Aalto-yliopisto
Perustieteiden korkeakoulu
Teknillisen fysiikan ja matematiikan tutkinto-ohjelma

Antti Salmela

Minimilatenssiongelman ratkaisualgoritmeja

Kandidaatintyö
Espoo 26.2.2014

Työn valvoja: Prof. Harri Ehtamo

Työn ohjaaja: Prof. Harri Ehtamo

Työn saa tallentaa ja julkistaa Aalto-yliopiston avoimilla verkkosivuilla.
Muilta osin kaikki oikeudet pidätetään.

Sisällysluettelo

1. Johdanto

Tarkastellaan korjausmiestä, joka lähtee liikkeelle toimipisteestänsä ja jonka tarkoitus on käydä määrättyjen asiakkaiden luona täsmälleen yhden kerran ja sitten palata takaisin toimipisteeseensä. Kunkin asiakkaan luona korjausmies korjaa epäkunnossa olevan laitteen. Kun minimoidaan yhteenlaskettua aikaa, jonka kukin laite ehtii olla epäkunnossa, puhutaan optimointiongelmasta nimeltään *matkustavan korjausmiehen ongelma* (travelling repairman problem). Toisin sanoen tarkoitus on summata saapumisajat eri asiakkaiden luokse ja minimoida tätä saapumisaikojen summaa. Yleisempi termi, jota tässä työssä jatkossa käytetään tästä samasta saapumisaikojen summaa minimoivasta optimointitehtävästä, on *minimilatenssiongelma* (minimum latency problem).

Tämä kandidaatintyö on kriittinen kirjallisuustutkimus siitä, minkälaisilla algoritmeilla minimilatenssiongelmaa voidaan olemassaolevan kirjallisuuden perusteella muun muassa ratkoa. Työssä sivutaan alkuvaiheessa myös optimointitehtävää nimeltään *matkustavan kauppamiehen ongelma* (travelling salesman problem, TSP), koska minimilatenssiongelma voidaan määritellä matkustavan kauppamiehen ongelmaa apuna käyttäen. Matkustavan kauppamiehen ongelmassa on tarkoitus etsiä paikasta toiseen kulkevalle kauppamiehelle nopein reitti, joka käy kaikissa määrätyissä paikoissa ja palaa alkupisteeseensä, mutta kuten todettua minimilatenssiongelmassa summataan jokainen saapumisaika eri paikkoihin, ja minimoidaan tätä saapumisaikojen summaa. Minimilatenssiongelma on laskennallisesti huomattavasti raskaampi kuin matkustavan kauppamiehen ongelma, mutta sen olennainen meriitti on, että siinä voidaan huomioida jokaisen kaupungin asiakkaat erikseen, eivätkä minkään yksittäisen kaupungin asiakkaat joudu odottamaan kohtuuttoman pitkää aikaa. Lisäksi eri kaupunkien tärkeyttä voidaan korostaa painokertoimilla. Näitä asioita ei voida ottaa huomioon esimerkiksi matkustavan kauppamiehen ongelmassa sen perusmuodossaan.

Työn alussa minimilatenssiongelma esitellään. Ongelman esittelyn jälkeen käydään läpi muutamia eksakteja sekä ei-eksakteja ratkaisualgoritmeja minimilatenssiongelmaan. Koska minimilatenssiongelma on monissa tapauksissa NP-kova [1,5], matemaattisia malleja voidaan muodostaa rajoitetusti. Kirjallisuudesta joka tapauksessa löytyy joitakin eksakteja polynomiaikaisia ratkaisualgoritmeja sellaisiin suppeisiin erikoistilanteisiin, joissa tehtävä kuuluu P-joukkoon. Aluksi esittelen näistä tilanteista sellaisen jossa ollaan samansuuruisten etäisyyksien muodostamassa puugraafissa (tree network) [1,2], sitten tilanteen jossa ollaan puugraafissa, jonka halkaisija on 3, ja lopuksi joitakin tapauksia *painotetusta minimilatenssiongelmasta* (minimum weighted latency problem, MWLP). Ei-eksakteista algoritmeista sivutaan hieman *approksimaatioalgoritmeja* (approximation alghorithm) ja sitten käydään läpi kaksi heuristisista algoritmia, jotka ovat *tabualgoritmi* (tabu search) [5] ja *geneettinen algoritmi* (genetic algorithm) [4].

2. Minimilatenssiongelman esittely

2.1 Minimilatenssiongelman määrittely matkustavan kauppamiehen ongelman pohjalta

Olkoon G verkosto, joka koostuu solmukohtien joukosta V ja niiden välisten etäisyyksien joukosta E. Verkostossa on n kappaletta solmukohtia. Ne voivat hyvin olla vaikkapa asiakkaiden epäkunnossa olevia koneita, kuten johdannossa kuvattiin, mutta tästä eteenpäin niitä kuvataan yleisemmillä graafitermeillä. Merkitään solmukohtien x ja y väliseen nopeimpaan mahdolliseen reittiin kulunutta aikaa merkinnällä d(x,y). Solmukohtien joukko $\{v_1, v_2, ...v_n\} \in V$ kuvaa sitä järjestystä, jossa solmukohdat vieraillaan, siten että i:ntenä vierailtu solmukohta on v_i. Jos lähdetään liikkeelle solmukohdasta s ja palataan lopuksi samaan solmukohtaan, on viimeisin vierailtava solmukohta v_n oltava sama kuin solmukohta s. Tavoitteena on tässä välissä käydä mahdollisimman nopeasti kaikissa muissa solmukohdissa, joita on n-1 kappaletta. Tällöin saadaan lyhimmäksi mahdolliseksi kierrokseen kuluvaksi ajaksi:

$$d(s, v_1, v_2, \dots, v_n) = d(s, v_1) + \sum_{i=1}^{i=n-1} \quad d(v_i, v_{(i+1)}) \qquad (1)$$

Ylläolevaa minimointitehtävää kutsutaan nimellä matkustavan kauppamiehen ongelma (TSP). Se muistuttaa paljon minimilatenssiongelmaa, mutta osoittautuu olevan monilta osin yksinkertaisempi ongelma kuin minimilatenssiongelma. Minimilatenssiongelma on joka tapauksessa helppo määritellä TSP:n avulla. Nimittäin jos solmukohta s:stä kuljetaan johonkin kierroksella olevaan solmukohtaan j, saadaan tähän keskeneräiseen kierrokseen kulunut aika seuraavalla lausekkeella,

$$T(v_j) = d(s, v_1) + \sum_{i=1}^{i=j-1} \quad d(v_i, v_{(i+1)}), \qquad (2)$$

joka on sama lauseke kuin matkustavan kauppamiehen ongelmassa sillä erotuksella, että ollaan vain laskettu aika, joka kului j:nnenteen kaupunkiin saapumiseen, sen sijaan että oltaisiin kuljettu koko kierros. Ja koska minimilatenssiongelmassa minimoidaan saapumisaikojen summaa, se voidaan kirjoittaa seuraavaan muotoon, jossa summataan TSP:n keskeneräisten kierrosten saapumisajat yksi kerrallaan ensimmäisestä solmukohdasta n:nteen solmukohtaan. Tälloin saadaan saapumisaikojen summa kaikkiin solmukohtiin, ja kun sitä minimoidaan, saadaan minimilatenssiongelmalle lauseke:

$$\min(\sum_{j=1}^{j=n} \quad T(v_j)). \qquad (3)$$

Tämän muotoilun samankaltaisuuden takia minimilatenssiongelmaa voidaan pitää yhtenä matkustavan kauppamiehen ongelman varianttina. Se vain on laskennallisesti perusmuotoa huomattavasti raskaampi, koska TSP:ssä perusmuodossaan minimoidaan vain viimeistä termiä tästä summalausekkeesta.

2.2 Minimilatenssiongelman virtausmääritelmä

Minimilatenssiongelma voidaan määritellä myös suoraan ilman viittausta matkustavan kauppamiehen ongelmaan. Yksi tällainen määritelmä on niin sanottu *virtausmääritelmä* (flow formulation) [3], joka on matemaattisesti suorempi ja myös laskennallisesti tehokkaampi muotoilu minimilatenssiongelmalle. Virtausmääritelmä on seuraavanlainen: Määritellään G,E ja V kuten aiemminkin, ja lisäksi asetetaan joukko K=V, jossa kullekin solmukohdalle k ϵ K täytyy tehdä yksi vierailu, ja minimilatenssiongelmalle tyypilliseen tapaan tarkoitus on minimoida näihin solmukohtiin kohdistuvien saapumisaikojen summaa. Määritellään lisäksi:

$$x_{ij} = \begin{cases} 1, jos\ kauppamies\ kulkee\ pisteestä\ i\ pisteeseen\ j \\ 0, muussa\ tapauksessa \end{cases}$$

$$p_{ij} = \begin{cases} 1, jos\ j\ on\ järjestyksessä\ i{:}nnes\ vierailtu\ solmukohta \\ 0, muussa\ tapauksessa \end{cases}$$

c_{ij} = aika, joka kuluu kuljettaessa i:stä j:hin

g_{ij} = summaus kuinka monta kertaa i:stä on kuljettu kokonaisuudessaan j:hin. Tätä kutsutaan myös virtaukseksi, josta kyseessä oleva minimilatenssiongelman muotoilu saa nimensä.

Nyt minimoitava kohdefunktio on seuraava,

$\min \sum_{(i,j)\epsilon E} c_{ij} q_{ij}$, (4)

ja rajoitusehdot ovat:

$\sum_{(i,j)\epsilon E} x_{ij} = 1, \quad \forall j \epsilon V$ (5)

$\sum_{(i,j)\epsilon E} x_{ij} = 1, \quad \forall i \epsilon V$ (6)

$\sum_{(0,j)\epsilon E} g_{0j} = |K|$ (7)

$\sum_{(i,k)\epsilon E} g_{ik} - \sum_{(k,j)\epsilon E} g_{kj} = 1, \quad \forall k \epsilon K$ (8)

$\sum_{(i,j)\epsilon E} g_{ij} = \sum_{t=1}^{n} t$ (9)

$g_{ij} \leq |K| x_{ij,} \quad \forall (i,j) \epsilon E$ (10)

$g_{ij} \geq 0, \quad \forall (i,j) \epsilon E$ (11)

$x_{ij} \epsilon \{0,1\} \quad \forall (i,j) \epsilon E$ (12)

$\sum_{i=1}^{n} p_{ij} = 1, \quad \forall j \epsilon V$ (13)

$\sum_{j=1}^{n} p_{ij} = 1, \quad \forall i \epsilon V$ (14)

$p_{1o} = 1$ (15)

$$\sum_{(j,k)\epsilon E} g_{jk} - (|K| - i + 1)p_{ij} \geq 0, \qquad \forall i\epsilon V, \forall j\epsilon V \tag{16}$$

$$\sum_{(i,k)\epsilon E} g_{ik} - \sum_{j\epsilon V} p_{ji} = 0, \qquad \forall i\epsilon V \tag{17}$$

$$p_{ij} \ \epsilon \{0,1\} \qquad \forall i\epsilon V, \forall j\epsilon V \tag{18}$$

Nyt minimoitava kohdefunktio (4) summaa kokonaisajan, joka verkostossa tulee kulutettua. Lausekkeet (5) ja (6) seuraavat tehtävänannosta, jossa x on määritelty. Ne takaavat sen, ettei kuljettu reitti voi haarautua moneen eri suuntaan samanaikaisesti. Rajoitus (7) varmistaa sen, että kokonaisvirtaus aloituspisteestä on yhtäsuuruinen verkoston solmukohtien lukumäärän kanssa, joissa kaikissa pitää käydä. Rajoitus (8) kertoo, että kun kuhunkin solmukohtaan saapuvasta virtauksesta vähennetään kyseisestä solmukohdasta lähtevä virtaus, saadaan tulokseksi 1. Rajoitus (9) varmistaa, että kaikki verkoston virtaus summattuna on yhtäsuuruinen kuin $\sum_{t=1}^{n} t$. Rajoitus (10) puolestaan varmistaa, että kaiken eri solmukohtien välillä tapahtuvan virtauksen on pakko olla pienempää tai yhtäsuurta kuin solmukohtien lukumäärä, eli pienempää kuin aiemmin määritelty kokonaisvirtaus. Rajoitus (11) kertoo etteivät virtausta kuvaavat muuttujat ole negatiivisia, ja se että muuttujat x ovat binäärisiä varmistetaan rajoituksella (12). Rajoitukset (13)-(14) takaavat että kullakin i:llä ja j:llä on vain yksi i:nnes ja j:nnes solmukohta jossa käydään. Rajoitus (15) kertoo, että ennen ensimmäistä vierailtua solmukohtaa lähdetään liikkeelle lähtöpisteestä. Rajoitukset (16)-(17) liittävät yhteen p:n ja g:n, ja viimeinen rajoite (18) takaa sen, että p on binäärinen.

Virtausmääritelmän hyödyt ovat lähteen [3] mukaan siinä, että tällä tavalla muotoiltua minimilatenssiongelmaa ratkaistaessa laskenta-aika on huomattavasti pienempi ja laskettavissa olevat tehtävät ovat selkeästi solmukohtien lukumääriltään suurempia, kuin esimerkiksi Fischettin [18] tai Eijln [19] muotoilemissa minimilatenssiongelmissa. Lisäksi *GAP LR* (linear relaxation gap) on virtausmääritelmässä selvästi pienempi verrokkeihinsa nähden. GAP LR tarkoittaa optimin ja alarajan suhteellista erotusta. Virtausmääritelmän tehokkuus verrattuna Fischettin ja Eijln muotoiluun sekä vaikkapa ylempänä esitettyyn matkustavan kauppamiehen ongelman pohjalta tehtyyn muotoiluun johtuu sen suuresta joukosta rajoitusehtoja, jotka parantavat mallin laskentakapasiteettia. Olennaisimpia rajoitusehtoja ovat (9) ja (13)-(18), joita ilman mallin ratkaisemistehokkuus palautuisi suurin piirtein samalle tasolle kuin se on Fischettin muotoilussa.

3. Eksakteja ratkaisualgoritmeja joihinkin erityistilanteisiin

Minimilatenssiongelma on perusmuodossa NP-ongelma, joka tarkoittaa sitä, että sille ei tunneta polynomiaikaista algoritmia. On kuitenkin joitakin erityistapauksia, joissa ongelma kuuluu P-joukkoon ja sille löytyy polynomiaikainen algoritmi, mutta niitä ei ole erityisen montaa. Kirjallisuudessa parhaiten tunnettuja tapauksia, joissa eksakti ratkaisu löytyy, ovat *puugraafi* (tree network) jossa solmukohtien etäisyydet ovat samat [6], puugraafi jonka halkaisija on 3 [2], verkosto joka on polku [8], puugraafi jossa niin sanottujen lehtien lukumäärä on vakio [9] sekä eräät yleisluontoisemmat tilanteet painotetusta minimilatenssiongelmasta [7]. Käyn läpi tässä luvussa kaksi ensinmainittua tapausta ratkaisualgoritmeineen näistä erityistilanteista, joissa minimilatenssiongelmalla on polynomiaikainen ratkaisu. Lopuksi käydään yhteenvetona läpi minimilatenssiongelmaa yleisemällä tasolla ottamalla huomioon painokertoimet, ja lisäksi käydään läpi joitakin periaatteita, joihin perustuu se, että luvun alkupuolella esitetyille tapauksille on olemassa eksaktit ratkaisut.

3.1 Puugraafi, jossa solmukohtien väliset etäisyydet ovat samat

Eräs yksinkertaisimmista minimilatenssiongelman ratkaisualgoritmeista löytyy lähteestä [6] ja koskee tapausta, jossa verkosto on puugraafi, ja kaikkien solmukohtien väliset etäisyydet ovat samat. Tällöin ratkaisualgoritmi mistä tahansa aloituspisteestä käsin on yksinkertaisesti mikä tahansa puugraafin *syvyyssuuntainen läpikäynti* (depth-first). Syvyyssuuntainen läpikäynti on käyty alla läpi lyhyesti.

Syvyyssuuntainen läpikäynti

1. Lähdetään aloituspisteestä johonkin puun haaraan ja edetään koko haaran syvyyteen, kunnes tulee vastaan umpikuja.

2. Umpikujasta lähdetään takaisinpäin sen verran, että tulee vastaan mikä tahansa toinen haara, jossa ei ole vielä käyty, ja ensimmäisen vaiheen tapaan kuljetaan sitä haaraa mahdollisimman pitkälle.

Toista vaihetta toistetaan niin kauan, että kaikki solmukohdat on käyty läpi, joihin on mahdollista päästä aloitussolmusta.

Todistus tämän syvyyssuuntaisen läpikäynnin pätevyydestä tässä nimenomaisessa minimilatenssiongelman tapauksessa löytyy ainakin lähteistä [6] ja [2], joista esittelen lyhyesti jälkimmäisen lähteen todistuksen. Se seuraa suoraan kahdesta alla olevasta lauseesta 3.1 ja 3.2:

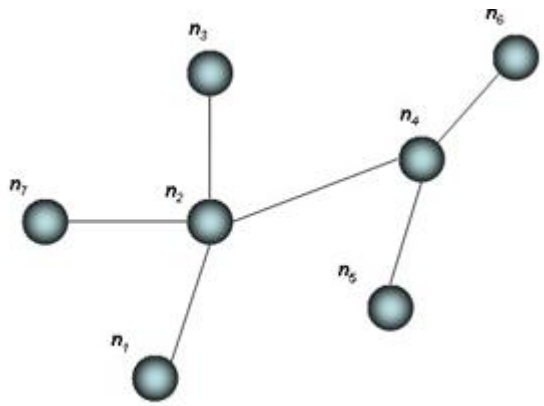
Lause 3.1. Tehdään mielivaltainen kierros puugraafissa, jossa solmukohtien väliset etäisyydet ovat samat. Merkitään etäisyyttä kaikkien solmukohtien välillä arvolla 1. Jos v_i on mikä tahansa i:nnes ensimmäistä kertaa vierailtu solmukohta ($i = 1,2, \dots, n$), v_i:hen tultaessa kuljettu matka M aloituspisteestä on aina enemmän tai yhtä suuri kuin $M = 2i$-syvyys(v_i), missä syvyys(v_i) tarkoittaa solmukohdan v_i etäisyyttä aloituspisteestä.

Lause 3.2. Tehdään syvyyssuuntainen läpikäynti mielivaltaisesta aloituspisteestä liikkeelle lähtien puugraafissa, jossa solmukohtien väliset etäisyydet ovat samat. Jos v_i on mikä tahansa i:nnes ensimmäistä kertaa vierailtu solmukohta ($i = 1,2, \dots, n$), v_i:hen tultaessa kuljettu matka M on täsmälleen $M = 2i$-syvyys(v_i).

Todistus lauseille 3.1 ja 3.2. Astetaan i:lle jokin vakioarvo, jolloin lause 3.1. osoittautuu ilmeiseksi: Palattaessa alkupisteeseen i:nnen ensimmäistä kertaa vieraillun solmukohdan jälkeen, on selkeästi kuljettu matka tällaisissa tapauksissa vähintään 2i askelta. Se voi olla myös enemmän kuin sen verran, koska mielivaltaisella kierroksella on voitu kulkea takaisinpäinkin ennen alkupisteeseen palaamista. Lisäksi syvyyssuuntaisessa läpikäynnissä se on lauseen 3.2. mukaisesti täsmälleen sen verran, koska algoritmissa ei kuljeta minkään solmukohtien välistä etäisyyttä koskaan useammin kuin 2 kertaa, ja aina kun verkoston haaroista palataan alkupisteeseen, pitkittyy kierros arvolla syvyys(v_i), ja saapumisaika i:nnenteen solmukohdaan on täsmälleen 2i-syvyys(v_i). □

3.2 Puugraafi, jonka halkaisija on 3

Puugraafilla, jonka halkaisija on 3, tarkoitetaan verkostoa, jossa on kaksi keskussolmukohtaa, joiden välillä on polku. Lisäksi molemmista keskussolmukohdista lähtee eri suuntiin muita polkuja, jotka päättyvät umpikujaan heti ensimmäiseen solmukohtaan tultaessa. Näin etäisyys verkoston toiselta laidalta toiselle laidalle on aina maksimissaan kolmen välimatkan mittainen. Esimerkkitapaus tällaisesta verkostosta on alla olevassa kuvassa 1.



Kuva 1. Esimerkki puugraafista, jonka halkaisija on 3.

Lähde [2] kertoo, että tällaiseen verkostoon saadaan ratkaisualgoritmi dynaamisella optimoinnilla. Dynaamisen optimoinnin tilat koostuvat kahdesta kokonaisluvusta k_L ja k_R, jotka tarkoittavat vasemmasta ja oikeasta keskussolmukohdasta lähtevien vierailtujen haarojen lukumäärää. Näiden tilojen lukumäärä on muotoa $O(n^2)$. Optimoinnin tuloksena saatu algoritmi toimii siten, että keskussolmukohdista lähdetään niistä haarautuviin polkuihin etäisyyksien mukaisesti järjestyksessä lyhyimmästä pisimpään. Aina kun toisesta keskussolmukohdasta kuljetaan toiseen, niiden välinen etäisyys huomioidaan määritettäessä keskussolmukohdista haarautuvien päätepisteiden etäisyyksiä.

3.3 Kategorisempi hahmottelu tilanteille, joihin löytyy polynomiaikainen ratkaisu

Tämän alaotsakkeen alla käydään pääosin lähteen [7] pohjalta yleisluontoisemmin läpi sitä, mihin perustuu että minimilatenssiongelmia voidaan joissakin tapauksissa optimoida dynaamisella ohjelmoinnilla polynomisessa ajassa. Tässä kappaleessa tarkastellaan minimilatenssiongelman eri variaatioita aiempaa laajemmin huomioimalla minimilatenssiongelman painokertoimet, jolloin ongelmaa kutsutaan *painotetuksi minimilatenssiongelmaksi* (minimum weighted latency problem, MWLP). Lisäksi tässä kappaleessa sivutaan usean eri kauppamiehen minimilatenssiongelmaa. Näistä minimilatenssiongelman yleisluontoisemmista variaatioista käydään läpi tuloksia, jotka ovat myös välttämättömiä edellytyksiä aiemmin läpikäytyjen erikoistapausten ratkaisualgoritmien olemassaololle. Aiemmin läpikäydyt minimilatenssiongelman muotoilut ovat itse asiassa vain tasakertoiminen osajoukko painotetusta minimilatenssiongelmasta, ja tässä mielessä tämä kappale toimii myös eräänlaisena yhteenvetona luvun 3 eksakteille ratkaisualgoritmeille.

Painotettu minimilatenssiongelma tarkoittaa sitä, että summataan saapumisaikoja, joilla kullakin on edessään erisuuruisia painokertoimia. Reaalimaailmassa tämä voisi tarkoittaa esimerkiksi tilannetta, jossa kiertelevällä kauppamiehellä on eriarvoisia asiakkaita, ja painokertoimet kuvaavat asiakkaiden erisuuruista rahallista arvoa. Määritellään painotettu minimilatenssiongelma formaalimmin:

Olkoon V, G ja $d(x, y)$määritelty kuten aiemminkin. Nyt lisäksi jokaiseen solmukohtaan $v \in V$ liittyy painokerroin $w(v)$, joka kuvaa kyseisen solmukohdan tärkeyttä. Olkoon P kierros, joka alkaa aloituspisteestä s, käy läpi kaikki verkoston G solmukohdat ja palaa aloituspisteeseen s. Solmukohtien $u, v \in G$ lyhin etäisyys verkostossa G ilmaistaan merkinnällä $d_G(u, v)$. Kun kierroksella P käydään ensimmäisen kerran u:ssa ja sen jälkeen ensimmäisen kerran v:ssä, näiden välillä kuljettua matkaa ilmaistaan merkinnällä $d_P(u, v)$. Painotettu latenssi kierrokselle P on:

$$L(P) = \sum_{v \in V(P)} w(v) d_p(s, v) \ . \qquad (19)$$

Määritelmä 3.1. Verkostolle G ja aloituspisteelle s, painotettu minimilatenssiongelma on minimointitehtävä, jossa on tarkoitus löytää s:stä alkava kierros P, joka minimoi lausekkeen (19).

Määritellään H osakierrokseksi ja merkitään sen solmukohtia termillä $V(H)$. Merkitään lisäksi H:n solmukohtien painokerrointen summaa termillä $w(H)$, koko verkoston solmukohtien painokerrointen summaa termillä $w(G)$ ja osakierroksen H pituutta termillä $d(H)$. Nyt voidaan kirjoittaa jatkoa ajatellen tarpeellinen lauseke:

$$c(G,P) = L(P) + \big(w(G) - w(H)\big)d(H)\ . \qquad (20)$$

Lähteen [7] mukaan painotetut minimilatenssiongelmat voidaan eräissä tapauksissa ratkaista dynaamisella optimoinnilla. Pääperiaate, johon lähteen [7] tulokset nojaavat, on että mikäli jokin minimilatenssiongelmaan liittyvä verkosto voidaan jakaa osaverkostoiksi siten, että kussakin osaverkostossa kulkeva osareitti tai osaratkaisu kulkee kyseisen osaverkoston solmukohdat samassa keskinäisessä järjestyksessä kuin ne kuljetaan globaalissa ratkaisussa, voidaan kyseinen ongelma ratkaista dynaamisella ohjelmoinnilla.. Alla on esitetty tämä pääajatus hieman muodollisemmin ja yksityiskohtaisemmin lauseeksi kirjoitettuna.

Määritelmä 3.2. Kun verkosto G jaetaan osaverkostoiksi G_1:ksi ja G_2:ksi solmukohdan v_l kohdalta, pätee $V(G_1) \cap V(G_2) = \{v_l\}$ ja G_1:n ja G_2:n unioni on G. Ositus siis säilyttää kaikki solmukohdat.

Lause 3.3. Oletetaan, että verkosto G voidaan jakaa k:ksi eri osaverkostoksi joissakin leikkauskohdissa. Mikäli osaverkostoissa kuljetuissa optimaalisissa osakierroksissa kuljetaan solmukohtien läpi samassa järjestyksessä kuin globaalissa ratkaisussa kuljetaan, ja osaratkaisut voidaan ratkaista ajassa $O(f(n))$, painotettu minimilatenssiongelma verkossa G voidaan ratkaista ajassa $O(k^2(\frac{n}{k}+1)^k + f(n))$ dynaamisella optimoinnilla.

Todistus. Koska kunkin osaverkoston solmukohtien vierailujärjestys on ennaltamäärätty koko globaalin verkon käsittävällä optimikierroksella, voidaan m:ssä ($m = 1,2,\dots n$) solmukohdassa kulkevalle osareitille määrittää tilavektori $l = (l_1, l_2, \dots l_k)$, missä $r \leq k$, ja jossa tilavektorin komponenttien summa on m. Tilavektorin komponentit l_i tarkoittavat kussakin osaverkostossa i vierailtujen solmukohtien lukumäärää, josta seuraa että niiden summa on m. r on rajoitus sille, missä osaverkostossa osakierroksen täytyy pysähtyä. Määritellään $T(l,r) = min_p\{c(G,P)\}$, joka tarkoittaa kaikkien sellaisten osakierrosten minimiä, jonka tilat voidaan esittää aiemmin kuvatuilla muuttujilla l ja r. Olkoon v_i viimeisin i:nnessä osaverkostossa vierailtu solmukohta jokaisella sellaisella osakierroksella jonka tila voidaan kuvata muuttujalla l ja olkoon u se solmukohta jossa vieraillaan seuraavaksi osaverkostossa r. Määritellään l' sellaiseksi vektoriksi, joka saadaan kun l:n r:ttä komponenttia suurennetaan yhdellä. $T(l',r)$ voidaan ilmaista seuraavasti:

$$T(l',r) = min\left\{T(l,r) + \Big(w(G) - w\big(V(l)\big)\Big)\, d_G(v_i,u)\right\}, \qquad (21)$$

jossa $V(l)$ tarkoittaa vierailtujen solmukohtien joukkoa osakierroksella, jota kuvataan muuttujalla l. Oletetaan että solmukohta u ei ole kierroksen alkupiste ja että v_r ei ole viimeinen vierailtu solmukohta r:nnellä osakierroksella. Määritellään myös $T(l,i) = \infty$, jos l:n i:nnes komponentti on nolla. Painotettu minimilatenssiongelma on yhtäsuuruinen lausekkeen $min_{1\leq i\leq k}\{T(l^*,i)\}$ kanssa, kun l^* on koko verkon solmukohtajoukkoa kuvaava vektori. Koska tällaisia vektoreita on enintään

$(\frac{n}{k}+1)^k$kappaletta, tilojen lukumäärä on $k(\frac{n}{k}+1)^k$. Aikakompleksisuus seuraa siitä, että funktio T voidaan ratkaista ajassa $O(k)$ jokaista tilaa kohden. □

Lauseesta 3.3. seuraa se, että luvun 3 johdannossa mainitut kirjallisuudessa parhaiten tunnetut minimilatenssiongelman muodot ovat ratkaistavissa polynomisessa ajassa. Esimerkiksi luvun 3 johdannossa mainittu verkosto, joka on yksittäinen polku, on selkeästi tälläinen osaverkoiksi jaettavissa oleva reitti, jonka osaratkaisuissa käydään solmukohdat läpi samassa keskinäisessä järjestyksessä kuin ne käytäisiin koko verkon ratkaisussa. Luvun 3 johdannossa mainittiin myös puugraafi, jossa on k eri lehteä, ja sekin on tällainen verkosto: se voidaan jakaa origon kohdalta (josta lehdet lähtevät eri suuntiin) k:ksi eri osaverkostoksi, ja kun ongelma ratkaistaan erikseen kussakin eri osaverkostossa eli kussakin lehdessä, on vierailujärjestys kunkin lehden solmukohdissa sama kuin se on kokonaisratkaisussakin. Tämä sama periaate pätee myös kappaleessa 3.2 käsiteltyyn puugraafiin, jonka halkaisija on 3. Tämäkin kyseinen verkosto voidaan jakaa kahden keskussolmukohdan kohdalta kahteen osaverkostoon, joissa solmukohtien keskinäinen vierailujärjestys on sama kuin se on globaalilla koko verkon kattavalla reitillä.

Lauseesta 3.3. seuraa myös, että minimilatenssiongelma on ratkaistavissa polynomisessa ajassa eräissä sellaisissa tilanteissa, joihin ei viitata kirjallisuudessa yhtä usein kuin yllä mainittuihin. Nämä tilanteet on listattu alle lauseiksi ja niistä johdetuiksi korollaareiksi. Lause 3.3. ei ole riittävä ehto kaikille näille tuloksille, mutta se on välttämätön ehto niille. Kukin tulos on todistettu tarkemmin lähteessä [7].

Lause 3.4. Oletetaan että H on tähtiverkosto, jossa on yksi keskusssolmukohta p, josta haarautuu eri suuntiin umpikujiin päättyviä polkuja. Jokaiselle H:n haaralle i ja j pätee: i:ssä käydään ennen j:tä optimaalisella kierroksella, jos $w(i)/d(p,i) > w(j)/d(p,j)$, missä $w(x)$ tarkoittaa minkä tahansa haaran x solmukohtien yhteenlaskettuja painokertoimia ja $d(p,x)$ tarkoittaa haaran x pituutta.

Lauseesta 3.4. ja ja lauseesta 3.3. seuraavat korollaari 3.1. ja korollaari 3.2.

Korollaari 3.1. Olkoon H sellainen tähtiverkosto, joka on verkosta G erilleen katkaistu osaverkosto. H:n solmukohtien vierailujärjestys G:llä kulkevalla optimireitillä voidaan ratkaista ajassa $O(nlogn)$.

Korollaari 3.2. Painotettu minimilatenssiongelma verkossa, jossa on k niin sanottua sisäistä solmukohtaa, voidaan ratkaista ajassa $O(k^2((n/k+1)k + nlogn)$.

Lause 3.5. Minimilatenssiongelman optimaalisen reitin aloituspiste 1-ulotteisella polulla, jossa on n solmukohtaa, voidaan ratkaista ajassa $O(n^2)$.

Viimeiset tulokset koskevat sellaista painotettua minimilatenssiongelmaa, jossa verkostossa on k eri kiertävää tahoa. Muilta osin se on samanlainen kuin tavanomainenkin minimilatenssiongelma, mutta ero on se, että kaikkien kiertäjien ei tarvitse käydä kaikissa solmukohdissa. Riittää, että kussakin solmukohdassa käy vain jokin kiertäjistä.

Lause 3.6. Painotettu minimilatenssiongelma, jossa k kiertävää tahoa kulkee yksiulotteisen polun muodostamassa verkostossa, voidaan ratkaista ajassa $O(n^3)$ kun $k = 2$, ja ajassa $O(n^4)$ kun $k > 2$.

Korollaari 3.3. Painotettu minimilatenssiongelma, jossa k kiertelevää tahoa kulkee ympyrän tai kehän muotoisessa verkostossa, voidaan ratkaista ajassa $O(n^4)$.

Luku 3 päättyy tähän. Tässä luvussa käytiin läpi kaikki tämän työn lähdeluettelosta löytyneet minimilatenssiongelman tapaukset, joissa polynomiaikainen ratkaisu tunnetaan. Seuraavassa luvussa käydään läpi joitakin approksimaatioalgoritmeja ja heuristisia algoritmeja minimilatenssiongelmaan.

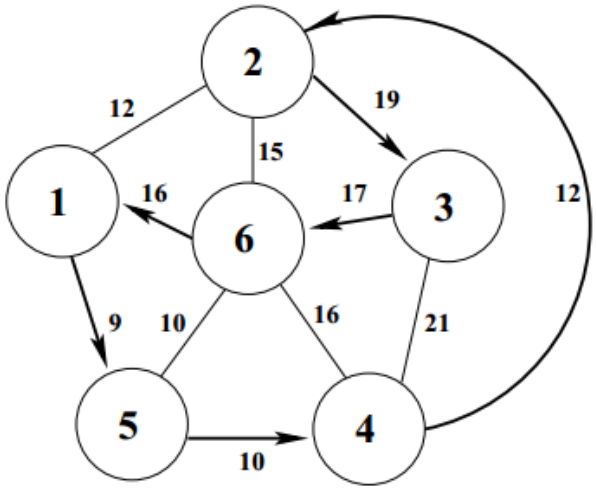
4. Ei-eksakteja ratkaisualgoritmeja

Eräs tapa ratkoa minimilatenssiongelmaa ei-eksaktilla tavalla ovat *approksimaatioalgoritmit*. Ensimmäiset tunnetut approksimaatioalgoritmit minimilatenssiongelmaan teki tiettävästi Blum et al. [2]. Kyseessä olivat 144-approksimaatio metriselle tapaukselle ja 8-approksimaatio puugraafeille. Molempia algoritmeja parantelivat Goemans ja Kleinberg [12]. Sittemmin erilaisia approksimaatioalgoritmeja minimilatenssiongelman eri tapauksiin ovat kehitelleet ainakin Grag [13], Arora ja Karakostas [14], ja Koutsopias, Papadimitriou sekä Yanakakis [9].

Toinen ei-eksakti tapa minimilatenssiongelman ratkaisuun, jota tässä työssä käydään läpi tarkemmin, ovat *heuristiset ratkaisualgoritmit*. Heuristiikka tarkoittaa yksinkertaistettuna yrityksen ja erehdyksen kautta tapahtuvaa ongelmanratkaisua, joka pohjautuu monesti iterointiin. Sekään ei anna eksaktia ratkaisua kuten eksaktit algoritmit antavat, mutta sen hyöty on siinä että sen avulla voidaan saada eksakteja algoritmeja pienemmällä vaivalla ja laskentateholla riittävän lähelle osuvia ratkaisuja. Heuristiset menetelmät eroavat siinä mielessä approksimaatioalgoritmeista, että niitä käytettäessä ei saada tietoa siitä, kuinka kauas jäädään optimiratkaisusta, kun taas approksimaatioalgoritmeja käytettäessä tiedetään kuinka kauas pahimmassa tapauksessa voidaan jäädä optimista. Tässä kappaleessa käyn läpi lähemmin kaksi hyvin yleistä heuristista algoritmia, joita voidaan käyttää minimilatenssiongelman ratkaisemiseen, ja jotka ovat paljon käytettyjä muissakin optimointitehtävissä. Toinen on *geneettinen algoritmi*, ja toinen on *tabualgoritmi*. Geneettisen algoritmin toiminta on referoitu lähteestä [4], ja tabualgoritmia käydään läpi lähteen [5] pohjalta. Aivan aluksi kuitenkin havainnollistetaan näiden algoritmien tarpeellisuutta esimerkillä 4.1. jossa yritetään ratkaista pientä malliverkostoa mielivaltaisiin kierroksiin perustuvalla Monte Carlo –simulaatiolla.

Esimerkki 4.1.

Tarkastellaan kuvan 2 esimerkkiverkostoa. Verkosto on peräisin lähteestä [3], jossa on myös kerrottu reitti, joka minimoi saapumisaikojen summan kyseisessä verkostossa. Ratkaisu on piirretty kuvaan 2, ja se on 9 + (9+10) + (9+10+12) + (9+10+12+19) + (9+10+12+19+17) + (9+10+12+19+17+16) = 259. Tässä esimerkissä on tarkoitus tutkia, kuinka lähelle kyseistä ratkaisua päästään Monte Carlo –simulaatiolla.



Kuva 2. Esimerkkiverkosto ja optimireitti joka minimoi saapumisaikojen summan.

Monte Carlo- simulaatiota tehtiin siten, että verkostossa kuljettiin täysin satunnaisia kierroksia, jotka aloitettiin solmukohdasta 1. Kussakin verkoston haarassa arvottiin mielivaltaisesti, että mihin suuntaan lähdetään. Satunnaiskierrosta jatkettiin niin kauan, että kaikissa solmukohdissa oli käyty ja lisäksi palattu aloitussolmukohtaan 1. Aluksi satunnaiskierroksia tehtiin 1000 kappaletta, ja katsottiin missä reitissä oli pienin latenssi eli saapumisaikojen summa. Tätä 1000 kierroksen simulaatiota toistettiin 10 kertaa. Tämän jälkeen satunnaiskierroksia tehtiin 10 000 ja etsittiin samaan tapaan pienilatenssisin reitti. Myös tämä simulaatio tehtiin 10 kertaa. Simulaatiot tehtiin numeeriseen matriisilaskentaan perustuvalla MATLAB 7.1 –ohjelmistolla. Ohjelmistoa käytettiin koneella, jossa oli 3.17GHz-nopeuksinen prosessori. Koodi jota simulaatioissa käytettiin on nähtävissä liitteessä A.

Kun 1000 satunnaiskierroksen simulaatiota toistettiin 10 kertaa, pienilatenssisimmat reitit kullekin simulaatiolle olivat 296, 259, 259, 269, 259, 271, 269, 259, 269 ja 281. Aika joka näihin kaikkiin simulaatiohin yhteensä kului oli 5.148 sekuntia. Saaduista tuloksista nähdään, että 1000 satunnaiskierroksen simulaatiossa ei lähellekään aina löydetä optimireittiä, joka on 259. Tässä simulaatiossa se löydettiin neljässä tapauksessa kymmenestä. Kun puolestaan 10 000 satunnaiskierrosta toistettiin 10 kertaa, pienilatenssisimmat kullekin simulaatiolle olivat 259, 259, 259, 259, 269, 259, 259, 259, 259, 259. 10 000 satunnaiskierroksen simulaatiossa optimireitti löydettiin yhdeksässä tapauksessa kymmenestä. Näihin 10 simulaatioon kului aikaa kokonaisuudessaan 97.095 sekuntia. Kummassakaan tapauksessa ei löydetty kaikilla kerroilla optimaalista reittiä, ja etenkin jälkimmäiseen simulaatioon kulunut aika oli huomattavan pitkä.

Minimilatenssiongelma on siis hyvin raskas ongelma ratkaistavaksi näinkin pienessä verkostossa, ja kehittyneemmät algoritmit sen ratkaisemiseksi ovat tarpeellisia. Seuraavassa kappaleessa tarkastellaan geneettistä algoritmia ja sen tehokkuutta verrataan tämän esimerkin Monte Carlo –iteraatioon, joka havaittiin jokseenkin tehottomaksi.

4.1 Geneettinen algoritmi

Geneettinen algoritmi on viime vuosina suosituksi tullut optimointimenetelmä, joka ottaa nimikkeistöä genetiikasta ja luonnossa tapahtuvasta evoluutiosta. Siinä eri osaratkaisujen voidaan vertauskuvallisesti sanoa olevan ”selviytymistaistelussa” ja ”evoluution alaisina” kuten eliöyksilöt luonnossa. Osaratkaisuja yhdistellään toisiinsa tai muunnellaan satunnaisesti, ja epäsopivimmat ratkaisut putoavat prosessin aikana pois. Algoritmin suorittamista jatketaan, kunnes riittävän hyvä osaratkaisu on löytynyt. Geneettisen algoritmin analogiat vastaaviin evoluution käsitteisiin ovat selkeitä: osaratkaisujoukkoja kutsutaan ”populaatioiksi”, joiden jäseniä eli bittijonoja kutsutaan usein ”kromosomeiksi”. Nämä puolestaan koostuvat alkioista, joita kutsutaan ”geeneiksi”. Geneettisen algoritmin hyvinä puolina pidetään sen tehokkuutta vaikeissa ja monia parametreja sisältävissä tehtävissä, joissa eksaktin ratkaisun löytäminen on haastavaa. Määritellään aluksi tarkemmin geneettisen algoritmin yleismuotoinen rakenne. Se on kirjoitettu alle koodikielissä usein käytetyn while-silmukan muotoon.

Geneettinen algoritmi yleisessä muodossaan:

Muodostetaan alkuperäinen osajoukkoratkaisu joillakin yksinkertaisemmilla algoritmeilla;
While not (algoritmin lopettamiskriteeri on täytetty)
Etsitään jokin määrä parhaita ratkaisuja osaratkaisujoukosta;
Yhdistellään parhaimpien ratkaisujen ominaisuuksia toisiinsa;
Muunnellaan osaratkaisuja satunnaisesti;
Lisätään yhdistelmäratkaisut osaratkaisujoukkoon;
poistetaan huonoimmat ratkaisut osaratkaisujoukosta;
end
Algoritmin ulostulo: paras ratkaisu, joka osaratkaisujoukosta löytyy.

Tämän algoritmin toimintaa on testattu eräillä parametreilla alla olevassa esimerkissä 4.2. Siinä käydään läpi simulaatioita, joissa geneettistä algoritmia testataan samassa verkostossa, jota tarkasteltiin aikaisemmin esimerkissä 4.1. Tällä tavoin geneettisen algoritmin tehokkuutta

päästään vertaamaan aikaisemman esimerkin Monte Carlo –simulaatioon minimilatenssiongelman ratkaisemisessa.

Esimerkki 4.2.

Geneettistä algoritmia testattiin kuvan 2 verkostoon, jota tarkasteltiin jo aiemmin. Tarkoituksena oli jälleen etsiä reitti, joka ratkaisee minimilatenssiongelman eli minimoi saapumisaikojen summan. Yllä esitettyä geneettistä algoritmia testailtiin useilla eri parametreilla, ja tähän esimerkkiin on kirjattu eräs verrattain hyvin toiminut toteutus. Aluksi muodostettiin täysin satunnaisista kierroksista 15 osaratkaisun osaratkaisujoukko. Tämän jälkeen etsittiin kaksi parasta ratkaisua ja yhdistettiin ne toisiinsa, mikäli ne ristesivät jonkun solmukohdan kohdalta. Risteävistä solmukohdista poissuljettiin ensimmäinen ja toinen solmukohta, koska siinä vaiheessa kuljetut reitit olisivat vielä keskenään aivan samat. Lisäksi risteävistä solmukohdista poissuljettiin viimeinen solmukohta, koska kaikki minimilatenssiongelmaan käyvät ratkaisuyritelmät risteävät viimeisessä solmukohdassa joka tapauksessa. Mikäli reitit eivät ristenneet, ei kyseisen iteraation aikana yhdistelty ratkaisuja toisiinsa. Kunkin iteraation aikana 5 huonointa ratkaisuyritelmää poistettiin osaratkaisujoukosta ja muodostetiin uusia satunnaisratkaisuja, jotta osaratkaisujoukon koko pysyi samana. Simulaatiot tehtiin esimerkin 4.1. tapaan MATLAB 7.1 –ohjelmistolla, jota ajettiin koneella, jossa oli 3.17GHz-nopeuksinen prosessori. Geneettiseen algoritmin simuloinnin Matlab-koodi on nähtävillä liitteessä B.

Geneettinen algoritmi löysi kuvan 2 esimerkkiverkostoon optimiratkaisut verrattain nopeasti. Kymmeneen simulaatioon, joissa kussakin tapauksessa geneettinen algoritmi löysi optimiratkaisun 259, kului aikaa 8.3929s. Iteraatioiden määrä kussakin simulaatiossa oli 184, 373, 426, 201, 20, 718, 491, 152, 161 ja 688. Verrattaessa iteraatioita Monte Carlo –simulaatin iteraatiolukuihin, on luvut kerrottava viidellä, koska kullakin geneettisen algoritmin iteraatiokierroksella luotiin 5 uutta osaratkaisua. Vertailujen perusteella tämän esimerkin geneettinen algoritmi on tehokkaampi kuin esimerkin 4.1 Monte Carlo -simulaatio. Testattujen iteraatioiden lukumäärä Monte Carlo –simulaatiossa oli 1000 ja 10 000, ja näilläkään iteraatiomäärillä ei aina löydetty optimiratkaisua. Tämän esimerkin simulaatiot antavat viitteitä siitä, että geneettinen algoritmi yleisessä muodossaan on toimiva työkalu minimilatenssiongelman ratkaisuun.

4.1.1 Kohennettu geneettinen algoritmi minimilatenssiongelmalle [4]

Lähde [4] ehdottaa minimilatenssiongelman ratkaisuun alla olevaa muotoilua geneettisestä algoritmista. Se on kirjoitettu koodikielistä tuttujen while- ja for-silmukoiden muotoon. Kyseinen algoritmi esitellään siksi, että se on toistaiseksi paras kirjallisuudesta löytyvä geneettinen algoritmi minimilatenssiongelman ratkaisuun, ja lisäksi se havainnollistaa geneettistä algoritmia yleisemminkin. Algoritmin eri vaiheet on selostettu myöhemmin tässä kappaleessa tarkemmin, mutta itse algoritmi on seuraavanlainen:

Kohennettu geneettinen algoritmi:

Muodostetaan alkuperäinen osaratkaisujoukko;

While not (algoritmin lopettamiskriteeri on täytetty)

If(osaratkaisujen samankaltaisuus ylittää määrätyn kriteerin)

Säilytetään optimaalisin osaratkaisu ja muodostetaan muu osaratkaisujoukko uudelleen ;

else

For (i=1; i=< osaratkaisujoukon koko; i++)

Valitaan parhaimmat osaratkaisut valintaopetaattorilla;

Yhdistellään parhaimpien ratkaisujen ominaisuuksia toisiinsa;

Tehdään osaratkaisuihin satunnaisia muutoksia;

Toteutetaan lokaalia optimia etsivä algoritmi;

Lisätään yhdistelmäratkaisut osaratkaisujoukkoon;

poistetaan ylimääräiset osaratkaisut osaratkaisujoukosta;

end

Ulostulo: paras osaratkaisu, joka osaratkaisujoukosta löytyy.

Lähteessä [4] on eritelty ylläolevan geneettisen algoritmin eri vaiheet täsmällisesti ja formaalisti, mutta niitä ei ole katsottu tarpeelliseksi kopioida aivan sellaisenaan kokonaisuudessaan tähän työhön. Sen sijaan alla on sanallisesti tiivistetty, kuinka kukin kohta algoritmissa toimii, ja mitä asiaa mikäkin termi tarkalleen ottaen tarkoittaa.

Osaratkaisujoukon jäsenet ovat minimilatenssiongelman verkostojen solmukohdista koostuvia jonoja, joita voidaan kirjoittaa muotoon (v_1, v_2, ..., v_n). Kukin jono tarkoittaa eri reittiä, jonka minimilatenssiongelman kauppamies voi valita, ja v_i on i:nnes vierailtu solmukohta reitillä. Mitä suurempi on eri solmukohtiin tapahtuvien saapumisaikojen summa, sitä huonompi on osaratkaisujoukon yksilö, eli saapumisaikojen summa on kääntäen verrannollinen yksilön hyvyyteen.

Tässä algoritmissa alkuperäinen osaratkaisujoukko muodostetaan seuraavasti: 70% muodostetaan siten, että arvotaan 70:30- painotuksella joko lähimmän naapurin algoritmilla [16] tai täysin satunnaisesti. Painotus tapahtuu niin, että ensin mainittu vaihtoehto on todennäköisempi, ja satunnaisyksilöt ovat epätodennäköisempiä. Loput 30% alkuperäispopulaatiosta luodaan Blum et al [2]- ja Goenmans & Kleinberg et al –approksimaatioalgoritmeilla [12]. Valintaa näidenkin

algoritmien välillä painotetaan 70:30 –suhdanteella. Erilaisuutta algoritmeihin on lähteen [4] mukaan haettu sen takia, että saataisiin osaratkaisujoukkoon mahdollisimman erilaisia jäseniä.

Yhdistelmäratkaisujen tuottaminen keskenään ristetetyistä osaratkaisuista on minimilatenssiongelmaan tehdyissä geneettisissä algoritmeissa hankalampaa kuin sen esiasteessa matkustavan kauppamiehen ongelmassa. Matkustavan kauppamiehen ongelmassa solmukohtien optimaalisia järjestyksiä tietyissä solmukohtajonojen kohdissa voidaan kopioida sellaisenaan yhdistettävistä osaratkaisuista yhdistelmäratkaisuun, mutta minimilatenssiongelman tapauksessa tämä ei usein ole optimaalista, koska pienet muutokset jonon rakenteessa voivat aiheuttaa merkittäviä ei-lokaaleja muutoksia. Perusidea tässä minimilatenssiongelmaan kehitetyssä geneettisessä algoritmissa on se, että kaikista yhdisteltävien solmukohtajonojen alkioista käydään läpi vaihtoehdot sekä reitillä eteenpäin- että taaksepäin kulkemisen väliltä, ja yhdistelmäratkaisuun valitaan pienempilatenssisempi vaihtoehto. Tässäkin tapauksessa eri suuntien valintaa voidaan painottaa ennaltamäärättyjen todennäköisyyksien perusteella suuremman diversiteetin saamiseksi osaratkaisujoukkoon. Yhdistelmäratkaisuja tuottavan risteytysoperaattorin tarkka toiminta on kuvattu lähteessä [4] sivulla 11. Yhdistettävät osaratkaisut valitaan siten, että otetaan populaatiosta satunnaisotos osaratkaisuja ja niistä valitaan pienilatenssisimmat, jotka risteytetään. Yhdistettävien osaratkaisujen valintapaine kasvaa sitä enemmän, mitä enemmän ryhmässä on kelvollisuuseroja, ja tästä syystä valintapainetta voidaan haluttaessa lisätä valitsemalla odotusarvoisesti suurempia satunnaisotoksia, koska suuremmissa ryhmissä on suurempia latenssieroja.

Algoritmin vaihe, jossa tehdään osaratkaisuihin satunnaisia muutoksia, tapahtuu siten että kukin satunnaisvaihtelun kohteena oleva osaratkaisu jaetaan kahteen osaan. Tämän jälkeen siirretään yksi kerrallaan toisen osan solmukohdat ensimmäiseen osayksilöön. Solmukohtien uudet sijainnit pyritään valitsemaan yksilön optimaalisuuden edesauttamiseksi. Tätä uudelleensijoittelua jatketaan niin kauan, että kaikki toisen osayksilön solmukohdat on sijoitettu ensimmäiseen osayksilöön. Myös tämän operaattorin tarkka ja eksakti kuvaus on lähteessä [4], ja se löytyy sivulta 11.

For-silmukan kohta, jossa etsitään lokaalia algoritmia tarkoittaa 2-opt-node [17] – tai 3-opt-node-algoritmien [15] käyttöä lokaalin minimin paikallistamiseksi. Tässä tapauksessa näistä valitaan toinen algoritmi aina täysin satunnaisesti.

Osaratkaisujoukon uudelleen luominen tarkoittaa geneettisessä algoritmissa toimintoa, jolla pyritään lisäämään osaratkaisujoukon jäsenten keskinäistä erilaisuutta ja näin välttämään lokaaliin optimiin päätymisen globaalin optimin sijaan. Kaikissa geneettisissä algoritmeissa ei ole tätä vaihetta, mutta tämän alaotsikon alla läpikäydyssä kohennetussa geneettisessä algoritmissa sitä käytetään, ja se on eräs tämän algoritmin ominaispiirteistä. Tässä algoritmissa se tarkoittaa sitä, että kun osaratkaisujoukon jäsenet on tarpeeksi paljon samanlaisia, vain kelvollisin osaratkaisu säilytetään, ja muut yksilöt muodostetaan uudelleen aiemmin kuvailluilla alkuperäisen osaratkaisujoukon muodostamiseen käytetyillä satunnaismenetelmillä jonkin määrätyn iteraatioiden lukumäärän jälkeen.

Algoritmi lopetetaan siihen, kun tietyn iteraatiokierrosmäärän jälkeen kelvollisin osaratkaisu ei ole kehittynyt lainkaan paremmaksi.

4.1.2 Kohennetun geneettisen algoritmin kokeelliset tulokset ja johtopäätökset niistä

Lukuisat ylläolevan kohennetun geneettisen algoritmin parametrit ovat määrittelemättömiä ja ne täytyy määrittää kokeellisesti. Tällaisia ovat yhdistelmäratkaisujen muodostamiseen valittavan satunnaisotoksen koko, osaratkaisujoukon koko, satunnaisvaihteluiden todennäköisyys, yhdistelmäratkaisuissa käytettävät ennaltamäärätyt todennäköisyydet, iteraatioiden lukumäärä joka odotetaan ennen osaratkaisujoukon uudelleen luomista ja iteraatioiden määrä joiden jälkeen algoritmi lopetetaan mikäli kelvollisin yksilö ei ole kehittynyt. Nämä voidaan määrittää kokeellisesti siten, että pidetään aina muita parametreja vakioina ja vaihdellaan yhtä parametria, ja etsitään sille yrityksen ja erehdyksen kautta paras mahdollinen arvo.

Lähteen [4] mukaan näitä parametreja on määritetty erilaisissa tapauksissa, joissa geneettistä algoritmia on sen jälkeen on testailtu TSLIB-tietopankista [20] löytyviin verkostoihin. Sitä on vertailtu Archerin, Levinin ja Williamsin usein käytettyyn approksimaatioalgoritmiin [11]. Vertailusuureena on käytetty suhdannetta f/f^*, joka tarkoittaa saadun ratkaisun suhdetta optimin alarajaan. Vertailujen perusteella yllä esitelty kohennettu geneettinen algoritmi antaa parempia tuloksia. Sille eri tapahtumista saadaan keskimäärin tulokseksi f/f^* = 1,95 ± 0,10, ja vertailukohteena ollut algoritmi antoi vastaavissa tilanteissa keskimääräiseksi tulokseksi 3 ± 0,6. Sen lisäksi että tässä kappaleessa referoitu geneettinen algoritmi antaa parempia tuloksia, se on myös on stabiilimpi, koska sen tulokset ovat huonoimmissa ja parhaissa tapauksissa verrattain lähellä toisiaan.

4.2 Tabualgoritmi ja minimilatenssiongelma, johon on lisätty palkkiot

Viimeisenä käydään läpi minimilatenssiongelman sellainen muunnelma, jossa jokaisesta vieraillusta solmukohdasta saa jonkun tietyn ajasta riippuvan tuoton tai palkkion, ja se kuinka tällaista ongelmaa ratkaistaan tabualgoritmilla. Tätä ongelmaa kutsutaan nimellä *palkkioihin pohjautuva minimilatenssiongelma* (travelling repairman problem with profits), ja se tarkoittaa muilta osin samankaltaista ongelman asettelua kuin tavallinen minimilatenssiongelma, mutta sen sijaan että minimoitaisiin solmukohtiin tapahtivien saapumisaikojen summaa, maksimoidaan verkoston solmukohdista saatavien palkkioiden summaa, joka on muotoa $\sum_i (p_i - t_i)$,

missä p_i kuvastavaa solmukohdan i tärkeyttä ja t_i on saapumisaika solmukohtaan i. Olennainen ero aiempaan minimilatensisongelmaan on palkkioihin pohjautuvassa minimilatenssiongelmassa se,

ettei kaikissa solmukohdissa välttämättä käydä ollenkaan, koska vierailemattomien solmukohtien palkkiot menevät ennen pitkää nollaan kun aikaa kuluu tarpeeksi paljon.

Tabualgoritmi, jonka käyttöä palkkioihin pohjautuvaan minimilatenssiongelmaan selostetaan alempana tässä kappaleessa, on heuristinen menetelmä, jossa kullakin askeleella siirrytään verkostossa siihen viereiseen solmukohtaan, jossa tuottofunktio saa suurimman arvon. Algoritmin nimi tulee siitä, että aina siirtymän jälkeen kirjataan niin sanottuun tabulistaan edellinen solmukohta, josta siirryttiin pois, ja tabulistan lopusta poistetaan vanhin siihen kirjattu solmukohta. Tabulistassa oleviin solmukohtiin ei ole luvallista siirtyä, ja tällä tavoin algoritmi takaa sen, ettei kuljeta liian pian takaisinpäin niihin solmukohtiin, joista ollaan tulossa pois päin. Ennen kuin siirrytään palkkioihin pohjautuvan minimilatenssiongelman ratkaisemiseen, tarkastellaan aluksi esimerkin 4.3 avulla yksinkertaisen tabualgoritmin toimintaa.

Esimerkki 4.3.

Tarkastellaan jälleen kuvan 2 esimerkkiverkostoa, johon sovellettiin aiemmin Monte Carlo –simulaatiota esimerkissä 4.1 ja geneettistä algoritmia esimerkissä 4.2. Tähän samaan verkostoon sovellettiin vertailun vuoksi hyvin yksinkertaista tabualgoritmia, joka toimi muilta osin samalla tavalla kuin Monte Carlo –simulaatio, mutta rajoituksena oli, ettei samaan solmukohtaan ollut luvallista siirtyä, josta oltiin tulossa. Tehtiin siis esimerkkiverkostossa mielivaltaisia kierroksia, mutta käytettiin tabulistaa, joka oli yhden solmukohdan mittainen. Tarkoituksena oli taas minimoida saapumisaikojen summa eri solmukohtiin, eli ratkaista minimilatenssiongelma.

Kun 1000 kierroksen tabualgoritmisimulaatiota toistettiin 10 kertaa, pienilatenssisin reitti 259 löytyi yhdekssä tapauksessa kymmenestä. Ainoassa simulaatiossa, jossa optimia ei löytynyt, tulos oli 269. Aika joka kuhunkin simulaatioon kului oli välillä 0.499-0.515 sekuntia. Esimerkkien 4.1-4.2 ja tämän esimerkin perusteella tabualgoritmi on testatuista algoritmeista tehokkain laskenta-ajallisesti ja iteraatiokierrosten perusteella. Lisäksi saadut tulokset ovat suhteellisen luotettavat. Tabualgoritmin toteuttaminen ja koodaaminen oli myös huomattavasti vähemmän työlästä, kuin geneettisen algoritmin. Tämänkin esimerkin simulaatiot tehtiin MATLAB 7.1 –ohjelmistolla, ja niihin liittyvä koodi on nähtävillä liitteessä C.

4.2.1 Palkkioihin pohjautuva minimilatenssiongelma

Alla on selostettu lähteen [5] mukaisesti palkkioihin pohjautuvan minimilatenssiongelman matemaattinen malli. Tarkastellaan jälleen verkostoa $G = (V, E)$, jossa V on solmukohtien joukko ja E on niiden välisten välimatkojen joukko. Kuten aiemmin mainittiin, kustakin solmukohdasta i saadaan palkkio $p_i - t_i$, joka riippuu solmukohdan antamasta ennaltamäärätystä tuotosta p_i ja ajasta t_i, jolloin kyseiseen solmukohtaan saavutaan. Tämän seurauksena ei sellaisiin solmukohtiin i mennä lainkaan, joissa $p_i \leq t_i$, koska niistä ei saataisi mitään palkkiota. Joukko $K = \{1,2,\dots,k\}$

tarkoittaa niitä solmukohtia, joiden palkkio kerätään, ja $d_{i,j}$ tarkoittaa aikaa, jonka kauppamies käyttää kulkiessaan matkan solmukohdasta i solmukohtaan j. Lisäksi (i,j) tarkoittaa välimatkaa solmukohdan i ja solmukohdan j välillä.

Määritellään binäärinen muuttuja y, $\forall i \in V, \forall j \in V_0 = V\backslash\{0\}, \forall l \in K$:

$$y_{i,j,l} = \begin{cases} 1, jos\ \ (i,j) on\ l{:}nnes\ kuljettu\ välimatka \\ 0\ muussa\ tapauksessa \end{cases} \quad (19)$$

Kun $y_{i,j,l} = 1$, (i,j) on l:nnes kuljettu välimatka ja i on $(l-1)$:nnes vierailtu solmukohta ja j on l:nnes vierailtu solmukohta. Jos $y_{i,j,l} = 1$, $d_{i,j}$ lasketaan $k+1-l$ –kertaisesti kokonaisaikaan. Tästä seuraa:

$$\sum_{i:vierailtu} t_i = \sum_{\{(i,j,l)|y_{i,j,l}=1\}} (k+1-l) d_{i,j} \quad (20)$$

Nyt voidaan muodostaa kohdefunktio, jota tässä tehtävässä maksimoidaan. Kohdefunktio summaa kunkin vieraillun solmukohdan tuoton ja saapumisajan erotuksen:

$$max \sum_{i \in V} \sum_{j \in V_0} \sum_{l \in K} \left(p_j - (k+1-l) d_{i,j}\right) y_{i,j,l} \quad (21)$$

Rajoitusehdot ovat:

$$\sum_{i \in V} \sum_{l \in K} y_{i,j,l} \leq 1, \ \ \forall j \in V_0 \quad (22)$$

$$\sum_{i \in V} \sum_{j \in V_0} y_{i,j,l} \leq 1, \ \ \forall l \in K \quad (23)$$

$$\sum_{i \in V} y_{i,j,l} - \sum_{i \in V_0} y_{i,j,l+1} = 0, \ \ \forall j \in V_0, \forall l \in K\backslash\{k\} \quad (24)$$

$$\sum_{j \in V_0} y_{0,j,1} = 1 \quad (25)$$

$$y_{i,j,l} \in \{0,1\}, \quad \forall i \in V, \forall j \in V_0, \forall l \in K. \quad (26)$$

Ensimmäinen rajoitusehto takaa sen, ettei missään solmukohdassa käydä useammin kuin kerran. Toinen takaa sen. että k:ssa eri solmukohdassa aloituspisteen lisäksi on käytävä. Kolmas ehto takaa liitettävyyden ja neljäs sen, että lähdetään liikkeelle valitusta aloituspisteestä. Viimeinen rajoitusehto takaa, että y on binäärinen.

Määritellään $f(k)$ kohdefunktion optimiksi, kun ollaan vierailtu k:ssa solmukohdassa, ja olkoon $f^* = f(k^*)$ globaali optimi, jolloin optimaalinen määrä vierailtuja solmukohtia on k^*. Lähteen [5] mukaan $f(k)$ ei ole unimodaalinen, ja solmukohtia lisäämällä k^*n arvo saattaa laskea.

4.2.2 Tabualgoritmi palkkioihin pohjautuvan minimilatenssiongelman ratkaisemiseksi

Lähde [5] esittelee tabualgoritmin, jolla voidaan ratkoa palkkioihin pohjautuvaa minimilatenssiongelmaa. Tätä algoritmia pohjustetaan konstruktiovaiheella, jossa paikallistetaan triviaaliratkaisu ja etsitään sen pohjalta jokin ei-triviaali ratkaisu. Tämän jälkeen siirrytään varsinaiseen tabualgoritmiin, johon sisältyy intensifikaatiovaihe ja diversifikaatiovaihe. Nämä mainitut vaiheet on selostettu lyhyesti alla.

Konstruktiovaiheessa osaratkaisuun lisätään vierailemattomia solmukohtia sen perusteella, kuinka paljon aikaa lisäykseen kuluu ja millainen tuotto lisäyksestä saadaan. Alkuperäinen osaratkaisu, jota lähdetään konstruoimaan voi olla esimerkiksi sellainen triviaaliratkaisu, jossa alkupisteestä ei olla lähdetty mihinkään.

Määritellään ratio,

$$ratio_{i,j}{}^{m} = \begin{cases} \frac{1}{d_{i,j}}, jos\ m = 0 \\ p_j * \frac{1}{d_{i,j}}, jos\ m = 1, \dots, 10 \end{cases} . \qquad (27)$$

Solmukohtien lukumäärä m määrittää ajan $d_{i,j}$ vaikutusta ratioon. Konstruktiovaiheessa ratiota käytetään uusien solmukohtien lisäämiseen osaratkaisuun siten, että lisättävä solmukohta kullakin askeleella toteuttaa lausekkeen (28),

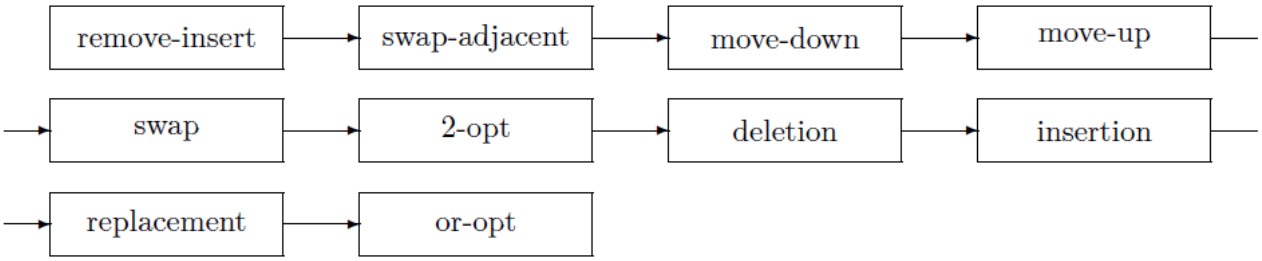
$$j^* = argmax_{j \in \bar{V}} ratio_{i,j}{}^{m}\ , \qquad (28)$$

missä $\bar{V}$ on vierailemattomien solmukohtien joukko.

Tabualgoritmia aletaan käyttää, kun konstruktiovaiheessa on saatu aikaiseksi osaratkaisu. Tabualgoritmi toimii siten, että sillä etsitään kussakin osaratkaisun reitillä olevassa solmukohdassa parasta siirtymävaihtoehtoa kyseisen solmukohdan naapurustosta. Paras siirtymä on sellainen, jossa kohdefunktio kasvaa eniten, tai jossa kohdefunktio vähenee vähiten. Reitillä kulkemista siis jatketaan, vaikka kohdefunktio alkaisi pienentyä, koska kyseessä saattaa olla vain lokaali optimi. Koska palkkioihin pohjautuvassa minimilatenssiongelmassa maksimoidaan solmukohdista saatavaa ajasta riippuvaa tuottoa, voidaan ratkaisua parantaa optimaalisemmaksi kahdella tavalla: muuttamalla vierailtujen solmukohtien määrää tai muuttamalla solmukohtien vierailujärjestystä. Näihin kahteen toimintoon perustuu kuvassa 3 olevat toimenpiteet, joita tätä ongelmaa ratkaiseva tabualgoritmi tekee. Kuvan järjestyksestä ilmenee myös se järjestys, jossa tabualgoritmi näitä toimenpiteitä tekee. Näitä toimintoja suoritetaan reitin solmukohdille yksi kerrallaan.

Kuvan 3 Remove-insert –toiminnossa katsotaan, mikä reitillä oleva solmukohta on kaikkein suurimman etäisyyden päässä tarkastelun kohteena olevasta solmukohdasta, ja lisätään se reitin viimeiseksi. Move-up– ja –down-toiminnoissa siirretään solmukohtaa reitillä ylös- tai alaspäin. Loput kuvan toiminnot ovat yleisemmin tunnettuja algoritmeja tai tarkoittavat suoraan sitä, mitä

niiden nimi antaa ymmärtää. Kukin toiminto kirjataan omaan tabulistaansa, jonka takia tulevilla iteraatiokierroksilla samoja toimintoja ei voi toistaa samoille solmukohdille uudelleen. Lisäksi algoritmin nopeuttamiseksi siihen on sisällytetty rajoitus, jonka mukaan move-up -, move-down -, swap-, 2-opt- ja or-opt –toiminnot saavat lisätä maksimissaan $n/2$ solmukohtaa kuljetulle reitille, jos n on suurin mahdollinen solmukohtamäärä, jonka ne muuten saisivat lisätä.



Kuva 3. Järjestys, jossa algoritmeja suoritetaan tutkittaessa solmukohtien vierailujärjestyksen tai vierailtujen solmukohtien vaihtamisen vaikutusta kohdefunktioon.

Kun yllämainittuja toimintoja käyttäen ollaan saavutettu lokaali optimi, alkaa intensifikaatiovaihe. Tässä vaiheessa muodostetaan matriisi M, jonka alkioihin $M(i,j)$ kirjataan se, kuinka monta kertaa reitti (i,j) oli osana optimiratkaisua. Tämän jälkeen saatua ratkaisua aletaan käyttää uutena osaratkaisuna kuvan 3 mukaiselle algoritmille, joka ajetaan nyt ilman yllä olevia rajoituksia solmukohtamäärille ja ilman tabulistoja.

Seuraava vaihe on diversifikaatiovaihe. Tässä vaiheessa tabulistat tyhjennetään, jonka jälkeen matriisia M käytetään siten, että sakotetaan siihen kirjattujen usein käytettyjen reittien kulkemisesta. Maksimoitavasta kohdefunktiosta vähennetään ennaltamäärätty sakko kerrottuna matriisin alkiolla $M(i,j)$ kullekin reitille (i,j). Tällä menetelmällä suositaan uudenlaisia reittejä. Tällaisella uudenlaisella alustuksella, johon mainitut sakot sisältyvät, tehdään 100 uutta iteraatiota, jonka aikana tabulistat muodostetaan uudestaan. Tästä vaiheesta saatua ratkaisua käytetään nyt uutena sisääntulona tabualgoritmille.

Lopuksi ajetaan läpi sellainen variaatio diversifikaatiovaiheesta, jossa annetaan palkkiot M:ään kirjatuista usein käytetyistä etäisyyksistä sen sijaan että niistä sakotettaisiin. Tällä tavoin taataan, että kaikki mahdollisimman erilaiset ratkaisut tulee kokeiltua. Algoritmi päättyy, kun ollaan saavutettu tietty määrä askelia, joiden aikana ratkaisu ei ole enää kehittynyt. Tämän askelmäärän määrittämisessä on tehtävä jonkunlainen kompromissi laskentatehon ja ratkaisun tarkkuuden välillä.

Lähteen [5] lopussa esitellään algoritmin antamat tulokset ja tehdään yhteenveto. Tulokset on laskettu testaamalla algoritmia TSLIB-tietopankista [20] löytyviin satunnaisesti valittuihin verkostoihin, joihin on lisätty satunnaiset palkkiot. Tuloksiin sisältyy tabualgoritmin antamien ratkaisujen vertailua Cplex-ohjelman laskemiin eksakteihin ratkaisuihin, ja lisäksi tabualgoritmin antamien ratkaisujen vertailua pelkän konstruktiovaiheen ratkaisuihin. Ratkaisuissa on oletettu

vierailtavien solmukohtien lukumäärä k tiedetyksi, ja sen pohjalta on laskettu myös optimin yläraja. Lähteen [5] tulokset koostuvat taulukoista, joista näkyy selkeästi että tabualgoritmi on antanut täsmälleen samat ratkaisut kuin mitä eksaktit ratkaisut ovat, silloin kun verkoston solmukohtien lukumäärät n saavat arvot n=10 ja n=20. Kun n=50, eksakteja ratkaisuja ei enää olla voitu laskea Cplexin laskentatehon tullessa vastaan, ja tässä vaiheessa tabualgoritmin ratkaisuja on verrattu vain LP-relaksaatiosta saatuun ratkaisuun, ja tulokseksi prosentuaaliselle erolle saatiin 13.99%. Odotetusti heuristinen tabualgoritmi käyttää kaikissa vaiheissa paljon vähemmän aikaa kuin eksakteja ratkaisuja etsivä Cplex-ohjelmisto. Lisäksi tuloksista ilmenee, että tabualgoritmi parantaa ratkaisua merkittävästi pelkkään konstruktiovaiheeseen nähden n:n kasvaessa suuremmaksi. Kun n on 10, parannusta konstruktiovaiheen löytämään ratkaisuun on tabualgoritmin käytön jälkeen 1.31%, mutta kun n on 500, on parannusta 16.02%. Kun n on välillä 10-500, tabualgoritmin antaman ratkaisun suhteellinen ero optimin ylärajaan on välillä 16.05-24.81%. Tabualgoritmin eduiksi voidaan sanoa palkkioihin pohjautuvassa minimilatenssiongelmassa sen ominaisuus löytää optimaaliset ratkaisut pienillä n:n arvoilla lyhyemmässä ajassa, kuin ei-heuristiset menetelmät tekevät. Suurilla n:n arvoilla tabualgoritmin käyttö on vielä hyödyllisempää, koska eksakteja ratkaisuja ei ole rajoitetun laskentatehon puitteissa voitu edes laskea. Lisäksi tabualgoritmin antamat ratkaisut ovat huomattavan paljon parempia kuin sen esiasteena toimivan konstruktiovaiheen ratkaisut, kun n saa suuria arvoja.

5. Yhteenveto

Minimilatenssiongelmaa käsittelevä kirjallisuus vaikuttaa vielä toistaiseksi olevan erittäin suppeaa verrattuna esimerkiksi sen yksinkertaisempaan versioon matkustavan kauppamiehen ongelmaan. Lisäksi monet algoritmit ovat vielä kehitysvaiheessa tai rajoittuvat erikoistilanteisiin ja yksittäistapauksiin. Lähitulevaisuudessa minimilatenssiongelmaa käsittelevä kirjallisuus tulee varmaankin olemaan laajempaa.

Koska matemaattinen malli eri tyyppisille minimilatenssiongelmille on usein NP-kova, se voidaan ratkaista vain pienissä määrätynlaisissa verkostoissa ja esimerkkitapauksissa, joita käytiin läpi eksakteja algoritmeja käsittelevissä kappaleissa. Sen takia ongelman ratkaisuun on tarpeellista käyttää heuristisia algoritmeja ja approksimaatioalgoritmeja, jotka vaativat vähemmän laskentatehoa, mutta voivat silti antaa riittävän hyviä likiarvoisia ratkaisuja. Tämä sama laskentatehoa koskeva ongelma pätee kuitenkin pienessä määrin myös heuristisiin algoritmeihin, koska niissäkin on rajoittavana tekijänä järkevän suuruinen laskentateho. On aina valittava jonkunlainen kompromissi laskenta-ajan ja algoritmin tehokkuuden väliltä. Esimerkiksi viimeisessä kappaleessa läpikäyty tabualgoritmi toimii vain rajoitetulla määrällä parametreja. Lisäksi kohennetun geneettisen algoritmin antama approksimaatiosuhdanne 1,9 on liian suuri moniin käytännön sovellutuksiin. Minimilatenssiongelman ratkaisualgoritmeja tullaan varmasti vastaisuudessa kehittämään lisää.

Vertailtaessa heuristisia algoritmeja keskenään esimerkkien 4.1-4.3 yksittäistapauksissa kaikkein tehokkain algoritmi oli hyvin yksinkertainen tabualgoritmi. Lisäksi kappaleessa 4.2.2 käsitellyn tabualgoritmin antamien tulosten suhteellinen ero optimin ylärajaan oli selkeästi pienempi kuin kappaleessa 4.1.1 käsitellyn geneettisen algoritmin antamien tulosten vastaava ero. Näitä tapauksia ei voi kuitenkaan verrata suoraan keskenään, koska minimilatenssiongelman muotoilut olivat erilaiset.

Kirjallisuuskatsauksen kirjoittaminen minimilatenssiongelmasta oli usealla eri tavalla opettavaista. Sen lisäksi että se perehdytti kirjoituksen aiheena olleeseen optimointiongelmaan, se edellytti myös ennen kaikkea tiedonlähteiden etsintää ja valintojen tekemistä sen suhteen, mitkä lähteet ja näkökulmat valitaan kirjoitettavaan työhön. Lisäksi luvun 4 testiesimerkit edellyttivät simulaatioiden koodaamista ja MATLAB-ohjelmiston käyttämiseen aiempaa parempaa perehtymistä. Voisi oikeastaan sanoa, että tätä työtä tehdessä tuli opeteltua ainakin yhtä paljon yleisluontoisiin kirjallisuustutkimuksen menetelmiin liittyvää asiaa, kuin varsinaista minimilatenssiongelman ratkomista. Tähän työhön käyttämäni aika- ja työmäärä oli jonkun verran suurempaa, kuin useimmiten vastaavaan opintopistemäärään käyttämäni työmäärä. Tämä johtuu luultavasti pääosin siitä, että tavallisesti kursseilla opiskeltava asia ja oppimateriaali on valmiiksi annettua, kun taas tätä työtä tehdessä aikaa kului huomattavasti sen miettimiseen, mihin suuntaan työtä lähtee viemään ja minkä lähteiden pohjalta. Kaiken kaikkiaan kuitenkin omatoimisuuden opetteleminen tuntui mielekkäältä.

Lähteet

[1] René Sitters, Minimum-Latency problem is NP-hard for weighted trees

[2] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, M. Sudan, The minimum latency problem, in: Proceedings of the twenty-sixth annual ACM symposium on the theory of computing (1994) pp. 163-171.

[3] Sarubbi J.F.M, Luna H.P.I, A new flow formulation for the minimum latency problem
[4] Ba Ha Bang, Nguyen Duc Nghia, Improved genetic algorithm for minimum latency problem
[5] Thijs Dewilde, Dirk Cattrysse, Sofie Coene, Frits C.R. Spieksma, Pieter Vansteenwegen, Heuristics for the Traveling Repairman Problem with Profits
[6] E. Minieka, The delivery man problem on a tree network
[7] Bang Ye Wu, Polynomial time algorithms for some minimum latency problems
[8] F. Afrati, S. Cosmadakis, C. Papadimitriou, G. Papageorgiou,N. Papakostantinou, The complexity of the traveling repairman problem
[9] E. Koutsoupias, C. Papadimitriou, M. Yannakakis, Searching a fixed graph
[10] E. Aarts, J.K. Lenstra (eds), Local search in combinatorial optimization, Wiley-interscience series in discrete mathematics and optimization, 1997.
[11] Aaron Archer, Asaf Levin, David Williamson, A faster, better approximation algorithm for the minimum latency problem
[12] M. Goemans & J. Kleinberg. An improved approximation ratio for the minimum latency problem
[13] N. Grag, A 3-approximation for the minimum tree spanning *k* vertices
[14] [2] S. Arora, G. Karakostas, Approximation schemes for minimum latency problems
[15] Ba Ha Bang, Nguyen Duc Nghia, Genetic algorithm for minimum latency problem
[16] C. Nilsson. Heuristics for the Traveling Repairman Problem
[17] Kamalika Chaudhuri, Brighten Goldfrey, Satish Rao, Kunai Talwar. Path, Trhee and mimimum latency tour
[18] M.Fischetti, G.Laporte, and S.Martelo. The deliveryman problem and cumulative methods. Operations research, 6:1055-1064, 1993
[19]C.Eijl. A polyhedral approach to the deliveryman problem. Memorandum COSOT 95, Eindhoven University of Technology, New York University 1995
[20]G. Reinelt, TSPLIB, *Institut für angewandte Mathematik, Universität Heidelberg (2001).* http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Liite A

MATLAB-koodi, jota käytettiin satunnaiskierroksen generoimiseen esimerkissä 4.1.

```
function [simulaatio,nodet,etaisyydet] = SK


%Nimetään verkoston solmukohdat A:sta F:ään.
verkosto=['A','B','C','D','E','F'];

%Merkitään kuljetun reitin solmukohdat muuttujaan nodet ja kuljetut
%etäisyydet muuttujaan etaisyydet. Reitti alkaa nodesta A.
nodet = ['A'];
etaisyydet = [];
%Aloitetaan solmukohdasta A.
sijainti='A';

%Katsotaan kolme reittivaihtoehtoa solmukohdasta 1, ja valitaan reitti
 %satunnaisesti.
    if sijainti=='A'
        x=rand(1);
        if x<1/3
            sijainti='B';
            etaisyydet=[etaisyydet 12];
            nodet=[nodet 'B'];
        end
        if x>1/3 && x<2/3
            sijainti='E';
            etaisyydet=[etaisyydet 9];
            nodet=[nodet 'E'];
        end
        if x>2/3
            sijainti='F';
            etaisyydet=[etaisyydet 16];
            nodet=[nodet 'F'];
        end
    end

%Aletaan muodostaa satunnaista reittiä verkostoon.
%Jatketaan verkostossa kulkemista niin kauan että kaikissa solmukohdissa on
%käyty ja ollaan tultu takaisin aloitussolmukohtaan.
while not(isequal(ismember(verkosto,nodet),[1 1 1 1 1 1]) && sijainti=='A')

    %Katsotaan kolme reittivaihtoehtoa solmukohdasta 1, ja valitaan reitti
    %satunnaisesti.
    if sijainti=='A'
        x=rand(1);
        if x<1/3
            sijainti='B';
            etaisyydet=[etaisyydet 12];
            nodet=[nodet 'B'];
        end
        if x>1/3 && x<2/3
            sijainti='E';
            etaisyydet=[etaisyydet 9];
            nodet=[nodet 'E'];
        end
```

```
    if x>2/3
        sijainti='F';
        etaisyydet=[etaisyydet 16];
        nodet=[nodet 'F'];
    end
end
%Katsotaan neljä reittivaihtoehtoa solmukohdasta 2, ja valitaan reitti
%satunnaisesti.
if sijainti=='B'
    x=rand(1);
    if x<1/4
        sijainti='A';
        etaisyydet=[etaisyydet 12];
        nodet=[nodet 'A'];
    end
    if x>1/4 && x<2/4
        sijainti='F';
        etaisyydet=[etaisyydet 15];
        nodet=[nodet 'F'];
    end
    if x>2/4 && x<3/4
        sijainti='C';
        etaisyydet=[etaisyydet 19];
        nodet=[nodet 'C'];
    end
    if x>3/4
        sijainti='D';
        etaisyydet=[etaisyydet 12];
        nodet=[nodet 'D'];
    end
end
%Katsotaan kolme reittivaihtoehtoa solmukohdasta 3, ja valitaan reitti
%satunnaisesti.
if sijainti=='C'
    x=rand(1);
    if x<1/3
        sijainti='B';
        etaisyydet=[etaisyydet 19];
        nodet=[nodet 'B'];
    end
    if x>1/3 && x<2/3
        sijainti='F';
        etaisyydet=[etaisyydet 17];
        nodet=[nodet 'F'];
    end
    if x>2/3
        sijainti='D';
        etaisyydet=[etaisyydet 21];
        nodet=[nodet 'D'];
    end
end
%Katsotaan neljä reittivaihtoehtoa solmukohdasta 4, ja valitaan reitti
%satunnaisesti.
if sijainti=='D'
    x=rand(1);
    if x<1/4
        sijainti='B';
        etaisyydet=[etaisyydet 12];
        nodet=[nodet 'B'];
    end
    if x>1/4 && x<2/4
```

```
        sijainti='C';
        etaisyydet=[etaisyydet 21];
        nodet=[nodet 'C'];
    end
    if x>2/4 && x<3/4
        sijainti='F';
        etaisyydet=[etaisyydet 16];
        nodet=[nodet 'F'];
    end
    if x>3/4
        sijainti='E';
        etaisyydet=[etaisyydet 10];
        nodet=[nodet 'E'];
    end
end
%Katsotaan kolme reittivaihtoehtoa solmukohdasta 5, ja valitaan reitti
%satunnaisesti.
if sijainti=='E'
    x=rand(1);
    if x<1/3
        sijainti='A';
        etaisyydet=[etaisyydet 9];
        nodet=[nodet 'A'];
    end
    if x>1/3 && x<2/3
        sijainti='F';
        etaisyydet=[etaisyydet 10];
        nodet=[nodet 'F'];
    end
    if x>2/3
        sijainti='D';
        etaisyydet=[etaisyydet 10];
        nodet=[nodet 'D'];
    end
end
%Katsotaan viisi reittivaihtoehtoa solmukohdasta 6, ja valitaan reitti
%satunnaisesti.
if sijainti=='F'
    x=rand(1);
    if x<1/5
        sijainti='A';
        etaisyydet=[etaisyydet 16];
        nodet=[nodet 'A'];
    end
    if x>1/5 && x<2/5
        sijainti='B';
        etaisyydet=[etaisyydet 15];
        nodet=[nodet 'B'];
    end
    if x>2/5 && x<3/5
        sijainti='C';
        etaisyydet=[etaisyydet 17];
        nodet=[nodet 'C'];
    end
    if x>3/5 && x<4/5
        sijainti='D';
        etaisyydet=[etaisyydet 16];
        nodet=[nodet 'D'];
    end
    if x>4/5
        sijainti='E';
```

```
            etaisyydet=[etaisyydet 10];
            nodet=[nodet 'E'];
        end
    end
end


%Määritellään reitin latenssi. Aluksi alustetaan muuttujaan T reitin pituus
%kuljettujen etäisyyksien lukumääränä.
T=size(etaisyydet);
S_ajat=[];
%Lasketaan saapumisajat kuhunkin solmukohtaan ja sijoitetaan ne vektoriin
S_ajat.
for i=1:T(2)

S_ajat(i)=sum(etaisyydet(1:i));

end

%Määritellään latenssi L eli saapumisaikojen summa
simulaatio=sum(S_ajat);
```

Liite B

MATLAB-koodi, jota käytettiin geneettisen algoritmin ajamiseen esimerkissä 4.2.

```
function [minimilatenssi,z] = GA

%luodaan aluksi tietty määrä satunnaiskierroksia
for i=1:15
    [A,B,C]=SK;
    latenssit(i)=A;
    reitit{i}=B;
    etaisyysvektorit{i}=C;
end

%asetetaan z kuvaamaan iteraatioiden määrää
z=1;
while(min(latenssit)>259)
    %etsitään minimilatenssin indeksi
    indeksi=find(latenssit==min(min(latenssit)));

    %katsotaan optimireitti ja etäisyydet
    optimireitti=reitit{indeksi(1)};

    %solmukohtien määrä optimireitillä
    S=size(reitit{indeksi(1)});

    optimietaisyydet=etaisyysvektorit{indeksi(1)};

    %pieninSK=min(latenssit)

    %mean(latenssit)

    %etsitään toisiksi pienin latenssi ja kyseisen reitin solmukohtien
    %määrä,
    templ=latenssit;
    templ(indeksi)=[];

    tempr=reitit;
    tempr(indeksi)=[];

    tempe=etaisyysvektorit;
    tempe(indeksi)=[];

    i2=find(templ==min(min(templ)));
    optimireitti2=tempr{i2(1)};
    S2=size(tempr{i2(1)});
    optimietaisyydet2=tempe{i2(1)};

    letaisyydet=0;
    lapsi=[];
    Llatenssi=0;

    %risteytetään pienilatenssisimmat yksilöt, mikäli ne risteävät 3:nnen
```

```
    %solmukohdan jälkeen. (1. tai 2. solmukohdassa risteävät reitit olisivat
    %samat)
    if S(2)<=S2(2)
        for i=3:S(2)
            if optimireitti(i)==optimireitti2(i) && length(optimireitti)<=i &&
length(optimireitti2)<=i
                lapsi=[optimireitti(1:i-1) optimireitti2(i:S2(2))];
                letaisyydet=[optimietaisyydet(1:i-1) optimietaisyydet2(i:end)];
            else
                eitehdamitaan=rand(1);
            end
        end
    else
        for i=3:S2(2)
            if optimireitti(i)==optimireitti2(i) && length(optimireitti)<=i &&
length(optimireitti2)<=i
                lapsi=[optimireitti(1:i-1) optimireitti2(i:S2(2))];
                letaisyydet=[optimietaisyydet(1:i-1) optimietaisyydet2(i:end)];
            else
                eitehdamitaan=rand(1);
            end
        end
    end

    %Määritellään lapsen latenssi. Aluksi alustetaan muuttujaan L reitin pituus
    %kuljettujen välimatkojen lukumääränä.
    L=size(letaisyydet);
    L_ajat=[];
    %Lasketaan saapumisajat kuhunkin solmukohtaan ja sijoitetaan ne vektoriin
L_ajat.
    for i=1:L(2)

    L_ajat(i)=sum(letaisyydet(1:i));

    end

    %Määritellään lapsen latenssi Llatenssi eli saapumisaikojen summa
    Llatenssi = sum(L_ajat);

    %uudistetaan populaatiota poistamalla 5 isolatenssisinta yksilöä ja
lisäämällä
    %sinne "lapsi" ja 4 satunnaiskierrosta. Jos "lasta" ei saada
    %olemassaolevista reiteistä, lisätään 5 satunnaiskierrosta.
    for i=1:5
        imax=find(latenssit==max(max(latenssit)));
        latenssit(imax)=[];
        reitit(imax)=[];
        etaisyysvektorit(imax)=[];
        i=i+1;
    end

    if Llatenssi>0
        latenssit=[latenssit Llatenssi];
        reitit=[reitit lapsi];
        etaisyysvektorit=[etaisyysvektorit letaisyydet];
        for i=1:4
            [D,F,G]=SK;
            latenssit(end+1)=D;
            reitit{end+1}=F;
            etaisyysvektorit{end+1}=G;
```

```
            i=i+1;
        end
    else
        for i=1:5
            [D,F,G]=SK;
            latenssit(end+1)=D;
            reitit{end+1}=F;
            etaisyysvektorit{end+1}=G;
            i=i+1;
        end
    end


    %merkitään iteraatioiden määrään yksi lisää
    z=z+1;
end

%etsitään pienin latenssi
minimilatenssi=min(latenssit);
```

Liite C

MATLAB-koodi, jota käytettiin tabualgoritmin ajamiseen esimerkissä 4.3.

```
function [simulaatio, reitti, etaisyydet] = tabu

%Nimetään verkoston solmukohdat A:sta F:ään.
verkosto=['A','B','C','D','E','F'];

%Merkitään kuljetun reitin solmukohdat muuttujaan nodet ja kuljetut
%etäisyydet muuttujaan etaisyydet. Reitti alkaa nodesta A.
nodet = ['A'];
etaisyydet = [];
%Aloitetaan solmukohdasta A.
sijainti='A';

%Muodostetaan tabulista, johon listataan viimeisimpänä vieraillut
%solmukodat. Aluksi siihen kuuluu vain lähtösolmukohta A. Toiseen
%solmukohtaan merkataan aluksi apumuuttuja Z, joka ei tarkoita mitään
%solmukohtaa.
tabulista = ['A', 'Z'];

%Katsotaan kolme reittivaihtoehtoa solmukohdasta 1, ja valitaan reitti
 %satunnaisesti.
    if sijainti=='A'
        x=rand(1);
        if x<1/3
            sijainti='B';
            etaisyydet=[etaisyydet 12];
            nodet=[nodet 'B'];
            tabulista=['B',tabulista];
        end
        if x>1/3 && x<2/3
            sijainti='E';
            etaisyydet=[etaisyydet 9];
            nodet=[nodet 'E'];
            tabulista=['E',tabulista];
        end
        if x>2/3
            sijainti='F';
            etaisyydet=[etaisyydet 16];
            nodet=[nodet 'F'];
            tabulista=['F',tabulista];
        end
    end


%Aletaan muodostaa satunnaista reittiä verkostoon lähtien solmukohdasta 1.
%Jatketaan verkostossa kulkemista niin kauan että kaikissa solmukohdissa on
%käyty ja ollaan tultu takaisin aloitussolmukohtaan. Mikäli solmukohta on
%tabulistassa toisena, siihen ei siirrytä, koska sieltä ollaan juuri
%tulossa, ja arvotaan reitti uudelleen.
while not(isequal(ismember(verkosto,nodet),[1 1 1 1 1 1]) && sijainti=='A')

    %Katsotaan kolme reittivaihtoehtoa solmukohdasta 1, ja valitaan reitti
```

```
%satunnaisesti.
if sijainti=='A'
    x=rand(1);
    if x<1/3
        if tabulista(2)~='B'
            sijainti='B';
            etaisyydet=[etaisyydet 12];
            nodet=[nodet 'B'];
            tabulista=['B',tabulista];
        else
            x=rand(1);
        end
    end
    if x>1/3 && x<2/3
        if tabulista(2)~='E'
            sijainti='E';
            etaisyydet=[etaisyydet 9];
            nodet=[nodet 'E'];
            tabulista=['E',tabulista];
        else
            x=rand(1);
        end
    end
    if x>2/3
        if tabulista(2)~='F'
            sijainti='F';
            etaisyydet=[etaisyydet 16];
            nodet=[nodet 'F'];
            tabulista=['F',tabulista];
        else
            x=rand(1);
        end
    end
end
%Katsotaan neljä reittivaihtoehtoa solmukohdasta 2, ja valitaan reitti
%satunnaisesti.
if sijainti=='B'
    x=rand(1);
    if x<1/4
        if tabulista(2)~='A'
            sijainti='A';
            etaisyydet=[etaisyydet 12];
            nodet=[nodet 'A'];
            tabulista=['A',tabulista];
        else
            x=rand(1);
        end
    end
    if x>1/4 && x<2/4
        if tabulista(2)~='F'
            sijainti='F';
            etaisyydet=[etaisyydet 15];
            nodet=[nodet 'F'];
            tabulista=['F',tabulista];
        else
            x=rand(1);
        end
    end
    if x>2/4 && x<3/4
        if tabulista(2)~='C'
            sijainti='C';
```

```
            etaisyydet=[etaisyydet 19];
            nodet=[nodet 'C'];
            tabulista=['C',tabulista];
        else
            x=rand(1);
        end
    end
    if x>3/4
        if tabulista(2)~='D'
            sijainti='D';
            etaisyydet=[etaisyydet 12];
            nodet=[nodet 'D'];
            tabulista=['D',tabulista];
        else
            x=rand(1);
        end
    end
end
%Katsotaan kolme reittivaihtoehtoa solmukohdasta 3, ja valitaan reitti
%satunnaisesti.
if sijainti=='C'
    x=rand(1);
    if x<1/3
        if tabulista(2)~='B'
            sijainti='B';
            etaisyydet=[etaisyydet 19];
            nodet=[nodet 'B'];
            tabulista=['B',tabulista];
        else
            x=rand(1);
        end
    end
    if x>1/3 && x<2/3
        if tabulista(2)~='F'
            sijainti='F';
            etaisyydet=[etaisyydet 17];
            nodet=[nodet 'F'];
            tabulista=['F',tabulista];
        else
            x=rand(1);
        end
    end
    if x>2/3
        if tabulista(2)~='D'
            sijainti='D';
            etaisyydet=[etaisyydet 21];
            nodet=[nodet 'D'];
            tabulista=['D',tabulista];
        else
            x=rand(1);
        end
    end
end
%Katsotaan neljä reittivaihtoehtoa solmukohdasta 4, ja valitaan reitti
%satunnaisesti.
if sijainti=='D'
    x=rand(1);
    if x<1/4
        if tabulista(2)~='B'
            sijainti='B';
            etaisyydet=[etaisyydet 12];
```

```
            nodet=[nodet 'B'];
            tabulista=['B',tabulista];
        else
            x=rand(1);
        end
    end
    if x>1/4 && x<2/4
        if tabulista(2)~='C'
            sijainti='C';
            etaisyydet=[etaisyydet 21];
            nodet=[nodet 'C'];
            tabulista=['C',tabulista];
        else
            x=rand(1);
        end
    end
    if x>2/4 && x<3/4
        if tabulista(2)~='F'
            sijainti='F';
            etaisyydet=[etaisyydet 16];
            nodet=[nodet 'F'];
            tabulista=['F',tabulista];
        else
            x=rand(1);
        end
    end
    if x>3/4
        if tabulista(2)~='E'
            sijainti='E';
            etaisyydet=[etaisyydet 10];
            nodet=[nodet 'E'];
            tabulista=['E',tabulista];
        else
            x=rand(1);
        end
    end
end
%Katsotaan kolme reittivaihtoehtoa solmukohdasta 5, ja valitaan reitti
%satunnaisesti.
if sijainti=='E'
    x=rand(1);
    if x<1/3
        if tabulista(2)~='A'
            sijainti='A';
            etaisyydet=[etaisyydet 9];
            nodet=[nodet 'A'];
            tabulista=['A',tabulista];
        else
            x=rand(1);
        end
    end
    if x>1/3 && x<2/3
        if tabulista(2)~='F'
            sijainti='F';
            etaisyydet=[etaisyydet 10];
            nodet=[nodet 'F'];
            tabulista=['F',tabulista];
        else
            x=rand(1);
        end
    end
```

```
    if x>2/3
        if tabulista(2)~='D'
            sijainti='D';
            etaisyydet=[etaisyydet 10];
            nodet=[nodet 'D'];
            tabulista=['D',tabulista];
        else
            x=rand(1);
        end
    end
end
%Katsotaan viisi reittivaihtoehtoa solmukohdasta 6, ja valitaan reitti
%satunnaisesti.
if sijainti=='F'
    x=rand(1);
    if x<1/5
        if tabulista(2)~='A'
            sijainti='A';
            etaisyydet=[etaisyydet 16];
            nodet=[nodet 'A'];
            tabulista=['A',tabulista];
        else
            x=rand(1);
        end
    end
    if x>1/5 && x<2/5
        if tabulista(2)~='B'
            sijainti='B';
            etaisyydet=[etaisyydet 15];
            nodet=[nodet 'B'];
            tabulista=['B',tabulista];
        else
            x=rand(1);
        end
    end
    if x>2/5 && x<3/5
        if tabulista(2)~='C'
            sijainti='C';
            etaisyydet=[etaisyydet 17];
            nodet=[nodet 'C'];
            tabulista=['C',tabulista];
        else
            x=rand(1);
        end
    end
    if x>3/5 && x<4/5
        if tabulista(2)~='D'
            sijainti='D';
            etaisyydet=[etaisyydet 16];
            nodet=[nodet 'D'];
            tabulista=['D',tabulista];
        else
            x=rand(1);
        end
    end
    if x>4/5
        if tabulista(2)~='E'
            sijainti='E';
            etaisyydet=[etaisyydet 10];
            nodet=[nodet 'E'];
            tabulista=['E',tabulista];
```

```
            else
                x=rand(1);
            end
        end
    end
end


%Määritellään reitin latenssi. Aluksi alustetaan muuttujaan T reitin pituus
%solmukohtien lukumääränä.
T=size(etaisyydet);
S_ajat=[];
%Lasketaan saapumisajat kuhunkin solmukohtaan ja sijoitetaan ne vektoriin
S_ajat.
for i=1:T(2)

S_ajat(i)=sum(etaisyydet(1:i));

end

%Määritellään latenssi L eli saapumisaikojen summa
simulaatio=sum(S_ajat);
reitti=nodet;
```