Katsaus simuloidun jäähdytyksen teoriaan ja eräisiin käyttökohteisiin

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Toukokuu 2024
Tuomas Koski

TURUN YLIOPISTO
Tietotekniikan laitos

Tuomas Koski: Katsaus simuloidun jäähdytyksen teoriaan ja eräisiin käyttökohteisiin

LuK-tutkielma, 21 s.
Tietojenkäsittelytiede
Toukokuu 2024

Simuloitu jäähdytys on metaheuristinen algoritmi, jolla voidaan ratkaista erilaisia optimointiongelmia useilla tieteenaloilla. Tässä kirjallisuuskatsauksessa selvitetään miten algoritmi toimii ja millaisiin komponentteihin sen voidaan ajatella jakautuvan. Lisäksi tarkastellaan millaisia variaatioita algoritmista on kehitetty. Lopuksi selvitetään millaisten ongelmien ratkaisuun algoritmia on käytetty ja kuinka algoritmin suorituskyky on vertautunut muihin optimointialgoritmeihin. Merkittävimpinä lähdeteoksina tutkielmassa käytetään tutkimusartikkeleita sekä kirjoja. Tutkielmassa havaitaan, että simuloitu jäähdytys on kilpailukykyinen vaihtoehto monissa ongelmissa. Lisäksi tullaan johtopäätökseen, että usein sopivan implementaation löytämiseksi ei ole teoreettisia tuloksia ja käyttäjä joutuu turvautumaan hyperparametrien empiiriseen muokkaukseen.

Asiasanat: simuloitu jäähdytys, optimointi, implementaatio, metaheuristiikka

Sisällys

1 Johdanto

Optimoinnilla pyritään saamaan jokin prosessi toimimaan mahdollisimman tehokkaasti. Se tehdään etsimällä parametreille tai muuttujille konfiguraatio, mikä tuottaa parhaan ratkaisun. Teknologian eri sektoreilla optimointiongelmat ovat päivittäinen haaste. Suuret datamäärät vaativat prosessointia ja analysointia, jota voidaan optimoimalla nopeuttaa ja toteuttaa pienemmillä resursseilla. Elektronisten komponenttien tietokoneavusteinen suunnittelu on kompleksinen optimointitehtävä, samoin kuin neuroverkkojen suunnittelu. Nämä ovat muutamia esimerkkejä nykypäivän ongelmista, joissa optimointi on tärkeässä roolissa.

Monien kompleksisten optimointiongelmien eksakti ratkaisu nykyteknologialla ei ole mahdollista hyväksyttävässä ajassa. Metaheuristiikat ovat keinoja, joilla tällaisiin ongelmiin voidaan etsiä hyvää ratkaisua tehokkaasti. Simuloitu jäähdytys on yksi vanhimmista metaheuristiikoista vuodelta 1983.[1] Sen tutkimus ja kehittäminen jatkuu edelleen 2020-luvulla. IEEE Xplore -tietokannasta löytyy viimeisen viiden vuoden ajalta lähes 500 konferenssi- ja aikakausijulkaisua, joiden otsikossa mainitaan simuloitu jäähdytys. Mikäli hakua laajennetaan abstrakteihin ja avainsanoihin on tuloksia yli 2000.

Tässä tutkielmassa perehdytään siihen, miten simuloitu jäähdytys toimii. Luvussa kaksi pohjustetaan aiheeseen liittyviä käsitteitä, käydään läpi algoritmin toimintaa ja sen komponentteja. Luvussa kolme käydään läpi tutkimuksia siitä, millaisiin ongelmiin simuloitua jäähdytystä on käytetty ja kuinka sen suorituskyky vertautuu

muihin algoritmeihin. Tutkielmassa pyritään vastaamaan seuraaviin tutkimuskysymyksiin:

- Tutkimuskysymys 1: Mitä on simuloitu jäähdytys ja sen eri variaatiot?
- Tutkimuskysymys 2: Miten hyperparametrit vaikuttavat simuloidun jäähdytyksen toimintaan?
- Tutkimuskysymys 3: Mitä ongelmia voidaan ratkaista simuloitua jäähdytystä käyttäen ja kuinka se vertautuu muihin algoritmeihin?

Tiedonhaun tärkeimmät hakukannat ovat olleet Volter, Web of science ja IEEE Xplore. Haussa on käytetty hakufraasia "*simulated annealing*" ja lisäksi algoritmin eri komponentteja, kuten "*cooling schedule*", "*neighborhood structure*" ja muita. Myös eri ongelmia koskien on haettu tietoa tarkentamalla hakua esimerkiksi fraaseilla "*traveling salesman*" tai "*neural network**". Lisää tutkielman kannalta kiinnostavia artikkeleita on löytynyt useita myös aiemmin löydettyjen lähteiden viittauksien kautta.

2 Simuloidun jäähdytyksen perusteet

2.1 Optimointiongelman ja metaheuristiikan käsitteet

Monet tietojenkäsittelytieteen, tilastotieteen ja muiden tieteenalojen keskeiset ongelmat liittyvät parametrien optimointiin. Optimointiongelmassa pyritään löytämään sellainen parametrien joukko, joka tuottaa mahdollisimman hyvän ratkaisun käsiteltävään ongelmaan. Ratkaisun hyvyyttä mitataan käyttämällä tavoitefunktiota, jonka muuttujina parametrit toimivat ja jolle pyritään löytämään mahdollisimman pieni arvo. [2] Tavoitefunktiosta käytetään myös muita nimityksiä, kuten hyöty-, tappio- tai kustannusfunktio. Hyötyfunktion tapauksessa sen arvoa pyritään maksimoimaan. Tässä tutkielmassa käsitellään ongelmia, joissa tavoitefunktiota pyritään minimoimaan, ellei toisin mainita.

Optimointiongelmat voidaan jakaa diskreetteihin ja jatkuviin ongelmiin niiden muuttujien laadun perusteella. Esimerkki diskreetistä ongelmasta on kauppamatkustajan ongelma, jossa pyritään löytämään lyhin reitti, joka käy läpi joukon kaupunkeja. Esimerkki jatkuvasta ongelmasta on parametrien etsiminen jonkin prosessin numeeriseen malliin siten, että se mallintaa prosessin havainnoitua käyttäytymistä mahdollisimman tarkasti. Kirjallisuudessa viitataan kahdentyyppisiin ongelmiin

vaikeina optimointiongelmina (engl. hard optimization problems): [2]

- Diskreetit optimointiongelmat, joihin ei ole tunnettua täsmällistä polynomisessa ajassa suoriutuvaa algoritmia. NP-täydelliset ongelmat ovat tätä tyyppiä.

- Jatkuvien muuttujien optimointiongelmat, joihin ei ole tunnettua algoritmia, joka varmuudella pystyisi löytämään parhaan mahdollisen ratkaisun äärellisessä ajassa. [2]

Tavoitefunktion arvojoukko on optimointiongelmissa usein epäkonveksi eli sillä on useita ääriarvokohtia, mikä aiheuttaa haasteita optimointimenetelmille [3]. Konveksin joukon tapauksessa paras ratkaisu löydetään helposti gradienttimenetelmää käyttäen. Gradienttimenetelmä (engl. gradient descent) etenee hakuavaruudessa siihen suuntaan, mihin siirtymällä tavoitefunktion arvo laskee suurimmalla nopeudella. Epäkonveksin joukon globaalin minimikohdan löytämiseksi on käytettävä kehittyneempää menetelmää, sillä gradienttimenetelmä ei kykene pakenemaan paikallisesta ääriarvokohdasta, vaan se jää jumiin ensimmäisen löytämänsä hakuavaruuden "laakson" pohjalle. Päästäkseen pois paikallisista ääriarvokohdista menetelmän on kyettävä etenemään hakuavaruudessa "ylämäkeen", eli sellaiseen suuntaan, missä tavoitefunktion arvo kasvaa. [4]

Metaheuristiikat ovat ratkaisutapoja, jotka yhdistävät lokaalin haun menetelmän ja korkean tason strategioita luodakseen prosessin, joka kykenee pakenemaan lokaaleista ääriarvoista ja suorittamaan robustin haun ratkaisuavaruuden läpi [5]. Metaheuristiikat ovat ainakin osittain stokastisia, mikä auttaa löytämään optimaalisia ratkaisuja kombinatorisesti suurista avaruuksista. Usein ne etenevät satunnaiseen suuntaan, joten haun aikana ei tarvitse laskea (joskus vaikeitakin) tavoitefunktion gradientteja. Monet metaheuristiikat perustuvat fysikaalisiin (esim. simuloitu jäähdytys), biologisiin (geneettinen algoritmi) tai etologisiin ("ant colony algorithms")

ilmiöihin[2]. Vaikka metaheuristiikat eivät takaa löytämänsä ratkaisun optimaalisuutta, niin monissa ongelmissa ne kykenevät löytämään paremman ratkaisun kuin deterministiset menetelmät.[5]

2.2 Algoritmin kuvaus

Simuloitu jäähdytys (engl. simulated annealing) tai simuloitu hehkutus[6] on eräs vanhimmista ja tutkituimmista metaheuristiikoista, joka on osoittautunut toimivan tehokkaasti moniin ongelmiin. Se on stokastinen lokaalin haun algoritmi, joka aloittaa jostain alustavasta ratkaisusta ja käy iteratiivisesti läpi senhetkisen ratkaisun naapurustoa. Algoritmi siirtyy aina parempaan ratkaisuun löytäessään sellaisen. Löydettäessä aiempaa huonompi ratkaisu haku päättää siirtymisestä siihen todennäköisyysperusteisesti riippuen ratkaisun huononemisesta ja algoritmin lämpötilaa jäljittelevästä parametristä.[7]

Simuloitu jäähdytys esiteltiin vuonna 1983 julkaisussa *Optimization by Simulated Annealing*[1], jonka kirjoittivat IBM:n statistisen mekaniikan tutkijat Kirkpatrick, Gelatt ja Vecchi. He olivat kiinnostuneet epäjärjestyneiden magneettisten materiaalien matalaenergisistä konfiguraatioista, joiden numeerinen määritys oli vaikea optimointiongelma. Heidän ehdotuksensa oli jäljitellä materiaalitieteissä käytettyä menetelmää nimeltä hehkutus (engl. annealing), jolla jokin materiaali pyritään saamaan alhaisimman energian tilaan.[7] Hehkutuksessa materiaali kuumennetaan sen uudelleenkiteytymislämpötilan yläpuolelle, mikä mahdollistaa atomien liikkumisen kiderakenteessa. Korkeassa lämpötilassa atomit voivat liikkua toisiinsa nähden ja siirtyä hetkittäin suuremman energian tilaan. Tällaisen siirtymän todennäköisyys riippuu kappaleen lämpötilasta ja laskee kappaleen jäähtyessä. Hitaasti jäähtyvässä kappaleessa atomit pyrkivät järjestäytymään globaaliin alhaisimman energian tilaan. Karkaisussa eli nopeassa jäähdytyksessä atomit voivat jäädä jumiin paikalliseen alhaisimman energian tilaan. Hehkutusta käytetään parantamaan metallien

mekaanisia ominaisuuksia.[8]–[10]

Simuloitu jäähdytys soveltaa metallurgian jäähdytyksen prosessia optimointiongelmaan. Matalaenergisen tilan asemasta pyritään minimoimaan tavoitefunktion arvoa käyttämällä apuna fiktiivistä parametriä *lämpötila*, joka on algoritmin hyperparametri. Kirkpatrick perusti algoritminsa aiempaan, vuonna 1953 esitettyyn Metropolis-algoritmiin, jonka voidaan ajatella olevan simuloidun jäähdytyksen muoto, jossa lämpötila pysyy vakiona.[2] Metropolis-algoritmi poikkeaa simuloidusta jäähdytyksestä siten, että se ei pyri löytämään yhtä ratkaisua, vaan sen avulla voidaan havainnoida millä todennäköisyydellä eri konfiguraatiot esiintyvät tietyssä lämpötilassa. Yksinkertaisimmillaan simuloitu jäähdytys voidaan kirjoittaa pseudokoodina seuraavasti:

Ohjelmalistaus 1 Simuloitu jäähdytys pseudokoodina:

```
Syöte: alkutila s, alkulämpötila T,
       maksimi iteraatiot n, jäähdytyskerroin a
Tuloste: haun aikana löydetty paras ratkaisu

paras ratkaisu s* := nykyinen ratkaisu s# := alkutila s
i := 0
while i < n do
   valitse satunnaisesti tilan s# naapurustosta tila t
   laske tavoitefunktion muutos dE = f(t) - f(s#)
   if dE < 0:
      aseta s# := t
   endif
   else if random(0,1) > exp(-dE/T)
      aseta s# := t
   endif
   if f(s#) < f(s*)
      aseta s* := s#
   endif
   T := T * a
   i := i + 1
endwhile
```

Yllä olevasta pseudokoodista nähdään, että algoritmi sisältää ainakin hyperparametrit alkulämpötila T, maksimi iteraatiot n ja jäähdytyskerroin a. Simuloidusta

jäähdytyksestä on olemassa useita erilaisia variaatioita, joita käsitellään aliluvussa 2.3. Suoritus alkaa valitsemalla jokin ratkaisu alkutilaksi s. Seuraavaksi toistetaan silmukkaa, jossa nykyiseen ratkaisuun $s\#$ tehdään jokin pieni muutos, mikä tuottaa uuden ratkaisuehdokkaan t. Tavoitefunktion f arvoja verrataan ratkaisuille $s\#$ ja t. Löydettäessä parempi ratkaisu siirrytään siihen aina, mutta löydettäessä huonompi ratkaisu siirrytään siihen todennäköisyydellä, joka pienenee lämpötilan laskiessa. Lisäksi tavoitefunktion arvojen erotus ratkaisuille vaikuttaa siihen hyväksytäänkö huonompi ratkaisu.

Usein huononevan ratkaisun hyväksymiseen käytetään Metropolis-algoritmin mukaista ehtoa, joka on pseudokoodissa mainittu $p(\Delta E) = e^{-\Delta E/T}$, missä ΔE on tavoitefunktion arvojen erotus ratkaisuille. Lämpötilan ollessa korkea lähestyy $e^{-\Delta E/T}$ arvo yhtä, vaikka ratkaisu olisi merkittävästikin huonompi, eli suuri osa ratkaisuista tulee hyväksytyksi. Vastaavasti lämpötilan lähestyessä nollaa todennäköisyys hyväksyä huononevia ratkaisuja laskee hyvin pieneksi, ja haku muuttuu käytännössä satunnaiseen suuntaan eteneväksi kukkulalle kapuamiseksi (engl. hill climbing algorithm).

2.3 Algoritmin komponentit ja variaatiot

Simuloitu jäähdytys voidaan implementoida useilla tavoilla riippuen ratkaistavasta ongelmasta. Käyttäjän on turvauduttava implementaation kokeelliseen mukauttamiseen, mikäli yleisiä teoreettisia tuloksia parhaasta implementaatiosta ei ole saatavilla. Joidenkin ongelmien tapauksessa saatu ratkaisu ja suoritusaika ovat erittäin herkkiä hyperparametrien säädölle, mikä on simuloidun jäähdytyksen ja muiden metaheuristiikkojen kiistämätön haittapuoli.[2] Tarkastellaan seuraavaksi mihin komponentteihin simuloidun jäähdytyksen implementaation voidaan ajatella jakautuvan. Osassa komponenteista käytetään parametriä k, joka on käyttäjän valitsema lukuarvo.

2.3.1 Alkulämpötila

Alkulämpötila T_0 vaikuttaa suoritukseen siten, että korkeampi alkulämpötila kasvattaa todennäköisyyttä hyväksyä huononevia ratkaisuja etenkin suorituksen alkuvaiheessa. Yksinkertaisin vaihtoehto on asettaa kiinteä alkulämpötila $T_0 = k$. Alkulämpötila voidaan myös suhteuttaa alkutilan s_0 tavoitefunktion f arvoon siten, että $T_0 = k \times f(s_0)$. Muunlaisia vaihtoehtoja on esimerkiksi suorittaa i:n askeleen mittainen satunnaiskulku hakuavaruudessa ja kirjata ratkaisut $s_0, s_1, ..., s_i$. Alkulämpötila voidaan suhteuttaa suurimpaan erotukseen kahden peräkkäisen ratkaisun tavoitefunktioissa.[7]

Alkulämpötila voidaan myös valita siten, että alustavan satunnaiskulun aikana keskimääräinen todennäköisyys hyväksyä huononeva ratkaisu saadaan ennalta päätettyyn arvoon, esimerkiksi $p = 0.5$. Johnson et al. [11] tutkivat graafin partitiointia simuloitua jäähdytystä käyttäen ja havaitsivat, että korkeissa lämpötiloissa suoritetut iteraatiot eivät edesauttaneet merkittävästi paremman ratkaisun löytämisessä. Keskimääräisen hyväksymistodennäköisyyden ollessa yli 40% algoritmi ei konvergoitunut kohti sen parempia ratkaisuja kuin satunnaisesta ratkaisusta aloitettu gradienttimenetelmä.

2.3.2 Tarkasteltujen ratkaisujen määrä lämpötilaa kohden

Yksinkertaisessa implementaatiossa jokaisen lämpötilan kohdalla tarkastellaan yhtä ratkaisua senhetkisen ratkaisun naapurustosta, minkä jälkeen lämpötilaa lasketaan. Algoritmin implementoinnissa on kuitenkin tavallista, että yhdessä lämpötilassa voidaan tehdä useita siirtymiä ratkaisujen välillä. Tarkasteltujen ratkaisujen määrästä käytetään nimeä lämpötilan pituus (engl. temperature length). Lämpötilan pituus L voi olla käyttäjän määrittelemä kiinteä arvo k. Diskreeteissä ongelmissa L voi olla suhteessa alkutilan naapuruston $\mathcal{N}(s_0)$ kokoon siten, että $L = k \times |\mathcal{N}(s_0)|$, ja sitä voidaan myös muuttaa esimerkiksi joka lämpötilan alussa suhteessa senhetkiseen

ratkaisuun.[7] Mikäli hyväksyttyjen siirtymien määrä oletetaan tärkeämmäksi kuin yritettyjen siirtymien, voidaan suorittaa katkaisu (engl. cutoff) jo ennen lämpötilan pituuden täyttymistä, mikäli hyväksyttyjä siirtymiä on tapahtunut riittävästi [12].

2.3.3 Pysäytyskriteeri

Suorituksen pysäyttäminen voi olla kiinteän tai muuttuvan kriteerin mukainen. Yksinkertainen kiinteä pysäytyskriteeri on pseudokoodissa mainittu jäähdytysaskeleiden maksimimäärä, minkä jälkeen suoritus pysähtyy. Muita vaihtoehtoja on pysäyttää suoritus tietyn ajan kuluttua tai lämpötilan laskettua ennalta määritetyn rajan alle.[7] Muuttuva pysäytyskriteeri on sellainen, joka riippuu käynnissä olevasta suorituksesta. Esimerkki tällaisesta on se, että suoritus pysähtyy, kun k ratkaisuehdotusta on hylätty peräkkäin tai viimeisen k ratkaisuehdotuksen hyväksymistodennäköisyys on laskenut ennalta määritetyn lukeman alle. [12]

2.3.4 Tutkimiskriteeri

Uusia ratkaisuja etsitään nykyisen ratkaisun ympäristöstä tutkimiskriteerin (engl. exploration criterion) mukaisesti. Alkuperäisessä algoritmissa Kirkpatrick et al. [1] käyttivät satunnaista tutkimiskriteeriä, eli naapurustosta valittiin satunnainen ratkaisu jokaisessa iteraatiossa. Connolly [13] sovelsi simuloitua jäähdytystä neliöllisen sijoitusongelman ratkaisemiseen. Hän havaitsi satunnaisen tutkimiskriteerin huonoksi, sillä satunnaisuuden vuoksi algoritmi saattoi jäädä ennenaikaisesti jumiin lokaaliin minimikohtaan. Hänen mukaansa seuraavaksi tarkasteltavaa naapuria varten kannatti tutkia koko naapurusto ennalta valitussa järjestyksessä, jolloin satunnaisuus ei voinut estää paranevien siirtymien löytämistä.

2.3.5 Hyväksymiskriteeri

Hyväksymiskriteeri ratkaisee, siirtyykö haku naapurustosta valittuun uuteen ratkaisuun vai jatketaanko hakua uudelleen samasta ratkaisusta. Alkuperäisen version stokastisen hyväksymiskriteerin mukaan haku siirtyy aina parempaan eli pienemmän tavoitefunktion arvon saavaan ratkaisuun löytäessään sellaisen. Huonompaan ratkaisuun eli suuremman tavoitefunktion arvoon siirrytään, mikäli $\xi > \exp\left(\frac{-\Delta E}{T}\right)$, missä $\Delta E = f(t) - f(s\#)$ on tavoitefunktion arvojen erotus ja ξ on satunnaisluku väliltä $[0, 1]$. [1] Ehto tulee Metropolis-algoritmista, jossa aineen todennäköisyys siirtyä toiseen tilaan riippuu tilojen potentiaalienergioiden erotuksesta ja aineen lämpötilasta [14].

Eksponenttifunktion laskemiseen kuluvaa aikaa voidaan mahdollisesti säästää approksimoimalla sitä funktiolla $1 - \Delta E/T$ tai taulukoimalla todennäköisyyksiä eri ΔE arvoille ja pyöristämällä algoritmissa saatua arvoa lähimpään taulukoituun arvoon. Tästä saatava hyöty riippuu käytettävän laitteiston prosessoinnin ja muistinkäsittelyn nopeudesta. Vuonna 1989 julkaistussa tutkimuksessa havaittiin molempia tapoja käyttäen säästyvän noin kolmasosa suoritusajasta ilman, että loppuratkaisussa tapahtui havaittavaa heikkenemistä. [12]

Hyväksymiskriteeri voi olla myös deterministinen. Tällaisissa variaatioissa algoritmi ei sisällä lämpötilaparametria. Threshold acceptance -variaatio hyväksyy huononevat ratkaisut aina, mikäli $\Delta E \leq \bar{\phi}$, missä $\bar{\phi}$ on parametri, jonka arvo vähenee haun aikana lämpötilan kaltaisesti. Great deluge -variaatiossa ratkaisuja ei verrata aiempiin löydettyihin ratkaisuihin, vaan parametriin $\bar{\phi}_k$, jonka arvo laskee haun edetessä. Variaation kehittäjä Dueck [15] kuvasi artikkelissaan great deluge -hakua henkilöksi, joka kulkee maastossa, jossa sade nostaa hitaasti veden pintaa. Henkilö kulkee vettä välttäen, mutta satunnaisesti niin kauan, kunnes veden pinta tavoittaa tämän, milloin todennäköisesti henkilö on päätynyt korkealle maastonkohdalle. Toinen Dueckin kehittämä variaatio on record-to-record travel, jossa hyväksytään

ratkaisu vain, mikäli se on korkeintaan parametrin γ verran huonompi, kuin haussa aiemmin löydetty paras ratkaisu. [15]

2.3.6 Jäähdytyskaava

Jäähdytyskaava (engl. cooling scheme) hallitsee lämpötilan päivittämistä haun edetessä. Tyypillisesti lämpötila laskee monotonisesti suorituksen aikana, mikä vähentää huononevien siirtymien hyväksymisen mahdollisuutta ja muuntaa algoritmia kohti gradienttimenetelmää. Kirjallisuudessa on olemassa myös variaatioita, joissa lämpötila voi nousta ajoittain. Algoritmin historiassa jäähdytyskaava on ollut tutkituin komponentti sekä teoreettisessa että kokeellisessa näkökulmassa. [7]

Kirkpatrickin alkuperäisessä algoritmissa käytettiin geometrista jäähdytystä, jossa alkulämpötila $T_0 = 10$ ja $T_{i+1} = 0.9 \times T_i$ [1]. Geman ja Geman [16] ehdottivat logaritmista jäähdytystä, missä lämpötila hetkellä k toteuttaa yhtälön $T(i) = \frac{c}{\log(1+i)}$ ja c on vakio. Kolmas paljon käytetty variaatio on Lundy-Mees -kaava, missä $T_{i+1} = \frac{T_i}{1+\delta_{LM}T_i}$ ja δ_{LM} on jäähdytyksen nopeutta kontrolloiva parametri [17].

Jäähdytys voi olla toteutettu aritmeettisella kaavalla, milloin lämpötila laskee joka iteraatiossa jonkin vakion a verran. Lämpötila voi myös pysyä vakiona suorituksen aikana, jolloin algoritmi toimii Metropolis-algoritmin kaltaisesti. Kirjallisuudessa tähän viitataan myös nimellä staattinen simuloitu jäähdytys tai yleistetty kukkulalle kapuaminen (engl. generalized hill climbing). Old Bachelor Acceptance -variaatiossa lämpötila laskee aina hyväksyttäessä ja nousee hylättäessä ratkaisu. Algoritmista on myös esitetty variaatiota, joissa lämpötilan laskettua tarpeeksi alas se nostetaan takaisin alkuperäiseen tai esimerkiksi arvoon, jossa toistaiseksi paras ratkaisu on löytynyt. [7]

2.3.7 Alkutila ja naapurusto

Edellä mainitut komponentit ovat algoritmille geneerisiä tekijöitä, joita voidaan soveltaa riippumatta siitä mitä ongelmaa algoritmilla pyritään ratkaisemaan. Näiden lisäksi implementoidessa algoritmia on määriteltävä kaksi ratkaistavalle ongelmalle ominaista komponenttia. Ensimmäinen näistä on alkutila eli ratkaisu, josta haku lähtee liikkeelle. Usein alkutila valitaan satunnaisesti. Myös paremman alkutilan käyttämistä on tutkittu, mutta on havaittu, ettei se takaa paremman ratkaisun löytymistä tai lyhyempää suoritusaikaa. [12]

Toinen ongelmalle ominainen komponentti on naapurusto $\mathcal{N}$, eli se ratkaisujen joukko, josta valitaan ehdokas haun uudeksi pisteeksi. Naapuruston rakenteen valinta määrittelee mahdolliset siirtymät, joita haku voi tehdä, joten se vaikuttaa merkittävästi konvergoitumisen nopeuteen. Liian maltillinen naapuruston rakenne tekee hakuavaruuden tutkimisesta hidasta. Liian vapaa rakenne mahdollistaa sen, että haku hyppää yli tärkeistä minimikohdista ja muuttuu sokeaksi satunnaishauksi. Erilaiset rakenteet voivat olla eduksi haun eri vaiheissa. Alkuvaiheessa vapaa rakenne mahdollistaa nopean ja karkean haun suorittamisen, kun taas loppuvaiheessa maltillinen rakenne edesauttaa minimikohdan löytämisessä. [18]

Diskreeteissä ongelmissa naapuruston muodostavat usein ne ratkaisut, jotka ovat saavutettavissa tekemällä yksi muutos konfiguraatioon. Esimerkiksi kauppamatkustajan ongelmassa tällainen muutos olisi vaihtaa kahden peräkkäisen kaupungin paikkaa reitillä. Mikäli naapuruston halutaan olevan suurempi, voidaan sallia myös muiden kuin perättäisten kaupunkien paikkojen vaihto tai vaihtaa useamman kuin kahden kaupungin paikkaa kerralla. Jatkuvassa ongelmassa naapuruston määrittely tai koko eivät ole yhtä selkeitä kuin diskreetissä. Optimoitaessa jatkuvia parametreja voidaan esimerkiksi yhtä tai useampaa parametria muuttaa standardinormaalijakaumasta poimitun luvun verran.

3 Simuloidun jäähdytyksen eräitä käyttökohteita

3.1 Kauppamatkustajan ongelma

Klassisista optimointiongelmista kauppamatkustajan ongelma on hyvin laajasti tutkittu ongelma. Kauppamatkustajan ongelma on paljon käytetty siksi, että se on helppo muotoilla, mutta samaan aikaan erittäin vaikea ratkaista. Ongelman suurimmat versiot, joiden optimiratkaisu on löydetty ja todistettu sisältävät joitakin tuhansia kaupunkeja.[2]

Useissa simuloidun jäähdytyksen eri variaatioita tutkivissa julkaisuissa käytetään kauppamatkustajan ongelmaa vertailemaan algoritmin tehokkuutta muihin vastaaviin algoritmeihin. Kirkpatrick et al. käyttivät niin ikään kauppamatkustajan ongelmaa esimerkkinä julkaistessaan algoritmin alkuperäisen version vuonna 1983. [1]

Kauppamatkustajan ongelmassa on annettuna N kaupunkia sekä jokaisen kahden kaupungin välillä kulkemisen hinta. Ratkaistakseen ongelman on löydettävä sellainen kauppamatkustajan reitti, joka vierailee jokaisessa kaupungissa ja palaa lähtöpisteeseen siten, että reitin kokonaiskustannus on mahdollisimman pieni. Deterministiset metodit ongelman eksaktiin ratkaisuun vaativat laskentatehoa, jonka tarve kasvaa eksponentiaalisesti N:n kasvaessa. Näin ollen käytännössä tarkkaa ratkaisua voidaan yrittää vain korkeintaan joidenkin tuhansien kaupunkien ongelmaan.

Mahdollisten ratkaisujen määrä N kaupungin tapauksessa on $(N-1)!/2$. Ongelma kuuluu NP-täydellisten ongelmien joukkoon. [1], [19]

Ongelmalle on useita käytännön sovelluskohteita. Esimerkiksi piirilevyn porauksessa koneen pään täytyy porata levyyn erikokoisia reikiä ja toisinaan käydä vaihtamassa poranterä telineestä. Porattavat reiät edustavat ongelmassa kaupunkeja ja niiden välillä siirtymiseen kuluva aika edustaa kustannusta. Toinen esimerkki käytännön sovelluksesta on varastolla poimittavan tilauksen ongelma. Varaston saadessa tilauksen jonkin ajoneuvon on kerättävä kaikki tilatut tuotteet voidakseen lähettää ne asiakkaalle. Lyhimmän reitin ratkaisu tuotteiden keräämiseksi edustaa kauppamatkustajan ongelmaa. Muita esimerkkejä löytyy muun muassa kaasuturbiinimoottoreiden huollosta, röntgenkristallografiasta ja komponenttien asettelusta tietokoneen emolevylle. [19]

Usean kauppamatkustajan ongelmassa on yhden matkustajan sijaan m matkustajaa, joiden on yhdessä vierailtava jokaisessa kaupungissa. Tälle variaatiolle on myös olemassa useita käytännön sovelluksia. [19] On siis selvää, että ongelma ei ole pelkästään teoreettinen suorituskykytesti algoritmeille, vaan sen perustapauksen ja eri variaatioiden ratkaisemiselle on myös käytännön perusteita.

Kirkpatrick et al. tutkivat kauppamatkustajan ongelmaa, jossa N kaupunkia sijaitsevat satunnaisesti N sivuisen neliön alueella. Tässä tapauksessa on mahdollista osoittaa, että optimaalisella reitillä kahden kaupungin välinen keskimääräinen etäisyys α on riippumaton N:stä. Vastatakseen paremmin todellisen maailman kaupunkeja kokeessa kaupungit eivät sijainneet täysin satunnaisesti neliön alueella, vaan yhdeksässä tasaisesti sijoitetussa ryppäässä, joiden välissä oli tyhjiä aukkoja. Tutkimuksessa verrattiin eroa α:ssa, mikäli käytettiin ahnetta heuristiikkaa tai simuloitua jäähdytystä. Ahneessa heuristiikassa reitin seuraavaksi pisteeksi valitaan lähin piste, joka ei ole vielä reitillä. [1]

Tutkimuksessa havaittiin, että ahnetta heuristiikkaa käyttäen kahden kaupungin

välinen etäisyys reitillä oli keskimäärin 1.12. Simuloitua jäähdytystä käyttämällä taas saatiin $\alpha \leq 0.95$ jopa $N = 6000$ kaupungin tapauksissa. Julkaisuvuonna 1983 suurin todistetusti ratkaistu kauppamatkustajan ongelma oli kooltaan $N = 318$, johon verrattuna algoritmin löytämiä ratkaisuja voidaan pitää edistyneinä. Naapuruston rakenteena menetelmässä käytettiin kahden perättäisen kaupungin paikkojen vaihtamista keskenään. [1]

Tuoreemmassa tutkimuksessa vuodelta 2009 Geng, Chen, Yang et al. [20] yhdistivät adaptiivisen simuloidun jäähdytyksen ahneeseen hakuun (engl. Adaptive simulated annealing with greedy search, ASA-GS) ja käyttivät tätä kauppamatkustajan ongelman ratkaisuun. Julkaisussa tutkittiin kolmen eri mutaation käyttöä reitin muuntamiseksi: yhden kaupungin siirtämistä reitillä, usean kaupungin lohkon siirtämistä reitillä sekä usean kaupungin lohkon kääntämistä reitillä.

Tutkijoiden mukaan tavallinen simuloitu jäähdytys vaatii hyvän ratkaisun löytämiseksi valtavasti aikaa. Nopeamman konvergoitumisen vuoksi he päättivät hyödyntää ahneen haun tekniikkaa saavuttaakseen kompromissin laskenta-ajan, ratkaisun laadun ja implementaation kompleksisuuden välillä. Ahneen haun tekniikka implementoitiin siten, että huononevia ratkaisuja hylättiin vähintään parametrin m verran, ennen kuin sellaiseen siirtyminen saattoi tapahtua hyväksymiskriteerin mukaisesti. Suurempi m:n arvo tekee algoritmista ahneemman, kun taas $m = 0$ vastaa perinteistä simuloitua jäähdytystä. [20]

Tutkimuksessa verrattiin ASA-GS:n suoriutumista 60 tunnetun kauppamatkustajan ongelman instanssissa, joiden koko vaihteli 51 ja 85900 kaupungin välillä. Jokaiseen instanssiin suoritettiin 5 hakua. Keskimäärin algoritmin löytämä ratkaisu oli 1.33% pidempi kuin lyhin tunnettu reitti. Kahdessa tapauksessa algoritmi löysi aiempaa parasta tunnettua ratkaisua paremman ratkaisun ja kolmessa muussa tapauksessa se vastasi aiemmin tunnettua parasta ratkaisua. [20]

Lisäksi verrattiin algoritmin suoritusaikaa yhdeksän muun kehittyneen algorit-

min kanssa. Suurimmassa osassa tapauksia ASA-GS kykeni löytämään paremman ratkaisun kuin muut algoritmit. Kolmen muun (memetic neural network, constructive-optimizer neural network ja generalized chromosome genetic algorithm) algoritmin konvergoitumisnopeus oli kuitenkin huomattavasti ASA-GS:ää nopeampi. [20]

3.2 Neliöllinen sijoitusongelma

Neliöllinen sijoitusongelma (engl. Quadratic assigment problem, QAP) on toinen optimointiongelma, johon simuloitua jäähdytystä on sovellettu useissa tutkimuksissa. Sen ratkaisemiseksi on sijoitettava n objektia n:n sijaintiin optimaalisesti. Jokaiselle objektien parille on määritelty niiden välillä kulkeva virtaus. Objektit tulee sijoittaa siten, että summa niiden välillä kulkevista virtauksista kerrottuna niiden välisillä etäisyyksillä saadaan minimoitua. [21]

Neliöllinen sijoitusongelma todistettiin NP-täydelliseksi vuonna 1976 ja se on eräs vaikeimmin ratkaistavista ongelmista tässä joukossa. Yleisesti $n > 30$ kokoisia ongelmia ei ole mahdollista ratkaista eksaktisti hyväksyttävässä ajassa. Sillä voidaan mallintaa monia reaalimaailman ongelmia useilla aloilla, kuten laitteistojen sijoitusta ja hajautettua samanaikaista laskentaa. Lisäksi muita ongelmia kuten kauppamatkustajan ongelmaa ja graafin partitiointia voidaan kuvantaa neliöllisenä sijoitusongelmana. Suurin ongelman instanssi, jota käytetään algoritmien testaamiseen, on tällä hetkellä kokoluokkaa $n = 150$. [21], [22]

Vuonna 1984 Burkard ja Rendl [23] sovelsivat ensimmäisenä simuloitua jäähdytystä neliöllisen sijoitusongelman ratkaisemiseksi ja onnistuivat löytämään erinomaisia ratkaisuja. Heidän käyttämässään algoritmissa valittiin satunnaisesti kaksi objektia, joiden paikat vaihdettiin keskenään. Lyhyessä suoritusajassa he löysivät ratkaisuja, jotka olivat vain 1-2% huonompia kuin siihen aikaan parhaat tunnetut ratkaisut. Suorittamalla ohjelmaa useita kertoja eri alkuarvoilla ohjelma löysi kaikki tunnetut tavoitefunktion minimiarvot.

Connolly [13] pyrki parantamaan simuloidun jäähdytyksen alkuperäistä versiota neliöllisen sijoitusongelman ratkaisemiseksi. Hänen mukaansa algoritmissa seuraavaa tarkasteltavaa ratkaisua ei ollut tehokasta valita satunnaisesti. Hän esitti, että potentiaaliset vaihdettavat objektiparit tutkittaisiin järjestyksessä $(1,2),(1,3),...,(1,n),(2,3),...,(n-1,n)$. Näin paranevia ratkaisuja ei jäisi huomioimatta satunnaisuuden vuoksi.

Connolly esitti myös optimaalisen ratkaisun löytämiseksi erilaista jäähdytyskaavaa, jossa päätettiin ennalta ajon aikana tapahtuvien vaihtojen määrä. Algoritmia kokeiltiin 100 kertaa tunnettuihin testiongelmiin, joiden koot olivat väliltä $n = 15$ ja $n = 100$. Algoritmi löysi useita aiempaa parasta julkaistua ratkaisua parempia ratkaisuja etenkin 50:n ja 100:n ongelman instansseissa ja ratkaisu oli lähes aina korkeintaan 1% huonompi kuin aiemmin tunnettu paras ratkaisu.[13]

Vuoden 2012 julkaisussaan Wang [24] tutki simuloidun jäähdytyksen yhdistämistä tabu-hakuun (engl. tabu search) neliöllisen sijoitusongelman ratkaisemiseksi. Hän esitti, että muistittomuutensa vuoksi simuloitu jäähdytys ei hyödynnä hakuhistoriaa ja paikallisista minimikohdista pakeneminen voi olla aikaa vievää etenkin alhaisissa lämpötiloissa. Tabu-haussa aiemmin vierailtuja ratkaisuja lisätään tabu-listalle, eikä niihin palata enää uudelleen.

Lisäksi Wangin algoritmissa käytettiin hajautettua uudelleenkäynnistystä. Systeemin lämpötilaa nostettiin, mikäli paranevia ratkaisuja ei löytynyt tietyssä määrässä iteraatiota. Samaan aikaan haku siirrettiin jatkumaan hieman eri pisteessä siten, että korkean virtauksen objektit säilytettiin paikoillaan ja matalan virtauksen objektien paikkoja vaihdettiin. Tutkiakseen kokeellisesti algoritminsa tehokkuutta Wang etsi sillä ratkaisuja tunnettuihin neliöllisen sijoitusongelman instansseihin, joiden koko vaihteli välillä 15 - 100. Kokeelliset tulokset osoittivat, että hajautettu uudelleenkäynnistys auttoi paikallisista minimikohdista pakenemisessa eikä hidastanut konvergoitumista merkittävästi. [24]

Simuloidun jäähdytyksen käytön tutkimus jatkuu edelleen nykyaikana. Vuoden 2018 julkaisussa Sonuc, Sen ja Bayir [25] tutkivat algoritmin käyttöä rinnakkaislaskennassa CUDA (Compute Unified Device Architecture) -arkkitehtuuria hyödyntäen. CUDA on NVIDIA:n kehittämä rinnakkaislaskennan alusta, jota käyttäen rinnakkaislaskentaa voidaan suorittaa grafiikkasuorittimilla. Tutkimuksessa grafiikkasuorittimen eri säikeet aloittivat hakualgoritmin eri permutaatioista ja sen tavoitteena oli kehittää tehokas tapa kommunikoida säikeiden välillä haun aikana. Säikeet kommunikoivat ajoittain keskenään, minkä jälkeen kaikissa säikeissä jatkettiin hakua parhaan löydetyn permutaation ympäristöstä.

Algoritmia testattiin neliöllisen sijoitusongelman 132:lla eri instanssilla, joiden koko vaihteli välillä 10 - 100. Tutkimuksessa havaittiin, että ongelman koon kasvaessa GPU-laskenta toimi selvästi perinteistä CPU-laskentaa tehokkaammin. $n = 100$ instanssissa GPU-laskennan nopeus oli 13 kertaa CPU-laskentaa parempi. Algoritmi löysi parhaan tunnetun ratkaisun 113:ssa tapauksessa, joista 99:ssä tapauksessa suoritusaika oli alle 60 sekuntia. Keskimäärin löydetyt ratkaisut olivat 0.09% huonompia, kuin paras tunnettu ratkaisu. [25]

3.3 Neuroverkon rakenteen optimointi

Tekoälytutkimuksessa neuroverkot soveltuvat moniin haastaviin tehtäviin, kuten kuvantunnistukseen. Neuroverkon suunnittelu tiettyä ongelmaa varten on haastava tehtävä, joka vaatii asiantuntemusta sekä aikaa vievää yritystä ja erehdystä. Suunnittelua automatisoivien metodien kehittäminen perustuu yleisesti optimointialgoritmeihin ja metaheuristiikkoihin. [26]

Kuo, Kuruoglu ja Chan [18] tutkivat simuloidun jäähdytyksen käyttöä neuroverkon rakenteen optimoinnissa. Heidän tutkimuksensa mukaan suurten neuroverkkojen ongelma on yliparametrisointi, mikä rajoittaa niiden käyttöä laitteistoissa, joilla ei ole riittävää laskentatehoa tai muistikapasiteettia neuroverkon vaatiman lasken-

nan suorittamiseksi. Julkaisussaan he pyrkivät osoittamaan, että neuroverkon rakennetta on mahdollista optimoida karsimalla sitä huomattavasti ilman, että sen suorituskyky kärsii merkittävästi. Rakenteen optimointiin he käyttivät simuloitua jäähdytystä.[18]

Prosessi aloitettiin hyvin koulutetusta täysin kytketystä neuroverkosta. Algoritmin suorituksen alussa passivoitiin satunnaisesti jokin ennalta päätetty määrä neuronien välisiä yhteyksiä. Tämän jälkeen ratkaisuja etsittiin naapurustosta siten, että valittiin yksi aktiivinen ja yksi passiivinen neuronien välinen linkki ja vaihdettiin niiden toiminnan statukset. Tämän jälkeen tarkasteltiin hyväksymiskriteerin mukaisesti hyväksytäänkö uusi konfiguraatio. [18]

Kokeellisessa tutkimuksessa käytettiin MNIST- ja FASHION -tietokantoja ja kuvantunnistusneuroverkkoa. Tutkimus osoitti, että simuloitu jäähdytys kykeni palauttamaan suurimman osan karsimisessa menetetystä suorituskyvystä. Parhaassa tapauksessa 90%:sti karsitun neuroverkon tapauksessa suorituskykyä menetettiin vain 4%. Johtopäätöksenä tutkijat pitivät, että simuloitua jäähdytystä käyttäen optimointiprosessi voidaan suorittaa ilman, että se vaatii suurta laskentatehoa. [18]

4 Yhteenveto

Tämän tutkielman tavoitteena oli luoda lukijalle yleinen kuva siitä, millainen algoritmi simuloitu jäähdytys on ja mihin tarkoituksiin sitä voidaan käyttää. Tavoitteeseen pyrittiin koostamalla tietoa algoritmia käsittelevistä artikkeleista sekä kirjoista. Tieto pyrittiin esittämään loogisessa muodossa sellaisella tavalla, että sen ymmärtämiseksi riittää tietämys algoritmiikan ja tietojenkäsittelyn perusteista. Johdantoluvussa esiteltiin aihe ja tutkimuskysymykset, toisessa luvussa käsiteltiin algoritmin perusteet sekä variaatiot ja kolmannessa luvussa tarkasteltiin millaisiin tarkoituksiin algoritmia on käytetty. Lähteitä käytettiin noin 40 vuoden ajalta algoritmin kehitysvuodesta 1983 nykyhetkeen.

Ensimmäiseen tutkimuskysymykseen vastattiin aliluvuissa 2.2 ja 2.3 käymällä läpi algoritmin perusteet lähtien alkuperäisestä implementaatiosta ja esittelemällä eri lähteissä käytettyjä variaatioita algoritmista. Tutkielmassa ei käsitelty algoritmin kaikkia variaatioita, sillä lähes jokaisessa julkaisussa implementaatiot poikkeavat jollain tavalla toisistaan. Toiseen tutkimuskysymykseen saatiin joitakin vastauksia aliluvussa 2.3 sekä luvussa 3. Vastaus tutkimuskysymykseen on kuitenkin se, että hyperparametrien vaikutus algoritmin toimintaan riippuu suuresti implementaatiosta sekä ratkaistavasta ongelmasta.

Kolmanteen tutkimuskysymykseen saatiin vastaukseksi se, että algoritmia voidaan käyttää useisiin kombinatorisiin optimointiongelmiin etenkin silloin, kun koko hakuavaruuden läpikäynti ei ole hyväksyttävä vaihtoehto. Algoritmin suorituskyky

oli viitatuissa lähteissä usein lähes yhtä hyvä tai parempi kuin muiden tutkittujen algoritmien, mutta parhaan implementaation ja hyperparametrien arvojen löytäminen oli usein ollut työläs tehtävä. Lukijan pohdittavaksi jää se, antaako kirjallisuus todellista paremman kuvan algoritmin suorituskyvystä, mikäli vain onnistuneet tutkimukset päätyvät julkaisuun.

Tässä tutkielmassa keskityttiin koostamaan tietoa aiemmin tehdystä tutkimuksesta. Jatkotutkimusta aiheesta voisi tehdä esimerkiksi tekemällä omia implementaatioita algoritmista ja käyttämällä niitä ratkaisemaan suorituskykytesti-instansseja mm. kauppamatkustajan ongelmasta ja neliöllisestä sijoitusongelmasta. Useimmissa lähteissä on ilmoitettu tutkimuksessa käytetty laitteisto (CPU/GPU), joten omia tuloksia olisi mahdollista verrata suoraan aiempaan tutkimukseen. Tällä tavalla olisi myös mahdollista nähdä suoraan, millä tavalla hyperparametrien manipulointi vaikuttaa algoritmin suoritukseen.

Lähdeluettelo

[1] S. Kirkpatrick, C. D. Gelatt ja M. P. Vecchi, "Optimization by Simulated Annealing", *Science*, vol. 220, nro 4598, s. 671–680, 1983. DOI: 10.1126/science.220.4598.671. eprint: https://www.science.org/doi/pdf/10.1126/science.220.4598.671. url: https://www.science.org/doi/abs/10.1126/science.220.4598.671.

[2] P. Siarry, *Metaheuristics*, eng. Cham: Springer International Publishing AG, 2017, ISBN: 3319454013.

[3] E. H. Houssein, M. R. Saad, F. A. Hashim, H. Shaban ja M. Hassaballah, "Lévy flight distribution: A new metaheuristic algorithm for solving engineering optimization problems", *Engineering Applications of Artificial Intelligence*, vol. 94, s. 103 731, 2020, ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2020.103731. url: https://www.sciencedirect.com/science/article/pii/S0952197620301482.

[4] C. Tsallis ja D. A. Stariolo, "Generalized simulated annealing", eng, *Physica A*, vol. 233, nro 1, s. 395–406, 1995, ISSN: 0378-4371.

[5] F. W. Glover ja G. A. Kochenberger, *Handbook of Metaheuristics* (International Series in Operations Research and Management Science), eng, 1. painos. Boston, MA: Springer, 2003, vol. 57, ISBN: 9781402072635.

[6] O. Kivinen, *Katsaus modulaaristen piirilevyladontakoneiden käyttöön liittyvistä optimointiongelmista*, fin, Luonnontieteiden ja tekniikan tiedekunta, 2019-04-24.

[7] A. Franzin ja T. Stützle, "Revisiting simulated annealing: A component-based analysis", eng, *Computers and operations research*, vol. 104, s. 191–206, 2019, ISSN: 0305-0548.

[8] M. Andresen, "Using Simulated Annealing for Open Shop Scheduling with Sum Criteria", eng, teoksessa *Simulated Annealing*, C. Ming Tan ja C. M. Tan, toim., IntechOpen, 2008, ISBN: 9789537619077.

[9] "Annealing (metallurgy)", Viitattu 25.10.2023. url: `https://www.chemeurope.com/en/encyclopedia/Annealing_%5C%28metallurgy%5C%29.html`.

[10] G. R. Karimi ja A. A. Verki, "Mean Field Annealing Based Techniques for Resolving VLSI Automatic Design Problems", eng, teoksessa *Simulated Annealing : Single and Multiple Objective Problems*, M. de Sales Guerra Tsuzuki, toim., IntechOpen, 2012, ISBN: 9789535107675.

[11] D. S. Johnson, C. R. Aragon, L. A. McGeoch ja C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning", eng, *Operations research*, vol. 39, nro 3, s. 378–406, 1991, ISSN: 0030-364X.

[12] D. S. Johnson, C. R. Aragon, L. A. McGeoch ja C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning", eng, *Operations research*, vol. 37, nro 6, s. 865–892, 1989, ISSN: 0030-364X.

[13] D. T. Connolly, "An improved annealing scheme for the QAP", eng, *European journal of operational research*, vol. 46, nro 1, s. 93–100, 1990, ISSN: 0377-2217.

[14] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller ja E. Teller, "Equation of State Calculations by Fast Computing Machines", *The Journal of Chemical Physics*, vol. 21, nro 6, s. 1087–1092, kesäkuu 1953, ISSN: 0021-9606. DOI: 10.1063/1.1699114. eprint: https://pubs.aip.org/aip/jcp/article-pdf/21/6/1087/8115285/1087_1_online.pdf. url: https://doi.org/10.1063/1.1699114.

[15] G. Dueck, "New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel", eng, *Journal of computational physics*, vol. 104, nro 1, s. 86–92, 1993, ISSN: 0021-9991.

[16] S. Geman ja D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images", eng, *IEEE transactions on pattern analysis and machine intelligence*, vol. PAMI-6, nro 6, s. 721–741, 1984, ISSN: 0162-8828.

[17] M. Huber, "Structural optimization of vapor pressure correlations using simulated annealing and threshold accepting: Application to R134a", eng, *Computers and chemical engineering*, vol. 18, nro 10, s. 929–932, 1994, ISSN: 0098-1354.

[18] C. L. Kuo, E. E. Kuruoglu ja W. K. V. Chan, "Neural Network Structure Optimization by Simulated Annealing", eng, *Entropy (Basel, Switzerland)*, vol. 24, nro 3, s. 348–, 2022, ISSN: 1099-4300.

[19] D. Davendra, *Traveling salesman problem, theory and applications*, eng. Rijeka, Crotia: IntechOpen, 2010, ISBN: 953-51-5501-6.

[20] X. Geng, Z. Chen, W. Yang, D. Shi ja K. Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search", eng, *Applied soft computing*, vol. 11, nro 4, s. 3680–3689, 2011, ISSN: 1568-4946.

[21] I. V. Sergienko, V. P. Shylo, S. V. Chupov ja P. V. Shylo, ”Solving the Quadratic Assignment Problem”, eng, *Cybernetics and systems analysis*, vol. 56, nro 1, s. 53–57, 2020, ISSN: 1060-0396.

[22] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn ja T. Querido, ”A survey for the quadratic assignment problem”, eng, *European journal of operational research*, vol. 176, nro 2, s. 657–690, 2007, ISSN: 0377-2217.

[23] R. Burkard ja F. Rendl, ”A thermodynamically motivated simulation procedure for combinatorial optimization problems”, eng, *European journal of operational research*, vol. 17, nro 2, s. 169–174, 1984, ISSN: 0377-2217.

[24] J.-C. Wang, ”A Multistart Simulated Annealing Algorithm for the Quadratic Assignment Problem”, eng, teoksessa *2012 Third International Conference on Innovations in Bio-Inspired Computing and Applications*, IEEE, 2012, s. 19–23, ISBN: 1467328383.

[25] E. Sonuc, B. Sen ja S. Bayir, ”A cooperative GPU-based Parallel Multistart Simulated Annealing algorithm for Quadratic Assignment Problem”, eng, *Engineering science and technology, an international journal*, vol. 21, nro 5, s. 843–849, 2018, ISSN: 2215-0986.

[26] L. Di Gaspero, P. Festa, A. Nakib ja M. Pavone, ”On Optimizing the Structure of Neural Networks Through a Compact Codification of Their Architecture”, eng, teoksessa *Metaheuristics*, sarja Lecture Notes in Computer Science, vol. 13838, Switzerland: Springer International Publishing AG, 2023, s. 133–142, ISBN: 9783031265037.