Jyri Leskinen

Memeettisistä algoritmeista

Tietotekniikan
pro gradu -tutkielma
19. kesäkuuta 2007

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Jyri Leskinen
Yhteystiedot: jyril@cc.jyu.fi
Työn nimi: Memeettisistä algoritmeista
Title in English: On Memetic Algorithms
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 85
Tiivistelmä: Memeettiset algoritmit ovat uudentyyppinen evoluutioalgoritmien luokka, jossa perinteiseen evoluutioalgoritmiin yhdistetään paikallisia hakumenetelmiä. Tutkielma keskittyy kauppamatkustajan ongelmalle kehitettyihin memeettisiin algoritmeihin. Käytännön osuudessa osoitetaan, että memeettinen algoritmi voi olla erillisiä heuristiikkoja tehokkaampi.
English abstract: Memetic algorithms are a new class of evolutionary algorithms, where traditional evolutionary algorithms are hybridized with local search methods. This thesis concentrates on memetic algorithms for Traveling salesman problem (TSP). In the experiemental section it is shown that a memetic algorithm can be more efficient than the separate heuristics alone.
Avainsanat: memeettiset algoritmit, evoluutioalgoritmit, paikallishaku, kauppamatkustajan ongelma
Keywords: memetic algorithms, evolutionary algorithms, local search, traveling salesman problem

Sanasto

ahne haku – greedy search
aikatauluttaja – scheduler
ei-perättäinen neljän kaaren vaihto – non-sequential four edge exchange
epistaasi – epistasis
esitysmuoto – representation
etsintä, diversifikaatio – exploration, diversification
evoluutioalgoritmi – evolutionary algorithm (EA)
evoluutiolaskenta – evolutionary computation (EC)
evoluutio-ohjelmointi – evolutionary programming (EP)
evoluutiostrategiat – evolutionary strategies (ES)
fenotyyppiavaruus, ratkaisuavaruus – phenotype space, solution space
geneettinen algoritmi – genetic algorithm (GA)
geneettinen ohjelmointi – genetic programming (GP)
generointifunktio – generating function
genotyyppiavaruus – genotype space
graafi – graph
heuristiikka – heuristics
hienosäätöaikatauluttaja – fine-grain scheduler
hyödyntäminen, tehostaminen – exploitation, intensification
iteroitu paikallishaku – iterated local search (ILS)
itsesopeutuva meemi – self-adaptive meme
jyrkimmän laskun menetelmä – steepest descent method
kaarenvaihto – edge exchange
kaari – edge
karkeasäätöaikatauluttaja – coarse-grain scheduler
kartoitusfunktio – mapping function
kauppamatkustajan ongelma – traveling salesman problem (TSP)
kaupunki, solmu – city, node, vertex
kelpoisuus – fitness
kelpoisuusfunktio, testifunktio, objektifunktio – fitness function, evaluation function, objective function
kelpoisuusmaasto – fitness landscape

kelpoisuuteen perustuva valinta – fitness-proportionate selection
kombinatorinen optimointiongelma – combinatorial optimization problem (COP)
kromosomi, genotyyppi – chromosome, genotype
lähimmän naapurin menetelmä – nearest neighbour method
meemi – meme
memeettinen algoritmi – memetic algorithm (MA)
meta-aikatauluttaja – metascheduler
metaheuristiikka – metaheuristic
mutaatio – mutation
muuttuja, lokus, geeni – variable, locus, gene
naapurusto – neighbourhood
NFL-teoreema – No Free Lunch theorem
NP-kova, NP-vaikea – Non-deterministic Polynomial time hard, NP-hard
NP-täydellinen – NP-complete
ohjattu paikallishaku – guided local search (GLS)
paikallinen haku, paikallishaku – local search (LS)
paremmuusjärjestykseen perustuva valinta – rank-based selection
parinvalinta – parent selection, mating selection
populaatio – population
päivitysfunktio – updating function
risteytys, rekombinaatio – crossover, recombination
selviytymisvalinta – survival selection
simuloitu jäähdytys – simulated annealing (SA)
sopeutuva meemi – adaptive meme
tabuhaku – tabu search (TS)
turnajaisvalinta – tournament selection
vakaan tilan valinta – steady state selection
vetovoima-alue – basin of attraction
yksilö, fenotyyppi, ratkaisu(ehdokas) – individual, phenotype, solution (candidate)

Sisältö

1 Johdanto

Jokapäiväisessä elämässä kohdataan usein tilanteita, joissa joudutaan tekemään valintoja kahden tai useamman laadultaan erilaisen vaihtoehdon välillä. Automatkaa suunniteltaessa yleensä pyritään valitsemaan välimatkaltaan lyhin reitti. Ostosta tehtäessä on usein järkevintä ostaa tuote, joka on hinta-laatusuhteeltaan paras. Nämä esimerkit ovat tyypillisiä arkipäivän optimointiongelmia. Niitä esiintyy kaikkialla, joten optimoinnilla on suuri merkitys jokapäiväisessä elämässä.

Useat optimointiongelmat ovat sellaisia, että niitä voidaan tutkia matemaattisesti. Muodostettu matemaattinen malli voidaan antaa tietokoneen ratkaistavaksi. Valitettavasti yllättävän monet ongelmat ovat laskennallisesti erittäin vaikeita; niitä ei voida ratkaista käyttämällä "raakaa voimaa" eli tutkimalla kaikkia mahdollisia ratkaisuja. Sen sijaan on käytettävä menetelmiä, jotka pyrkivät jollakin keinoin löytämään optimaalisen ratkaisun mahdollisimman vähällä vaivalla.

Osa optimointimenetelmistä on sellaisia, että ne kykenevät löytämään nopeasti ratkaisuja, mutta niiden löytämät ratkaisut eivät useinkaan ole optimaalisia. Luonnollista evoluutiota mukailevat menetelmät ovat parempia hyvien ratkaisualueiden löytämisessä, mutta ne kuluttavat paljon aikaa näillä alueilla ennen parhaan ratkaisun löytämistä.

Memeettiset algoritmit ovat uudenlainen menetelmä, joka pyrkii yhdistämään näiden kahden lähestymistavan hyvät ominaisuudet. Hyödyntämällä evoluutiota ne kykenevät löytämään kiinnostavia alueita, ja paikallisten menetelmien avulla ne pystyvät siirtymään nopeasti optimaaliseen ratkaisuun.

Tämä tutkielma käsittelee memeettisiä algoritmeja. Tutkittavaksi ongelmaksi on valittu kauppamatkustajan ongelma, joka on eräs tärkeimmistä ja kuuluisimmista optimointiongelmista. Siinä pyritään löytämään lyhin reitti annettujen kaupunkien välillä siten, että jokaisessa kaupungissa käydään kerran. Tutkielma painottuu tälle ongelmalle suunniteltuihin algoritmeihin.

1.1 Tutkielman rakenne

Tutkielma jakaantuu pääpiirteittäin kolmeen osaan. Luvussa 2 esitellään optimoinnin peruskäsitteitä, kombinatoriset optimointiongelmat, sekä erityisesti tutkielman esimerkkitapauksena käytetty kauppamatkustajan ongelma. Luvuissa 3 ja 4 käsitellään memeettisten algoritmien kannalta olennaisia metaheuristiikkoja ja niihin liittyviä kä-

sitteitä. Luvussa 3 esitellään evoluutioalgoritmien eri suuntaukset sekä niissä käytetyt operaattorit esimerkein. Luvussa 4 käydään läpi yleisesti käytettyjä paikallishakualgoritmeja ja niitä ohjaavia menetelmiä.

Toinen osa käsittelee evoluutio- ja paikallishakualgoritmien hybridisointia, sekä varsinaisia memeettisiä algoritmeja. Luvun 5 alussa kerrotaan evoluutioalgoritmien ja paikallisten hakualgoritmien hybridisoinnista koituvista hyödyistä. Seuraavaksi tutustutaan memeettisten algoritmien toimintaperiaatteeseen, niiden historiaan ja sovelluskohteisiin. Memeettisten algoritmien formaalin muodon avulla esitellään, kuinka monipuolisesti paikallishakumenetelmiä voidaan yhdistellä evoluutioprosessiin. Lopuksi luetellaan seikkoja, joita täytyy ottaa huomioon algoritmeja suunniteltaessa. Luvussa 6 esitellään joitakin kauppamatkustajan ongelmalle suunniteltuja memeettisiä algoritmeja.

Tutkielman viimeisessä osassa, luvussa 7 keskitytään tapaustutkimukseen, jossa tutkitaan tyypillisen memeettisen algoritmin tehokkuutta verrattuna perinteisiin menetelmiin.

2 Johdattelua heuristiseen optimointiin

Seuraavassa esitellään tutkielmassa käytetyt optimoinnin peruskäsitteet ja perustellaan heuristisen menetelmien tarve. Kombinatoristen optimointiongelmien käsite selitetään lyhyesti. Lopuksi tutustutaan tutkielman esimerkkitapauksena käytettyyn kauppamatkustajan ongelmaan.

2.1 Optimoinnin peruskäsitteitä

Optimointi on prosessi, jossa etsitään parasta ratkaisua annetuissa olosuhteissa [34]. Useat ratkaistavat ongelmat ovat niin vaikeita, ettei parasta ratkaisua voida löytää, jolloin joudutaan tyytymään heikompaan ratkaisuun. Lisäksi ratkaisun "paremmuus" voi myös riippua arvioijasta, eikä aina ole selvää, miten ratkaisun paremmuutta voidaan arvioida.

Jotta optimointitehtävä voidaan ratkaista matemaattisin menetelmin, siitä on kyettävä muodostamaan matemaattinen malli. Tyypillisesti kyseessä on minimointitehtävä, joka voidaan esittää muodossa

$$\begin{aligned} &\text{minimoi} && f(\mathbf{x}) \\ &\text{rajoittein} && g_i(\mathbf{x}) \leq 0 \text{ kaikilla } i = 1, ..., m, \\ &&& h_i(\mathbf{x}) = 0 \text{ kaikilla } i = 1, ..., l. \end{aligned}$$

Optimoitavaa funktiota f kutsutaan *objektifunktioksi*. Evoluutioalgoritmien tapauksessa sitä kutsutaan yleensä *kelpoisuusfunktioksi*. f voi olla yhden tai useamman muuttujan funktio. Jos kyseessä on maksimointitehtävä, on yhtäpitävää minimoida funktiota $-f$. *Rajoitefunktiot* $g_i(\mathbf{x})$ ja $h_i(\mathbf{x})$ määrittelevät tehtävän *sallitun alueen* S. Vektorin $\mathbf{x} \in \mathbb{R}$ komponentit $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ ovat tehtävän *muuttujia*. Piste $\mathbf{x}^*$ on *sallittu*, mikäli se kuuluu S:ään.

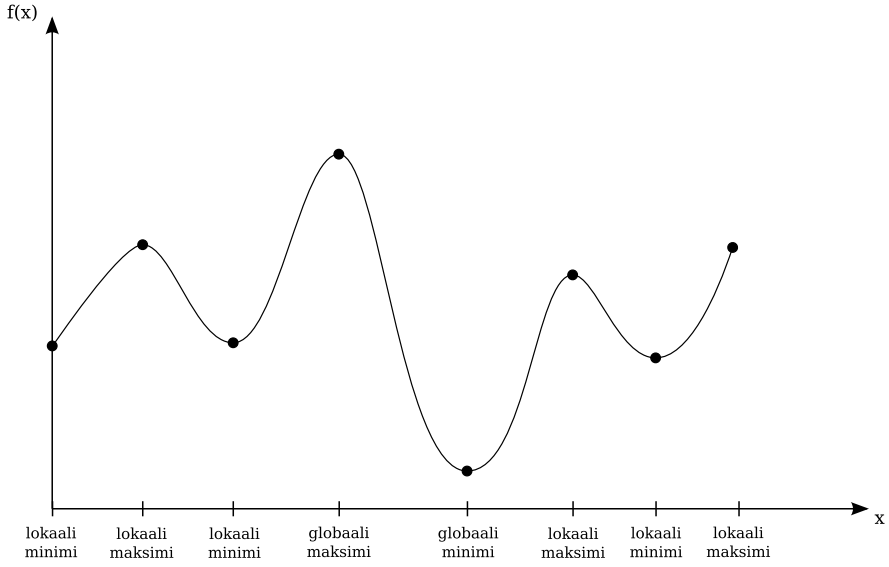
Minimointitehtävän tapauksessa optimipisteet sijaitsevat objektifunktion minimeissä. Objektifunktion globaali minimi on optimointitehtävän paras mahdollinen ratkaisu eli *globaali optimi*. Piste $\mathbf{x}^*$ on globaali minimi, jos

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

Piste x^* on puolestaan *paikallinen (lokaali) optimi*, mikäli

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \text{ kaikilla } \mathbf{x} \in S, \text{ joilla } \|\mathbf{x} - \mathbf{x}^*\| \leq \delta, \delta > 0.$$

Kuvassa 2.1 on esimerkki eräästä objektifunktiosta. Useat optimointimenetelmät kykenevät siirtymään vain ”alamäkeen” kohti lähintä paikallista minimipistettä. Aluetta, jonka pisteistä siirrytään samaan minimiin, kutsutaan *vetovoima-alueeksi* (basin of attraction).



Kuva 2.1: Eräs objektifunktio ja sen optimipisteet.

2.2 Kombinatoriset optimointiongelmat

Kombinatoriseksi optimointiongelmaksi (combinatorial optimization problem, COP) voidaan luokitella ongelma, jonka ratkaisut sijaitsevat äärellisessä, mutta usein hyvin suuressa, diskreetissä ratkaisuavaruudessa [24]. Ratkaisujen äärellisyydestä johtuen jokaiselle COP-ongelmalle on aina olemassa algoritmi, joka kykenee löytämään globaalin optimipisteen [30].

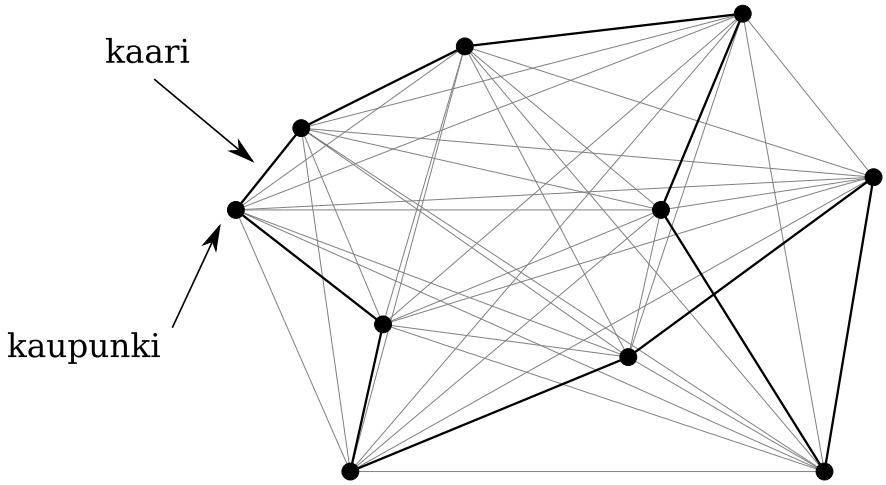
Seuraavassa alaluvussa esiteltävän kauppamatkustajan ongelman tapauksessa mahdollisten ratkaisujen lukumäärä on $(n-1)!/2$ [4]: kymmenen kaupungin tapauksessa erilaisia ratkaisuja löytyy 181 440, mikä on vielä helposti tietokoneella laskettavissa. 100 kaupungin tapauksessa ratkaisuja löytyy jo noin $4.7 \cdot 10^{155}$ eli määrä, jota on mahdoton tutkia tietokoneella. Huomattakoon, että sadan yksikön suuruista ongelmaa voidaan pitää vielä varsin pienenä.

Tällainen käyttäytyminen on tyypillistä COP-ongelmille. Ongelmakoon kasvaessa mahdollisten ratkaisujen määrä kasvaa eksponentiaalisesti tai jopa sitäkin nopeammin. Näin käyttäytyvät ongelmat kuuluvat vaativuusluokkaan NP (NP tulee termistä *Nondeterministic Polynomial time*), ja niitä kutsutaan *NP-koviksi* (NP-hard) ongelmiksi.

Vaativuusluokkaan NP kuuluvia ongelmia ei ilmeisesti pystytä ratkaisemaan polynomiaalisessa ajassa deterministisellä Turingin koneella [1]. Ongelmat, jotka voidaan ratkaista polynomisessa ajassa ongelmakoosta riippumatta, kuuluvat vastaavasti vaativuusluokkaan P (*Polynomial time*) ja ne ovat yleensä ratkaistavissa. Luokkaan NP kuuluvaa ongelmaa kutsutaan *NP-täydelliseksi*, jos kaikki luokan NP ongelmat voidaan palauttaa siihen polynomisilla muutoksilla. NP-täydellinen ongelma voi kuulua P:hen vain jos kaikki luokan NP ongelmat kuuluvat P:hen, eli P = NP. Tätä pidetään käytännön havaintojen perusteella erittäin epätodennäköisenä, joskaan sitä ei ole vielä kyetty todistamaan [38]. Laskennallisessa kompleksisuusteoriassa tämä onkin tärkein avoin kysymys [24].

Tunnettuja NP-kovia COP-ongelmia ovat kauppamatkustajan ongelman lisäksi muun muassa solmupeiteongelma, riippumaton joukko -ongelma, klikkiongelma, Hamiltonin kehä -ongelma, ajoneuvojen reititysongelma, verkonväritysongelma, kvadraattinen sijoitteluongelma, pakkausongelma, aikataulutusongelma sekä proteiinien laskostusongelma. COP-ongelmat eivät suinkaan ole pelkästään teoreettisia, vaan niitä esiintyy hyvin useissa ja monentyyppisissä käytännön tilanteissa.

2.2.1 Kauppamatkustajan ongelma



Kuva 2.2: Verkko G ja eräs ei-optimaalinen reitti.

Kauppamatkustajan ongelma (Traveling Salesman Problem, TSP) on eräs tärkeimmistä ja tutkituimmista NP-kovista ongelmista [19]. Ongelma on seuraavanlainen [42]:

[1] *Turingin koneella* tarkoitetaan automaattia, jolle annetaan syötteeksi mielivaltaisen pitkä työnauha, jota se pystyy lukemaan ja kirjoittamaan nauhapään välityksellä merkki kerrallaan. Mikä tahansa tietokoneen ratkaistavissa oleva ongelma voidaan ratkaista Turingin koneen avulla [38].

Olkoon annettu n kaupunkia ja niiden väliset etäisyydet. Tehtävänä on etsiä lyhin reitti, joka kulkee jokaisen kaupungin kautta tasan kerran palaten takaisin lähtökaupunkiin. Kyseessä on siten koko verkon käsittävän Hamiltonin kehän minimointi [30]. Kuvassa 2.2 on esitelty eräs kaupunkien muodostama verkko ja eräs validi, mutta selvästikin ei-optimaalinen reitti. Verkon solmua kutsutaan *kaupungiksi* ja kaksi kaupunkia yhdistävä jana on nimeltään *kaari*.

Ensimmäinen maininta ongelmasta löytyy saksalaisesta vuodelta 1832 peräisin olevasta kauppamatkustajan ohjekirjasta [42]. Ongelmaan läheisesti liittyviä matemaattisia ongelmia tutkittiin 1800-luvulla (Hamilton, Kirkman) [4]. Karl Mengel julkaisi ensimmäisen matemaattisen tutkielman aiheesta vuonna 1928. Tietokoneiden ilmaantuessa kyettiin ratkaisemaan aiempaa suurempia ongelmatapauksia. Laskentatehon ja varsinkin uusien, paljon entistä tehokkaampien algoritmien ansiosta entistä suurempien ongelmatapausten ratkaisu on ollut mahdollista. Ratkaisutehokkuuden dramaattinen kehitys on selkeästi havaittavissa kuvassa 2.3. Tällä hetkellä suurin ratkaistu ongelmatapaus on kooltaan 24 978 kaupunkia. Se laadittiin Ruotsin kylien ja kaupunkien perusteella.

vuosi	kaupunkeja	nimi
1954	49	`dantzig42`
1971	64	64 satunnaista pistettä
1975	67	67 satunnaista pistettä
1977	120	`gr120`
1980	318	`lin318`
1987	532	`att532`
1987	666	`gr666`
1987	2 392	`pr2392`
1994	7 397	`pla7397`
1998	13 509	`usa13509`
2001	15 112	`d15112`
2004	24 978	`sw24798`

Kuva 2.3: Suurimmat ratkaistut kauppamatkustajan ongelman tapaukset eri vuosina.

Suuri osa kauppamatkustajan ongelmaan liittyvästä tutkimuksesta on teoreettista, sillä ongelma tarjoaa käyttökelpoisen alustan tutkittaessa yleisiä kombinatoristen ongelmien ratkaisumenetelmiä [4]. Ongelma on tärkeä myös käytännössä, sillä se ilmenee usein jokapäiväisessä elämässä. Reitinsuunnittelun lisäksi ongelma kohdataan niinkin erilaisissa tapauksissa kuin piirilevyjen porauksessa, röntgenkristallografiassa, lentoko-

neiden turbiinimoottorien huollossa, varastomyymälän tavaroiden noutojärjestyksessä, tietokoneiden johdotuksessa, taulukkojen elementtien ryvästyksessä, arkeologisten kohteiden ikäjärjestyksen määrittämisessä, aikataulutuksessa ja robottien ohjauksessa [40].

Kauppamatkustajan ongelmasta on olemassa erilaisia muunnelmia. Yllä määriteltyä tapausta kutsutaan toisinaan *symmetriseksi kauppamatkustajan ongelmaksi* (symmetric traveling salesman problem, STSP) erotukseksi *epäsymmetrisestä kauppamatkustajan ongelmasta* (asymmetric TSP, ATSP). STSP:n tapauksessa verkon G kaaret ovat saman mittaisia riippumatta kumpaan suuntaan kaarta kuljetaan (eli $d_{ab} = d_{ba}$ jokaiselle $a, b \in G$). Epäsymmetrisessä tapauksessa tämä ei päde.

Ongelma voidaan yleistää myös graafeihin, joissa kaupungit eivät välttämättä ole suorassa yhteydessä toisiinsa. Tällöin saatetaan joutua luopumaan vaatimuksesta, että samassa kaupungissa voidaan käydä vain kerran. Tämä vaatii monimutkaisempaa reittien laskentaa, mutta käytännössä nämä niin sanotut *graafiset kauppamatkustajan ongelmat* (graphical TSPs) ovat yleisiä. *Yleistetyssä kauppamatkustajan ongelmassa* (generalized TSP) kaupungit on jaettu ryppäiksi, ja kauppamatkustajan tulee käydä jokaisessa ryppäässä ainakin yhdessä kaupungissa. *Usean kauppamatkustajan ongelmassa* jokaiselle n kauppamatkustajalle muodostetaan alireitti siten, että jokaisessa verkon kaupungissa käy tasan yksi kauppamatkustaja. *Pullonkaula-kauppamatkustajan ongelmassa* (bottleneck TSP) pyritään löytämään reitti, jonka pisin kaari on mahdollisimman lyhyt.

2.3 Heuristiset menetelmät

Kombinatoristen optimointiongelmien ratkaisemiseksi on kehitetty lukuisia erilaisia algoritmeja [2]. Menetelmiä, jotka löytävät aina globaalin optimin rajoitetussa ajassa, kutsutaan *täydellisiksi* (complete). Koska NP-ongelmille ei ole olemassa polynomisessa ajassa ratkeavia deterministisiä menetelmiä, eivät täydelliset menetelmät ole useinkaan kovin käyttökelpoisia. Sen sijaan NP-ongelmat voidaan ratkaista polynomiaalisessa ajassa epädeterministisellä menetelmällä, toisin sanoen arvaamalla ja tarkistamalla [38]. Siksi valtaosa näille ongelmille suunnitelluista menetelmistä onkin *likimääräisiä* (approximate).

Likimääräiset menetelmät voidaan vielä jakaa alaluokkiin, *konstruktiivisiin* (constructive) ja *paikallishakuihin* (local searches). Konstruktiivisissa menetelmissä ratkaisuehdokas muodostetaan lisäämällä aluksi tyhjään alkuratkaisuun komponentteja, kunnes saadaan käypä ratkaisu. Lähimmän naapurin menetelmä, jossa reitti muodostetaan valitsemalla lähin vapaa naapuripiste, on eräs tällainen menetelmä. Konstruktiiviset menetelmät ovat nopeimpia likimääräisiä menetelmiä, mutta niillä saadut ratkaisut

ovat yleensä paikallishakuihin verrattuna huonompia. Paikallishakumenetelmät puolestaan tutkivat annetun alkuratkaisun ympäristöä paremman ratkaisun löytämiseksi. Ne esitellään tarkemmin luvussa 4.

Yksinkertaisia likimääräisiä menetelmiä voidaan yhdistää ylemmän tason strategioiksi, niin kutsutuiksi *metaheuristiikoiksi* (metaheuristics). Metaheuristiikat voidaan luokitella useisiin eri luokkiin. Tämän tutkielman kannalta olennaisin on jako populaatioita hyödyntäviin menetelmiin ja menetelmiin, jotka tutkivat vain yhtä pistettä kerrallaan. Populaatioon perustuvat menetelmät etsivät ratkaisuja usean ratkaisuehdokkaan muodostaman *populaation* avulla. Näiden menetelmien toimintaperiaatteet ovat yleensä peräisin luonnosta. Memeettisten algoritmien kannalta olennaisia ovat *evoluutioalgoritmit*, jotka esitellään luvussa 3. Muita menetelmiä ovat mm. muurahaisyhteisöt. Algoritmeja, jotka käsittelevät yhtä pistettä kerrallaan, kutsutaan *polkumenetelmiksi* (trajectory methods). Näiden menetelmien toimintaa voidaan kuvailla poluksi kohti optimaalista pistettä. Polkumenetelmiä ovat esimerkiksi myöhemmin esiteltävät ohjattu paikallishaku, simuloitu jäähdytys ja tabuhaku.

Metaheuristiikkojen toiminta jaetaan kahteen osaan: *diversifikaatioon* (diversification) ja *tehostamiseen* (intensification). Diversifikaatiovaiheessa algoritmi etsii uusia optimipisteitä, tehostamisvaiheessa puolestaan pyritään siirtymään löydettyyn optimipisteeseen. Evoluutioalgoritmien yhteydessä käytetään tyypillisesti termejä *etsintä* (exploration) ja *hyödyntäminen* (exploitation).

3 Luonnolliseen evoluutioon pohjautuvat menetelmät

Evoluutioalgoritmien perustana toimii luonnollisen evoluution soveltaminen optimointiprosessissa. Luvussa esitellään luonnollinen evoluutio ja siihen olennaisesti liittyvät käsitteet. Evoluutioalgoritmien historia, erilaiset suuntaukset ja toimintaperiaate käsitellään yleisesti. Evoluutioalgoritmeissa käytettyihin muuntelu- ja valintaoperaattoreihin tutustutaan yksityiskohtaisemmin.

3.1 Luonnollinen evoluutio

Kirjassaan *Lajien synty* (1859) Darwin [8] esitteli teorian *luonnollisesta evoluutiosta*. Sen mukaan jokainen elävä olento on kehittynyt evoluution tuloksena. Evoluutio perustuu kolmeen periaatteeseen [30]: *kopiointiin*, *muunteluun* ja *luonnonvalintaan*. Uusia eliölajeja ei voi syntyä puhtaan kopioinnin tuloksena, sillä tällöin uudet yksilöt olisivat edeltäjiensä täydellisiä kopioita. Siksi tarvitaan perimässä tapahtuvia virheitä, *mutaatioita*. Seksuaalisessa lisääntymisessä on tarjolla toinen varioinnin muoto, *risteytyminen* [1], jossa jälkeläisen perimä valikoituu vanhemmilta satunnaisesti. Maapallon rajallisista resursseista johtuen eliöiden rajoittamaton kopioituminen on mahdotonta. Tästä syystä eliöt joutuvat kilpailemaan saatavilla olevista resursseista, jolloin vain kelpoisimmilla yksilöillä on mahdollisuus menestyä.

Eliön *kelpoisuus* eli *fitness* voidaan määritellä joko *elinkykyisyytenä* eli todennäköisyytenä, jolla eliö pysyy hengissä ja lisääntyy, tai *hedelmällisyytenä* eli kuinka monta jälkeläistä eliö tuottaa.

Jokaisen eliön geneettinen tieto on tallentunut solujen *kromosomeihin*, jotka ovat DNA:sta (deoksiribonukleiinihappo) koostuvia rihmoja. Kromosomit sisältävät kaiken tarvittavan tiedon eliön rakenteesta. Ne voidaan jakaa käsitteellisesti *geeneihin*, jotka ovat jaksoja DNA-molekyylissä. Jokainen geeni tuottaa tiettyä proteiinia. Geenin paikkaa DNA:ssa kutsutaan *lokukseksi*. Samassa lokuksessa sijatsevan geenin eri versiot, *alleelit*, tuottavat erilaisia *ominaisuuksia*, kuten silmien värin. Yksilön koko geenis-

[1] Variointia syntyy myös meioosin aikana *rekombinaatioksi* kutsutussa prosessissa. Kromosomiluvun puolittuessa sukusolun kromosomi rakentuu vastinkromosomeilta valikoituneista jaksoista [35]. Evoluutiolaskennassa risteytystä ja rekombinaatiota käytetään usein toistensa synonyymeinä.

töä kutsutaan *genomiksi*. *Genotyyppi* tarkoittaa puolestaan yksilön perimää. Perimän ja ympäristön yhteisvaikutus muodostaa eliön ilmiasun, *fenotyypin*.

Geenien välinen vuorovaikutus, *epistaasi*, tekee mutaatiosta ja risteytyksestä vaikeasti ennustettavia. Ilmiötä, jossa yksittäinen geeni vaikuttaa samanaikaisesti useaan fenotyypin ominaisuuteen, kutsutaan *pleiotropiaksi*. Toisaalta yhteen fenotyypin ominaisuuteen voi vaikuttaa useampi geeni (*polygenia*).

3.2 Evoluutiolaskennan suuntaukset

Jo kauan ennen tietokoneiden yleistymistä pohdittiin keinoja, joilla evoluutiota voitaisiin soveltaa automaattisessa ongelmanratkaisussa [16]. Vuonna 1948 Turing kehitti niin kutsutun "geneettisen eli evolutionäärisen haun". Vuonna 1962 Bremenmann suoritti tietokoneella optimointitestejä, joissa hyödynnettiin evoluutiota ja risteytystä. Samalla vuosikymmenellä laadittiin toisistaan riippumattomasti kolme evoluutioon perustuvaa menetelmää. Seuraavassa esitellään nämä sekä myöhemmin kehitetty neljäs, suurta suosiota saavuttanut menetelmä.

3.2.1 Evoluutiostrategiat

1960-luvulla saksalaiset Bienert, Rechenberg ja Schwefel kehittivät evoluutiota hyödyntävän menetelmän nimeltään *evoluutiostrategiat* (evolution strategies, ES; alkuperäinen saksankielinen termi on *Evolutionstrategie*) [1]. Menetelmää sovellettiin suunnitteluun ja rakenteiden optimointiin. Tosin aluksi mallinnuksessa ei käytetty tietokoneita, vaan konkreettisia malleja, joita testattiin ja muunneltiin niveliä ja osia lisäämällä tai poistamalla.

Schwefel esitteli ensimmäisen ES-tietokonealgoritmin vuonna 1965 ja Rechenberg kehitti sitä edelleen vuonna 1973. Alkuperäisissä ES-algoritmeissa käytettiin kahta yksilöä, vanhempaa ja jälkeläistä. Yksilöt esitettiin reaaliarvoisina vektoreina. Jälkeläinen muodostettiin muuttamalla vanhemman arvoja normaalijakautuneella satunnaismuuttujalla, jonka odotusarvo oli nolla ja keskihajonta jokin ennalta valittu arvo. Jos jälkeläinen oli kelpoisuudeltaan vanhempaa parempi, vanhempi korvattiin. Muussa tapauksessa vanhemmasta mutatoitiin uusi jälkeläinen.

Alkuperäisen (1+1)-valintastrategian ongelmana oli se, että algoritmi jumiutui helposti paikalliseen optimiin. Evoluutiostrategioiden kehittäminen jatkui, ja 1980-luvun alkupuolelle mennessä nykyisenkaltaiset menetelmät olivat kehittyneet. Niissä käytetään populaatioita, risteytystä sekä (λ, μ)- ja $(\lambda + \mu)$-valintaa (nämä esitellään selviytymisvalinnan yhteydessä sivulla 19).

3.2.2 Evoluutio-ohjelmointi

Evoluutio-ohjelmointi (evolutionary programming, EP) muistuttaa läheisesti evoluutiostrategioita, vaikka se kehitettiinkin niistä riippumattomasti Yhdysvalloissa (Fogel, Owens ja Walsh, 1966) [1] [35]. Alkuperäisessä menetelmässä sovelluskohteena käytettiin äärellisten automaattien siirtymätaulukoita, joita mutatoitiin. Automaatin kelpoisuuden mittarina oli se, kuinka hyvin sen tuloste vastasi ennalta valittua jaksoa. Menetelmä käytti yhtä ratkaisupopulaatiota, ja uusien yksilöiden muodostamisessa käytettiin pelkästään mutaatiota.

Evoluutio-ohjelmointi jäi vähälle huomiolle aina 1980-luvun lopulle saakka, jolloin David Fogel, menetelmän kehittäjän Lawrence Fogelin poika, laajensi menetelmän nykyiseen muotoonsa. Alun perin diskreeteistä parametreista tehtiin reaaliarvoisia. Algoritmiin lisättiin myös itsesopeutuvuus, jolloin evoluutio-ohjelmoinnista tuli hyvin evoluutiostrategioiden kaltainen.

3.2.3 Geneettiset algoritmit

John Holland tutki 1960-luvulla Yhdysvalloissa luonnon sopeutumisprosesseja ja kuinka niitä voitaisiin käyttää keinotekoisissa järjestelmissä [1]. Kehittelyn tuloksena syntyi niin kutsuttu *geneettinen algoritmi* (genetic algorithm, GA). Toisin kuin evoluutiostrategioita ja evoluutio-ohjelmointia, sitä ei kehitetty jotain tiettyä sovelluskohdetta varten. Menetelmää on helppo soveltaa monenlaisiin ongelmiin ja heikostakin toteutuksesta huolimatta se tuottaa hyviä ratkaisuja. Tästä johtuen geneettiset algoritmit ovatkin selvästi eniten käytetty evoluutiota hyödyntävä menetelmä.

Edellisistä menetelmistä poiketen geneettissä algoritmeissa yksilöitä ei muokata suoraan, vaan ne koodataan ensin kromosomeihin, jotka olivat alkuperäisessä geneettisessä algorimissa bittijonoja. GA oli ensimmäinen evoluutiomenetelmä, jossa käytettiin parinvalintaa ja risteytystä.

3.2.4 Geneettinen ohjelmointi

Geneettinen ohjelmointi (genetic programming, GP) on neljäs, John Kozan vuonna 1992 kehittämä tekniikka [1]. Se muistuttaa toiminnaltaan geneettisiä algoritmeja, ja tarkkaan ottaen se voidaan luokitella niiden erikoistuneeksi tyypiksi. Geneettistä ohjelmointia käytetään ohjelmien luomiseen, sen avulla tietokoneet pystyvät periaatteessa ohjelmoimaan itse itseään. Menetelmä on julkaisustaan lähtien ollut varsin suosittu.

Geneettinen ohjelmointi poikkeaa geneettisistä algoritmeista siinä, että muutokset kohdistuvat itse yksilöihin eikä erillisiin kromosomeihin. Menetelmän tuottamien

lausekkeiden tulee olla syntaktisesti korrekteja, ja evoluution edetessä niiden pituus vaihtelee, joten esitysmuotona käytetään puurakennetta. Yksilöitä, siis tietokoneohjelmia, testataan suorittamalla niitä ja kelpoisuuden mittarina käytetään niiden tulostetta annetuilla syötteillä.

3.2.5 Evoluutioalgoritmit

Menetelmät kehittyivät kauan toisistaan erillään kuitenkin lainaillen ominaisuuksia toisiltaan. Viimein 1990-luvun alkupuolelta lähtien niitä on alettu pitää saman teknologian eri ilmentyminä. Menetelmiä kutsutaan nykyisin yhteisnimityksellä *evoluutioalgoritmit* (evolutionary algorithms, EA) [16].

3.3 Evoluutioalgoritmin toimintaperiaate

Evoluutioalgoritmien toiminta voidaan yleistää samaan toimintaperiaatteeseen [16]: populaation yksilöihin sovelletaan luonnonvalinnasta sovellettuja mekanismeja sekä geneettistä muuntelua tarkoituksena saada kelpoisuudeltaan hyviä yksilöitä eli lähellä optimia sijaitsevia ratkaisuja.

Evoluutioalgoritmin yleinen periaate on esitelty kuvassa 3.1. Aluksi muodostetaan *alkupopulaatio*, joka koostuu yleensä satunnaisesti valituista ratkaisuehdokkaista. Ehdokkaita testataan *kelpoisuusfunktiolla*, joka määrää niiden kelpoisuuden. Osa yksilöistä valitaan kelpoisuuden perusteella vanhemmiksi ja risteytetään toistensa kanssa, jolloin saadaan yksi tai useampi jälkeläinen. Muodostettuja jälkeläisiä mutatoidaan ja testataan. Kelpoisiksi havaitut yksilöt valitaan seuraavaan sukupolveen, josta muodostetaan uusi populaatio. Prosessia jatketaan kunnes annettu lopetusehto on saavutettu.

```
procedure EA
begin
    /* muodosta yksilöt satunnaisesti tai jollakin reitinmuodostusalgoritmilla */
    AlustaPopulaatio(P);
    foreach s ∈ P do LaskeKelpoisuus(s);
    repeat
        P_j := 0; /* P_j jälkeläisten populaatio */
        for i := 1 to risteytysten lkm do
        begin
            P_v := ValitseVanhemmat(P); /* P_v vanhempien populaatio */
            s := Risteytä(P_v); /* valitse vanhemmat satunnaisesti */
            n := random(0, 1);
            /* sovella mutaatiota yksilöön todennäköisyydellä p_m */
            if n < p_m then s := Mutatoi(s);
            /* lisätään uusi yksilö jälkeläisten populaatioon */
            Lisää(P_j, s);
        end;
        foreach s ∈ P_j do LaskeKelpoisuus(s);
        P := ValitseJälkeläiset(P ⋃ P_j); /* selviytymisvalinta */
    until lopetusehto saavutettu;
end;
```

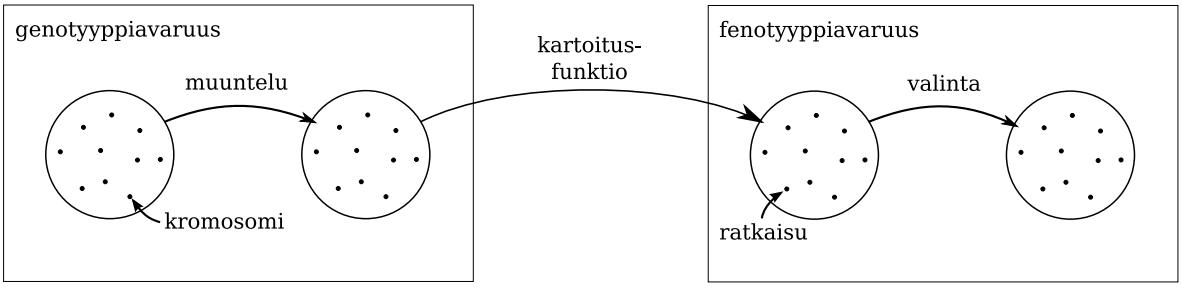
Kuva 3.1: Evoluutioalgoritmin yleinen toimintaperiaate.

3.4 Ongelman esitysmuoto

Optimointitehtävän *ratkaisuehdokkaat* (*fenotyypit* eli *yksilöt*) ovat pisteitä s, jotka sijaitsevat mahdollisten ratkaisujen joukossa [16]. Evoluutioalgoritmien tapauksessa tätä joukkoa kutsutaan *fenotyyppiavaruudeksi* $\mathcal{P}$. Evoluutioalgoritmit eivät tuota ratkaisuja suoraan, vaan ne käyttävät diskreettiä tietorakennetta, *kromosomia* (myös termejä *yksilö* ja *genotyyppi* käytetään). Kromosomit sijaitsevat vastaavasti *genotyyppiavaruudessa* $\mathcal{G}$.

Algoritmi tuottaa ratkaisuehdokkaita, jotka on pystyttävä koodaamaan yksikäsitteisesti reaalimaailman ratkaisuksi ja päinvastoin. Tätä esitysmuodon ”tulkkina” toimivaa funktiota kutsutaan *kartoitusfunktioksi* $\mathcal{M}$. Se on injektio, joka kuvaa jokaisen genotyyppiavaruuden ratkaisun fenotyyppiavaruuteen. Avaruuksien välistä relaatiota on havainnollistettu kuvassa 3.2 (mukailtu kuva lähteestä [30]).

Käytännössä esitysmuoto on usein alkuperäisen ratkaisun kaltainen, jolloin myös kartoitusfunktiosta tulee hyvin yksinkertainen. Kartoitusfunktion toiminta poikkeaa huomattavasti luonnollisesta prosessista, sillä luonnossa genotyypin koodautuminen fenotyypiksi on erittäin monimutkainen prosessi [30].



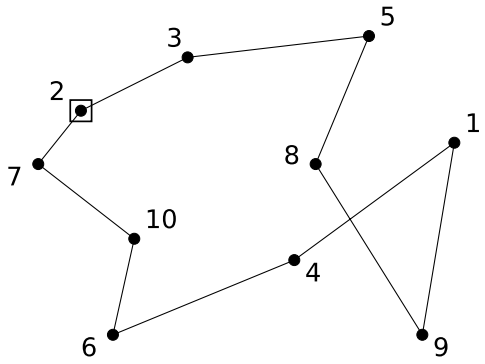
Kuva 3.2: Fenotyyppi- ja genotyyppiavaruudet.

Ongelman esitysmuoto voidaan kuvata l:n mittaisena n-kantaisten lukujen jonona [23]

$$I = \{q_1, ..., q_n\}^l$$

Alkuperäisissä geneettisissä algoritmeissa esitysmuotona käytettiin binäärilukujonoja (eli $I = \{0, 1\}^l$). Permutaatio-ongelmissa käytetään tyypillisesti kokonaislukujonoja. Algoritmin toteutuksessa ei suinkaan tarvitse rajoittua lukujonoihin: joissakin tapauksissa esitysmuotona kannattaa käyttää esimerkiksi puurakennetta tai binäärimatriisia. Lähteessä [44] on lueteltu lukuisia erilaisia tapoja koodata reitti kromosomiin.

Tarkastellaan kuvaa 3.3. Se esittää kymmenen kaupungin kauppamatkustajan ongelmaa. Kaupungit on numeroitu yhdestä kymmeneen. Tällöin eräs ratkaisuehdokas –



Kuva 3.3: Eräs kaikki kaupungit kiertävä reitti.

siis fenotyyppi – on reitti $2 \to 3 \to 5 \to 8 \to 9 \to 1 \to 4 \to 6 \to 10 \to 7 \to 2$. Lienee johdonmukaisin tapa muodostaa näistä lukujono – kromosomi eli genotyyppi – on yksinkertaisesti merkitä kauppamatkustajan vierailemat kaupungit järjestyksessä

2	3	5	8	9	1	4	6	10	7

Biologisen esikuvan tapaan lukujonon yhtä muuttujaa voidaan kutsutaan myös *geeniksi* tai *lokukseksi* [16]. Muuttujan arvo on puolestaan *alleeli*.

Etenkin kombinatorisissa optimointiongelmissa muuttujien välillä esiintyy *epistaasia*, eli kuten biologisten geenien tapauksessa muuttujat vaikuttavat toisiinsa [32]. Esimerkiksi yhtäkään yllämainitun kromosomin muuttujaa ei voida vaihtaa niin, ettei se vaikuttaisi johonkin toiseen kromosomin muuttujaan. Epistaasia voidaan käyttää ongelman vaikeuden arvointiin, sillä se määrittää ongelman epälineaarisuuden määrän.

3.5 Kelpoisuuden arviointi

Optimoitaessa jotain ongelmaa on annetun ratkaisukandidaatin laatu tunnettava. Sama pätee luonnollisesti myös evoluutioalgoritmeihin, ja yksilön kelpoisuuden arviointi onkin keskeinen osa algoritmien toimintaa [30]. Kelpoisuuden arviointi voidaan esittää *kelpoisuusfunktiona*

$$f : I \to \mathbb{R}^+,$$

joka kuvaa esitysmuotoa I olevan kromosomin positiiviseksi reaaliluvuksi.

Kelpoisuusfunktio toimii valinnan perustana, sillä se määrää, milloin yksilön kelpoisuus on parantunut. Funktion valinta riippuu täysin käsiteltävästä ongelmasta ja se on määriteltävä jokaiselle ongelmalle erikseen. Usein ainoa algoritmiin sisällytetty informaatio itse ongelmasta sijaitsee kelpoisuusfunktiossa [10].

Rajoitettujen optimointiongelmien tapauksessa evoluutio-operaattorit saattavat tuottaa yksilöitä, joita ei voida hyväksyä [30]. Ne täytyy joko korjata jollakin virheenkorjausalgoritmilla, tai vaihtoehtoisesti voidaan käyttää sakkotermejä, jotka ohjaavat algoritmin hyväksyttävien arvojen alueelle.

Yksilön kelpoisuuden laskeminen on usein suhteellisen vaativaa, joten järkevällä suunnittelulla voidaan säästää laskenta-aikaa. Esimerkiksi kauppamatkustajan tapauksessa koko reittiä ei tarvitse laskea uudelleen, jos tiedetään, mitkä kaaret ovat vaihtuneet. Samoin, jos tutkittava ongelma ei ole erityisen suuri, kaupunkien väliset etäisyydet kannattaa tallentaa taulukoihin ennen varsinaista optimointivaihetta.

Tarkastellaan edellisessä alaluvussa esiteltyä reittiä. Kauppamatkustajan ongelman tapauksessa kelpoisuus on helppo määrittää – se on selvästikin reitin kokonaispituus. Seuraavassa taulukossa on listattuina kauppamatkustajan vierailemien kaupunkien väliset matkat:

kaari	**etäisyys**
(2–3)	3,2
(3–5)	2,8
(5–8)	4,5
(8–9)	1,8
(9–1)	2,2
(1–4)	4,2
(4–6)	6,2
(6–10)	3,2
(10–7)	4,4
(7–2)	2,1
Σ	34,6

Kyseisen reitin, siis ”yksilön”, kelpoisuus on siten arvoltaan 34,6. Taulukosta nähdään, että alireitin (2–3–5–8) pituus on 10,5. Olkoon kaupunkien 2 ja 5 välinen etäisyys 2,2 sekä kaupunkien 3 ja 8 välinen etäisyys 3,0. Olkoon kyseessä symmetrinen tapaus, jolloin kaupunkien 3 ja 5 väli pysyy samana. Tällöin alireitti (2–5–3–8) onkin pituudeltaan vain 8,0. Yksilö, joka on muuten esimerkin kaltainen mutta sisältää muutetun alireitin, on kelpoisuudeltaan 32,1. Koska kyseessä on minimointitehtävä, tämä yksilö on kelpoisempi eli lähempänä globaalia optimia.

3.5.1 Kelpoisuusmaasto

Ratkaisuavaruuden pisteiden voidaan ajatella muodostavan eräänlaisen maaston, jossa pisteen kelpoisuus määrittää sen korkeuden jostakin tietystä vertailutasosta. Hy-

vien ratkaisujen alueiden voidaan ajatella muodostavan ”vuoria” ja huonojen ”laaksoja”. Evoluutiobiologiassa ja -laskennassa tällaista kelpoisuuden määrittelemää aluetta kutsutaan *kelpoisuusmaastoksi* (fitness landscape) [32]. Kombinatoristen optimointiongelmien tapauksessa kelpoisuusmaasto on verkko, jossa solmukohdat ovat mahdollisia ratkaisuja ja niiden väliset kaaret ovat siirtymiä, joita pitkin valitulla operaattorilla voidaan liikkua.

Kelpoisuusmaaston muoto vaikuttaa huomattavasti heuristisiten menetelmien tehokkuuteen: maasto voi olla hyvin epätasainen, eli kelpoisuus vaihtelee suuresti lähekkäisten pisteiden välillä. Maastossa voi olla runsaasti paikallisia optimeja (huippuja). Optimien sijoittumisella ja paikallisten optimien vetovoima-alueiden muodoilla on myös suuri vaikutus. Näitä ominaisuuksia voidaan tutkia tilastollisin menetelmin, ja onkin ehdotettu, että kelpoisuusmaastoa voitaisiin käyttää ongelman vaikeuden arviointiin.

3.6 Populaatio

Populaatiolla tarkoitetaan [16] evoluutioalgoritmien yhteydessä ratkaisujen s_i monijoukkoa[2] P:

$$P = (s_1, ..., s_\mu) \in I^\mu,$$

missä μ on populaation yksilöiden lukumäärä. Se pidetään tavallisesti vakiona, mutta joissakin algoritmeissa populaation kokoa saatetaan muuttaa dynaamisesti.

Populaation *diversiteetti* kertoo, kuinka paljon erilaisia ratkaisuja populaatio sisältää. Se voidaan määrittää esimerkiksi laskemalla kuinka monta erilaista kelpoisuusarvoa populaatiosta löytyy, tai kuinka monta erilaista genotyyppiä tai fenotyyppiä populaatio sisältää. Diversiteetin heikkeneminen yleensä hidastaa algoritmin etenemistä kohti globaalia optimia.

3.7 Valintamekanismit

Evoluutioalgoritmien valintamekanismeilla valitaan populaatiosta kelvolliset yksilöt. Ne ohjaavat algoritmin kehittymistä kohti optimaalisia alueita. Valinta kohdistuu yksilöihin, joten se tapahtuu fenotyyppiavaruudessa, eikä vaikuta suoraan yksilöiden genotyyppeihin.

Valintamekanismeja on kaksi: *parinvalinta* ja *eloonjäänti*. Algoritmin konvergenssia voidaan säätää muuttamalla valintapainetta. Jos valintapainetta kasvatetaan, eli pienempi osa populaatiosta valitaan, algoritmi konvergoi nopeammin, mutta toisaalta

[2]Monijoukko on joukko, joka voi sisältää useita kopioita samasta alkiosta.

samalla populaation diversiteetti heikkenee.

3.7.1 Parinvalinta

Parinvalinta vaikuttaa osaltaan populaation laadun paranemiseen [16]. Sen tarkoituksena on valita populaatiosta yksilöt, jotka sisältävät ominaisuuksia, joiden halutaan lisääntyvän populaatiossa. Yksilöä, joka tuottaa jälkeläisiä, kutsutaan *vanhemmaksi*. Tyypillisesti parinvalinta perustuu todennäköisyyteen, eli kelpoisuudeltaan paremmat yksilöt valitaan vanhemmiksi. Liian nopea konvergenssi johtaa algoritmin juuttumiseen paikalliseen optimiin, mistä johtuen myös heikommille yksilöille annetaan yleensä pieni todennäköisyys tulla valituksi.

Parinvalintaa varten on kehitetty lukuisia erilaisia menetelmiä, joista yleisimmin käytettyjä ovat

- **Kelpoisuuteen verrannollinen valinta:** [30] Useat geneettiset algoritmit suhteuttavat yksilön kelpoisuuden todennäköisyyteen, jolla se tulee valituksi. Esimerkiksi yksilöllä, jonka kelpoisuus on kaksi, on kaksi kertaa suurempi todennäköisyys tulla valituksi kuin yksilöllä, jonka kelpoisuus on yksi. Valintatodennäköisyys voidaan esittää todennäköisyysfunktiona

  $$p(s_i) = \frac{f(s_i)}{\sum_{s_j \in P} f(s_j)},$$

  jossa $f(s_i)$ on yksilön s_i kelpoisuus. Tyypillisin tätä tekniikkaa käyttävä menetelmä on *rulettivalinta* [35], jossa valinta perustuu "rulettipyörän" pyörittämiseen. Jokaiselle populaation yksilölle annetaan oma siivu rulettipyörästä. Siivun koko riippuu yksilön kelpoisuudesta. Pyörää pyöräytetään yhtä monta kertaa kuin populaatiossa on yksilöitä. Yksilö, jonka kohdalle pyörä pysähtyy, tulee valituksi.

- **Paremmuusjärjestykseen perustuva valinta:** [35] Tässä menetelmässä yksilöt lajitellaan paremmuusjärjestykseen, jolloin ei tarvitse laskea yksilöiden absoluuttista kelpoisuutta, eikä kelpoisuuksia tarvitse suhteuttaa. Valintapaine säilyy varianssin muuttuessa, koska perättäisten yksilöiden valintatodennäköisyyden välinen suhde pysyy koko ajan vakiona.

  Tästä on hyötyä verrattuna edelliseen menetelmään, jossa poikkeuksellisen kelvollinen yksilö tulee valituksi liian usein. Samaten edellinen menetelmä on ongelmallinen, kun yksilöiden kelpoisuuksien varianssi on pieni: valinta muuttuu satunnaiseksi, kun kaikki yksilöt saavat hyvin samanlaisen valintatodennäköisyyden. Haittapuolena paremmuusjärjestykseen perustuvassa menetelmässä on se,

että joissakin tapauksissa saattaa olla tärkeää tietää, kuinka kelvollinen tarkasteltava yksilö on verrattuna sen lähimpään kilpailijaan.

Eräs (lineaarinen) tapa [30] laskea todennäköisyys, jolla yksilö s valitaan on

$$p(s_i) = p_{\max} - (p_{\max} - p_{\min})\frac{i-1}{n-1},$$

missä $p_{\max}$ on populaation kelvollisimman yksilön valintatodennäköisyys ja $p_{\min}$ heikoimman yksilön valintatodennäköisyys.

- **Turnajaisvalinta** [35] on esitellyistä tekniikoista laskennallisesti vähiten vaativa, sillä se ei tarvitse tietoa koko populaatiosta. Menetelmän ideana on valita joukko yksilöitä, joita kilpailutetaan keskenään. Voittaja valitaan vanhemmaksi. Eräs tapa toteuttaa tämä on valita kaksi yksilöä täysin satunnaisesti. Arvotaan luku $r \in [0, 1]$, ja verrataan sitä valittuun parametriin k (esim. 0,75). Mikäli r on pienempi kuin k, valitaan yksilöistä kelpoisempi, muussa tapauksessa heikompi. Ongelmaksi voi koitua se, että kelpoisimmat yksilöt voivat jäädä valitsematta. Valintapainetta voidaan kasvattaa lisäämällä yksilöiden määrää [15].

3.7.2 Eloonjäänti

Eloonjääntivalinta määrittää sen, mitkä yksilöt selviytyvät seuraavaan sukupolveen. Valinta perustuu yleensä kelpoisuuteen ja on tyypillisesti determinististä, eli heikommat yksilöt eivät siirry seuraavaan sukupolveen.

Tyypillisimpiä valintamenetelmiä ovat [30]

- **Vakaan tilan valinta:** Jälkeläisiä tuotetaan vähemmän kuin mitä vanhempia on. Osa vanhemmista korvataan jälkeläisillä. Korvaaminen voi perustua esimerkiksi kelpoisuuteen, eli jokin (usein heikoin) yksilö korvataan paremmalla jälkeläisellä. Joissakin algoritmeissa käytetään ikään perustuvaa valintaa, eli populaation vanhimmat yksilöt korvataan.

- **(μ+λ)-valinta:** Tämä valintamenetelmä on peräisin evoluutiostrategioista. μ kuvaa vanhempien ja λ jälkeläisten lukumäärää. Tyypillisesti λ on yksi tai kaksi kertaa niin suuri kuin μ. Tässä menetelmässä μ seuraavan sukupolven yksilöä valitaan väliaikaisesta populaatiosta, joka sisältää sekä vanhemmat että jälkeläiset.

- **(μ, λ)-valinta**: Tämä menetelmä poikkeaa edellisestä siinä, että valinta kohdistuu pelkästään jälkeläisiin. Vanhemmat korvataan μ:lla jälkeläisellä. Valintapainetta voidaan kasvattaa kasvattamalla λ:aa. Huomattakoon, että λ:n täytyy olla

μ:ta suurempi, jotta valintaa syntyisi. Geneettisten algoritmien yhteydessä tätä menetelmää kutsutaan *sukupolvenvaihdoksi* (generational replacement).

(μ, λ)-valinnan tapauksessa algoritmin toimintaa voidaan tehostaa *elitismillä*, jossa yksi tai useampi parasta yksilöä (*eliittiyksilöä*) valitaan suoraan seuraavaan sukupolveen. Tällöin vältytään tilanteelta, jossa paras löydetty ratkaisu menetetään sukupolvenvaihdoksen yhteydessä.

Uutta populaatiota muodostettaessa yksilöiden kopiot voidaan poistaa, jolloin estetään populaation näivettyminen muutaman yksilön kopioiden joukoksi.

3.8 Muunteluoperaattorit

Muunteluoperaattorien tehtävä on muodostaa uusia yksilöitä vanhoista näiden genotyyppejä yhdistelemällä ja muokkaamalla. Fenotyyppiavaruudessa tämä tarkoittaa uudenlaisten ratkaisuehdokkaiden muodostamista. Operaattoreita on kahdentyyppisiä, mutaatiota ja risteytystä.

3.8.1 Risteytys

Risteytys eli *rekombinaatio* on yleensä binäärinen muunteluoperaattori [16]. Se yhdistää vanhempien genotyypit yhdeksi tai useammaksi jälkeläisgenotyypiksi. Pyrkimyksenä on yhdistää vanhempien hyvät ominaisuudet ja saada näin aikaiseksi jälkeläinen, joka on niitä kelpoisempi. Risteytys perustuu satunnaisuuteen: valittavat kromosomin jaksot ja tapa, jolla jaksot yhdistetään, ovat satunnaisia tapahtumia.

Risteytysoperaattori voidaan esittää relaationa [43]

$$R : I^{\mu} \times \delta \rightarrow I,$$

jossa μ on operaattorin käytettävissä olevien vanhempien lukumäärä ja δ todennäköisyys, jolla risteytys suoritetaan. Yleensä risteytysoperaatiossa käytetään kahta vanhempaa jälkeläisyksilön muodostamiseen, mutta on havaittu, että joissakin tapauksissa on hyödyllistä risteyttää useampi vanhempi keskenään. Tällaista ilmiötä ei tunneta luonnosta.

Perinteisissä geneettisissä algoritmeissa, jotka käsittelevät esimerkiksi binäärimuotoisia lukuja, risteytysoperaattorien muodostaminen on varsin suoraviivaista. Kauppamatkustajan ongelma on luonteeltaan permutointia, mikä vaatii erityistä huomiota operaattorin muodostamisessa, sillä samat kaupungit tai kaaret eivät saa toistua. Seuraavassa on lueteltu toimintaideoiltaan erilaisia kauppamatkustajan ongelmalle suunniteltuja risteytysoperaattoreita kymmenen kaupungin tapauksessa:

- **IX (Inversion Crossover)** [24] Valitaan ensin kaksi satunnaista risteytymiskohtaa, esimerkiksi 3. ja 4. sekä 7. ja 8. kaupungin väliset kaaret:

s_a	1	2	3	4	5	6	7	8	9	10
s_b	7	6	2	3	1	5	4	10	8	9

Pisteiden välinen jakso s_b-kromosomista kopioidaan yhteen neljästä mahdollisesta pisteestä s_a-kromosomissa. Olkoon satunnaisesti valittu piste 3, jolloin saadaan uusi kromosomi s'_c:

s'_c	1	2	3	1	5	4	4	5	6	7	8	9	10

Kromosomista poistetaan kaupunkien kopiot, jolloin saadaan lopullinen risteytetty kromosomi:

s_c	2	3	1	5	4	6	7	8	9	10

Kromosomi s_d tuotetaan samalla tavoin, mutta nyt kopioidaan s_a kromosomiin s_b.

- **PMX (Partially Matched Crossover)** [24] Kuten edellä, valitaan ensin kaksi satunnaista risteytymiskohtaa:

s_a	1	2	3	4	5	6	7	8	9	10
s_b	7	6	2	3	1	5	4	10	8	9

Vaihdetaan alireitit 4–5–6–7 ja 3–1–5–4, jolloin saadaan

s'_c	1	2	3	3	1	5	4	8	9	10
s'_d	7	6	2	4	5	6	7	10	8	9

Havaitaan, että molemmissa kromosomeissa on samoja lukuja kahteen kertaan. Kromosomit korjataan korvaamalla risteytymispisteiden ulkopuoliset kopiot puuttuvilla arvoilla. Lopputulokseksi saadaan esimerkiksi

s_c	6	2	7	3	1	5	4	8	9	10
s_d	3	1	2	4	5	6	7	10	8	9

Jälkeläiset poikkeavat vanhemmistaan monin tavoin: esimerkiksi s_c:n kaarta (2–7) tai s_d:n kaarta (7–10) ei esiinny kummassaan vanhemmassa: niissä on tapahtunut *implisiittistä mutaatiota*. Tämä voi olla ongelmallista perinteisissä evoluutioalgoritmeissa, sillä jos uusia kaaria syntyy paljon, algoritmi rappeutuu satunnaiskävelyksi eikä kykene lähestymään optimia. Useimmat risteytysoperaattorit sisältävät tällaista mutaatiota.

- **MPX (Maximal Preservative Crossover)** on risteytysoperaattori [33], joka on kehitetty estämään implisiittistä mutaatiota. Sitä on käytetty menestyksekkäästi [30] ASPARAGOS-menetelmässä, joka on ollut jo vuosien ajan paras kauppamatkustajan ongelmalle kehitetty evoluutiostrategia. MPX-operaattori toimii seuraavasti: kahden satunnaisesti valitun risteytymispisteen välinen alireitti kopioidaan ensimmäisestä vanhemmasta jälkeläiseen. Ensimmäiseksi risteytymispisteeksi on valittu piste, joka sijaitsee kaarella, jota ei ole toisessa vanhemmassa:

s_a	5	3	9	1	2	8	10	6	7	4
s_b	1	2	5	3	9	4	8	6	10	7
s'_c	9	1	2	8	-	-	-	-	-	-

Alireittiä kasvatetaan lisäämällä kaupunkeja toisesta vanhemmasta. Kaari (8–6) on vapaa, joten kaupunki 6 voidaan lisätä. Samaten voidaan lisätä kaupungit 10 ja 7.

s'_c	9	1	2	8	6	10	7	-	-	-

Kaupunki 1 on jo lisätty, joten seuraavaksi lisätään ensimmäisen vanhemman kaupunki 4, joka on kaupunki 7:n naapuri. Ainoa vapaa naapurikaupunki on ensimmäisen vanhemman kaupunki 5, joka lisätään jälkeläiseen. Kaupunki 3 lisätään samalla tavoin:

s_c	9	1	2	8	6	10	7	4	5	3

Jos sopivaa kaarta ei löydy, joudutaan lisäämään uusi kaari, jota ei esiinny kummassakaan vanhemmassa. Menetelmä kykenee säilyttämään suurelta osin vanhemmilta perityt kaaret.

- **DPX (Distance Preserving Crossover)** [11] Muunteluoperaattorien voidaan ajatella tekevän eräänlaisia ”hyppyjä” kelpoisuusmaastossa. Sopivassa tilanteessa tuloksena on jälkeläinen, joka sijaitsee nykyistä paremman optimin vetovoima-alueella. Kuitenkin jos kahden vanhemman välinen etäisyys (poikkeavien kaarien

lukumäärä) on pieni, on selvää, että ”hyppy” jää pieneksi ja jälkeläisestä tulee hyvin vanhempiensa kaltainen. Mikäli kyseessä on paikallista hakua käyttävä hybridialgoritmi, jälkeläinen todennäköisesti jää samaan paikalliseen optimiin.

DPX-operaattori säilyttää kahden vanhemman välisen etäisyyden, eli jälkeläinen sijaitsee yhtä kaukana kuin vanhemmat toisistaan. Tällöin on todennäköisempää, että syntyvä ”hyppy” on tarpeeksi suuri, jotta jälkeläinen siirtyy pois optimialueelta. Perinteisten evoluutioalgoritmien tapauksessa tämä estää konvergenssin, joten operaattori soveltuu vain hybridialgoritmeille, jossa paikallinen haku optimoi jälkeläisen.

Menetelmä toimii seuraavasti: ensin etsitään alireitit, jotka löytyvät molemmista vanhemmista:

s_a	5	3	9	1	2	8	10	6	7	4
s_b	1	2	5	3	9	4	8	6	10	7

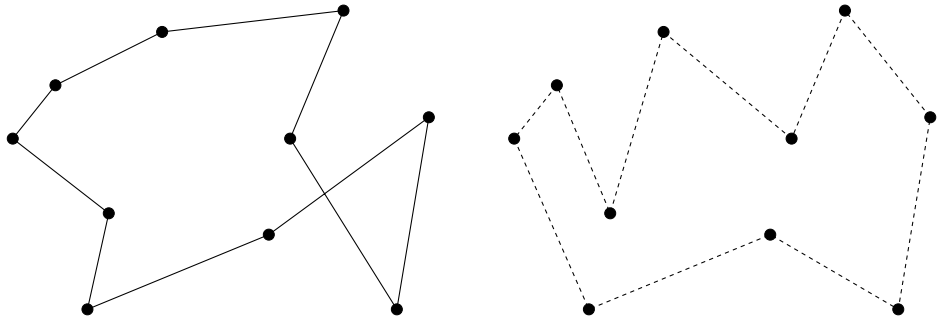
Näitä ovat (5–3–9), (1–2) ja (10–6). Jotta etäisyys säilyisi, osat yhdistetään järjestyksessä, jota ei löydy kummastakaan vanhemmasta. Alireittien yhdistämisessä voidaan käyttää esimerkiksi ahnetta lähimmän naapurin menetelmää. Lopputulokseksi saadaan esimerkiksi

s_c	6	10	5	3	9	8	7	2	1	4

- **EAX (Edge Assembly Crossover)** on Nagatan ja Kobayashin kehittämä tehokas risteytysoperaattori [46], joka poikkeaa tyypillisistä operaattoreista siinä, ettei se ole ”sokea”, toisin sanoen informaatiota paikallisesta ympäristöstä käytetään hyväksi alireittien muodostamisessa ja operaattori sisältää implisiittisesti paikallishaun.

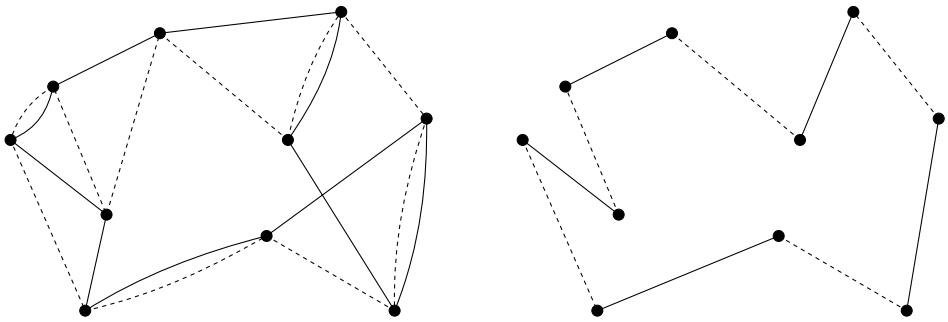
  Operaattori toimii seuraavasti: Ensin yhdistetään kahden vanhemman (A ja B, kuva 3.4) reitit uudeksi graafiksi G (kuva 3.5). Jos molemmat vanhemmat sisältävät saman kaaren, tulee G:hen tällöin kaksinkertainen kaari. G:stä valitaan satunnainen kaupunki, joka kuuluu vanhempaan A. Seuraavaksi valitaan kaupungista lähtevä kaari, joka kuuluu vanhempaan B. Jos vaihtoehtoja on useampia, kaari valitaan satunnaisesti. Seuraavat kaupungit valitaan vuorotellen A:sta ja B:stä.

  Kaaria lisätään kunnes vapaata kaarta ei enää löydy. Mikäli viimeinen kaupunki ei ole lähtökaupunki, eli reitti sisältää muutakin kuin silmukan, ylimääräinen osa poistetaan. Tällöin reitistä on muodostunut niin kutsuttu *AB-silmukka* (kuva 3.5). AB-silmukka on G:hen kuuluva alisilmukka, jossa on parillinen määrä



Kuva 3.4: Vanhemmat A ja B.

kaaria siten, että joka toinen kaarista kuuluu A:han ja joka toinen B:hen. AB-silmukassa kaupungit voivat toistua, mutteivät kaaret. Kaksinkertaiset kaupunkien väliset kaaret sallitaan, sillä on selvää, että kaaret ovat peräisin eri vanhemmista. Kun AB-silmukka on löydetty, se tallennetaan ja poistetaan G:stä. Uusia AB-silmukoita muodostetaan samalla tavoin, kunnes R:ssä ei ole enää yhtään kaarta eli se on hajoitettu kokonaan AB-silmukoiksi.

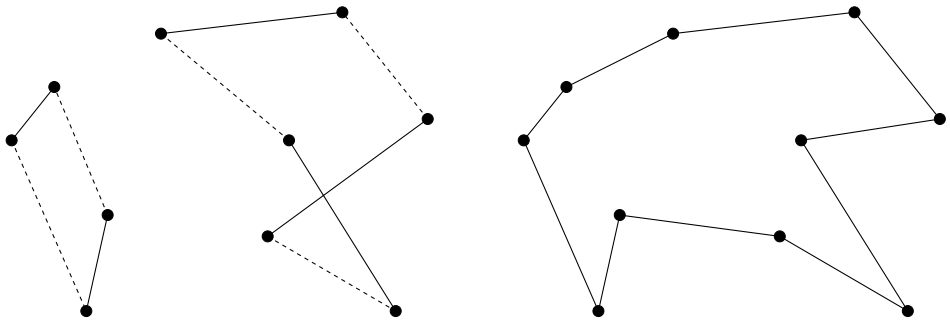


Kuva 3.5: Graafi G ja AB-silmukka, joka muodostaa E-joukon.

Uusi jälkeläinen C muodostetaan *E-joukon* avulla, joka puolestaan saadaan valitsemalla osa AB-silmukoista. E-joukko voidaan muodostaa valitsemalla silmukat satunnaisesti tai käyttämällä heuristista menetelmää, jossa otetaan huomioon muodostettavan reitin pituus sekä populaation diversiteetin säilyttäminen.

Väliaikaisen jälkeläisen C muodostaminen aloitetaan kopioimalla vanhempi A. Jos E-joukosta löytyy kaari, joka kuuluu A:han, se poistetaan jälkeläisestä. Vastaavasti, jos E-joukosta löytyy B:hen kuuluva kaari, se lisätään jälkeläiseen. Lopulta C sisältää joukon toisiinsa liittymättömiä alireittejä (kuva 3.6).

Viimeisessä vaiheessa jälkeläisestä muodostetaan sallittu reitti. Sen rakentaminen aloitetaan valitsemalla alireitti, joka sisältää vähiten kaaria. Siihen lisätään ahneella menetelmällä alireitti siten, että kokonaisreitti kasvaa mahdollisimman



Kuva 3.6: Syntyneet alireitit ja valmis jälkeläinen C.

vähän. Näin jatketaan, kunnes jälkeläisestä on saatu sallittu reitti (kuva 3.6).

3.8.2 Mutaatio

Mutaation tarkoitus on luoda uudentyyppisiä genotyyppejä [16], jotka auttavat algoritmia siirtymään uusien optimien vetovoima-alueille. Mutaatiota sovelletaan aina yhteen genotyyppiin kerrallaan ja se muodostaa yksilöstä hieman erilaisen mutantin.

Kuten risteytys, mutaatio on aina satunnaista. Kun tyypillisesti lähes jokainen jälkeläinen tuotetaan risteyttämällä, mutaatiota sovelletaan huomattavasti harvemmin, geneettisten algoritmien tapauksessa usein korkeintaan muutamaan prosenttiin. Syynä tähän on se, että mutaatiot yleensä heikentävät yksilöä. Liian suuri mutaatiotiheys hidastaa algoritmin konvergenssia kohti optimipistettä. Toisaalta massiivisesta mutaatiosta voi olla hyötyä, kun algoritmin eteneminen pysähtyy diversiteetin kadotessa. Mutatoimalla voimakkaasti populaation yksilöitä populaation diversiteetti kasvaa räjähdysmäisesti, jolloin toivottavasti päästään pois paikallisesta optimista.

Mutaatio-operaattoria vastaava relaatio on [43]

$$M : I \times \delta \rightarrow I.$$

Kuten risteytyksen tapauksessa, δ kuvaa mutaation todennäköisyyttä.

Permutaatio-ongelmille on olemassa lukuisia mutaatiomenetelmiä [24]. Jokaisessa tapauksessa alireitit valitaan satunnaisesti:

- **Reitinvaihto (swap):** Kaksi alireittiä vaihdetaan keskenään:

s_a	1	2	3	**4**	**5**	6	**7**	**8**	9	10
s_m	1	2	3	**7**	**8**	6	**4**	**5**	9	10

- **Lisäys (insertion):** Alireitti siirretään satunnaiseen paikkaan:

s_a	1	2	3	4	5	**6**	**7**	**8**	9	10
s_m	1	2	3	**6**	**7**	**8**	4	5	9	10

- **Kääntäminen (inversion):** Alireitti käännetään päinvastaiseksi:

s_a	1	2	3	**4**	**5**	**6**	**7**	8	9	10
s_m	1	2	3	**7**	**6**	**5**	**4**	8	9	10

- **Kaarenvaihto (exchange):** Ensin katkaistaan k kaarta ja uusi kelvollinen reitti muodostetaan lisäämällä k uutta kaarta. Tapauksessa $k = 4$ saadaan esimerkiksi

s_a	1	2	3	4	5	6	7	8	9	10
s_m	1	2	7	8	5	6	3	4	9	10

  Katkaistut linkit ovat tässä tapauksessa (2, 3), (4, 5), (6, 7) ja (8, 9). Tapaus $k = 2$ on käytännössä sama kuin alireitin kääntäminen.

4 Paikalliset hakuheuristiikat

Paikalliset hakuheuristiikat tutkivat nimensä mukaisesti tutkittavan pisteen lähiympäristöä. Seuraavassa tutustutaan näiden menetelmien toimintaperiaatteeseen. Varsinaisista paikallishakumenetelmistä esitellään kauppamatkustajan ongelmalle kehitettyjä menetelmiä. Lisäksi käydään läpi joukko yleisimpiä paikallishakuja ohjaavia metaheuristiikkoja.

4.1 Naapurustot

Pisteen s *naapurustolla* tarkoitetaan niiden pisteiden joukkoa $\mathcal{N}(s)$, johon voidaan siirtyä yhdellä operaattorin M toimenpiteellä [30]. Olkoon esimerkiksi M operaattori, joka vaihtaa kaksi kaupunkia keskenään ja s reitti

s	1	2	3	4	5	6	7	8	9	10

Tällöin reitti

s'	1	2	**7**	4	5	6	**3**	8	9	10

kuuluu s:n naapurustoon.

4.2 Paikalliset hakualgoritmit

Menetelmää, joka tutkii annetun pisteen ympäristöä paremman ratkaisun löytämiseksi, kutsutaan *paikalliseksi hauksi* (local search) [16]. Kuvassa 4.1 on esitelty paikallinen hakualgoritmi yleistetyssä muodossa. Paikallisia hakualgoritmeja on käytetty jo kauan tietotekniikassa. Esimerkiksi kauppamatkustajan ongelmalle kehitettiin ensimmäinen paikallinen haku jo 1950-luvulla [5].

Paikallisen optimin löytämisessä ne ovat tehokkaita [30], sillä useimmiten naapurin arvoa ei tarvitse laskea suoraan. Riittää, että lasketaan nykyisen ehdokkaan s ja naapuripisteen s' välinen muutos $\Delta f = f(s) - f(s')$. Tällöin algoritmin vaativuus naapuriratkaisua laskettaessa putoaa $\mathcal{O}(n)$:stä $\mathcal{O}(1)$:een. Esimerkiksi kauppamatkustajan ongelmaa ratkaistaessa näin voidaan tehdä.

```
procedure LS
begin
    s_paras = s; /* s jokin alkuratkaisu */
    it = 0;
    repeat
        repeat
            /* etsi uusi naapuri s' jollakin siirtymäoperaattorilla */
            s' := Operaattori(s);
            if f(s') < f(s_paras) then
                s_paras = s';
        until valintaehto saavutettu;
        s := s_paras;
        it := it + 1;
    until it = it_max; /* syvyysehto */
end;
```

Kuva 4.1: Paikallishakualgoritmi pseudokoodina.

Paikallisen haun tehokkuus riippuu voimakkaasti naapuruston valinnasta. Naapuruston kokoa kasvattamalla algoritmi kykenee löytämään parempia ratkaisuja. On kuitenkin huomattava, että algoritmin tehokkuus heikkenee nopeasti naapuruston kasvaessa. Algoritmien paikallisuudesta seuraa luonnollisesti se, etteivät ne sovellu sellaisenaan globaaliin optimointiin. Mikäli algoritmiin ei sisälly jonkinlaista mahdollisuutta ”peruuttaa”, se ei kykene paikalliseen optimiin saavuttuaan enää jatkamaan etenemistään.

Paikallisesta hausta voidaan erottaa kolme eri komponenttia [16]: tärkein on sääntö, joka määrää mihin pisteeseen algoritmi siirtyy nykyisestä pisteestä. Esimerkiksi *jyrkimmän laskun* menetelmässä algoritmi siirtyy arvoltaan parhaimpaan tutkittuun naapuripisteeseen, eli sen voi kuvitella minimointitehtävän ollessa kyseessä kulkevan aina jyrkimmän rinteen kautta. *Ahne haku* siirtyy välittömästi ensimmäiseen löytyneeseen nykyistä parempaan pisteeseen.

Siirtymäoperaattori M määrittelee algoritmin naapuruston ja sen myötä kelpoisuusmaaston, jossa paikallishaku voi liikkua. Kelpoisuusmaaston muoto riippuu valitusta siirtymäoperaattorista, joten sen valinnalla voi olla ratkaiseva merkitys paikallisen haun tehokkuuteen. Jonkin operaattorin paikallinen optimi ei olekaan optimipiste toisen operaattorin tapauksessa, ja käyttämällä erilaisia siirtymäoperaattoreita voidaan algoritmia estää pysähtymästä.

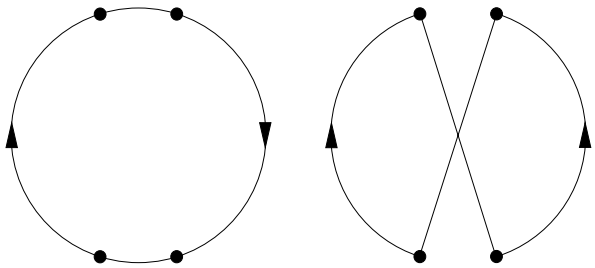
Haun syvyys kertoo, kuinka kauan algoritmin iteraatiota jatketaan. Paikallinen ha-

ku voidaan esimerkiksi pysäyttää heti kun se on siirtynyt parempaan pisteeseen, tai sitä voidaan jatkaa, kunnes parempia pisteitä ei enää löydy (eli se on saavuttanut paikallisen optimin).

4.3 k-Opt -menetelmät

k-Opt -menetelmät ovat klassisia kaarenvaihtomenetelmiä [19], joissa vaihdetaan k kaarta keskenään. Ne perustuvat λ-optimaalisuuteen [17]: reitti on λ-*optimaalinen*, jos on mahdotonta korvata reitin λ:n suuruista alijoukkoa toisella λ:n kokoisella alijoukolla niin, että saadaan aikaiseksi lyhyempi reitti. Luonnollisesti se on myös λ'-optimaalinen, jos $0 \leq \lambda' \leq \lambda$.

2-Opt on yksinkertainen k-Opt -menetelmä, joka on peräisin jo vuodelta 1958 [19]. Siinä reitistä poistetaan kaksi kaarta ja kaaret yhdistetään uudelleen vaihtaen kohdekaupungit keskenään (kuva 4.2; siirtymän havainnollistamiseksi kaupungit on järjestetty renkaaksi). Siis jos esimerkiksi alkuperäinen reitti kulki kaupungista 1 kaupunkiin 2 ja kaupungista 5 kaupunkiin 6, 2-Opt -operaation jälkeen reitti kulkee kaupungista 1 kaupunkiin 6 ja kaupungista 5 kaupunkiin 2. Samalla kaupunkien 2 ja 6 välinen reitti vaihtaa suuntaansa.

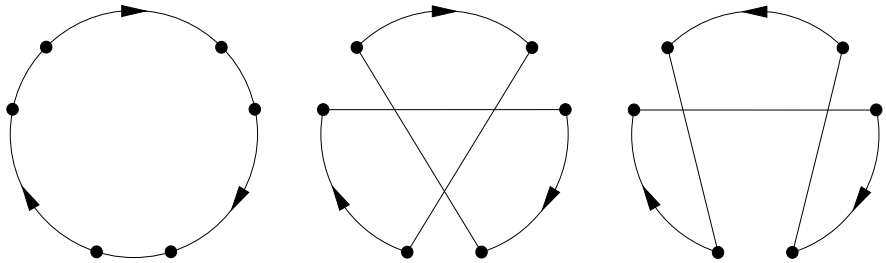


Kuva 4.2: Kahden kaaren vaihto.

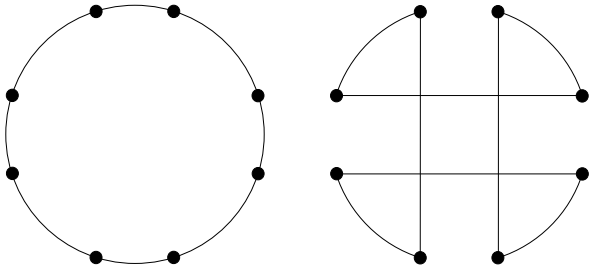
3-Opt -menetelmä on samankaltainen kuin 2-Opt, mutta nyt vaihdetaan kolme kaarta keskenään. Tällöin tulokseksi saadaan kaksi uutta reittivaihtoehtoa (kuva 4.3). Samalla naapuruston koko kasvaa 2-Opt-menetelmän $O(N^2)$:sta $O(N^3)$:een.

2-Opt ja 3-Opt -menetelmien tapauksessa kaarenvaihdot ovat aina peräkkäisiä. Kun $k \geq 4$ voidaan suorittaa myös kaarenvaihtoja, jotka eivät ole perättäisiä. Eräs tällainen kaarenvaihto on esitelty kuvassa 4.4. Edellisessä luvussa mutaatioiden yhteydessä esimerkkinä annettu kaarenvaihto-operaattori on tällainen *ei-perättäinen neljän kaaren vaihto* (non-sequential four edge exchange).

Kaarenvaihdot voidaan yleistää k:hon kaarenvaihtoon. Luonnollisestikin algoritmin tarkkuus paranee k:ta kasvattamalla, mutta koska suoritusaika kasvu on eksponenti-



Kuva 4.3: Kaksi tapaa vaihtaa kolme kaarta.



Kuva 4.4: Ei-perättäinen neljän kaaren vaihto.

aalinen k:n suhteen ($O(N^k)$), vain pienet k:n arvot ovat järkeviä. Menetelmät, joissa $k > 5$ ovat harvinaisia.

4.4 Lin-Kernighan -algoritmi

Lin-Kernighan -algoritmi on tehokas kauppamatkustajan ongelmalle suunniteltu ratkaisualgoritmi. Sen kehittivät Lin ja Kernighan vuonna 1973 [28] ja aina vuoteen 1989 se oli paras tunnettu kauppamatkustajan ratkaisualgoritmi [30]. Se perustuu k-Opt -menetelmien tavoin λ-optimaalisuuteen. Algoritmin tehokkuus piilee siinä, että se kykenee muuttamaan λ:n arvoa, eli se kykenee tekemään monimutkaisia kaarenvaihtoja ilman suoritusajan eksponentiaalista kasvua.

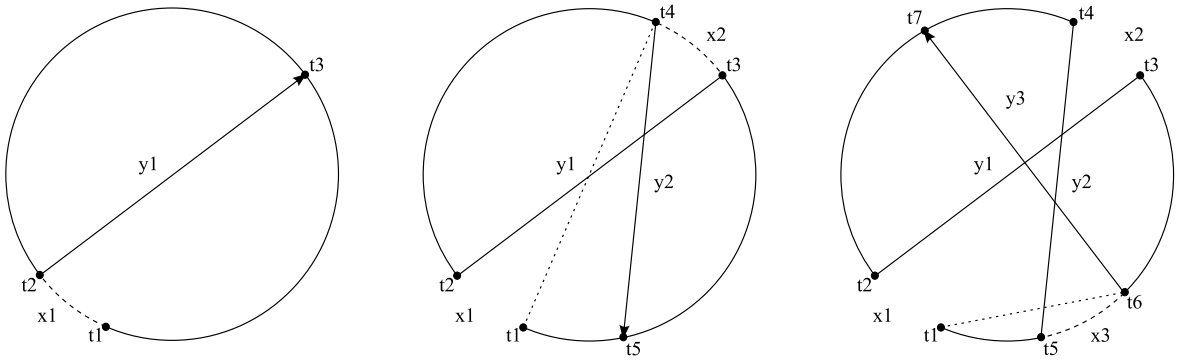
Algoritmin tehokas toteutus on mutkikas, mutta sen perusidea on varsin yksinkertainen:

- Ensin valitaan piste t_1 ja sitä seuraava piste t_2.
- Seuraavaksi valitaan piste t_3 siten, että muodostettava uusi kaari (y_1) on x_1:tä lyhyempi (kuva 4.5, vasen tapaus).
- Jos tällainen piste löytyy, valitaan uusi poistettava kaari x_2, jonka toinen päätepiste on piste t_3.

- Muodostetaan uusi kaari y_2, joka yhdistää pisteet t_4 ja t_1. Mikäli uusi reitti on entistä lyhyempi, aloitetaan operaatio alusta valitsemalla uusi t_1.
- Muussa tapauksessa y_2:n uudeksi päätepisteeksi valitaan uusi piste t_5 niin, että y_2 on x_2:sta lyhyempi (kyva 4.5, keskimmäinen tapaus).
- Kaari x_3 valitaan kuten x_2. Kun t_6 yhdistetään t_1:een, reitistä pitää tulla ehjä. Jos tämä ei ole mahdollista, etsitään uusi t_3. Samoin tehdään aina, kun kaarenvaihto on edennyt kahta kaarta syvemmälle eikä ehjää reittiä voida muodostaa.
- Näin jatketaan, kunnes vaihdettavia kaaria ei enää löydy.

Lin-Kernighan -haku siis vaihtaa ensin kaksi kaarta, eli se vastaa tällöin normaalia 2-Opt -menetelmää. Algoritmin tekemät kaarenvaihdot ovat aina perättäisiä, joten kun $k \geq 4$ se ei pysty löytämään kaikkia mahdollisia vaihtoja.

Etsiessään uusia kaarenvaihtoja haku suorittaa runsaasti silmukoita, jonka seurauksena algoritmin suoritusaika on varsin pitkä. Lin ja Kernighan lisäsivät algoritmiinsa joukon heuristisia menetelmiä, joilla sitä voitiin lyhentää. Suoritusaikaa on saatu edelleen lyhennettyä käyttämällä kehittyneempiä tietorakenteita. Eräs hyvin tehokas useita muita menetelmiä hyödyntävä parannus algoritmiin on esitelty lähteessä [17].



Kuva 4.5: Lin-Kernighan -algoritmin eteneminen.

4.5 Paikallishakuja ohjaavat metaheuristiikat

Tehokkaatkin paikallishakualgoritmit pysähtyvät yleensä ennen saapumistaan globaaliin optimipisteeseen. Suoraviivaisin tapa on yksinkertaisesti aloittaa uudesta, satunnaisesta pisteestä. Valitsemalla alkupisteet joko satunnaisesti tai esimerkiksi sijoittamalla ne mahdollisimman kauaksi toisistaan tutkittavan alueen maksimoimiseksi saadaan niin kutsuttu *multistart-paikallishaku*.

4.5.1 Iteroitu paikallishaku

```
procedure ILS
begin
    s_0 := LuoAlkuRatkaisu();
    s* := PaikallisHaku(s_0);
    repeat
        s' := Häiritse(s*);
        s*' := PaikallisHaku(s');
        s* := Valitse(s*, s*');
    until lopetusehto saavutettu;
end;
```

Kuva 4.6: Iteroitu paikallishaku.

Iteroitu paikallishaku (iterated local search) on yksinkertainen paikallishakua ohjaava metaheuristiikka [29]. Siinä paikallishaun annetaan ensin edetä paikalliseen optimipisteeseen. Saatua ratkaisua häiritään, jolloin algoritmi siirtyy pois paikallisen optimipisteen vetovoima-alueelta ja toivon mukaan jatkaa etenemistään kohti globaalia optimipistettä. Algoritmi on esitetty kuvassa 4.6. Iteroitu paikallishaku on käytännössä sama kuin evoluutioalgoritmi, jossa ei käytetä risteytystä ja jossa populaation kooksi on valittu yksi.

4.5.2 Simuloitu jäähdytys

Simuloidun jäähdytyksen (simulated annealing) toimintaperiaate on peräisin hehkutuksesta. Se on fysikaalinen prosessi, jossa kiinteä kappale laitetaan lämpökylpyyn [20]. Aluksi lämpötila nostetaan niin korkeaksi, että kappale sulaa. Sen jälkeen lämpötilaa lasketaan hyvin varovaisesti, jolloin aluksi satunnaisessa järjestyksessä olevat hiukkaset järjestäytyvät mahdollisimman alhaiseen energiatilaan, eli järjestäytyneeksi hilaksi.

Tätä fysikaalista prosessia voidaan simuloida tietokoneella [30] (kuva 4.7). Olkoon kiinteän kappaleen nykyisen tilan i energia E_i. Jos systeemiä häiritään, esimerkiksi siirtämällä yhtä hiukkasta, saavutaan uuteen tilaan j, jonka energia on vastaavasti E_j. Mikäli systeemi siirtyy alempaan energiatilaan, eli muutos $\Delta E = E_j - E_i \leq 0$, valitaan j nykyiseksi tilaksi. Muussa tapauksessa tila j hyväksytään todennäköisyydellä $p = e^{-\frac{\Delta E}{kT}}$, missä T on lämpökylvyn lämpötila ja k Boltzmannin vakio. Hyväksymissääntö tunnetaan *Metropolis-kriteerinä*. Kuvan 4.7 algoritmissa yksilön kelpoisuus vastaa energiatilaa. Boltzmannin vakio sisältyy implisiittisesti lämpötilan päivitykseen

```
procedure SA
begin
    t := T(0); n := 1; /* T(0) valittu alkulämpötila */
    s_paras := s; /* s valittu alkuratkaisu */
    repeat
        s' := EtsiNaapuri(s); /* s' ∈ N(s) */
        Δf := f(s) − f(s');
        /* Valitse uusi ratkaisu jos se on edellistä parempi tai raja-arvo ylittyy. */
        n := random(0, 1);
        if Δf ≤ 0 or e^(−Δf/t) > n then s := s';
        if f(s) > f(s_paras) then s_paras := s;
        t := T(n); /* Päivitä lämpötila */
        n := n + 1;
    until lopetusehto saavutettu;
    return s_paras;
end;
```

Kuva 4.7: Simuloitu jäähdytys

$t := T(n)$. Se voi olla esimerkiksi nykyisen ja edellisen iteraatiokierroksen lämpötilojen suhde $r \in]0, 1[$.

Simuloidun jäähdytyksen toiminta perustuu siihen, että alussa lämpötilan ollessa korkea etsintäprosessi kattaa suuren alueen, jolloin se muistuttaa globaalia hakua [27]. Jäähtymisen aikana hypyt pienenevät ja etsintä kohdistuu entistä pienemmälle alueelle optimipisteen läheisyyteen ja se alkaa käyttäytyä normaalin paikallishaun tapaan.

4.5.3 Tabuhaku

Tabuhaku (tabu search) on suosittu hakumenetelmä, jonka idean esitti ensimmäisenä Glover vuonna 1977 [24]. Yksinkertainen tabuhakualgoritmi on esitelty kuvassa 4.8. Menetelmän olennaisin piirre on muistin hyödyntäminen haun aikana. Muiden paikallishakumenetelmien tapaan tabuhaku etenee naapuripisteestä toiseen. Se on luonteeltaan deterministinen, sillä se valitsee aina kelpoisuudeltaan parhaan naapurin. Jotta haku ei jäisi jumiin paikalliseen optimiin tai jäisi pyörimään silmukkaan, niin ratkaisut, joissa menetelmä on äskettäin vieraillut määritellään ”kielletyiksi” eli *tabuiksi*. Nämä ratkaisut tallennetaan erityiseen *tabulistaan*. Joissakin tapauksissa tämä rajoittaa algoritmin etenemistä liikaa, joten menetelmä sisältää myös erityisiä *aspiraatiokriteereitä* (aspiration criteria), jotka sallivat siirron, vaikka se olisikin tabulistalla. Esimerkiksi jos pisteen tabulistalla oleva naapuri johtaa nykyistä parempaan ratkaisuun, se hyväk-

```
procedure TS
begin
    T := 0; /* tabulista */
    s_paras := s; /* s valittu alkuratkaisu */
    repeat
        do
            s' := EtsiParasNaapuri(s); /* etsi paras naapuriratkaisu */
        while s' ∈ T; /* s' ei saa kuulua tabulistaan */
        s := s';
        T := T ∪ s; /* lisätään uusi ratkaisu tabulistaan */
        if f(s) < f(s_paras) then s_paras := s;
    until lopetusehto saavutettu;
    return s_paras;
end;
```

Kuva 4.8: Tabuhaku

sytään. Tabuhaku voi sisältää useita tabulistoja ja aspiraatiokriteereitä [30].

4.5.4 Ohjattu paikallishaku

Ohjattu paikallishaku (Guided local search) on menetelmä [36], jonka periaatteena on estää haun pysähtyminen paikalliseen optimiin erityisen ohjausfunktion avulla.

Jokaisen etsintäavaruuden S ratkaisun s voidaan ajatella sisältävän tiettyjä *piirteitä* (features). Piirteeksi voidaan käsittää mitä tahansa ratkaisun ominaisuus, jota ei ole kaikilla S:n ratkaisupisteillä. Jokaiselle piirteelle r_k on olemassa funktio Ind_k, missä $Ind_k = 1$, jos s:llä on kyseinen ominaisuus, ja $Ind_k = 0$, jos sillä tätä ominaisuutta ei ole.

Funktiota Ind kerrotaan sakkolaskurilla p_k, jota kasvatetaan aina, kun r_k:ta sakotetaan. Sakkolaskuri on yleensä kokonaisluku. Yhdistämällä nämä objektifunktioon f saadaan *ohjausfunktio*

$$f(s) = f + \lambda \sum_{k=1}^{M} p_k Ind_k(s)$$

Säätelyparametri λ tasoittaa sakkolaskurin vaikutusta. Kaikki M sakkolaskuria saavat alkuarvokseen nollan.

Jotta algoritmi toimisi optimaalisesti, λ:n arvoa joudutaan virittämään. Liian pienillä λ:n arvoilla algoritmi etenee liian hitaasti, kun taas liian suurilla λ:n arvoilla algoritmi löytää vain vähän hyviä optimipisteitä. Voudouris ja Tsang [48] käyttivät

```
procedure GLS
begin
    i := 0;
    foreach piirre k do p_k := 0;
    repeat
        f(s) := f + λ Σ_k^M p_k Ind_k;
        s_{i+1} := Haku(s_i, f(s_i)); /* hakuoperaattori */
        foreach piirteelle k joka maksimoi funktion util_k do
            p_k := p_k + 1;
        i := i + 1;
    until lopetusehto saavutettu;
    return paras ratkaisu;
end;
```

Kuva 4.9: Ohjattu paikallishaku.

kauppamatkustajan ongelman tapauksessa arvoa

$$\lambda = a \times \frac{f_{2-opt}}{N}, 0 \leq a < 1,$$

missä f_{2-opt} on tyypillisen 2-optimaalisen reitin pituus. Algoritmi toimi tutkituissa tapauksissa tehokkaimmin arvon $a = 0,3$ läheisyydessä.

Piirteiden sakottamisella on haittapuolensa, sillä potentiaalisesti arvokkaat piirteet ohjaavat helposti algoritmin paikalliseen optimiin. Tällöin niitä sakotetaan useasti, eikä algoritmi enää hyödynnä niitä. Tämän estämiseksi jokaiselle piirteelle annetaan vielä kustannusarvo c_k

$$util_k(s) = Ind_k(s) \times \frac{c_k}{1 + p_k}$$

Vakio 1 estää nollalla jakamisen, kun sakkolaskuria p_k ei ole vielä kasvatettu.

Ohjattu paikallishakualgoritmi on esitelty kuvassa 4.9. Aliohjelma `Haku()` on jokin hakuoperaattori, esimerkiksi 2-Opt. Jotta algoritmi ei pysähtyisi paikalliseen optimipisteeseen, operaattoria pitää rajata niin, ettei se aina saavuta paikallista optimipistettä.

5 Memeettiset algoritmit

Evoluutioalgoritmeista voidaan kehittää entistä tehokkaampia menetelmiä, memeettisiä algoritmeja, lisäämällä niihin paikallisia hakuheuristiikkoja. Ennen varsinaisten memeettisten algoritmien esittelyä tutustutaan lamarckistiseen evoluutioon, Baldwinin ilmiöön sekä meemeihin. Memeettisten algoritmien formaalin muodon avulla voidaan muodostaa monipuolinen memeettisten algoritmien luokittelumenetelmä. Lopuksi kerrotaan yleisesti, mitä tehokkaiden algoritmien suunnittelussa tulee ottaa huomioon.

5.1 Evoluutioalgoritmien hybridisointi

Usein on havaittu, että yhdistämällä evoluutioalgoritmeja ja muita menetelmiä saadaan aikaiseksi tehokkaita hybridimenetelmiä [16]. Jotkin monimutkaiset ongelmat voidaan jakaa osaongelmiin, joille löytyy valmiita eksakteja menetelmiä tai hyviä heuristiikkoja. Laaja empiirinen todistusaineisto ja niin sanottu NFL-teoreema[1] (No Free Lunch, "ilmaisia lounaita ei ole") viittaavat vahvasti siihen, että kaikkiin ongelmiin soveltuvia yleisiä ongelmanratkaisijoita ei ole olemassa [47]. Hybridialgoritmien on havaittu käytännön kokeissa toimivan niiden yksittäisiä osa-algoritmeja tehokkaammin, koska ne voivat yhdistää eri algoritmien tunnettuja vahvuuksia.

Evoluutioalgorimien tehokkuus perustuu niiden kykyyn toimia koko etsintäavaruudessa, ja ne kykenevätkin löytämään usein hyvien ratkaisujen alueita. Kuitenkin optimipisteen läheisyydessä ne usein konvergoivat hitaasti. Paikallisten hakualgoritmien tapauksessa konvergenssi on nopeaa, mutta ne puolestaan rajoittuvat vain paikallisiin optimeihin. Niiden tehokkuus on myös riippuvainen alkuratkaisusta, kun taas evoluutioalgoritmeilla tätä ongelmaa ei ole. Kun evoluutioalgoritmiin yhdistetään paikallista hakua, voidaan algoritmista näin saada tehokas [30].

Kun evoluutioalgoritmeihin yhdistetään paikallista hakua, niiden luonne muuttuu. Evoluutioprosessin ei itse tarvitse enää löytää hyviä ratkaisuja; riittää vain, että se muodostaa ratkaisuehdokkaita, jotka sijaitsevat edellistä parempien paikallisten optimien vetovoima-alueilla. Paikallinen haku suorittaa optimiin siirtymisen. Lisäksi se voi korjata evoluutioalgoritmin muodostamia kelvottomia ratkaisuja, mikä voi helpottaa

[1] Teoreeman mukaan jokaisen algoritmin tehokkuus on keskimäärin sama, kun sitä verrataan jokaiseen mahdolliseen optimointiongelmaan. Toisin sanoen mikä tahansa algoritmi on toista algoritmia tehokkaampi jossain ongelmassa, mutta huonompi toisessa.

huomattavasti ratkaisujen esitysmuodon suunnittelua. Kehittyneemmät hakumenetelmät, kuten simuloitu jäähdytys, voivat auttaa hybridimenetelmää poistumaan paikallisista optimeista.

5.1.1 Lamarckismi ja Baldwinin ilmiö

Evoluutioalgoritmien ja paikallisten hakujen hybridit voidaan jakaa kahteen luokkaan [30]: niihin, jotka perustuvat *lamarckilaiseen evoluutioon* ja niihin, jotka hyödyntävät *Baldwinin ilmiötä*.

Lamarck esitteli vuonna 1809 evoluutioteorian, jonka mukaan yksilöiden kehitys on seurausta edellisten sukupolvien toiminnasta: klassisen esimerkin mukaan kirahvin kaula pitenee sukupolvien saatossa kunkin sukupolven kurkotellessa ylhäällä puissa olevia lehtiä kohti. Teorian perusajatuksena on se, että hankitut ominaisuudet siirtyvät seuraavaan sukupolveen. Darwin osoitti myöhemmin tämän teorian paikkaansapitämättömäksi.

Luonnosta ei tunneta mekanismia, jolla yksilön fenotyyppi voidaan koodata takaisin genotyyppiin. Evoluutioalgoritmeissa tämä on mahdollista, ja on havaittu, että optimoinnissa siitä voi olla hyötyä. Lamarckistisissa hybridialgoritmeissa — jotka muodostavat valtaosan hybridialgoritmeista — paikallista hakua sovelletaan itse yksilöön, jolloin ominaisuudet periytyvät. Näin ollen lamarckistista lähestymistapaa hyödyntävät algoritmit poikkeavat toimintaperiaatteeltaan huomattavasti luonnollisesta evoluutiosta.

Baldwinin ilmiöllä tarkoitetaan yksilön kykyä sopeutua muutoksiin [45]. Esimerkiksi ihon ruskettuminen auringonpaisteen vaikutuksesta on tällainen kyky. Rusketuksen ansiosta vaaleaihoinen henkilö kykenee oleskelemaan auringonpaahteessa. Useiden sukupolvien kuluttua populaatiossa alkaa esiintyä aitoa tummaihoisuutta, kun luonnonvalinta suosii tummempaa ihoa. Sen sijaan vanhempien rusketus ei siirry heidän genotyyppeihinsä, joten Baldwinin ilmiö on puhtaasti darwinistinen.

Baldwinin ilmiötä hyödyntävissä algoritmeissa yksilöä testataan paikallisella haulla [27]. Haun tulos tallennetaan yksilön kelpoisuudeksi, mutta sen sijaan muuttunutta genotyyppiä ei tallenneta, eikä paikallisen haun muutokset näin ollen tallennu jälkeläisiin. Baldwinistisia hybridialgoritmeja on käytetty erityisesti neuroverkkojen yhteydessä [35].

5.2 Meemit ja kulttuurievoluutio

Evoluutio käsitetään yleensä biologiseksi ilmiöksi, mutta biologinen evoluutio ei suinkaan ole ainoa mahdollinen evoluution muoto. Kirjassaan "Geenin itsekkyys" (1976) Dawkins esitteli uudenlaisen kopioitujan, *meemin* [9]. Meemi voidaan käsittää eräänlaiseksi "kulttuurigeeniksi", joka kopioituu ihmisen mielestä toiseen imitaation avulla. Esimerkiksi ajatukset, hokemat, vaatemuodit, saviruukkujen ja rakennusten kaarien muodot ovat erilaisia meemejä.

Ihmisen aivot kykenevät vastaanottamaan ja muodostamaan uusia ideoita rajallisesti, minkä seurauksena meemeihin kohdistuu valintaa [30]. Kelpoisuuden mittarina toimii tällöin esimerkiksi yksilön mieltämä hyöty. Meemien kehittyminen poikkeaa kuitenkin huomattavasti geeneistä. Satunnaiset mutaatiot ja risteytymiset ovat merkitykseltään vähäisempiä. Sen sijaan uudet meemit syntyvät usein vanhoja ideoita yhdistelemällä ja uusia keksimällä; niiden muodostamisessa voidaan hyödyntää myös muiden alojen tietoa ja ideoita. Tällaista uusien, vanhemmilta puuttuvien alleelien muodostamista ja testaamista ei luonnollisessa evoluutiossa esiinny lainkaan.

Kulttuurievoluutio vaatii vähän resursseja verrattuna luonnolliseen evoluutioon. Lisäksi kulttuurievoluutio suuntautuu jotain päämäärää kohti, kun taas luonnollisessa evoluutiossa kehitys johtaa vain paremmin sopeutuneisiin yksilöihin. Tämä ja meemien tapa kehittyä johtavat siihen, että kulttuurievoluutio on huomattavasti luonnollista evoluutiota nopeampaa, mikä on selvästi havaittavissa verrattaessa ihmisen kulttuurin kehitystä lajimme biologiseen kehitykseen.

5.3 Memeettinen algoritmi

Evoluutioalgoritmia, jonka yhdessä tai useammassa vaiheessa yksilöihin sovelletaan tehostamista jonkin etsintästrategian avulla, kutsutaan *memeettiseksi algoritmiksi* [16]. Tyypillisimmillään tämä tarkoittaa geneettistä algoritmia, jossa sovelletaan paikallista hakua jossain evoluutiosyklin vaiheessa. Kuvassa 5.1 on tyypillinen memeettinen algoritmi, jossa yksilöitä tehostetaan joka kerta niitä muuteltaessa.

Termin "memeettinen algoritmi" esitti ensimmäisenä Moscato vuonna 1989 [37]. Memeettisten algoritmien yhteydessä *meemillä* tarkoitetaan algoritmeissa käytettäviä etsintästrategioita, jotka ohjaavat hakuprosessia kohti haluttuja pisteitä. Kuten myöhemmin käy ilmi, nämä hakustrategiat käsittävät laajan joukon erillaisia menetelmiä.

Kirjallisuudessa memeettiset algoritmit tunnetaan asiayhteydestä ja rakenteesta riippuen useilla eri nimityksillä, kuten *hybridiset geneettiset algoritmit* (hybrid GAs), *baldwinilaiset evoluutioalgoritmit* (Baldwinian EAs), *lamarckistiset evoluutioalgoritmit*

```
procedure MA
begin
    /* muodosta yksilöt satunnaisesti tai jollakin reitinmuodostusalgoritmilla */
    AlustaPopulaatio(P);
    foreach s ∈ P do
    begin
        s := Paikallishaku(s); /* yksilön tehostaminen paikallishaulla */
        LaskeKelpoisuus(s);
    end;
    repeat
        P_j := 0; /* P_j jälkeläisten populaatio */
        for i := 1 to risteytysten lkm do
        begin
            P_v := ValitseVanhemmat(P); /* P_v vanhempien populaatio */
            s := Risteytä(P_v); /* valitse vanhemmat satunnaisesti */
            n := random(0, 1);
            /* sovella mutaatiota yksilöön todennäköisyydellä p_m */
            if n < p_m then s := Mutatoi(s);
            s := Paikallishaku(s); /* paikallishakuvaihe */
            LaskeKelpoisuus(s);
            Lisää(P_j, s);
        end;
        P := ValitseJälkeläiset(P ⋃ P_j); /* selviytymisvalinta */
    until lopetusehto saavutettu;
end;
```

Kuva 5.1: Memeettinen algoritmi.

(Lamarckian EAs) ja *geneettiset paikallishakualgoritmit* (genetic local search algorithms). On kuitenkin huomattava, että termi on edelleen kiistanalainen [24]. On myös kritisoitu sitä, etteivät yksinkertaiset staattiset geneettisten algoritmien ja paikallisten hakujen hybridit juurikaan vastaa alkuperäistä meemien ideaa [25].

Memeettiset algoritmit ovat osoittautuneet erittäin tehokkaaksi menetelmäksi joissakin optimointitehtävissä. Ne vaativat usein vähemmän laskentaa verrattuna perinteisiin evoluutioalgoritmeihin saavuttaen kuitenkin parempia ratkaisuja, mikä on tehnyt niistä suosittuja. Memeettisiä algoritmeja on käytetty muun muassa kauppamatkustajan ongelman, proteiinien laskostuksen, aikataulujen suunnittelun, väritysongelman, selä kvadraattisen ohjelmointiongelman ratkaisemisessa.

5.4 Formaali muoto

Geneettinen algoritmi voidaan formalisoida muotoon [43]

$$GA = (P^0, \delta^0, \lambda, \mu, l, f, G, U),$$

missä $P^0 \in I^\mu$ on alkupopulaatio, μ populaation koko ja λ jälkeläisten lukumäärä. I on ongelman esitysmuoto ja l on sen pituus. Reaaliluku δ^0 määrittää operaattorien alkuasetukset, esimerkiksi mutaation todennäköisyyden algoritmin käynnistyessä. f on kelpoisuusfunktio. Risteytys- ja mutaatio-operaattorit sisältyvät generointifunktioon $G : I^\mu \rightarrow I^\lambda$ ja päivitysfunktio $U : I^\mu \times I^\lambda \rightarrow I^\mu$ sisältää valintaoperaattorien lisäksi menetelmät, jotka huolehtivat diversiteetin säilymisestä.

Jälkeläisten joukko on $O \in I^\lambda$, jolloin algoritmin iteraatiosta saadaan

$$\begin{aligned} O_i^t &= M(R(P^t, \delta^t), \delta^t) \; \forall \; i \in \{1, ..., \lambda\} \\ P^{t+1} &= U(O^t, P^t), \end{aligned}$$

missä t on aika-askel.

Memeettisiä algoritmeja varten relaatiota laajennetaan [23] paikallisten hakualgoritmien joukolla

$$\mathcal{L} = \{L_1, ..., L_m\}$$

Algoritmeja, jotka käyttävät useampaa hakualgoritmia ($m > 1$) kutsutaan *multimemeettisiksi algoritmeiksi* (multimemetic algorithms). Yksittäiset paikalliset hakualgoritmit ovat tällöin

$$L_j : I^{c_1} \times \zeta \rightarrow I^{c_1},$$

missä ζ on kyseiseen hakuoperaattoriin liittyvä parametri (vrt. δ), j on indeksi joukossa $\mathcal{L}$ ja c_1 vakio, joka määrää, kuinka montaa ratkaisua paikallinen haku käsittelee ja kuinka monta ratkaisua se palauttaa. Normaalisti hakuoperaattori käsittelee yhtä yksilöä kerrallaan (esimerkiksi mutaation tai risteytyksen jälkeen), jolloin $c_1 = 1$. $c_1 = 2$, jos paikallishakua käytetään risteytyksessä eli paikallishaku käsittelee kahta vanhempaa samanaikaisesti, ja niin edelleen.

5.5 Kehittyneemmät memeettiset algoritmit

Memeettisillä algoritmeilla voidaan tarkoittaa hyvin eri tyyppisiä algoritmeja, joille on yhteistä yksilöiden tehostaminen evoluutiosyklin eri vaiheissa. Memeettisistä algoritmeista voidaan muodostaa malli, jonka avulla prosessin eri vaiheita voidaan havaita.

5.5.1 Aikatauluttajat

Paikallishakuoperaattorien integrointia evoluutioalgoritmiin sekä niiden ja evoluutiooperaattorien säätelyä voidaan kuvata niin kutsutuilla *aikatauluttajilla* (schedulers) [23].

Hienosäätöaikatauluttaja (Fine-Grain Scheduler) ohjaa missä, milloin ja millä parametreilla joukon $\mathcal{L}$ hakuoperaattoreita sovelletaan risteytys- ja mutaatiovaiheissa. Se voidaan esittää relaationa

$$fS : (I^{c_1} \times \delta \rightarrow I) \times \mathcal{L} \times I^{c_1} \times \delta \times \zeta \rightarrow I$$

fS hyödyntää kolmea argumenttia. Ensimmäinen on uusia ratkaisuja muodostava funktio $I^{c_1} \times \delta \rightarrow I$. Se on risteytys (kun $c_1 = \mu$) tai mutaatio (kun $c_1 = 1$). Toinen argumentti on aikataulutettavien hakuoperaattorien joukko $\mathcal{L}$, jota kuvaava funktio on muotoa $I^{c_2} \times \zeta \rightarrow I^{c_2}$. Tyypillisesti c_2 saa arvon 1, eli hakuoperaattoria käytetään vasta risteytyksen tai mutaation jälkeen syntyneeseen yksilöön. Toisaalta jos esimerkiksi paikallista hakua käytetään vanhempiin ennen hakua, niin c_2 on sama kuin c_1 (eli tavallisen risteytyksen tapauksessa $c1 = c2 = 2$, siis molempia vanhempia tehostetaan paikallisella haulla). Kolmas argumentti on muodostettujen jälkeläisten joukko I^{c_1}. Lisäksi se saa aiemmin mainitut operaattorikohtaiset parametrit δ ja ζ.

Yksinkertaisissa tapauksissa hienosäätö voidaan jakaa erillisiin mutaatio- ja risteytysaikatauluttajiin:

$$\begin{aligned} fS_M &: (I \times \delta \rightarrow I) \times \mathcal{L} \times I \times \delta \times \zeta \rightarrow I \\ fS_R &: (I^{\mu} \times \delta \rightarrow I) \times \mathcal{L} \times I^{\mu} \times \delta \times \zeta \rightarrow I \end{aligned}$$

Esimerkiksi jos algoritmissa yksilöä tehostetaan paikallisella haulla mutaation jälkeen mutaatioaikatauluttaja on muotoa $fS_M(M, L_1, s, \delta, \zeta_1)$. M on mutaatio-operaattori, jota sovelletaan todennäköisyydellä δ. L_1 on paikallishaku parametreilla ζ_1 ja s on käsiteltävä yksilö.

Aikatauluttajat määritellään hyvin yleisellä tasolla. Siten memeettinen algoritmi, joka soveltaa paikallista hakua ensin risteytyksen jälkeen ja sitten mutaation jälkeen, käyttää näitä aikatauluttajia. Samaten jos paikallista hakua käytetään risteytysoperaattorin sisällä, on kyseessä fS_R-aikatauluttaja.

Karkeasäätöaikatauluttaja (Coarse-Grain Scheduler) koordinoi populaation tasolla toimivaa paikallista hakua:

$$cS : (I^{\mu} \times I^{\lambda} \rightarrow I^{\mu}) \times \mathcal{L} \times I^{\mu} \times I^{\lambda} \times \delta \times \zeta \rightarrow I^{\mu}$$

Parametreina ovat päivitysfunktio $U : (I^{\mu} \times I^{\lambda} \rightarrow I^{\mu})$, paikallisten hakuoperaattorien joukko $\mathcal{L}$ sekä vanhempien (I^{μ}) ja jälkeläisten joukot (I^{λ}). Lisäksi tarvitaan operaattoreiden parametrien joukot δ ja ζ. Tämä aikatauluttaja säätää hakuoperaattoria, jota

käytetään vanhempien tai jälkeläisten joukkoihin tai niiden yhdisteeseen. Olennaisin ero karkeasäädön ja hienosäädön välillä on siinä, että jälkimmäinen prosessoi yhtä yksilöä kerrallaan, kun taas karkeasäätö toimii populaation tasolla.

Karkeasäätöaikatauluttajan avulla voidaan muodostaa tehokkaita algoritmeja, jotka hyödyntävät dynaamisesti useita erilaisia hakuoperaattoreita. Karkeasäätö mahdollistaa myös osittaisen lamarckismin, jossa paikallista hakua käytetään hyvien alueiden etsintään ilman, että hakua suoritetaan loppuun asti. Samaten populaatiosta saatavaa tietoa voidaan käyttää hyödyksi siten, että paikallishakua käytetään valikoiden vain joihinkin yksilöihin.

Meta-aikatauluttaja (Metascheduler) on ylimmän tason operaattori. Se kuvataan relaatiolla

$$mS : \mathcal{L} \times H_P^t \times I^\mu \times \delta \times \zeta^t \rightarrow I^\mu,$$

missä $H_P^t \subseteq P_1 \cup P_2 \cup ... \cup P_{t-1}$ eli kaikkien edellisten populaatioiden yhdiste.

Meta-aikatauluttaja toimii eräänlaisena evolutionaarisena muistina, eli se käyttää edellisten populaatioiden informaatiota hyväkseen ohjatessaan algoritmin toimintaa. Toisin kuin edellisten aikatauluttajien tapauksessa, parametrien joukko ζ voi sisältää yksinkertaisten todennäköisyysjakaumien sijaan monimutkaisia tietorakenteita, esimerkiksi tabulistoja vanhoista populaatioista.

Meta-aikatauluttajan lisääminen mahdollistaa uuden metaheuristiikkojen luokan, jossa iteraatioksi tulee

$$\begin{aligned} O_i^t &= fS_M(M, \mathcal{L}, fS_R(R, \mathcal{L}, P^t, \delta^t, \zeta^t), \delta^t, \zeta^t) \ \forall \ i \in \{1, ..., \lambda\} \\ P^{t+1} &= mS(\mathcal{L}, H_P^t, cS(U, \mathcal{L}, P^t, O^t, \delta^t, \zeta^t), \delta^t, \zeta^t) \end{aligned}$$

Yläindeksi t tarkoittaa sitä, että monet parametreista voivat olla ajasta riippuvaisia.

Krasnogor ja Smith huomauttavat [23], että tämän kaltaista memeettistä algoritmia ei esiinny kirjallisuudessa, joten kyseessä voi olla uusi, tehokas memeettisten algoritmien luokka.

Aikatauluttajista voidaan muodostaa nelinumeroinen binääriluku

$$D(A) = b_{mS} b_{cS} b_{fS_R} b_{fS_M},$$

missä kukin luku saa arvon 0 (ei kyseistä aikatauluttaja) tai 1 (aikatauluttaja on olemassa). Tällöin esimerkiksi algoritmi, joka tehostaa yksilöitä paikallisella haulla risteytyksen jälkeen, muttei muulloin, saa arvon $D = 2$ (0010). Memeettisiä algoritmeja on suunniteltu arvoille yhdestä kahdeksaan.

Kauppamatkustajan ongelmalle kehiteltyjä algoritmeja on olemassa vain tyyppejä kahdesta neljään, joista kustakin on annettu esimerkkejä seuraavassa luvussa. Geneettinen paikallishaku -algoritmi on tyyppiä kaksi, sillä se ei käytä lainkaan mutaatiota.

Ohjattua paikallishakua käyttävä memeettinen algoritmi, STSP-GA ja multimemeettinen algoritmi ovat tyyppiä kolme, eli niissä yksilöt optimoidaan sekä risteytyksen että mutaation jälkeen. Adaptiivinen memeettinen algoritmi on tyyppiä neljä, sillä siinä hakuoperaattoria käytetään paikallishaun lisäksi myös populaation diversiteetin säätämiseen, mutta vain ennen muuntelu- ja valintaoperaattorien käyttämistä.

5.5.2 Meemit

Paikalliset hakuoperaattorit $L \in \mathcal{L}$, eli "meemit", voidaan luokitella kolmeen ryhmään niiden toimintaperiaatteen mukaan [23]:

- *Staattinen meemi* (static meme) on paikallinen hakuoperaattori, joka ei muutu lainkaan algoritmin ajon aikana ($L^t = L^0 \; \forall \; t$). Valtaosassa kehitetyistä memeettisistä algoritmeista käytetään pelkästään staattisia meemejä.

- *Sopeutuva meemi* (adaptive meme) on hakuoperaattori, joka sopeutuu muutoksiin muuttamalla parametrejaan ζ^t:ssa t:n kasvaessa. Sopeutuva meemi voi esimerkiksi hyväksyä huonompia paikallishaun muodostamia ratkaisuja, kun populaation diversiteetti heikkenee. Jos meemin parametrit riippuvat käsiteltävästä yksilöstä, voi olla järkevää tallentaa ne suoraan yksilön genotyyppiin. Multimemeettisen algoritmin ollessa kyseessä riittää, että vain yksi meemeistä on sopeutuva, jotta meemien joukkoa $\mathcal{L}$ voidaan kutsua sopeutuvaksi.

- *Itsesopeutuva meemi* (self-adaptive meme) kykenee muokkaamaan itse paikallista hakua L:ää esimerkiksi geneettisen ohjelmoinnin avulla. Itsegeneroivat meemit ovat myös sopeutuvia, eli nekin voivat muuttaa ζ^t:tä. Kuten sopeutuvan meemin tapauksessa, vain yksi itsesopeutuva meemi riittää tekemään koko joukosta $\mathcal{L}$ itsesopeutuvan.

5.6 Memeettisten algoritmien suunnittelusta

Kirjassaan "The Design of Innovation: Lessons from and for Competent Genetic Algorithms" Goldberg [13] määrittelee tehokkaan geneettisen algoritmin algoritmiksi, joka "ratkaisee vaikeat ongelmat nopeasti, luotettavasti ja tarkasti". Määritelmä voidaan yleistää memeettisten algoritmien tapaukseen. Vaikka memeettinen algoritmi usein suoriutuukin perinteisiä evoluutioalgoritmeja paremmin, ei pelkkä paikallishaun lisääminen evoluutioprosessiin välttämättä tuota tehokkaampaa algoritmia [23]. Memeettistä algoritmia suunniteltaessa tuleekin ottaa huomioon useita seikkoja, joista seuraavassa.

5.6.1 Alkupopulaation valinta

Usein alkupopulaation yksilöt valitaan täysin satunnaisesti eri puolilta etsintäavaruutta [32]. Tällöin algoritmilla on suurempi todennäköisyys osua hyvän optimin vetovoima-alueelle. Toisaalta on kuitenkin järkevää täyttää populaatio satunnaista paremmilla ratkaisuilla etsinnän nopeuttamiseksi. Alkupopulaatio voidaan esimerkiksi täyttää lähimmän naapurin menetelmällä muodostetuilla reiteillä. Toinen vaihtoehto on etsiä yksi reitti ja muodostaa populaatio siitä mutatoiduilla yksilöillä. Mutaation on oltava riittävän voimakasta, jottei populaation diversiteetti jää liian köyhäksi.

5.6.2 Evoluutio-operaattorit

Lamarckistisissa memeettisissä algoritmeissa yksilöt sijaitsevat usein valmiiksi paikallisissa optimipisteissä ennen muunteluoperaattorien käyttämistä, joten perinteiset muunteluoperaattorit keskimäärin heikentävät niiden kelpoisuutta. Tämänkaltaisissa algoritmeissa on syytä käyttää hakuoperaattoria muuntelun jälkeen ja ennen valintaa, sillä muussa tapauksessa muunnetut yksilöt todennäköisesti jäävät valitsematta ja muuntelusta ei saada tavoiteltua hyötyä.

Memeettisissä algoritmeissa risteytysoperaattoreille voidaan antaa tehtäviä, joita ei ole perinteisissä evoluutioalgoritmeissa [30]. Normaalisti risteytysoperaattorin tarkoituksena on luoda entistä kelpoisempi yksilö, mutta koska memeettisen algoritmin paikallishaku hoitaa tämän tehtävän, risteytys voidaan valjastaa muihin tehtäviin, kuten uusien vetovoima-alueiden etsintään ja diversiteetin säilyttämiseen (vrt. DPX-risteytysoperaattori).

Mutaatiota voidaan käyttää huomattavasti enemmän, esimerkiksi alkuperäisessä STSP-GA -algoritmissa sitä sovelletaan joka viidenteen populaation yksilöistä ja algoritmin uudistetussa versiossa jopa puolet yksilöistä valitaan mutatoitavaksi.

5.6.3 Hakuoperaattorit

Paikallinen haku on vastuussa memeettisen algoritmin konvergenssista kohti optimipistettä. Samaten valtaosa algoritmin suoritusajasta menee paikalliseen hakuun, joten sen valinnalla on ratkaiseva merkitys algoritmin tehokkuuteen.

Paikallishakua valittaessa on päätettävä, onko järkevää tehostaa kaikkia yksilöitä, mikä hidastaa algoritmia ja heikentää diversiteettiä, vai pelkästään joitakin yksilöitä, jolloin konvergenssi hidastuu. Haun laajuudella ja syvyydellä on suuri vaikutus algoritmin toimintaan; algoritmin tehokkuus heikkenee nopeasti haun laajetessa, ja toisaalta liian suppea haku voi olla tehoton.

Hakuoperaattorin tehokkuus riippuu myös ratkaistavan ongelman ominaisuuksista. Lisäksi jokainen hakuoperaattori toimii erilaisessa kelpoisuusmaastossa: jokin operaattori saattaa jumiutua paikalliseen optimiin, mitä toisentyyppinen hakuoperaattori ei havaitse. Siksi algoritmeissa päädytään usein ratkaisuun, jossa käytetään samanaikaisesti useita hakualgoritmeja. Jottei algoritmi kuluttaisi turhaan aikaa heikompiin operaattoreihin, hakuja voidaan ohjata jonkinlaisella karkeasäätöaikatauluttajalla.

Erityisen tärkeää on varmistaa, ettei hakuoperaattori tee liian samanlaisia siirtymiä kuin risteytys- tai mutaatio-operaattori [16]. Tällä on todistettavasti vaikutusta algoritmin pahimman tapauksen tehokkuuteen. Sen voi päätellä myös intuitiivisesti: Jos esimerkiksi mutaatio-operaattori kääntää alireitin päinvastaiseksi ja paikallishakuna käytetään 2-Opt -operaattoria, on todennäköistä, että paikallishaku palauttaa yksilön mutaatiota edeltävään tilaan, jolloin molempien operaattorien työ menee hukkaan.

Algoritmin toimintaa voidaan tehostaa hyödyntämällä tietoa populaatiosta tai algoritmin toimintahistoriasta – toisin sanoen käyttämällä karkeasäätö- ja meta-aikatauluttajia. Diversiteetin ylläpitämiseksi voidaan algoritmia esimerkiksi estää siirtymästä tiettyihin pisteisiin pitämällä yllä tabulistaa jo käydyistä paikoista. Toinen menetelmä on käyttää simuloidun jäähdytyksen tapaan lämpötilaa, joka kasvaa populaation diversiteetin heiketessä, jolloin paikallishaku alkaa hyväksyä heikompia ratkaisuja. Tällainen algoritmi on esitelty seuraavassa luvussa.

6 Esimerkkejä memeettisistä algoritmeista

Seuraavassa tarkastellaan eri tyyppisiä kauppamatkustajan ongelmalle suunniteltuja memeettisiä algoritmeja. Geneettisessä paikallishaussa mutaatio on korvattu paikallishaulla. Ohjattuun paikallishakuun perustuvassa memeettisessä algoritmissa yksilöitä tehostetaan ohjatulla paikallishaulla. STSP-GA -algoritmi hyödyntää tehokasta paikallishakumenetelmää. MsMA on multimemeettinen algoritmi. Viimeiseksi esiteltävä memeettinen algoritmi perustuu adaptiivisten meemien ideaan.

6.1 Geneettinen paikallishaku

```
procedure GLS
begin
    /* λ, μ ≥ 1 */
    AlustaPopulaatio(P);
    foreach s ∈ P do
    begin
        s := Paikallishaku(s);
    end;
    repeat
        P_j := 0; /* jälkeläisten populaatio */
        for i := 1 to λ do
        begin
            ValitseVanhemmat(P, s_a, s_b); /* parinvalinta */
            s_c := Risteytä(s_a, s_b); /* risteytys */
            s_c := Paikallishaku(s_c); /* paikallishaku mutaatio-operaattorin sijalla */
            LaskeKelpoisuus(s_c);
            Lisää(P_j, s_c);
        end;
        P := ValitseJälkeläiset(P ⋃ P_j); /* selviytymisvalinta */
    until lopetusehto saavutettu;
end;
```

Kuva 6.1: Geneettinen paikallishaku

Geneettinen paikallishaku (Genetic Local Search, GLS) on eräs varhainen memeet-

tinen algoritmi [23]. Se poikkeaa tyypillisestä geneettisestä algoritmista siinä, ettei se hyödynnä lainkaan mutaatiota, vaan sen on korvannut operaattori `Paikallishaku()`. Algoritmi soveltaa $(\mu + \lambda)$-valintastrategiaa. Se on esitelty kuvassa 6.1.

6.2 Ohjattuun paikallishakuun perustuva memeettinen algoritmi

```
procedure GLS-MA
begin
    AlustaPopulaatio(P);
    foreach s ∈ P do
    begin
        s := Paikallishaku(s);
    end;
    repeat
        for i := 1 to risteytysten lkm do
        begin
            P_v := ValitseVanhemmat(P);
            s := Risteytä(P_v);
            s := Paikallishaku(s);
            LaskeKelpoisuus(s);
            Lisää(P, s);
        end;
        for i := 1 to mutaatioiden lkm do
        begin
            s := ValitseYksilö(P);
            s_m := Mutatoi(s);
            s_m := Paikallishaku(s_m);
            LaskeKelpoisuus(s_m);
            Lisää(P, s_m);
        end;
        P := ValitseJälkeläiset(P);
        if PopulaatioKonvergoitunut := true then P := ResetoiPopulaatio(P);
    until lopetusehto saavutettu;
end;
```

Kuva 6.2: Ohjattuun paikallishakuun perustuva memeettinen algoritmi.

Ohjattuun paikallishakuun perustuva memeettinen algoritmi on Holsteinin ja Moscaton [18] suunnittelema suhteellisen yksinkertainen ja tehokas metaheuristiikka. Holstein

ja Moscato päättivät tutkia, kuinka ohjattu paikallishaku voitaisiin parhaiten yhdistää memeettiseen algoritmiin. Kirjoittajien perimmäinen motivaatio oli lähinnä pedagoginen, eikä menetelmää suunniteltu erityisen tehokkaaksi. Tästä huolimatta se osoittautui varsin lupaavaksi menetelmäksi. Algoritmi on esitelty kuvassa 6.2.

Aliohjelma `AlustaPopulaatio()` luo ensimmäisen yksilön muodostamalla kaupungeista satunnaisen permutaation, jonka jälkeen sitä tehostetaan paikallishaulla. Seuraavaksi ensimmäisestä yksilöstä tehdään kopio. Uudesta yksilöstä poistetaan neljä kaarta tasavälein siten, että jäljelle jäävät alireitit ovat suunnilleen yhtä pitkiä. Ehjä yksilö muodostetaan yhdistämällä alireitit seuraavasti: jos alkuperäiset kaaret olivat (c_i, c_{i+1}), (c_j, c_{j+1}), (c_k, c_{k+1}) ja (c_m, c_{m+1}), niin uusiksi kaariksi tulevat (c_i, c_j), (c_{i+1}, c_k), (c_{j+1}, c_m) ja (c_{k+1}, c_{m+1}). Samoin kuin ensimmäisen yksilön tapauksessa, uutta yksilöä tehostetaan vielä paikallishaulla. Muut yksilöt muodostetaan samalla tavoin.

Ohjattu paikallishaku sisältää säätelyparametrin λ, joten myös jokaiselle populaation yksilölle on sellainen määriteltävä. Algoritmin suunnittelijat päätyivät arvoon $\lambda = 0,3 \times L_{alkureitti}/N_c$, missä $L_{alkureitti}$ on yksilöön liitetyn reitin pituus ensimmäisen optimointivaiheen jälkeen ennen evoluutioprosessia. N_c on kaupunkien lukumäärä.

Paikallishakuoperaattori `Paikallishaku()` soveltaa 2-Opt-menetelmää. Koska normaali 2-Opt-menetelmä tuottaa vain optimaalisia ratkaisuja, algoritmia on muutettu siten, että hyväksymiskriteerinä käytetään reitin pituuden sijasta erityistä ohjausfunktiota.

Parinvalinta on seuraavanlainen: ensimmäiseksi vanhemmaksi valitaan aina populaation paras yksilö. Toiseksi vanhemmaksi valitaan satunnainen yksilö jäljelle jäävien joukosta. Risteytysoperaattorina toimii DPX:n kaltainen operaattori. Algoritmin kehittäjät päätyivät risteytysten lukumäärässä arvoon 10.

Populaation diversiteetin säilyttämiseksi `Lisää()`-komento tarkistaa, onko populaatiossa jo ennestään sen kaltaista yksilöä. Mikäli sellainen löytyy, uutta yksilöä mutatoidaan perättäisillä 2-Opt-siirtymillä samaan tapaan kuin alkupopulaatiota muodostettaessa toimittiin.

Mutaatiota käytetään yhteen jälkeläiseen. Mutaatio-operaattorina toimii ei-perättäinen neljän kaaren vaihto.

`ValitseJälkeläiset()`-aliohjelma valitsee populaation 20 parasta yksilöä niiden kelpoisuuden perusteella. Jos kolmen perättäisen sukupolven aikana paras yksilö ei muutu, paikallishaku ei ilmeisesti enää kykene tehostamaan yksilöitä, joten `Populaatio-Konvergoitunut`-muuttujalle annetaan arvo tosi.

Holstein ja Moscato käyttivät [18] testauksessa kolmea erilaista metaheuristiikkaa: tässä esiteltyä ohjattuun paikallishakuun perustuvaa memeettistä algoritmia (GLS-MA), pelkkää ohjattua paikallishakua (GLS), sekä memeettistä algoritmia, joka käyt-

tää samaa 2-Opt-menetelmää kuin ohjattu paikallishaku mutta ilman ohjausstrategiaa (MA). Näitä verrattiin yksinkertaiseen multistart- paikallishakuun (MSLS).

Testiaineistoksi Holstein ja Moscato valitsivat 23 TSPLIB-kirjaston [41] pientä tai keskisuurta tapausta, jotka ovat suuruudeltaan 48–783 kaupunkia. Jokaisen tapauksen optimaaliset ratkaisut tunnettiin. Multistart-hakualgoritmia toistettiin 300 kertaa laskien samalla kuinka monta kaarenvaihtoa algoritmi yritti suorittaa. Tämä lukumäärä toimi muiden menetelmien lopetuskriteerinä.

MSLS ei kyennyt löytämään yhtäkään optimia. MA suoriutui vain vähän paremmin, sillä se kykeni löytämän optimin kuudessa tapauksessa. Algoritmin heikkous on ymmärrettävää, sillä se ei sisällä hyvää etsintästrategiaa eikä ole robusti. GLS pärjäsi huomattavasti paremmin löytäessään optimiratkaisun 13 tapauksessa. GLS-MA osoittautui testatuista algoritmeista parhaimmaksi, sillä se löysi optimaalisen ratkaisun 17 tapauksessa. Toisaalta GLS kykeni löytämään paremman ratkaisun viiden suurimman testitapauksen kohdalla yhtä lukuunottamatta. Kumpikaan algoritmeista ei kuitenkaan kyennyt löytämään optimiratkaisua näille tapauksille.

Valittuja heuristiikkoja testattiin myös kahdeksaan fraktaalitapaukseen, joiden koot vaihtelevat 294:stä 768:aan. Tulokset olivat samansuuntaisia paitsi GLS:n tapauksessa, joka suoriutui näistä huomattavasti heikommin.

6.3 STSP-GA

Freislebenin ja Merzin kehittämä [12] STSP-GA ja siitä edelleen kehitetyt algoritmit kuuluvat parhaimpiin kauppamatkustajan ongelmalle suunniteltuihin metaheuristiikoihin [23].

Alkupopulaation yksilöt muodostetaan yksinkertaisella lähimmän naapurin menetelmällä. Lähtökaupunki valitaan kullekin yksilölle satunnaisesti. Esitysmuotona toimii luvussa 3.4 esitelty permutaatiolista.

STSP-GA:n hakuoperaattorina toimii Lin-Kernighan -haku, jonka algoritmin kehittäjät valitsivat sen tehokkuuden vuoksi. Alkupopulaation yksilöt tehostetaan paikallishaulla, jolloin ne siirtyvät paikallisiin minimeihin. Paikallishakua käytetään joka kerta muunteluoperaattorin jälkeen, jolloin populaation yksilöt pysyvät aina paikallisissa minimeissä. Siten paikallishaun tehtävä on puhtaasti hyödyntävä, kun muunteluoperaattorien tehtävänä on uusien optimipisteiden etsiminen.

Vanhemmiksi valitaan kaksi satunnaista populaation yksilöä. Kelpoisuuteen perustuvia valintamenetelmiä ei voida käyttää, koska kaikki populaation yksilöt ovat optimoituja ja hyviä yksilöitä suosittaessa jälkeläisistä saattaisi tulla liian toistensa kaltaisia.

```
procedure STSP-GA
begin
    /* alusta populaatio P lähimmän naapurin heuristiikalla */
    AlustaPopulaatio(P);
    /* yksilöiden tehostus */
    foreach s ∈ P do
    begin
        s := Paikallishaku(s);
    end;
    repeat
        for i := 1 to risteytysten lkm do
        begin
            ValitseVanhemmat(P, s_a, s_b);
            s_c := DPX(s_a, s_b);
            s_c := Paikallishaku(s_c);
            n := random(0, 1);
            if n > p_m then /* p_m mutaation todennäköisyys */
            begin
                Mutatoi(s_c);
                s_c := Paikallishaku(s_c);
            end;
            LaskeKelpoisuus(s_c);
            Korvaa(P, s_c); /* korvaa jokin populaation yksilö s_c:llä */
        end;
    until lopetusehto saavutettu;
end;
```

Kuva 6.3: STSP-GA

Risteytysoperaattorina käytetään DPX:ää. Operaattorin luonteesta johtuen se mahdollistaa ”hypyt” uusien optimien vetovoima-alueille ja toivon mukaan siirtää algoritmin kohti globaalia optimia.

Mutaatiota sovelletaan pieneen osaan risteytyksessä muodostettuja yksilöitä. Mutaatio-operaattorina käytetään ei-perättäistä neljän kaaren vaihtoa. Operaattorin aiheuttama hyppy on suhteellisen pieni, joten uusi yksilö ei siirry liian kauaksi nykyisestä ratkaisusta. Sen lisäksi ei-perättäisyys varmistaa sen, ettei paikallishaku heti kumoa muutosta.

Muunteluoperaattorien jälkeen kaikki populaation yksilöt sijaitsevat paikallisissa optimeissa. Tämä tuottaa vakavia ongelmia diversiteetin säilymisen kannalta, joten kirjoittajat ovat päätyneet seuraavaan ratkaisuun: Uutta yksilöä lisättäessä ensin et-

sitään populaation samankaltaisin yksilö (siis se yksilö, jonka etäisyys eli poikkeavien kaarien lukumäärä, on pienin). Mikäli ero on pienempi kuin jokin ennalta määrätty raja-arvo, yksilö korvataan jälkeläisellä. Paras yksilö korvataan vain, jos jälkeläinen on sitä parempi. Kyseessä on siten vakaan tilan valinta.

Merz ja Freisleben testasivat [12] algoritmia kuudella eri TSPLIB-kirjastosta löytyvällä tapauksella, jotka ovat kooltaan 51–1577 kaupunkia. Populaation kooksi he valitsivat pienimpien testitapausten kohdalla kymmenen ja suurempien tapausten kohdalla kaksikymmentä yksilöä. Risteytysten todennäköisyydeksi he valitsivat 0,5 (eli puolet yksilöistä risteytettiin), ja mutaation todennäköisyydeksi 0,2. Suurinta tapausta lukuunottamatta algoritmi kykeni löytämään optimiratkaisun suhteellisen nopeasti ja pienellä määrällä operaatioita.

6.3.1 Uudistettu STSP-GA

Merz ja Freisleben kehittivät edelleen algoritmiaan [31]. Algoritmin rungosta tuli edellisessä luvussa esitellyn ohjattuun paikallishakuun perustuvan memeettisen algoritmin kaltainen (kuva 6.4). Mutaation määrää he kasvattivat 50 %:iin.

Suurimmat muutokset kohdistuivat paikalliseen hakuun. Käyttämällä tabulistoja, lähimpien naapureiden listoja ja muita kehittyneempiä menetelmiä Lin-Kernighan -hakuun käytetty aika lyheni huomattavasti, esimerkiksi 783 kaupungin tapauksen (`rat783`) ratkaisuaika lyheni jopa yli 35-kertaisesti 14 880 sekunnista 424 sekuntiin.

```
procedure STSP-GAv2
begin
    /* alusta populaatio P lähimmän naapurin heuristiikalla */
    AlustaPopulaatio(P);
    foreach s ∈ P do
    begin
        s := Paikallishaku(s); /* yksilöiden tehostus */
        LaskeKelpoisuus(s);
    end;
    repeat
        for i := 1 to risteytysten lkm do
        begin
            /* valitse kaksi satunnaista vanhempaa s_a, s_b ∈ P */
            ValitseVanhemmat(P, s_a, s_b);
            s_c := DPX(s_a, s_b);
            s_c := Paikallishaku(s_c);
            LaskeKelpoisuus(s_c);
            Lisää(P, s_c);
        end;
        for i := 1 to mutaatioiden lkm do
        begin
            s := ValitseYksilö(P); /* valitse satunnainen yksilö s */
            s_m := Mutatoi(s);
            s_m := Paikallishaku(s_m);
            LaskeKelpoisuus(s_m);
            Lisää(P, s_m);
        end;
        P := ValitseJälkeläiset(P);
    until lopetusehto saavutettu;
end;
```

Kuva 6.4: STSP-GA:n uudistettu versio.

6.4 MsMA: multimemeettinen algoritmi

MsMA on Zoun, Zhoun, Chening ja Yaon kehittämä memeettinen algoritmi [49], jonka perusideana on käyttää useaa erilaista paikallishakua populaation diversiteetin säilyttämiseksi.

```
procedure MsMA
begin
    /* alusta populaatio P lähimmän naapurin heuristiikalla */
    alusta_populaatio(P);
    foreach s ∈ P do
        MLS(s, k); /* monipaikallishaku */
    repeat
        for i := 0 to risteytysten lkm do
        begin
            /* valitse kaksi satunnaista vanhempaa s_a, s_b ∈ P */
            ValitseVanhemmat(P, s_a, s_b);
            s_c := EAX(s_a, s_b);
            MLS(s_c, k);
            Mutatoi(s_c);
            s'_c := Lin-Kernighan(s_c);
            LaskeKelpoisuus(s'_c);
            /* valitse yksilöistä kelvollisempi */
            if f(s'_c) < f(s_c) then s_c := s'_c;
            s_a := s_c;
        end;
    until lopetusehto saavutettu;
end;
```

Kuva 6.5: MsMA-algoritmi.

Algoritmi on esitelty kuvassa 6.5. Alkupopulaatio muodostetaan satunnaisesti permutoiduista reiteistä, jonka jälkeen jokaiseen yksilöön sovelletaan monipaikallishakua MLS (multi-local search, kuva 6.6 sivulla 54). Haun jälkeen jokainen yksilö on siirtynyt paikalliseen minimiin. MLS-operaattorissa käytettävät hakuoperaattorit valitaan satunnaisesti samalla todennäköisyydellä.

Risteytysoperaattorina käytetään EAX-operaattorin muunnelmaa M-EAX. Se poikkeaa tavallisesta EAX-operaattorista siinä, että kelvolliset AB-syklit hyväksytään E-joukkoon suurella todennäköisyydellä. Epäkelvot AB-syklit puolestaan hyväksytään hyvin pienellä todennäköisyydellä. Lisäksi AB-syklejä, joissa on vain yksi kaksinkertainen kaari, ei koskaan hyväksytä. Ahneen reitinkorjausmenetelmän sijaan uusi yk-

```
procedure MLS(s, k)
begin
    r := random(0, 1);
    if 0 ≤ r < 1/k then paikallishaku_1(s);
    if 1/k < r < 2/k then paikallishaku_2(s);
        ...
    if (k − 1)/k < r ≤ k then paikallishaku_k(s);
end;
```

Kuva 6.6: Paikallishaun valinta-algoritmi.

silö rakennetaan valitsemalla alireitit satunnaisesti. Nämä kaksi muutosta lyhentävät reilusti operaattorin suorittamiseen kuluvaa aikaa ja tehostavat operaattorin sisäistä paikallishakua. Samalla varmistetaan se, että syntyvät jälkeläiset ovat parempia.

Ominaisuuksistaan johtuen M-EAX heikentää jonkin verran populaation diversiteettiä. MLS-operaattori on tässä kohtaa hyödyllinen, sillä luonteestaan johtuen se tuottaa erilaisia ratkaisuja. MLS-haun jälkeen yksilöä mutatoidaan (operaattorina eiperättäinen neljän kaaren vaihto). Yksilöä tehostetaan vielä Lin-Kernighan -haulla, jolloin saadaan vielä uusi yksilö. Lopuksi ensimmäinen vanhemmista korvataan paremmalla jälkeläisellä.

Algoritmi käyttää kolmea lopetusehtoa: se pysähtyy, jos globaali optimi löytyy, tai kaikki populaation yksilöt ovat samoja, tai jos kaikki muodostetut jälkeläiset kymmenessä perättäisessä sukupolvessa ovat huonompia kuin vanhempansa.

Zou ym. käyttivät [49] MLS-haussa kahta hakumenetelmää: Lin-Kernighan -hakua ja naapurit yhdistävää paikallishakua. He vertasivat uutta algoritmia memeettisiin algoritmeihin, jotka käyttivät kyseisiä menetelmiä erikseen. Testauksessa käytettiin kahdeksaa TSPLIB-kirjaston 318–1889 kaupungin tapausta.

MsMA osoittautui molempia tehokkaammaksi. Useimmissa tapauksissa se oli nopeampi ja jokaisessa tapauksessa optimit löytyivät paremmin. Lisäksi uutta risteytysoperaattoria vertailtiin EAX-operaattoriin käyttämällä samaa algoritmia. M-EAX:lla varustettu algoritmi osoittautui useimmissa tapauksissa jonkin verran tehokkaammaksi sekä nopeuden että tarkkuuden suhteen.

6.5 Memeettinen algoritmi itsesopeutuvalla paikallisella haulla

Krasnogorin ja Smithin vuonna 2000 [22] suunnittelema adaptiivinen memeettinen algoritmi (kuva 6.7) poikkeaa edellisistä siinä, että se hyödyntää sopeutuvaa paikallista hakua. Algoritmia testattiin kauppamatkustajan ongelmaan ja proteiinien laskostusongelmaan. Vaikka algoritmin päätarkoitus olikin lähinnä tutkia menetelmän hyötyjä, tulokset olivat lupaavia yksinkertaisista operaattoreista huolimatta.

```
procedure Adaptiivinen_MA
begin
    AlustaPopulaatio(P);
    repeat
        P := Paikallishaku(P, p_h);
        /* valitse risteytettävät yksilöt */
        P_v := ValitseVanhemmat(P);
        P_j := Risteytä(P_v);
        Mutatoi(P_j);
        P := ValitseJälkeläiset(P, P_j);
    until lopetusehto saavutettu;
end;
```

Kuva 6.7: Adaptiivinen memeettinen algoritmi.

```
procedure Paikallishaku(P, p_ls)
begin
    /* T lämpötila, f_max maksimi- ja f_min minimikelpoisuus */
    T := 1/|f_max − f_min|; /* Populaation lämpötila */
    foreach s ∈ P do
    begin
        /* p_ls paikallishaun todennäköisyys */
        if p_ls ≤ random(0, 1) and s ≠ s_paras then
            s := Siirry(s);
    end;
    return P;
end;
```

Kuva 6.8: Adaptiivinen paikallishaku.

Algoritmi koostuu kahdesta optimointiprosessista: perinteisestä geneettisestä algoritmista ja Monte Carlo -tyyppisestä paikallishaku- ja diversifikointiprosessista. Algo-

ritmin alkuvaiheessa paikallishaku pyrkii lähestymään alkupisteen läheisyydessä olevaa paikallista optimia. Myöhemmin, kun populaation diversiteetti heikkenee ja haun eteneminen hidastuu, sen rooli muuttuu. Paikallishaku alkaa hyväksyä enenevässä määrin nykyistä heikompia ratkaisuja, jolloin algoritmi toivon mukaan pääsee poistumaan paikallisesta optimista ja siirtymään paremman optimin vetovoima-alueelle. Se on esitelty kuvassa 6.8.

`ValitseJälkeläiset()` on $(\mu + \lambda)$- tai (μ, λ)-valintastrategia, jossa $+$-strategialla on suurin valintapaine ja ,-strategialla pienin. `ValitseVanhemmat()` on turnajaisvalinta. $+$-strategian tapauksessa annettua yksilöä saatetaan muuttaa useaan kertaan mutaatiolla tai paikallishaulla sen elinaikana, koska strategia antaa yksilön säilyä.

```
procedure Siirry(s)
begin
    s_o := s;
    f_o := LaskeKelpoisuus(s); /* tallennetaan yksilön alkuperäinen kelpoisuus */
    s := Muuta(s); /* muutetaan s:ää jollakin operaatiolla */
    f_n := LaskeKelpoisuus(s);
    if f_n < f_o then
        return s;
    else
    begin
        ΔE := f_n − f_o;
        r := e^(−k∗ΔE/T); /* säätelyparametri */
        if r < random(0, 1) then
            return s; /* hyväksy ratkaisu vaikka se olisikin edellistä huonompi */
        else
            return s_o; /* hylkää muutokset */
    end;
end;
```

Kuva 6.9: Siirtyminen paikallishaussa.

Paikallishaun siirtymäoperaattori on esitelty tarkemmin kuvassa 6.9. `Muuta()` voi olla mikä tahansa paikallishakusiirtymä (esimerkiksi kahden kaupungin vaihto tai lisäys).

Hyväksyessään tai hylätessään paikallishaku käyttää hyödykseen simuloidun jäähdytyksen yhteydessä sivulla 33 esiteltyä Metropolis-kriteeriä. Se riippuu populaation lämpötilasta T, joka määritellään populaation parhaimman ja heikoimman yksilön kelpoisuuden perusteella (kuva 6.8). Kun populaation diversiteetti heikkenee, lämpötila alkaa nousta ja paikallishaku alkaa hyväksyä enenevässä määrin heikompia ratkaisuja,

eli paikallishaun tehtävä vaihtuu löydettyjen optimipisteiden hyödyntämisestä uusien optimialueiden etsimiseksi. Vastaavasti kun diversiteetti on kasvanut riittävän suureksi, lämpötila on laskenut alhaiseksi eikä huonompia ratkaisuja enää hyväksytä. Siten populaation diversiteetti heilahtelee edestakaisin ja tällöin säästytään ennenaikaiselta konvergenssilta. Kelvollisinta yksilöä ei käsitellä paikallishakuoperaattorilla, jottei sen arvo heikkenisi.

Krasnogor ja Smith valitsivat [22] populaation kooksi 50 yksilöä. Risteytyksen todennäköisyydeksi he asettivat 80 % ja mutaation todennäköisyydeksi 5 %. Selviytymisvalinnassa he käyttivät sekä (μ, λ)-, että $(\mu + \lambda)$-strategiaa. Boltzmannin vakion k arvoksi he antoivat ensimmäisessä tapauksessa arvon 0,01 ja jälkimmäisessä tapauksessa 0,001. Jokainen yksilö kävi läpi paikallishaku-/diversifiointivaiheen lukuunottamatta populaation parasta yksilöä. Pariutumisvalinnassa kirjoittajat käyttivät kahden yksilön turnajaisvalintaa.

Reitit koodattiin siten, että jos kromosomin paikassa i on arvo j, niin kaupunkien i ja j välillä on kaari. Risteytysoperaattorina käytettiin menetelmää, jossa valitaan ensin satunnainen kaupunki ja jälkeläinen rakennetaan lisäämällä lyhin vapaa kaari jommastakummasta vanhemmasta. Jos yhtään vapaata kaarta ei löydy, jälkeläiseen lisätään satunnainen kaari. Mutaationa käytettiin kahden kaaren vaihtoa. Alkupopulaatio alustettiin satunnaisilla reiteillä.

Säätämällä algoritmin paikallishakua Krasnogor ja Smith muodostivat algoritmista useita erilaisia variantteja. Mikäli paikallishakua ei käytetty lainkaan, oli kyseessä normaali geneettinen algoritmi. Jos algoritmi hyödynsi hakuoperaattoria, joka hyväksyi vain yksilöiden parannukset, oli kyseessä memeettinen mäennousualgoritmi. Memeettinen Boltzmann-mäennousualgoritmi puolestaan toimi kuten itsesopeutuva memeettinen algoritmi, mutta lämpötila pidettiin vakiona. Lineaarinen memeettinen jäähdytysalgoritmi käytti Boltzmann-kriteeriä siirtymien hyväksymiseen tai hylkäämiseen.

Testattavaksi ongelmaksi Krasnogor ja Smith valitsivat TSPLIB-kirjastosta reitin `eil76`, joka ei ole erityisen haastava ja joka sisältää vain 76 kaupunkia. Jokaisella variantilla suoritettiin 30 testiajoa (50, 50)- ja (50 + 50)-valintastrategialla. Algoritmien annettiin edetä 2000 sukupolven verran lukuun ottamatta geneettistä algoritmia, joka käytti 6000 sukupolvea. Ajot suoritettiin MAFRA-kehitysalustalla [21].

Toisessa kokeessa kirjoittajat vaihtoivat esitysmuodon permutaatioksi ja risteytysoperaattorin PMX:ään, muuten algoritmi pidettiin muuttumattomana. Saadut tulokset olivat samansuuntaisia.

Tuloksista ilmeni, että adaptiivinen memeettinen algoritmi on muita variantteja tehokkaampi. Se löysi lyhyempiä reittejä muita menetelmiä paremmin ja kykeni säilyttämään populaation diversiteetin muita korkeampana lukuun ottamatta Boltzmann-

mäennousualgoritmia (50, 50)-strategian tapauksessa.

7 Tapaustutkimus

Tapaustutkimuksessa vertaillaan STSP-GA:n kaltaista memeettistä algoritmia kolmeen perinteiseen heuristiseen menetelmään. STSP-GA valittiin testattavaksi menetelmäksi, koska se kuuluu esitellyistä menetelmistä tehokkaimpiin ja se on suhteellisen yksinkertainen toteuttaa. Vertailussa käytettyjä menetelmiä voidaan pitää jossain määrin memeettisten algoritmien komponentteina, joten evoluutioprosessin ja paikallisten hakumenetelmien yhteisvaikutus tulee selkeästi esille. Paikallishaussa käytettiin kahta erilaista menetelmää, jotta haun vaikutusta memeettisen algoritmin etenemiseen voitaisiin havainnollistaa.

7.1 Testausjärjestelyt

Testauslaitteistona käytettiin 2.13 GHz Intel Core2 Duo 6400 -suorittimella ja 1024 Mt muistilla varustettua tietokonetta. Käyttöjärjestelmänä oli Ubuntu Linux 6.10 (”Edgy Eft”). Testaussovellus toteutettiin Java-ohjelmointikielen versiolla 1.5.0.

7.2 Käytetyt algoritmit

Testattavan memeettisen algoritmin runkona käytettiin STSP-GA -algoritmia. Populaation kooksi valittiin 20 yksilöä, joka on sopiva koko valitunkokoisille tapauksille. Pienempi lukumäärä johtaa helposti algoritmin ennenaikaiseen konvergenssiin, kun taas suurempi määrä hidastaa algoritmin etenemistä liiaksi. Risteytyksessä käytettiin DPX-operaattoria ja mutaationa ei-perättäistä neljän kaaren vaihtoa. Alkuperäisen algoritmin tapaan vanhemmat valittiin populaatiosta satunnaisesti. Evoluutioalgoritmin parametrien arvoja ei muutettu, sillä niiden muuttamisesta ei havaittu olevan juurikaan hyötyä. Näin ollen risteytyksellä tuotettiin kymmenen ja mutaatiolla neljä yksilöä kunkin sukupolven aikana.

Alkuperäisen algoritmin tapaan selviytymisvalintana käytettiin vakaan tilan valintaa. Jos uuden yksilön ja samankaltaisimman populaatiosta löytyvän yksilön välinen ero oli riittävän pieni ja jos uuden yksilön kelpoisuus oli parempi, vanhempi yksilö korvattiin uudella. Yksilöt saivat poiketa toisistaan korkeintaan kaksi prosenttia eli ongelmatapauksen koosta riipuen 1–9 kaarta. Tämän arvon valinnalla havaittiin olevan

varsin merkittävä vaikutus algoritmin etenemiseen. Liian suurilla arvoilla algoritmin konvergenssi kärsi, kun taas liian pienillä arvoilla algoritmi konvergoi liian nopeasti diversiteetin heiketessä.

Paikallishakuna toimi perinteinen Lin-Kernighan -haku Keld Helsgaunin esittelemin koodausta helpottavin parannuksin [17]. Testisovellus toteutettiin ”naiivilla” ohjelmoinnilla ilman koodin optimointia. Lin-Kernighan -haun edetessä kahta kaarenvaihtoa syvemmälle se tutki vain viisi lähintä naapuria. Tällä oli jonkin verran vaikutusta haun tarkkuuteen, mutta suurempien tapausten kohdalla huonoja suoritusaikoja saatiin lyhennettyä merkittävästi. Muita Linin ja Kernighanin esittelemiä heuristisia parannuksia ei toteutettu, millä oli suuri vaikutus algoritmin nopeuteen. Paikallishaun vaikutuksen havainnollistamiseksi memeettinen algoritmi ajettiin myös tavallisella 2-Opt -algoritmilla.

Memeettistä algoritmia vertailtiin seuraaviin perinteisiin menetelmiin:

- Multistart-paikallishaussa sekä Lin-Kernighan, että 2-Opt -haku ajettiin tuhat kertaa aloittaen joka kerta satunnaisesti muodostetusta permutaatiosta.

- Iteroidussa paikallishaussa populaation kooksi valittiin yksi. Joka sukupolvessa yksilöön sovellettiin ei-perättäistä neljän kaaren vaihtoa viisi kertaa, jotta se siirtyisi tarpeeksi kauas paikallisesta minimistä.

- Geneettinen algoritmi muodostettiin poistamalla käytöstä paikallishakuoperaattori. Toimintatavastaan johtuen DPX-operaattori ei sovellu perinteisen geneettisen algoritmin risteytysoperaattoriksi ja se vaihdettiin sopivampaan PMX-operaattoriin. Neljän kaaren vaihto aiheuttaa tässä tapauksessa liian suuria hyppyjä, joten mutaatio-operaattoriksi valittiin alireitin kääntö. Populaatio oli GA:n tapauksessa kooltaan 50 yksilöä, risteytyksen todennäköisyys 80 % ja mutaation todennäköisyys 10 %. Risteytettävien yksilöiden vanhemmat valittiin turnajaisvalinnalla. Selviytymisvalinnaksi valittiin $(\mu + \lambda)$.

Algoritmien nopeuttamiseksi alkupopulaatiot alustettiin lähimmän naapurin menetelmällä muodostetuista yksilöistä. Multistart-paikallishaun tapauksessa alustusta ei käytetty, sillä se heikensi algoritmin tehokkuutta.

7.3 Tutkitut testitapaukset

Testauksessa käytettiin kymmentä TSPLIB-kirjastosta [41] peräisin olevaa pientä ja keskisuurta testitapausta (kuva 7.1). Ne pyrittiin valitsemaan siten, että joukossa on

vaativuudeltaan erityyppisiä tapauksia. Testitapaukset on esitelty tarkemmin liitteessä A.

tapaus	koko	optimi
danzig42	42	699
eil51	51	426
eil76	76	538
pr76	76	108159
eil101	101	629
gr120	120	6942
kroA200	200	21282
gr202	202	40160
pr226	226	80369
lin318	318	42029

Kuva 7.1: Tutkitut tapaukset optimiratkaisuineen.

7.4 Tulosten analysointi

Jokainen ongelmatapaus ajettiin 20 kertaa, jotta satunnaisuus ei vaikuttaisi liiaksi tuloksiin. Lin-Kernighan -algoritmilla varustettu memeettinen algoritmi (MA+LK) pysäytettiin sadannen sukupolven kohdalla, iteroidun paikallishaun (ILS) ja 2-Opt -memeettisen algoritmin (MA+2-Opt) annettiin edetä tuhat sukupolvea. Geneettisen algoritmin (GA) annettiin edetä 5000 sukupolven ajan. Algoritmien suoritus pysäytettiin, jos globaali optimipiste saavutettiin. Multistart-menetelmässä (MSLS+2-Opt, MSLS+LK) paikallishaut ajettiin tuhat kertaa peräkkäin.

Testitulokset on esitelty seuraavassa. Taulukoiden sarakkeet on merkitty seuraavasti: n on löydettyjen globaalien optimien lukumäärä, sp_{ka} keskimääräinen sukupolvien määrä (iteroidun paikallishaun tapauksessa iterointien lukumäärä it_{ka}), pr_{ka} algoritmin keskimäärin saavuttama paras ratkaisu verrattuna tunnettuun globaalin optimiratkaisuun. t_{ka} keskimääräinen ratkaisuun kulunut aika. Tähti ($\star$) tarkoittaa sitä, että algoritmi löysi kyseisessä tapauksessa eniten optimiratkaisuja ollen samalla nopein.

7.4.1 Multistart-algoritmi

Multistart-algoritmin tulokset on esitelty kuvassa 7.2. Taulukosta on helposti nähtävissä, että 2-Opt -haku on huomattavasti Lin-Kernighan -hakua nopeampi. Sen sijaan Lin-Kernighan -haku kykenee löytämään huomattavasti parempia ratkaisuja. danzig42:n tapauksessa se löytää optimiratkaisun lähes joka toisella yrityksellä. Tuloksista on ha-

	MSLS+2-Opt			MSLS+LK		
tapaus	n	pr_{ka}	t_{ka}	n	pr_{ka}	t_{ka}
danzig42	14	5,42 %	0,00074 s	442	0,81 %	0,0058 s
eil51	0	6,10 %	0,0015 s	26	2,14 %	0,0062 s
eil76	0	7,80 %	0,0036 s	4	2,98 %	0,014 s
pr76	0	4,97 %	0,0040 s	67	1,79 %	0,025 s
eil101	0	8,21 %	0,0097 s	1	3,31 %	0,029 s
gr120	0	7,16 %	0,018 s	1	2,25 %	0,057 s
kroA200	0	7,96 %	0,096 s	0	2,40 %	0,22 s
gr202	0	6,80 %	0,10 s	0	2,80 %	0,39 s
pr226	0	4,75 %	0,16 s	0	1,81 %	0,57 s
lin318	0	7,37 %	0,40 s	0	2,89 %	0,94 s

Kuva 7.2: Multistart-paikallishaku 2-Opt ja Lin-Kernighan -hakujen tapauksessa.

vaittavissa myös, kuinka ongelmatapauksen rakenne vaikuttaa algoritmin etenemiseen. Esimerkiksi pr76:n tapauksessa molemmat algoritmit selviytyivät paremmin kuin pienemmän eil51:n tapauksessa. Kaiken kaikkiaan menetelmä on hyvin heikko verrattuna muihin tutkittuihin algoritmeihin, sillä kuten tuloksista ilmenee, globaalin ratkaisun vetovoima-alueelle osuminen on hyvin epätodennäköistä. Menetelmä vaatii myös paljon laskentaa, kun haku joudutaan joka kerta aloittamaan alusta.

7.4.2 Iteroitu paikallishaku

	ILS+LK				
tapaus	n	it_{ka}	pr_{ka}	t_{ka}	
danzig42	20	1,25	0,00 %	0,03 s	
eil51	20	57,50	0,00 %	0,69 s	
eil76	20	8,75	0,00 %	0,29 s	
pr76	20	4,30	0,00 %	0,22 s	
eil101	20	98,35	0,00 %	4,65 s	
gr120	20	38,15	0,00 %	4,10 s	⋆
kroA200	20	37,95	0,00 %	12,82 s	⋆
gr202	18	264,75	0,04 %	130,22 s	
pr226	20	91,90	0,00 %	58,99 s	
lin318	7	715,80	0,19 %	731,64 s	

Kuva 7.3: Iteroitu Lin-Kernighan paikallishaku.

Iteroitu Lin-Kernighan -haku osoittautui varsin tehokkaaksi menetelmäksi (kuva 7.3). Algoritmi löysi optimipisteen joka kerta lukuun ottamatta tapauksia gr202

ja lin318. Tapauksissa gr120 ja kroA200 se oli myös tutkituista algoritmeista nopein. Suurempien tapausten kohdalla algoritmin tehokkuus kuitenkin heikkeni selvästi.

7.4.3 Geneettinen algoritmi

	GA			
tapaus	n	sp_{ka}	pr_{ka}	t_{ka}
danzig42	6	3669,75	2,69 %	1,36 s
eil51	0	5000,00	2,48 %	1,97 s
eil76	0	5000,00	5,36 %	2,35 s
pr76	0	5000,00	4,53 %	2,57 s
eil101	0	5000,00	5,43 %	2,70 s
gr120	0	5000,00	5,56 %	2,96 s
kroA200	0	5000,00	5,54 %	4,48 s
gr202	0	5000,00	7,07 %	4,51 s
pr226	0	5000,00	5,39 %	4,89 s
lin318	0	5000,00	9,97 %	5,46 s

Kuva 7.4: Geneettinen algoritmi.

Lukuunottamatta tapausta danzig42 geneettinen algoritmi ei kyennyt löytämään yhtäkään globaalia optimipistettä (kuva 7.4). Algoritmin ongelmaksi koitui diversiteetin romahtaminen ja siitä seurannut konvergenssin heikentyminen. Tarkkuudeltaan se on useimmissa tapauksissa hieman 2-Opt -paikallishakua parempi. Algoritmista olisi luultavasti saatu tehokkaampi, jos sen toteutukseen olisi kiinnitetty enemmän huomiota, joten esitellyt tulokset saattavat antaa algoritmista todellista pessimistisemmän kuvan. Positiivisena puolena voidaan nähdä se, että testiongelman koko vaikutti geneettisen algoritmin nopeuteen huomattavasti vähemmän kuin paikallishakua käyttävissä menetelmissä ja se oli tutkituista algoritmeista selvästi helpoin toteuttaa.

7.4.4 Memeettinen algoritmi

Memeettinen algoritmi osoittautui tutkituista algoritmeista selkeästi tehokkaimmaksi (kuva 7.5). Evoluution tuoma hyöty tulee selkeästi esille hakuoperaattorin löytäessä nopeasti globaalin optimiratkaisun. Kuvasta on myös selvästi nähtävissä, kuinka suuri vaikutus paikallishaulla on algoritmin toimintaan. Pienimmissä ongelmatapauksissa yksinkertainen hakuoperaattori on nopea ja usein riittävä, jolloin algoritmin suoritusaika on lyhyt ja globaali optimiratkaisu löytyy nopeasti. Sen sijaan tapauksissa, joissa 2-Opt -haulla oli vaikeuksia löytää globaali optimiratkaisu, tehokkaampi Lin-Kernighan

	MA+2-Opt					MA+LK				
tapaus	n	sp_{ka}	pr_{ka}	t_{ka}		n	sp_{ka}	pr_{ka}	t_{ka}	
danzig42	20	3,55	0,00 %	0,02 s	⋆	20	1,00	0,00 %	0,04 s	
eil51	17	342,70	0,03 %	1,68 s		20	6,40	0,00 %	0,24 s	⋆
eil76	20	21,00	0,00 %	0,26 s	⋆	20	3,20	0,00 %	0,31 s	
pr76	20	6,85	0,00 %	0,10 s	⋆	20	1,10	0,00 %	0,17 s	
eil101	20	114,35	0,00 %	1,76 s		20	5,50	0,00 %	0,92 s	⋆
gr120	10	629,55	0,13 %	15,66 s		20	15,00	0,00 %	4,54 s	
kroA200	2	956,55	0,10 %	70,61 s		20	18,85	0,00 %	17,88 s	
gr202	11	524,45	0,17 %	41,86 s		19	46,80	0,00 %	63,91 s	
pr226	6	775,85	0,28 %	60,96 s		20	23,45	0,00 %	46,04 s	⋆
lin318	1	965,15	0,28 %	173,61 s		6	92,80	0,15 %	276,54 s	

Kuva 7.5: Memeettinen algoritmi 2-Opt ja Lin-Kernighan -haulla.

-haku osoittautui selvästi paremmaksi huolimatta sen suuresta laskentamäärästä. Paremmalla implementaatiolla tästä memeettisestä algoritmista voidaan saada merkittävästi tehokkaampi.

8 Yhteenveto

Tämä tutkielma käsittelee memeettisiä algoritmeja, uutta ja laajaa suosiota saavuttanutta evoluutioalgoritmien luokkaa. Tutkielman alussa tarkasteltiin optimoinnin peruskäsitteistöä ja kombinatorisia optimointiongelmia sekä esiteltiin tutkielman esimerkkiongelmaksi valittu kauppamatkustajan ongelma. Evoluutioalgoritmien lisäksi tutkielmassa esiteltiin kauppamatkustajan ongelmalle suunniteltuja paikallisia hakualgoritmeja ja niitä ohjaavia metaheuristiikkoja. Varsinaisessa memeettisiä algoritmeja käsittelevässä luvussa selvitettiin näiden menetelmien yhdistämisestä koituvia hyötyjä, erilaisia tapoja muodostaa memeettisiä algoritmeja, sekä seikkoja joita tulee ottaa huomioon niitä suunniteltaessa. Memeettisistä algoritmeista esiteltiin toimintaperiaatteeltaan erilaisia kauppamatkustajan ongelmalle suunniteltuja algoritmeja.

Tutkielman empiirisessä osuudessa tutkittiin erään memeettisen algoritmin (STSP-GA) tehokkuutta verrattuna kolmeen perinteiseen heuristiseen menetelmään kymmenen erilaisen kauppamatkustajan ongelman tapauksessa. Menetelmiä arvoitiin silmämääräisesti suoritukseen kuluneen ajan ja saavutettujen ratkaisujen laadun perusteella.

Tuloksista käy ilmi, että hyödyntämällä evoluutiota haku voidaan ohjata tehokkaammin globaaliin optimiin kuin satunnaisesti käynnistetyn tai iteratiivisesti etenevän paikallishaun tapauksessa. Toisaalta paikallishaun avulla memeettinen algoritmi saavutti optimaalisia ratkaisuja paljon perinteistä geneettistä algoritmia tehokkaammin.

Memeettistä algoritmia testattiin myös kahdella erilaisella paikallishaulla ja havaittiin, että haun valinnalla on suuri merkitys algoritmin nopeuteen ja tarkkuuteen. Kelvollisempia ratkaisuja tuottava mutta laskennallisesti vaativampi paikallishaku osoittautui paremmaksi hankalammissa testitapauksissa, koska se konvergoi globaaliin optimiratkaisuun nopeammin.

Tutkitulla memeettisellä algoritmilla on myös ilmeiset heikkoutensa. Tehokas paikallinen haku on laskennallisesti vaativa, mistä on haittaa populaatioita käyttävien algoritmien tapauksessa. Algoritmin implementaatiolla oli siihen osuutensa, mutta siitä huolimatta yksinkertaisia siirtymiä toteuttava algoritmi on laskennallisesti huomattavasti kevyempi. Tällaisen algoritmin tehokas toteutus on myös varsin työlästä, mutta suurin puute kuitenkin on sen riippuvuus käsiteltävästä ongelmasta, joten ”black box” -tyyppiseen optimointiin se ei sovellu. Vaikka kauppamatkustajan tapauksessa simuloidun jäähdytyksen tai geneettisen algoritmin kaltaiset menetelmät eivät pärjääkään

tehokkuudessa STSP-GA:n kaltaisille algoritmeille, niitä voidaan käyttää hyvin erilaisissa ongelmatapauksissa ja ne ovat myös yksinkertaisempia toteuttaa.

Kuten memeettisten algoritmien luokittelun yhteydessä kävi ilmi, memeettiset algoritmit eivät rajoitu paikallishaulla tehostettuihin geneettisiin algoritmeihin. Ne käsittävät laajan ryhmän erilaisia menetelmiä, joille on yhteistä paikallishaun hyödyntäminen evoluutioprosessin yhteydessä. Esimerkiksi esitelty adaptiivinen algoritmi voi olla kilpailukykyinen perinteisten menetelmien kanssa niiden vahvoilla alueilla.

Adaptiivinen memeettinen algoritmi, joka käyttää paikallishakuoperaattoria yksilöiden tehostuksen lisäksi uusien ratkaisualueiden etsintään, on perinteistä geneettistä algoritmia tehokkaampi, mutta paikallishaun luonteesta johtuen algoritmi konvergoi varsin hitaasti. Esimerkiksi multimemeettinen algoritmi, joka valitsee paikallishakuoperaattorin populaation diversiteetin perusteella, saattaisi olla tehokkaampi.

Yksikään esitellyistä algoritmeista ei ota huomioon ongelman rakennetta. Jos memeettinen algoritmi osaisi tutkia kelpoisuusmaastoa ennen paikallishaun valitsemista, se voisi valita paikallishaun maaston rakenteen perusteella. Tosin maaston tutkiminen voi olla liian vaativa tehtävä.

Adaptiiviset ja multimemeettiset algoritmit ovat yleistymässä, mutta esimerkiksi kauppamatkustajan ongelmalle ei ole kehitetty menetelmää, joka ohjaisi hakua sukupolvien tasolla. Myöskään itsegeneroivia meemejä ei ole käytetty tässä yhteydessä. Tosin voidaan kysyä, voidaanko kauppamatkustajan ongelman tapauksessa näiden avulla muodostaa tehokkaita menetelmiä. On kuitenkin selvää, että joidenkin ongelmatyyppien tapauksessa tällaiset ”älykkäät” menetelmät ovat tehokkaimpia.

No Free Lunch -teoreeman perusteella memeettiset algoritmit ovat kaikkien muiden menetelmien tapaan tehokkaita vain joissakin tapauksissa. Tämä tarkoittaa sitä, että yleispätevästi tehokasta memeettistä algoritmia ei voida kehittää, mutta toisaalta siitä seuraa se, että uusille memeettisille algoritmeille löytyy aina kysyntää.

Yhteenvetona voidaan sanoa, että memeettiset algoritmit ovat monipuolinen ja tehokas heuristisiten menetelmien luokka. Tietokoneiden tehokkuuden kasvun ja uusien, entistä hienostuneempien algoritmien ansiosta memeettisten algoritmien tulevaisuus näyttää lupaavalta.

9 Viitteet

[1] Peter J. Bentley, *An Introduction to Evolutionary Design by Computers*, kirjassa "Evolutionary Design by Computers", Peter J. Bentley (toim.), Morgan Kaufmann, San Francisco, 1999, s. 1–73.

[2] Christian Blum ja Andrea Roli, *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*, ACM Computing Surveys, Vol. 35, No. 3, s. 268–308 (September 2003).

[3] N. Christofides, S. Eilon, *An Algorithm for the Vehicle-Dispatching Problem*, Operations Research, Vol. 20, No. 3, (September 1969), s. 309–318.

[4] William Cook, "Traveling Salesman Problem", avoin Internet-sivusto, saatavilla WWW-muodossa <URL: `http://www.tsp.gatech.edu/`>, viitattu 23.3.2007.

[5] G. A. Croes, *A Method for Solving Traveling-Salesman Problems*, Operations Research, Vol. 6, No. 6, (Nov. – Dec. 1958), s. 791–812.

[6] Harlan Crowder ja Manfred W. Padberg, *Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality*, Management Science, Vol. 26, No. 5, (May, 1980), s. 495–509.

[7] G. Dantzig, R. Fulkerson, S. Johnson, *Solution of a Large-Scale Traveling-Salesman Problem*, Journal of the Operations Research Society of America, Vol. 2, No. 4, (November 1954), s. 393–410.

[8] Charles Darwin, "On the Origin of Species", John Murray, Lontoo, 1859, saatavilla tekstimuodossa <URL: `http://www.gutenberg.org/etext/1228`>, viitattu 23.3.2007.

[9] Richard Dawkins, "Geenin itsekkyys", Art House, Jyväskylä, 1993.

[10] A. E. Eiben, M. Schoenauer, *Evolutionary computing*, Information Processing Letters, Vol. 82, No. 1, (15 April 2002), s. 1–6.

[11] Bernd Freisleben ja Peter Merz, *New Genetic Local Search Operators for the Traveling Salesman Problem*, Lecture Notes In Computer Science, Vol. 1141 (1996), s. 890–899.

[12] Bernd Freisleben ja Peter Merz, *A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems*, Proceedings of IEEE International Conference on Evolutionary Computation, 1996, s. 616–621.

[13] David E. Goldberg, "The Design of Innovation: Lessons from and for Competent Genetic Algorithms", Kluwer Academic Publishers, Norwell, 2002.

[14] Martin Grötschel, *Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme*, Mathematical Systems in Economics, Vol. 36 (January 1977).

[15] Peter J. B. Hancock, "An empirical comparison of selection methods in evolutionary algorithms", Evolutionary Computing, AISB Workshop, 1994, s. 80–94.

[16] William E. Hart, Natalio Krasnogor, J. E. Smith, *Memetic Evolutionary Algorithms*, kirjassa Recent Advances in Memetic Algorithms, William E. Hart, Natalio Krasnogor, J. E. Smith (toim.) (Springer-Verlag, Piscataway, 2004), s. 3–27.

[17] Keld Helsgaun, *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic*, European journal of operational research, Vol. 126, No. 1, (2000), s. 106–130.

[18] Diana Holstein ja Pablo Moscato, *Memetic Algorithms using Guided Local Search: A Case Study*, kirjassa New Ideas in Optimization, McGraw-Hill Ltd., Maidenhead, 1999, s. 235–243.

[19] David S. Johnson ja Lyle A. McGeoch, *The Traveling Salesman Problem: A Case Study in Local Optimization*, kirjassa Local Search in Combinatorial Optimization, E. H. L. Aarts ja J. K. Lenstra (toim.), John-Wiley and Sons, Ltd., New York, 1997, s. 215–310.

[20] S. Kirkpatrick, C. D. Gelatt Jr. ja M. P. Vecchi, *Optimization by Simulated Annealing*, Science, Vol. 220, No. 4598, (13 May 1983), s. 671–681.

[21] Natalio Krasnogor ja Jim Smith, *MAFRA: A Java Memetic Algorithms Framework*, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Darrell Whitley et al. (toim.), Morgan Kaufmann, Las Vegas, 2000, s. 125–130.

[22] Natalio Krasnogor ja Jim Smith, *A Memetic Algorithm With Self-Adaptive Local Search: TSP as a case study*, kirjassa Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Darrell Whitley et al. (toim.), Morgan Kaufmann, Las Vegas, 2000, s. 987–994.

[23] Natalio Krasnogor ja James Smith, *A tutorial for competent memetic algorithms: model, taxonomy, and design issues*, IEEE Transactions on Evolutionary Computation, Vol. 9, No. 5 (Oct. 2005), s. 474–488.

[24] Natalio Krasnogor, "Studies on the Theory and Design Space of Memetic Algorithms", väitöskirja, University of the West of England, Bristol, 2002, saatavilla PDF-muodossa <URL: `http://www.cs.nott.ac.uk/ nxk/PAPERS/thesis.pdf`>, viitattu 23.3.2007.

[25] Natalio Krasnogor ja Steven Gustafson, "Toward Truly "Memetic" Memetic Algorithms: discussion and proofs of concept", 2002, saatavilla WWW-muodossa <URL: `http://citeseer.ist.psu.edu/krasnogor02toward.html`>, viitattu 23.3.2007.

[26] Patrick Krolak, Wayne Felts ja George Marble, *A man-machine approach toward solving the traveling salesman problem*, Communications of the ACM, Vol. 14, No. 5 (May 1971), s. 327–334.

[27] Mark William Shannon Land, "Evolutionary algorithms with local search for combinatorial optimization", väitöskirja, University of California, Computer Science & Engr. Dept., San Diego, 1998, saatavilla WWW-muodossa <URL: `http://citeseer.ist.psu.edu/land98evolutionary.html`>, viitattu 23.3.2007.

[28] S. Lin ja B. W. Kernighan, *An Effective Heuristic Algorithm for the Traveling-Salesman Problem*, Operations Research, Vol. 21, No. 2 (Mar. – Apr., 1973), s. 498–516.

[29] Helena R. Lourenço, Olivier C. Martin ja Thomas Stüzle, "Iterated Local Search", Economics Working Papers 513, Department of Economics and Business, Universitat Pompeu Fabra, saatavilla PDF-muodossa <URL: `http://www.econ.upf.edu/docs/papers/downloads/513.pdf`>, viitattu 23.3.2007.

[30] Peter Merz, "Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies", väitöskirja, University of Siegen, Department of Electrical Engineering and Computer Science, 2000, saatavilla PDF-muodossa <URL: `http://www.ub.uni-siegen.de/pub/diss/fb12/2001/merz/merz.pdf`>, viitattu 23.3.2007.

[31] Peter Merz ja Bernd Freisleben, *Genetic Local Search for the TSP: New Results*, kirjassa Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, IEEE Press, 1997, s. 159–164.

[32] Peter Merz ja Bernd Freisleben, *Fitness landscapes and memetic algorithm design*, kirjassa New Ideas in Optimization, McGraw-Hill Ltd., Maidenhead, 1999, s. 245–260.

[33] Peter Merz, *A Comparison of Memetic Recombination Operators for the Traveling Salesman Problem*, kirjassa Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann, San Francisco, 2002.

[34] Kaisa Miettinen, "Epälineaarinen optimointi", luentomoniste 10, Jyväskylän yliopisto, tietotekniikan laitos, Jyväskylä, 2003.

[35] Melanie Mitchell, "An Introduction to Genetic Algorithms", MIT Press, Cambridge, 1996.

[36] Pablo Moscato, *Memetic Algorithms: A Short Introduction*, kirjassa New Ideas in Optimization, McGraw-Hill Ltd., Maidenhead, 1999, s. 219–234.

[37] Pablo Moscato, "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms", 1989, saatavilla PostScript-muodossa <URL: `http://www.densis.fee.unicamp.br/~moscato/papers/bigone.ps`>, viitattu 23.3.2007.

[38] Pekka Orponen, "Laskennan teoria", luentomoniste, Jyväskylän yliopisto, matematiikan laitos, Jyväskylä, 1994.

[39] Manfred Padberg ja Giovanni Rinaldi, *A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems*, SIAM Review, Vol. 33, No. 1 (March 1991), s. 60–100.

[40] Gerhard Reinelt, "The Traveling Salesman", Springer-Verlag, Heidelberg, 1994.

[41] Gerhard Reinelt, "TSPLIB", Heidelbergin yliopisto, tietotekniikan laitos, avoin Internet-sivusto, saatavilla WWW-muodossa <URL: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>, viitattu 23.3.2007.

[42] A. Schrijver, "On the history of combinatorial optimization (till 1960)", saatavilla WWW-muodossa <URL: `http://www.cwi.nl/~lex/files/histco.ps`>, viitattu 23.3.2007.

[43] James Edward Smith, "Self Adaptation in Evolutionary Algorithms", väitöskirja, Faculty of Computer Studies and Mathematics, University of the West of England, Bristol, 1998, saatavilla PDF-muodossa <URL: http://www.cems.uwe.ac.uk/~jsmith/thesis.pdf>, viitattu 23.3.2007.

[44] Hisashi Tamaki, Hajime Kita, Nobuhiko Shimizu, Keiji Maekawa ja Yoshikazu Nishikawa, *A Comparison Study of Genetic Codings for the Traveling Salesman Problem*, kirjassa Proceedings of the First IEEE World Congress on Computational Intelligence. IEEE Computer Society Press, Orlando, 1994, s. 1–6.

[45] P. Turney, *Myths and Legends of the Baldwin Effect*, kirjassa Proceedings Workshop on Evolutionary Computation and Machine Learning at the 13th International Conference on Machine Learning (1996), s. 135–142.

[46] J. Watson, C. Ross, V. Eisele, J. Denton, J. Bins, C. Guerra, D. Whitley ja A. Howe, *The Traveling Salesrep Problem, Edge Assembly Crossover, and 2-opt*, kirjassa Proceedings of the Fifth Conference on Parallel Problem Solving from Nature, Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, Hans-Paul Schwefel (toim.), Springer, Amsterdam, 1998, s. 823–832.

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

[48] Chris Voudouris ja Edward Tsang, "Guided Local Search", Technical Report CSM-247, University of Essex, Department of Computer Science, 1995.

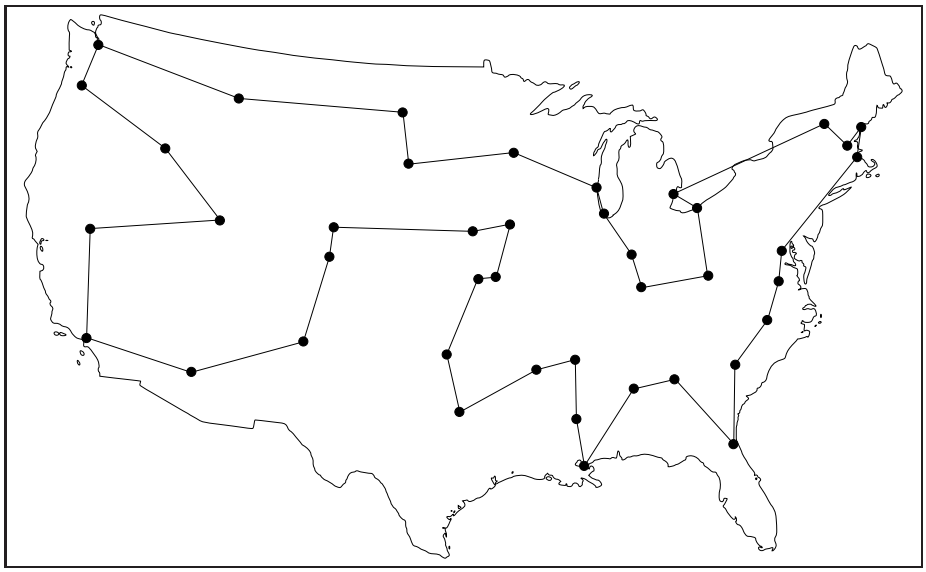
[49] Peng Zou, Zhi Zhou, Guoliang Chen ja Xin Yao, *A novel memetic algorithm with random multi-local-search: a case study of TSP*, Congress on Evolutionary Computation, Vol. 2 (19–23 June 2004), s. 2335–2340.

A Testitapaukset

Seuraavassa on listattuna testauksessa käytetyt ongelmatapaukset optimiratkaisuineen. Ne ovat saatavilla TSPLIB-kirjastosta [41].

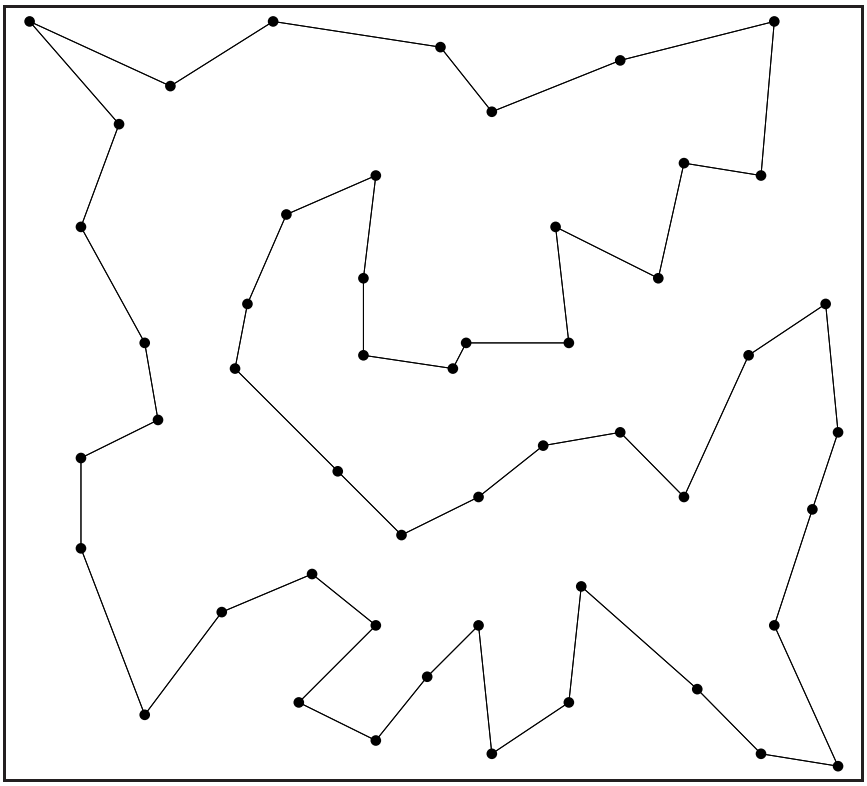
A.1 danzig42

`danzig42` on klassinen vuodelta 1954 peräisin oleva ongelmatapaus [7], joka koostuu 42 Yhdysvaltain mantereella sijaitsevasta kaupungista.

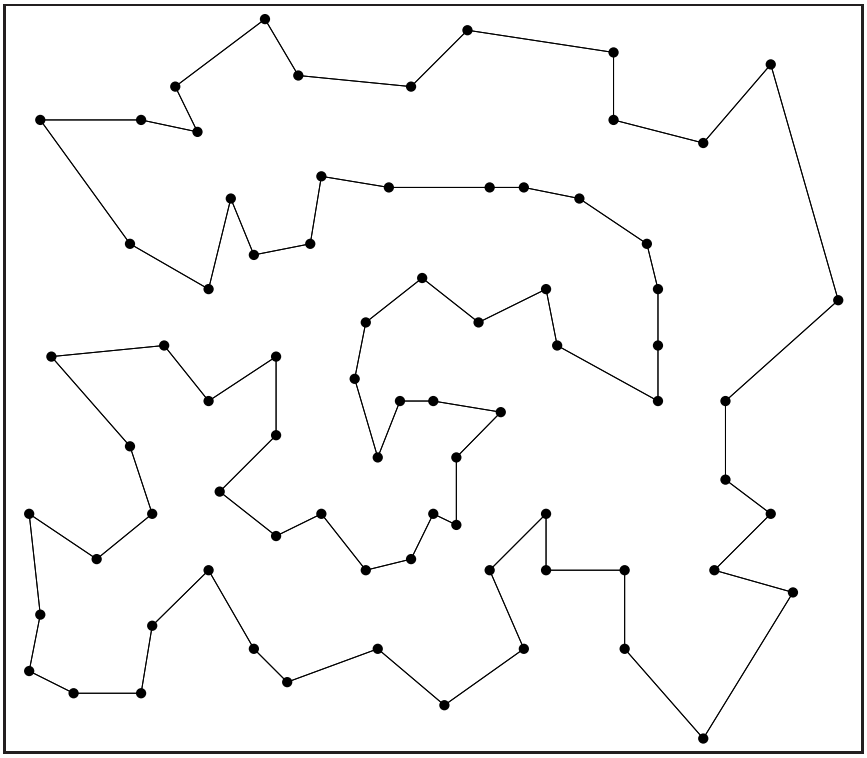


A.2 eil51, eil76 ja eil101

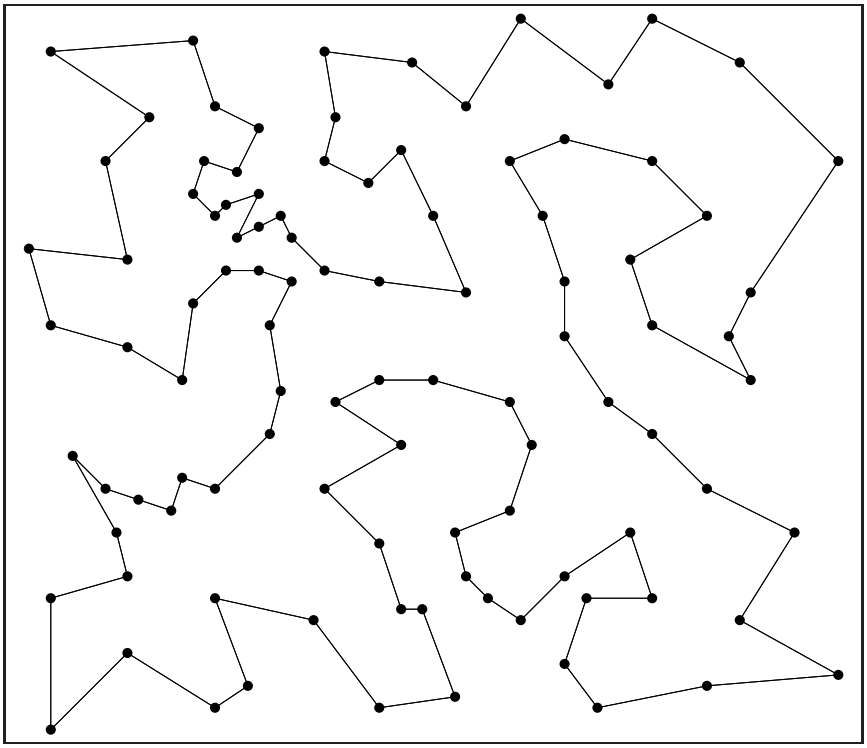
Tapaukset `eil51`, `eil76` ja `eil101` ovat vuodelta 1969 peräisin olevasta artikkelista [3], jossa tutkittiin ajoneuvojen reititysongelmaa.



Kuva A.1: Tapaus `eil51`.



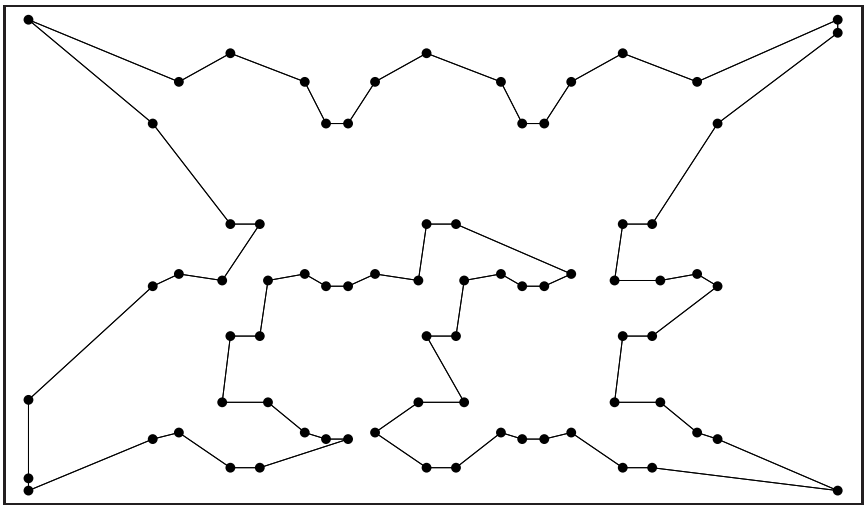
Kuva A.2: Tapaus `eil76`.



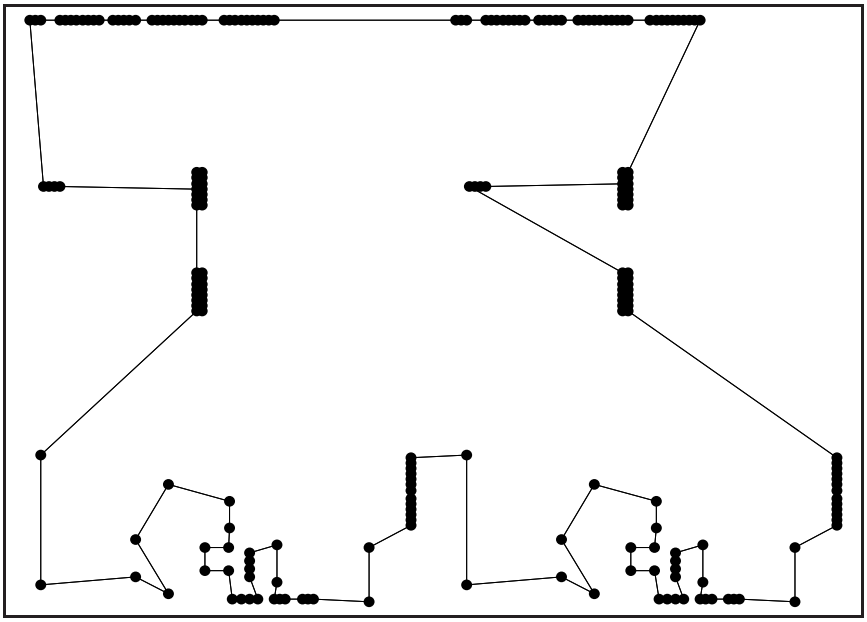
Kuva A.3: Tapaus `eil101`.

A.3 pr76 ja pr226

Tapaukset `pr76` ja `pr226` ovat peräisin Padbergin ja Rinaldin artikkelista, jossa tutkittiin Branch & Cut -algoritmia [39].



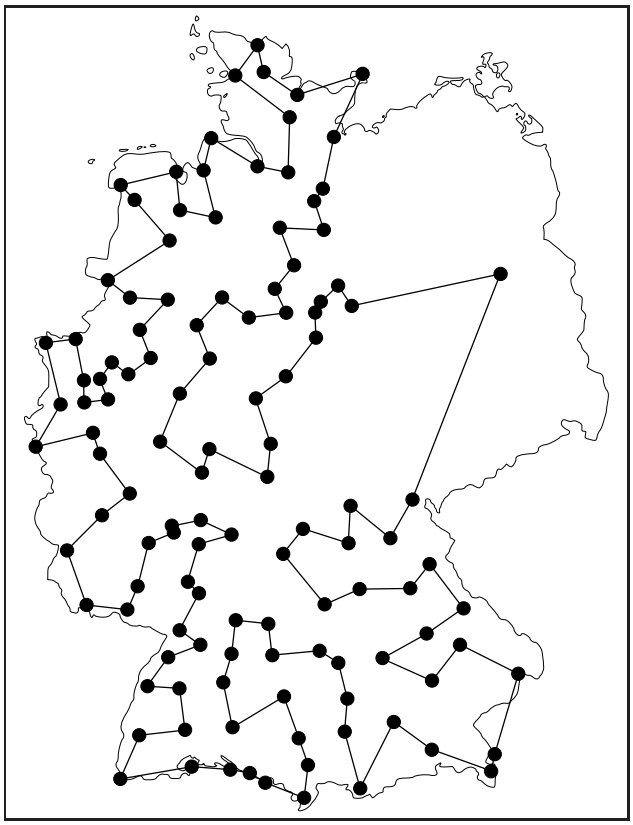
Kuva A.4: Tapaus `pr76`.



Kuva A.5: Tapaus `pr226`.

A.4 gr120

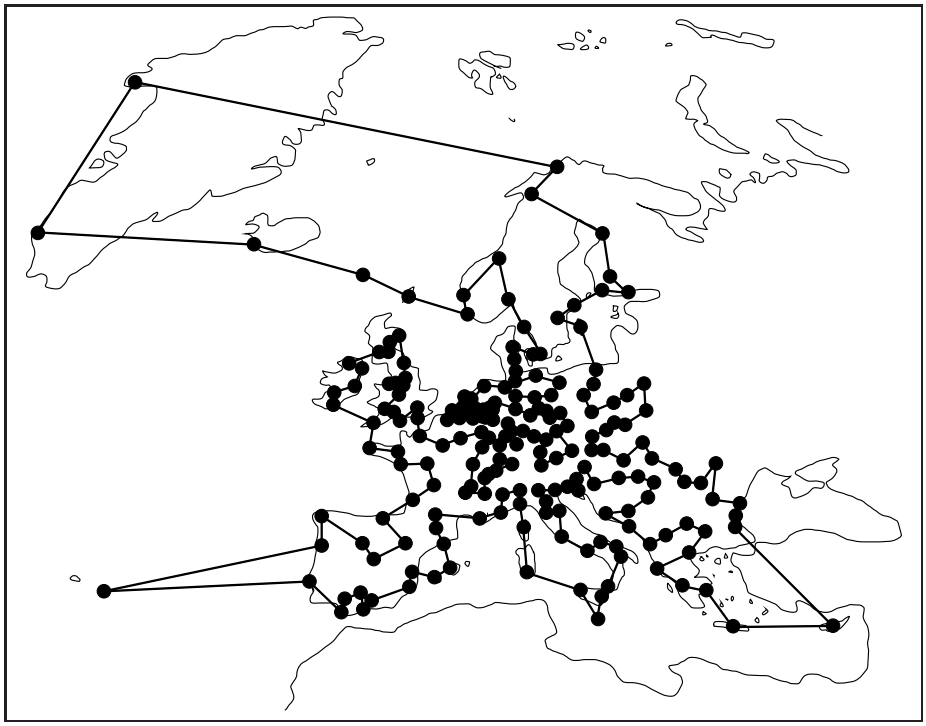
Tapaus `gr120` sisältää 120 Länsi-Saksan kaupunkia Berliini mukaanlukien. Se ratkaistiin ensi kertaa vuonna 1977 [14], mistä lähtien sitä on käytetty standarditestitapauksena.



A.5 gr202

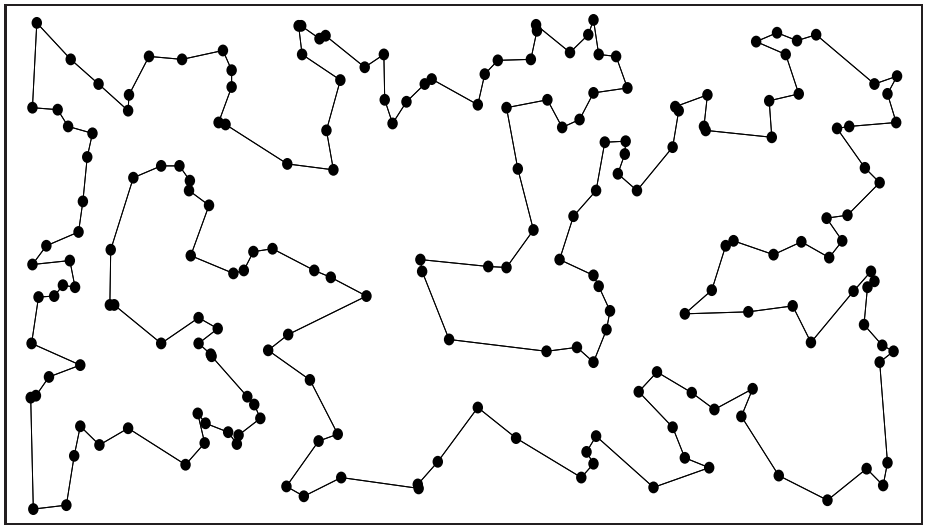
gr202 on osa koko maailman käsittävää 666 kaupungin tapausta, joka ratkaistiin vuonna 1987. Se sisältää 202 Euroopassa ja Grönlannissa sijaitsevaa kaupunkia. Kuva täydellisestä 666 kaupungin tapauksesta on saatavilla osoitteesta:

<URL: http://www.tsp.gatech.edu/history/pictorial/gr666.html>



A.6 kroA200

Tapaus `kroA200` on peräisin artikkelista [26]. Siinä tutkittiin puoliautomaattista heuristista menetelmää, jossa tietokone jakoi 100–200 kaupungin suuruisia tapauksia osiin. Osia yhdistämällä ihmisen oletettiin kykenevän löytämään optimaalisen ratkaisun käyttämällä hyväkseen luonnollista päättelykykyään.



A.7 lin318

Tapaus `lin318` esiteltiin ensi kertaa vuonna 1973 Lin-Kernighan -algoritmin yhteydessä [28]. Se on peräisin eräästä poraussovelluksesta, jossa porana käytettiin laserpulssia. Ongelma ratkaistiin vuonna 1980 [6]. Se oli tuolloin suurin ratkaistu kauppamatkustajan ongelman tapaus.