioon *sakkoterminä*, eli niin, että jos rajoitetta $x^2+y^2 \geq 4$ rikotaan, niin siitä sakotetaan kohdefunktiossa. Tällöin simuloidun jäähdytyksen algoritmi voi kulkea ratkaisuavaruudessa vapaammin, ja muokatulla tehtävällä on edelleen sama optimi kuin tehtävässä (7).

Siinä missä ratkaisuavaruus Ω määrittää pistejoukon, josta optimia haetaan, naapurifunktio N määrittää sen, miten joukon Ω pisteet ovat yhteydessä toisiinsa. Erityisesti ratkaisuavaruutta Ω, naapurifunktiota N ja kohdefunktiota f voidaan mietiä yhtenä kokonaisuutena. Tälle kokonaisuudelle sileähkö muoto, jossa lokaalit mi-

nimit ovat vähäisiä ja matalia, on toivotumpi kuin sellainen muoto, jossa on paljon syviä lokaaleja minimejä. Naapurifunktion N antamien vaihtoehtojen lukumäärällä on myös väliä. Jos vaihtoehtojen lukumäärä on pieni, niin algoritmi voi vaatia liian monta iteraatiokierrosta ratkaisuavaruuden Ω eri pisteiden välillä kulkemiseen. Toisaalta jos vaihtoehtojen määrä on suuri, niin siirtyminen sallitun alueen sisällä on satunnaisempaa, jolloin algoritmilla voi olla vaikeuksia lähestyä lokaaleja optimeja ratkaisuavaruudessa.

Lämpötilojen t_k alustaminen voidaan aloittaa alkulämpötilan t_0 määrittelystä. Eräs tapa alkulämpötilan määräämiseen on ensin päättää, kuinka monta kohdefunktion arvoa huonontavaa pistettä halutaan algoritmin hyväksyvän, kun $k = 0$, ja tämän perusteella laskea alkulämpötilan t_0 arvo. Esimerkiksi voitaisiin hyväksyä noin 20% tai 50% kohdefunktiota huonontavista arvoista. Jos lämpötila haluttaisiin määrittää todennäköisyydelle 50%, se onnistuisi soveltamalla kaavaa (6) ja selvittämällä millä lämpötilalla t_0 toteutuu

$$0{,}5 = \exp\left(\frac{-(f(\boldsymbol{\omega}') - f(\boldsymbol{\omega}))}{t_0}\right).$$

Arvon t_0 määrittäminen tässä tapauksessa vaatii myös sen, että erotuksesta $f(\boldsymbol{\omega}') - f(\boldsymbol{\omega})$ on olemassa jokin arvio. Alkulämpötilan kanssa voidaan toimia myös niin, että käyttäjä antaa haluamansa arvon alkulämpötilalle ennen algoritmin suoritusta.

Alkulämpötilan lisäksi on myös päätettävä jäähdytysfunktio α. Eräs yksinkertainen ja hyväksi todettu tapa on laskea lämpötilaa geometrisesti. Tässä tapauksessa jäähdytysfunktio saa muodon $\alpha(t_k) = at_k$, kun $a \in (0, 1)$. Näin ollen uusi lämpötila saadaan kaavalla $t_{k+1} = \alpha(t_k)$. Eräs suositus aloittelijoille on määrätä kertoimen a arvoksi 0,9, ja toinen on valita kertoimen a arvo vapaammin väliltä $[0{,}8; 0{,}99]$.

Kun lämpötilat t_k on määritelty, on vielä päätettävä kussakin lämpötilassa t_k suoritettavien iteraatiokierrosten lukumäärä M_k. Yksinkertaisimmillaan M_k voi olla muotoa $M_k = \mu$ kaikilla k, kun μ on jokin positiivinen kokonaisluku. Erään ohjeistuksen mukaan toistojen lukumääräksi M_k voi valita optimointiongelman parametrien lukumäärästä riippuvan arvon. Näin ollen, jos parametrien lukumäärä on $P \in \mathbb{N}$, niin iteraatiokierrosten lukumäärä M_k voi olla muotoa $100 \cdot P$ kaikille indekseille k. Iteraatiokierrosten lukumäärät M_k voi myös määrätä niin, että algoritmi hyväksyy lämpötilassa t_k uuden iteraatiopisteen $12 \cdot P$ kertaa. Näin ollen parametrin M_k arvo olisi tapauskohtainen ja riippuisi algoritmin toiminnasta.

Eräs keskeinen asia simuloidun jäähdytyksen toteutuksessa on se, että pystytään generoimaan satunnaislukuja tasaisesta jakaumasta, sillä niitä hyödynnetään useammassa kohtaa menetelmän suoritusta. Ensinnäkin satunnaislukuja käytetään testattaessa ratkaisuehdokkaan $\boldsymbol{\omega}'$ hyväksymistä. Toiseksi satunnaislukuja tarvitaan valitsemaan ratkaisuehdokas $\boldsymbol{\omega}'$ naapuripisteiden joukosta $\{\boldsymbol{\omega}^1, \boldsymbol{\omega}^2, \ldots, \boldsymbol{\omega}^n\} \in N(\boldsymbol{\omega})$, kun $\boldsymbol{\omega}$ on nykyinen iteraatiopiste.

3 Sudokun ratkaiseminen

Sudoku on suosittu ja yksinkertainen pulmapeli, jossa on käytössä 9×9-ruudukko, joka on jaettu yhdeksään pienempään 3×3-aliruudukkoon. Sudokussa on tarkoi-

tuksena täyttää ruudukko luvuilla yhdestä yhdeksään niin, ettei yksikään luvuista toistu kertaakaan yhdelläkään ruudukon pysty- ja vaakarivillä tai yhdessäkään 3×3-aliruudukossa. Osa sudokun luvuista on annettu valmiiksi, ja näitä lukuja ei saa ratkaisussa muuttaa. Jatkossa valmiiksi annettuja lukuja kutsutaan vihjenumeroiksi ja sudokun muita lukuja muutettaviksi numeroiksi. Liitteessä A on esitetty esimerkkejä erilaisista sudoku-tehtävistä, joita käytetään myöhemmin tässä luvussa simuloidun jäähdytyksen toiminnan havainnollistamiseen.

Sudoku soveltuu hyvin ratkaistavaksi simuloidulla jäähdytyksellä, sillä pelin säännöt ovat selkeät ja helpot ilmaista optimointitehtävän kohdefunktiona ja rajoitteina. Sudokun ratkaisemiseen on koodattu yksinkertainen versio simuloidusta jäähdytyksestä käyttäen Javaa, jonka löytää sivulta `https://github.com/risliu/SA-sudoku`. Seuraavissa aliluvuissa käsitellään tarkemmin erimerkiksi algoritmissa tehtyjä valintoja, kuten sitä, miten kohdefunktio f ja naapurifunktio N ovat määritelty. Lopuksi esitellään menetelmällä saatuja numeerisia tuloksia, ja vertaillaan miten eri parametrien arvot vaikuttavat saatuihin tuloksiin. Tarkoituksena on havainnollistaa, miten simuloitua jäähdytystä voi käytännössä soveltaa ja miten menetelmässä eri parametrivalinnat vaikuttavat saatuihin tuloksiin. Lopuksi pohditaan, mitä johtopäätöksiä tuloksista voi tehdä. Näin ollen tässä luvussa ei verrata saatuja tuloksia muihin sudokulle kehitettyihin algoritmeihin.

3.1 Menetelmän implementointi

Simuloidun jäähdytyksen implementoinnissa sudokun ratkaisu esitetään 9×9-kokonaislukutaulukkona, jossa sudokuun tulevat numerot yhdestä yhdeksään on kuvattu kokonaislukuina $1, 2, \ldots, 9$ ja luku 0 vastaa toistaiseksi tyhjää ruutua. Tämän lisäksi jokaiseen taulukon kokonaislukuun eli ruutuun linkitetään tieto siitä, onko ruudussa vihjenumero vai muutettava numero. Kun menetelmän suoritus aloitetaan, taulukkoon asetetaan jokaiseksi muuttuvaksi numeroksi jokin satunnaisesti generoitu kokonaisluku $1, 2, \ldots, 9$, minkä avulla päästään eroon tyhjistä ruuduista ja saadaan aikaiseksi lähtöratkaisu.

Sudokua ratkaistaessa on olennaista, ettei vihjenumeroita muuteta. Tästä syystä on tärkeää, ettei naapurifunktio N ehdota uusia ratkaisuehdokkaita, joissa olisi muutettu jotain vihjenumeroa. Tämän lisäksi esimerkiksi sudokussa A.1 on ylimmällä vaakarivillä vihjenumerot 4 ja 1, minkä takia ratkaisun ylimmän rivin muissa ruuduissa ei voi esiintyä näitä lukuja. Nämä kaksi seikkaa riittävät naapurifunktion tarjoamien ratkaisuehdokkaiden rajaamiseen. Menetelmän toteutuksessa naapurifunktio toimii niin, että sen tuottamassa ratkaisuehdokkaassa $\boldsymbol{\omega}' \in N(\boldsymbol{\omega})$ on vaihdettu yksi muutettava numero joksikin toiseksi niin, ettei muutos ole ristiriidassa vihjenumeroiden asettamien rajoitusten kanssa. Jokainen sallittu muutos on naapurifunktiossa yhtä todennäköinen.

Kohdefunktion arvon ja sen muutoksen nopea laskeminen on simuloidun jäähdytyksen kannalta olennaista. Erityisen tärkeää on, että kohdefunktion pienin mahdollinen arvo vastaa sudokun globaalia optimia. Menetelmän implementoinnissa kohdefunktion arvo saadaan laskemalla yhteen jokaisen kokonaisluvun $1, 2, \ldots, 9$ toistojen määrä jokaisessa vaakarivissä, pystyrivissä ja 3×3-aliruudukossa. Esimerkiksi jos jossain rivissä esiintyy numero 5 kolme kertaa, niin tällöin lukua on toistettu yh-

teensä kahdesti, ja kohdefunktion arvoon lisätään kyseinen toistojen määrä. Näin määritettynä kohdefunktion optimiarvo on nolla, sillä tällöin jokaisessa vaakarivissä, pystyrivissä sekä 3×3-aliruudukossa esiintyy numerot $1, 2, \ldots, 9$ tasan kerran.

Kohdefunktion arvo lasketaan menetelmän alussa lähtöratkaisussa käymällä läpi jokaisen kokonaisluvun $1, 2, \ldots, 9$ esiintyminen jokaisella vaakarivillä, pystyrivillä, ja 3×3-aliruudukossa. Tämän jälkeen uudessa iteraatiopisteessä riittää vain määrittää kohdefunktion arvon muutos, joka onnistuu nopeammin. Alustetaan ensin muutoksen arvo olemaan nolla. Seuraavaksi selvitetään, mitä ruutua ratkaisuehdokkaassa $\boldsymbol{\omega}'$ on muutettu ja mitkä sen alkuperäinen ja uusi lukuarvo ovat. Tämän jälkeen kyseistä ruutua vastaavan vaakarivin, pystyrivin ja 3×3-aliruudukon kohdalla selvitetään, onko niissä toistoja alkuperäisen tai uuden lukuarvon kanssa. Jos joko vaakarivillä, pystyrivillä tai 3×3-aliruudukossa on toistoja alkuperäisen lukuarvon kanssa, jokaista kohden muutos pienenee yhdellä. Jos toistoja on puolestaan uuden lukuarvon kanssa vaakarivillä, pystyrivillä tai 3×3-aliruudukossa, muutos kasvaa jokaista kohden yhdellä. Muutos voi kasvaa näin ollen esimerkiksi kahdella, jos toistoja on vaikka sekä vaakarivillä että pystyrivillä. Lopputuloksena muutos on aina jokin kokonaisluku arvojen -3 ja 3 välillä.

Menetelmän toteutuksessa lämpötilaparametrin t_k määräämisen voi jakaa kolmeen osaan: alkulämpötilaan t_0, jäähdytysfunktioon α ja iteraatiokierrosten lukumäärään M_k lämpötilassa t_k. Menetelmän implementoinnissa käytössä oleva jäähdytysfunktio on muotoa $t_{k+1} = at_k$, jossa $a \in (0, 1)$, ja iteraatiokierrosten lukumäärä on jokin $\mu \in \mathbb{N}$ kaikille lämpötiloille t_k. Seuraavassa aliluvussa esitellään tarkemmin, miten lämpötilaan liittyvien parametrien lukuarvot on valittu.

Algoritmissa on kaksi eri lopetusehtoa. Ensimmäisessä ehdossa algoritmin suoritus pysäytetään, jos kohdefunktion arvo on nolla. Tällöin sudoku on saatu ratkaistua. Toisessa lopetusehdossa menetelmän suoritus puolestaan pysäytetään, jos kolmessa peräkkäisessä lämpötilassa t_k yhtään kohdefunktion arvoa huonontavaa ratkaisuehdokasta ei olla hyväksytty. Tässä toisessa lopetusehdossa pyritään arvioimaan, onko algoritmi jumittanut johonkin lokaaliin optimiin. Jos on, niin algoritmin suoritus lopetetaan, jotta voidaan välttää laskenta-ajan tuhlaamista ajoon, jossa globaalia optimia ei todennäköisesti saavuteta.

3.2 Parametrien valinta

Tässä aliluvussa kerrotaan, miten on valittu lukuarvot alkulämpötilalle t_0, jäähdytysfunktion kertoimelle a ja iteraatiokierrosten lukumäärälle M_k. Kullekin parametrille on valittu vähintään kaksi mahdollista lukuarvoa, ja algoritmin suoritusta näillä eri arvoilla vertaillaan aliluvussa 3.3. Alkulämpötilan ja jäähdytysfunktion kertoimen valinnat perustuu teoksen [18] ehdotuksiin.

Alkulämpötila t_0 on valittu niin, että joko 20% tai 50% kohdefunktion arvoa huonontavista ratkaisuehdokkaista hyväksytään. Tämä on huomioitu menetelmän koodissa niin, että käyttäjä voi valita kahdesta alkulämpötilasta, jotka vastaavat kyseisiä 20% ja 50% todennäköisyyksiä. Jotta alkulämpötiloille on saatu oikeat lukuarvot, on selvitetty odotusarvo kohdefunktion arvon muutokselle niissä tapauksissa, jossa kohdefunktion arvo on ratkaisuehdokkaassa suurempi kuin nykyisessä iteraatiopisteessä. Jatkossa kyseistä odotusarvoa merkitään kirjoittamalla Δ, ja esi-

merkiksi sudokun A.1 tapauksessa tämä odotusarvo on laskennallisesti määritetty olemaan noin $\Delta = 1{,}5873394$. Tarkastelemalla taulukkoa 1 huomataan, että likimääräinen arvo odotusarvolle vaihtelee tehtävän mukaan. Alkulämpötilat on kuitenkin laskettu arvion $\Delta = 1{,}5873394$ avulla, joten implementoinnissa ei olla esimerkiksi laskettu alkulämpötiloja jokaisen sudokun kohdalla erikseen. Alkulämpötilat saadaan puolestaan määrättyä yhtälöstä

$$t_0 = \frac{-\Delta}{\ln p},$$

jossa p on todennäköisyys hyväksyä kohdefunktion arvoa huonontava ratkaisuehdokas. Sijoittamalla $\Delta = 1{,}5873394$ ja $p = 0{,}5$ saadaan ensimmäiseksi alkulämpötilaksi $t_0(50\%) = 2{,}290046681$. Sen sijaan sijoituksella $p = 0{,}2$ saadaan toiseksi alkulämpötilaksi $t_0(20\%) = 0{,}986269422$.

Taulukko 1: Jokaiselle liitteen A sudokulle mahdollisten naapuripisteiden määrä $|N(\boldsymbol{\omega})|$, ratkaisuavaruuden pisteiden lukumäärä $|\Omega|$ sekä laskennallinen arvio kohdefunktion arvoa huonontavan muutoksen keskiarvosta Δ.

Sudokutehtävä	$\lvert N(\boldsymbol{\omega})\rvert$	$\lvert\Omega\rvert$	Δ
Sudoku A.1	217	$1{,}496{\cdot}10^{32}$	1,5873394
Sudoku A.2	188	$4{,}207{\cdot}10^{27}$	1,6424732
Sudoku A.3	50	$4{,}977{\cdot}10^{5}$	1,8571243
Sudoku A.4	171	$6{,}065{\cdot}10^{25}$	1,6252895
Sudoku A.5	199	$1{,}179{\cdot}10^{29}$	1,5838349

Jäähdytysfunktiolle $\alpha(t_k) = at_k$ kerroin a on valittu olemaan joko 0,9 tai 0,8. Kertoimen arvo $a = 0{,}9$ on melko tyypillinen valitulle jäähdytysfunktiolle, kun taas valinta $a = 0{,}8$ laskee lämpötilaa tätä nopeammin.

Iteraatiokierrosten lukumääräksi M_k lämpötilassa t_k voi menetelmässä valita minkä tahansa kokonaislukuarvon $\mu \in \mathbb{N}$, eli $M_k = \mu$ jokaisella lämpötilalla t_k. Kuten taulukosta 1 nähdään, ratkaisuavaruuden koko sekä mahdollisten ratkaisuehdokkaiden lukumäärä voivat olla suuria, ja näin ollen iteraatiokierrosten määrän μ on myös oltava riittävän suuri. Kvalitatiivisen tarkastelun perusteella algoritmin on todettu ratkaisevan tehtävän nopeasti, kun iteraatiokierrosten määrä μ oli kokonaisluku väliltä $[500, 4500]$. Tarkempaa vertailua varten iteraatiokierrosten määrä μ on valittu olemaan joko 500, 1500 tai 4500.

3.3 Numeeriset tulokset

Seuraavaksi esitetään simuloidulla jäähdytyksellä saatuja tuloksia sudokuille ja vertaillaan, miten erilaiset parametrivalinnat vaikuttavat menetelmän toimintaan. Heuristisen menetelmän suoriutumista arvioidessa on otettava huomioon sekä ratkaisun laatu että kuinka nopeasti siihen on päästy. Sudokun tapauksessa voidaan ajatella, että ainoa hyvä ratkaisu on sellainen, missä kohdefunktion arvo on nolla, eli on saavutettu globaali optimi ja tehtävä on ratkennut. Kuitenkin jos simuloidun jäähdytyksen algoritmia käytetään yksittäisen sudokun ratkaisemiseen vain kerran, eli suoritetaan yksi ajo, niin saavutetun ratkaisun kohdefunktion arvo ei välttämättä

ole nolla. Siksi ohjelman suoriutumista tarkastellaan tässä aliluvussa vertailemalla sitä, kuinka kauan kestää saavuttaa globaali optimi, jos ajoja suoritetaan, kunnes tällainen ratkaisu löytyy. Useamman ajokerran suorittaminen eli uudelleenkäynnistys on simuloidun jäähdytyksen kaltaisissa iteratiivisissa heuristisissa menetelmissä melko tavallista [14]. Uudelleenkäynnistystä soveltava heuristinen menetelmä voi joissain tapauksissa saavuttaa optimin todennäköisemmin kuin samassa ajassa toiminut vastaavanlainen heuristinen menetelmä ilman uudelleenkäynnistystä [7].

Seuraavaksi siirrytään tarkastelemaan simuloidulla jäähdytyksellä saatuja numeerisia tuloksia. Kaikki tulokset on laskettu tietokoneella, jossa on käytössä Windows 11 käyttöjärjestelmä, Intel Core i5 prosessori, 8Gt keskusmuistia ja NVIDIA GeForce GTX 1070 näytönohjain. Tuloksia laskettaessa on lisäksi käytetty Javan ajoympäristöä (JRE) 17.0.1.

Taulukoihin 2−4 on kerätty keskimääräiset laskenta-ajat eri sudokuille eri parametriyhdistelmillä, ja taulukoiden tulokset on ilmoitettu sekunteina. Numeeriset tulokset on kerätty niin, että jokaista taulukoissa mainittua parametriyhdistelmää soveltamalla on 500 kertaa löydetty globaali optimi, ja näiden 500 optimin löytämiseen vaaditusta ajasta on otettu keskiarvo. Ajokerrat, jolla ei olla saavutettu optimia, otetaan kuitenkin huomioon mittaustuloksissa. Voi siis esimerkiksi käydä niin, että ohjelma ensin aloittaa ajan mittaamisen, minkä jälkeen se suorittaa muutaman ajon, joissa saatu tulos $f(\boldsymbol{\omega}) > 0$, ja vasta tämän jälkeen tekee ajon, jossa $f(\boldsymbol{\omega}) = 0$, jolloin ajan mittaaminen lopetetaan. Taulukoiden 2−4 tulokset eivät ota kantaa siihen kuinka monta ajokertaa on vaadittu, jotta globaali optimi on löytynyt 500 kertaa, mutta vaadittujen ajokertojen lukumäärä on kuitenkin merkitty taulukoihin 5−7. Taulukoissa 2−4 ilmoitettu keskiarvo on puolestaan sudokujen A.1−A.5 tulosten keskiarvo tietyillä kiinnitetyillä parametreilla.

Taulukko 2: Keskimääräiset laskenta-ajat sekunteina eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 500$ kaikissa lämpötiloissa t_k.

$M_k = 500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	1,432228	3,631798	1,000860	2,37612
Sudoku A.2	0,287314	0,626330	0,190148	0,459352
Sudoku A.3	0,000228	0,000180	0,000052	0,000050
Sudoku A.4	0,031818	0,060198	0,020192	0,042244
Sudoku A.5	0,151498	0,417822	0,102580	0,297520
Keskiarvo	0,380617	0,947266	0,262766	0,635057

Taulukko 3: Keskimääräiset laskenta-ajat sekunteina eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 1500$ kaikissa lämpötiloissa t_k.

$M_k = 1500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	0,657084	1,210568	0,461284	0,777584
Sudoku A.2	0,165602	0,258836	0,112358	0,184896
Sudoku A.3	0,000326	0,000282	0,000050	0,000052
Sudoku A.4	0,030332	0,033016	0,016674	0,021224
Sudoku A.5	0,085082	0,130666	0,056296	0,102112
Keskiarvo	0,187685	0,326674	0,129332	0,217174

Taulukko 4: Keskimääräiset laskenta-ajat sekunteina eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 4500$ kaikissa lämpötiloissa t_k.

$M_k = 4500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	0,530644	0,710682	0,368248	0,476828
Sudoku A.2	0,153498	0,17261	0,112886	0,132846
Sudoku A.3	0,000424	0,000374	0,000050	0,000052
Sudoku A.4	0,036822	0,031792	0,024402	0,022242
Sudoku A.5	0,093714	0,101812	0,058340	0,070284
Keskiarvo	0,163020	0,203454	0,112785	0,140450

Taulukoiden 2−4 tuloksista nähdään, että eri sudokutehtävien laskenta-ajoissa on merkittäviä eroja. Jos merkitään sudokulle A.i laskentaan vaadittua aikaa kirjoittamalla $\tau(i)$ kaikilla $i = 1,2,3,4,5$, niin huomataan että kaikilla valituilla parametriyhdistelmillä pätee, että

$$\tau(1) > \tau(2) > \tau(5) > \tau(4) > \tau(3). \tag{8}$$

Näin ollen sudoku A.1 on vaikein ratkaista ja sudoku A.3 puolestaan helpoin. Sudokutehtävien A.1−A.5 vaikeutta voidaan arvioida myös taulukon 1 ratkaisuavaruuden pisteiden lukumäärien $|\Omega|$ avulla. Käyttämällä merkintää, jossa $|\Omega(i)|$ on ratkaisuavaruuden koko sudokulle A.i, tehtävät voidaan laittaa järjestykseen

$$|\Omega(1)| > |\Omega(5)| > |\Omega(2)| > |\Omega(4)| > |\Omega(3)|.$$

Tehtävien järjestys on tällä tavalla muuten sama kuin kaavassa (8), paitsi että sudokut A.2 ja A.5 ovat vaihtaneet paikkaansa. Taulukoiden 2−4 tulokset yhtyvät taulukkoon 1 myös siinä mielessä, että sudoku A.3 vaikuttaa olevan selkeästi muita helpompi ratkaista, kun taas sudoku A.1 on selkeästi muita vaikeampi. Näin ollen numeeriset tulokset vaikuttavat johdonmukaisilta sudokujen ratkaisuavaruuksien kokojen kanssa.

Vertailtaessa alkulämpötilan t_0 vaikutusta taulukoiden 2−4 tuloksiin huomataan, että alkulämpötila $t_0(20\%) = 0{,}986269422$ on jokaisessa sudokussa A.1−A.5 ja kiinnitetyillä parametrien arvoilla a ja $M_k = \mu$ parempi kuin valinta $t_0(50\%) =$

2,290046681. Tämä voi johtua siitä, että alkulämpötilaa $t_0(20\%)$ käyttäessä lämpötila t_k laskee nopeammin riittävän pieneksi ja näin ollen mahdollistaa suppenemisen kohti jotain lokaalia optimia. On mahdollista, että yksittäisellä ajokerralla alkulämpötilalla $t_0(50\%)$ saatetaan saavuttaa todennäköisemmin ratkaisu, jossa $f(\boldsymbol{\omega}) = 0$. Menetelmä alkulämpötilalla $t_0(20\%)$ voidaan kuitenkin todeta keskimääräisesti nopeammaksi, jos ajokertoja toistetaan peräjälkeen niin kauan, että globaali optimi löytyy.

Saaduista tuloksista nähdään myös, että laskenta-aika on usein selkeästi lyhyempi, jos jäähdytysfunktion $\alpha(t_k) = at_k$ kerroin a on 0,9 ja jos muut parametrit pidetään samoina. Taulukoista 2−4 kuitenkin huomataan, että sudokun A.3 kaltaisten helppojen tehtävien kohdalla valinta $a = 0{,}8$ on parempi. Tämä voi johtua esimerkiksi siitä, että hyvin helppojen sudokutehtävien kohdalla kaikki lämpötilaparametrin t_k laskua nopeuttavat valinnat nopeuttavat myös ratkaisun löytämistä. Vaikeammissa tehtävissä vaikuttaa olevan puolestaan hyötyä siitä, että lämpötilaa lasketaan hitaammin kertoimella 0,9, jolloin ratkaisuavaruutta käydään läpi perusteellisemmin. Voi myös olla, että jos tehtävä on vaikea ja lämpötilaa lasketaan liian nopeasti, niin yksi simuloidun jäähdytyksen lopetusehdoista täyttyy liian aikaisin, sillä algoritmi tulkitsee päätyneensä lokaaliin optimiin ennenaikaisesti. Näin ollen uusi ajo joudutaan aloittamaan, vaikka nykyisen ajon aikana olisi edelleen voitu saavuttaa globaali optimi.

Taulukossa 2 ilmoitetut laskenta-ajat ovat pääsääntöisesti hitaampia kuin taulukoissa 3 ja 4, paitsi helpoimman sudokun A.3 kohdalla. Tämä voi johtua siitä, että jos iteraatiokierrosten lukumäärä M_k on 500, lopetusehto täyttyy todennäköisemmin ennenaikaisesti. Tämä voi myös selittää sen, miksi laskenta-ajat jäähdytysfunktion kertoimella $a = 0{,}8$ ja iteraatiokierrosten lukumäärällä $M_k = 4500$ ovat nopeampia kuin kertoimella $a = 0{,}8$ ja iteraatiokierrosten lukumäärällä $M_k = 1500$. Sen sijaan kertoimella $a = 0{,}9$ erot laskenta-ajoissa valintojen $M_k = 1500$ ja $M_k = 4500$ välillä ovat melko pieniä.

Taulukoihin 2−4 merkittyjen laskenta-aikojen keskiarvojen perusteella keskimääräisesti paras parametriyhdistelmä sudokutehtävien ratkaisemiseen on $t_0(20\%)$, $a = 0{,}9$ ja $M_k = 4500$. Lisäksi kyseisellä parametriyhdistelmällä saavutetaan paras laskenta-aika sudokutehtävissä A.1 ja A.2. Kuitenkin parametriyhdistelmällä $t_0(20\%)$, $a = 0{,}9$ ja $M_k = 1500$ saadaan paremmat keskimääräiset laskenta-ajat sudokutehtävissä A.4 ja A.5, jotka ovat toisaalta tehtäviä A.1 ja A.2 helpompia. Molemmilla mainituilla parametriyhdistelmillä saavutetaan myös keskimääräisesti paras laskenta-aika sudokussa A.3.

Tarkastellaan vielä taulukkoja 5−10. Taulukoihin 5−7 on merkittynä kuinka monta ajokertaa vaaditaan, jotta globaali optimi on löytynyt 500 kertaa, kun taas taulukoihin 8−10 on merkitty, kuinka monta kertaa kohdefunktion f arvo on laskettu jaettuna vaadittujen ajokertojen lukumäärällä.

Taulukko 5: Vaadittujen ajokertojen lukumäärä eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 500$ kaikissa lämpötiloissa t_k.

$M_k = 500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	392479	1779072	423169	1661011
Sudoku A.2	79216	311282	83213	332945
Sudoku A.3	500	500	500	500
Sudoku A.4	8842	30575	9307	30943
Sudoku A.5	42055	206540	44310	210178
Yhteensä	523092	2327969	560499	2235577

Taulukko 6: Vaadittujen ajokertojen lukumäärä eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 1500$ kaikissa lämpötiloissa t_k.

$M_k = 1500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	58987	196967	62939	181987
Sudoku A.2	15043	43037	15635	44272
Sudoku A.3	500	500	500	500
Sudoku A.4	2959	5778	2644	5489
Sudoku A.5	7798	21761	8082	24535
Yhteensä	85287	268043	89800	256783

Taulukko 7: Vaadittujen ajokertojen lukumäärä eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 4500$ kaikissa lämpötiloissa t_k.

$M_k = 4500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	15626	38080	16378	36812
Sudoku A.2	4602	9491	5184	10497
Sudoku A.3	500	500	500	500
Sudoku A.4	1287	1979	1394	2067
Sudoku A.5	2924	5612	2810	5675
Yhteensä	24939	55662	26266	55551

Taulukko 8: Keskiarvo siitä, kuinka monta kertaa kohdefunktion f arvo on määrätty jokaista suoritettua ajoa kohden eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 500$ kaikissa lämpötiloissa t_k.

$M_k = 500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	12388	6893	8392	5005
Sudoku A.2	12251	6824	8259	4935
Sudoku A.3	1424	1107	281	274
Sudoku A.4	11860	6699	7871	4813
Sudoku A.5	12238	6839	8243	4955
Keskiarvo	12335	6873	8345	4987

Taulukko 9: Keskiarvo siitä, kuinka monta kertaa kohdefunktion f arvo on määrätty jokaista suoritettua ajoa kohden eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 1500$ kaikissa lämpötiloissa t_k.

$M_k = 1500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	38680	21380	26722	15712
Sudoku A.2	38491	21220	26514	15560
Sudoku A.3	2095	1794	290	288
Sudoku A.4	35834	20290	23639	14519
Sudoku A.5	37898	21094	25889	15432
Keskiarvo	38262	21271	26373	15604

Taulukko 10: Keskiarvo siitä, kuinka monta kertaa kohdefunktion f arvo on määrätty jokaista suoritettua ajoa kohden eri parametriyhdistelmillä, kun toistojen lukumäärä $M_k = 4500$ kaikissa lämpötiloissa t_k.

$M_k = 4500$	$t_0(50\%)$ ja $a = 0.9$	$t_0(50\%)$ ja $a = 0.8$	$t_0(20\%)$ ja $a = 0.9$	$t_0(20\%)$ ja $a = 0.8$
Sudoku A.1	119881	65860	83902	48853
Sudoku A.2	118401	65263	83152	48317
Sudoku A.3	2748	2394	277	283
Sudoku A.4	101330	57785	66019	41319
Sudoku A.5	114583	63739	78086	46763
Keskiarvo	115681	64687	80591	47821

Taulukoista 5−7 huomataan, että ajokertojen lukumäärä noudattaa yhtälön (8) järjestystä. Lisäksi sudokutehtävien ja parametrivalintojen välillä on monen kertaluokan eroja vaadittujen ajokertojen lukumäärässä. Esimerkiksi sudokussa A.1 parhaalla parametriyhdistelmällä globaali optimi saavutetaan noin 3,2% ajoista, kun taas huonoimmalla parametriyhdistelmällä globaali optimi saavutetaan vain noin 0,028% ajoista. Taulukoista 5−7 huomaa myös sen, että sudokun A.3 tapauksessa globaali optimi saavutettiin jokaisella ajokerralla.

Sudokua A.3 lukuun ottamatta taulukoiden 8−10 tulokset vaihtelevat lähinnä eri parametriyhdistelmien välillä, eivätkä niinkään tehtävien välillä. Tämä voi selittyä esimerkiksi sillä, että jos ajokerta ei johda globaaliin optimiin, niin menetelmän suoritus lopetetaan, kun kolmessa peräkkäisessä lämpötilassa t_k yhtäkään kohdefunktion arvoa huonontavaa ratkaisuehdokasta ei hyväksytä. Tämän lopetusehdon täyttyminen riippuu lähinnä lämpötilaparametrista, eikä niinkään ratkaistavasta tehtävästä. Taulukoiden 5−7 perusteella tiedetään, että valtaosa ajoista ei johda globaaliin optimiin, jonka takia taulukoiden 8−10 tulokset painottuvat tietyillä parametriyhdistelmillä kohti samoja arvoja.

Taulukoiden 5−10 tulokset voivat myös kertoa siitä, että menetelmän suorituksen aikana lämpötilaa on laskettu liian nopeasti. Sen lisäksi keskimääräisesti parhaat laskenta-ajat on saatu parametriyhdistelmällä $M_k = 4500$, $t_0(20\%)$ ja $a = 0{,}9$, jossa, aloituslämpötilaa lukuun ottamatta, on tehty lämpötilan t_k laskemista hidastavia valintoja. Toisaalta helpommissa tehtävissä lämpötilan laskemista nopeuttavat valinnat ovat parantaneet myös joskus keskimääräisiä laskenta-aikoja. Jatkossa ohjelmaa voisi testata suuremmilla toistojen M_k ja kertoimen a arvoilla ja tarkastelemalla, kuinka paljon edellä mainittujen parametrien arvoja pitää nostaa, että laskenta-ajat muuttuvat hitaammiksi. Simuloidun jäähdytyksen algoritmia voisi olla mahdollista tehostaa myös siten, että ohjelma muuttaisi lämpötilaan t_k liittyviä parametreja yksittäisistä ajoista saatavan tiedon perusteella. Esimerkiksi algoritmi voisi ensin laskea lämpötilaa nopeasti, mutta hidastaa lämpötilan laskua, jos se ei saavuta globaalia optimia riittävän monen ajon jälkeen.

4 Yhteenveto

Työssä on perehdytty simuloituun jäähdytykseen. Simuloitu jäähdytys on metaheuristinen optimointimenetelmä ja yksi tunnetuimmista metaheuristiikoista. Erityisesti työssä on käsitelty simuloidun jäähdytyksen ominaisuuksia ja toimintaa. Aluksi työssä on kerrottu mihin menetelmiin simuloidun jäähdytyksen kehittäminen on perustunut, minkä lisäksi on mainittu muutamia sovelluskohteita simuloidulle jäähdytykselle. Tämän jälkeen on esitelty simuloidussa jäähdytyksessä käytetyt keskeiset parametrit. Algoritmin toiminta on selitetty sekä sanallisesti, että esittämällä algoritmi pseudokoodina. Simuloidun jäähdytyksen konvergenssia on kommentoitu lyhyesti, minkä lisäksi on tarjottu aloittelijalle sopivia ohjeistuksia menetelmän implementointiin.

Työn lopussa menetelmän toimintaa on havainnollistettu sudoku-tehtävällä. Saatujen tulosten perusteella on selvää, että sudokun kaltaisia yksinkertaisia pulmapelejä voidaan ratkaista melko helposti simuloidulla jäähdytyksellä, ja että menetelmän implementointi ei vaadi kuin suppeat ohjelmointitaidot ilman kattavampaa koulu-

tusta alalta. Numeeristen tulosten perusteella on nähty, että jos lämpötilaa pienennetään liian nopeasti, globaalin optimin löytämiseen vaadittava laskenta-aika kasvaa. Tästä syystä, jotta saavutetaan mahdollisimman hyvät laskenta-ajat, iteraatiokierrosten lukumäärän kussakin lämpötilassa ja lämpötilojen laskemiseen käytetyn kertoimen on oltava riittävän suuria. Lisäksi tuloksista on havaittu, että liian korkea aloituslämpötila myös pidentää laskenta-aikoja.

Yleisesti ottaen metaheuristiikkojen implementoinnissa vaaditaan kykyä tulkita menetelmälle saatuja numeerisia tuloksia sekä taitoa säätää menetelmän toimintaa tulosten perusteella. Koska simuloidussa jäähdytyksessä usein tasapainotellaan ratkaisujen hyvyyden ja käytetyn laskenta-ajan välillä, menetelmän implementoinnissa on myös osattava tehdä tapauskohtaisia päätöksiä. Näin ollen on osattava päättää, missä määrin halutaan varmistaa saavutettujen ratkaisujen laatu ja missä määrin halutaan vähentää laskenta-aikaa.

Viitteet

[1] D. Abramson ja M. Randall. A simulated annealing code for general integer linear programs. *Annals of Operations Research*, 86:3–21, 1999.

[2] A. Anagnostopaulos, L. Michel, P. Van Hentenryck, ja Y. Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9:177–193, 2006.

[3] L. Boltzmann. Studien über das gleichgewicht der lebendigen kraft zwischen bewegten materiellen punkten. *Wiener Berichte*, 58:517–560, 1868.

[4] D. Bovet ja P. Crescenzi. *Introduction to the theory of complexity.* Prentice Hall, New York, USA, 1994.

[5] M. Fleischer ja S. H. Jacobson. Information theory and the finite-time behavior of the simulated annealing algorithm: Experimental results. *INFORMS Journal on Computing*, 11:35–43, 1999.

[6] W. K. Hastings. Monte carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[7] S. H. Jacobson. Analyzing the performance of generalized hill climbing algorithms. *Journal of Heuristics*, 10:387–405, 2004.

[8] S. Kirkpatrick, C. D. Gelatt, ja M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[9] D. A. McQuarrie. *Statistical Mechanics.* Harper's chemistry series. Harper Collins, New York, 1976.

[10] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, ja A. H. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

[11] D. Mitra, F. Romeo, ja A. Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. *Advances in Applied Probability*, 18:747–771, 1986.

[12] M. M. Mäkelä. *Matemaattinen Optimointi I.* Turun Yliopisto, Matematiikan ja tilastotieteen laitos, 2019.

[13] M. M. Mäkelä. *Matemaattinen Optimointi II.* Turun Yliopisto, Matematiikan ja tilastotieteen laitos, 2019.

[14] A. G. Nikolaev ja S. H. Jacobson. Simulated annealing. Editorit M. Gendreau ja J. Potvin, *Handbook of Metaheuristics*, sivut 1–40. Springer, New York, USA, 2010.

[15] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving.* Addison-Wesley Publishing Company, USA, 1984.

[16] G. R. Raidl, J. Puchinger, ja C. Blum. Metaheuristic hybrids. Editorit M. Gendreau ja J. Potvin, *Handbook of Metaheuristics*, sivut 469–496. Springer, New York, USA, 2010.

[17] Y. Rossier, M. Troyon, ja T. M. Liebling. Probabilistic exchange algorithms and euclidean traveling salesman problems. *OR Spektrum*, 8:151–164, 1986.

[18] P. Siarry. *Metaheuristics.* Springer, Cham, Switzerland, 2016.

A Sudokutehtävät

4				1				
		2		5		8		
				7		3		4
			7	3		6		
					1		7	
	2	3	4				5	
		1		4		9		
	6						2	
	8		3					

Sudoku A.1

5		1			9			3
		7		6				
					5			
			9				4	
7				3		1		
9						8		6
4				8		9		5
		6			2	4	8	
3	8	9			4		6	

Sudoku A.2

8			2	5		1		6
7		1	6	9			3	5
6	4	5	1	7		2	9	8
4		8		2	7	3	5	
2	9	3	4	1	5	6		7
5	1					9	2	4
	7		8			5	6	
1	5		7			8	4	
3	8				2	7	1	9

Sudoku A.3

				5		2	4	
	2				6	9		
					3		7	
	8		9				6	
4				3			8	
6	3	1						
					7		3	8
8	7	5			4		1	2
	6			8			9	

Sudoku A.4

			1		8			
						3		
3	8	9			4			
		6	8					
7			6				5	
2	5				7	8		
								4
4	1	5				9	2	
	6			7	5			3

Sudoku A.5