PARVIÄLYKKYYS JA MUURAHAISPOHJAISET ALGORITMIT

Mikko Suominen

Pro gradu -tutkielma

ITÄ-SUOMEN YLIOPISTO

Tietojenkäsittelytieteen laitos

Tietojenkäsittelytiede

Heinäkuu 2013

ITÄ-SUOMEN YLIOPISTO, Luonnontieteiden ja metsätieteiden tiedekunta, Kuopio
Tietojenkäsittelytieteen laitos
Tietojenkäsittelytiede

Opiskelija, Suominen Mikko: Parviälykkyys ja muurahaispohjaiset algoritmit
Pro gradu -tutkielma, 61 s., 6 liitettä (34 s.)
Pro gradu -tutkielman ohjaaja: FT Maija Marttila-Kontio
Heinäkuu 2013

Tutkielmassa parvikäyttäytyminen ja parviälykkyys esitellään esimerkkien avulla, pääasiassa mehiläisten ja muurahaisten avulla. Tutkielmassa esitellään myös kaksi muurahaispohjaista parviälykkyysalgoritmia ja verrataan algoritmien suoritusnopeutta keskenään kolmella Kauppamatkustajan ongelman ilmentymällä. Tutkielmassa käytetyt muurahaispohjaiset parviälykkyysalgoritmit ovat Ant System ja Ant Colony System.

Tutkielmassa on verrattu Ant System ja Ant Colony System -algoritmien suoritusaikoja keskenään riippumattomien otosten t-testillä ja Mann-Whitneyn U-testillä Kauppamatkustajan ongelman ilmentymissä rd100, berlin52 sekä pr152. Riippumattomien otosten t-testillä ja Mann-Whitneyn U-testillä on haluttu osoittaa algoritmien suoritusaikojen tilastolliset erot. Ant Colony System löysi lyhimmän reitin kaikista kolmesta ilmentymistä selvästi nopeammin kuin Ant System, joten erot olivat tilastollisesti merkitseviä.

Tutkielman pohjalta jatkotutkimuksena suositellaan vertaamaan algoritmien löytämien lyhimpien reittien pituuksia keskenään. Tutkielman kokeellinen osuus osoitti, että Ant System -algoritmi on hitaampi kuin Ant Colony System -algoritmi, mutta Ant System -algoritmilla löydettiin muutamia kertoja lyhyempi reitti kuin Ant Colony System -algoritmilla kaikista kolmesta Kauppamatkustajan ongelman ilmentymistä.

Avainsanat: parvikäyttäytyminen, parviälykkyys, muurahaispohjaiset algoritmit, Ant System, Ant Colony System

ACM-luokat (ACM Computing Classification System, 2013 version): Machine learning, Multi-agent reinforcement learning, Paths and connectivity problems, Probabilistic reasoning

UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry,
Kuopio
School of Computing
Computer Science

Student, Suominen Mikko: Swarm Intelligence and Ant-Based Algorithms
Master's Thesis, 61 p., 6 appendixes (34 p.)
Supervisor of the Master's Thesis: PhD Maija Marttila-Kontio
July 2013

In this thesis swarm behavior and swarm intelligence is introduced by examples of nature, mainly with examples of bees and ants. Algorithm´s running times are compared with each other with three Travelling Salesman Problem instances. The ant-based swarm intelligence algorithms used in this thesis are Ant System and Ant Colony System.

Running times of the algorithms with the instances of Travelling Salesman Problem rd100, berlin42 and pr152, are compared with each other with an independent sample t-test and with Mann-Whitney U test. With the tests the statistical difference in algorithms´ running times is proved. Ant Colony System found the shortest path from all instances faster than Ant System and the differences between running times are statistically significant.

Based on this thesis and for further studies, the shortest paths found with Ant System and Ant Colony System algorithms are recommended to be compared with each other. Based on the results of this thesis, Ant System algorithm occasionally managed to find a shorter path than Ant Colony System from all three instances of Travelling Salesman Problem. The running time of Ant System -algortihm was slower than the running time of Ant Colony System -algoritm in every scenario with the tested Travelling Salesman problem instances.

Keywords: swarm behaviour, swarm intelligence, ant-based algorithms, Ant System, Ant Colony System

CR Categories (ACM Computing Classification System, 2013 version): Machine learning, Multi-agent reinforcement learning, Paths and connectivity problems, Probabilistic reasoning

Esipuhe

Tämä tutkielma on tehty Itä-Suomen yliopiston Kuopion kampuksen Tietojenkäsittelytieteen laitokselle aikavälillä marraskuu 2011 ja heinäkuu 2013.

Haluan erityisesti kiittää tämän tutkielman ohjaaja Maija Marttila-Kontiota kärsivällisestä ja neuvovasta ohjauksesta. Tutkielman teko venyi reilusti pidemmäksi alkuperäisestä suunnitelmasta muun muassa terveydellisistä syistä johtuen, joten ohjaajalta on vaadittu kärsivällisyyttä tutkielman tekoprosessin ajan.

Viimein kaikkien vaiheiden jälkeen olen saanut tutkielman valmiiksi ja yksien kansien sisään ja voin huokaista helpotuksesta sekä keskittyä elämässä muihin asioihin. Kiitos kannustuksesta ja tukemisesta myös vanhemmilleni sekä veljilleni, joilta olen saanut myös neuvoja tämän tutkielman tekoprosessin aikana.

Lyhenteet ja termit

ADAPTIIVINEN

Muutoksiin sopeutuva

DYNAAMINEN JÄRJESTYS

Olosuhteiden mukaan liikkuva ja mukautuva järjestys

EI-MINIMAALINEN REITITYSALGORITMI

Reititysalgoritmi, jonka tehtävänä on etsiä reitti verkosta jollain muulla heuristiikalla kuin lyhimmällä polun heuristiikalla

FEROMONI

Muurahaisten levittämä kemikaali, jolla houkutellaan muita muurahaisia hajun avulla

HAIHTUMISKERROIN

Numeerinen arvo väliltä [0,1), minkä mukaan feromonit haihtuvat kaarilta

HAMILTONIN POLKU

Verkosta löytyvä polku, jossa jokaisessa verkon solmussa käydään vain kerran

HEURISTIIKKA

Kokemukseen perustuva oppiminen

HIUKKASPARVIOPTIMOINTI

Parviälykkyyden kehittynein sovellus, jossa käytetään apuna liikkuvia partikkeleita

KAARI

Verkon osa, joka yhdistää solmut keskenään

KAUPPAMATKUSTAJAN ONGELMA

Lyhimmän polun etsintäongelma, jossa kauppamatkustajan tehtävänä on vierailla jokaisessa kaupungissa vain kerran ja palata lähtöpisteeseen

NEGATIIVINEN PALAUTE

Parviälykkyydessä esiintyvä ilmiö, jolla pyritään estämään muutosta

NP-VAIKEA PARADIGMA

Ongelma, johon ei ole löydetty yleispätevää ratkaisua

NOLLAHYPOTEESI

Tilastollisen päätöksenteon oletus, jonka oikeellisuutta testataan tilastollisella testillä

MANET

Matkapuhelinten AdHoc -verkko eli langattoman kommunikoinnin matkapuhelinverkko

MERKITSEVYYSTASO

Tilastotieteen termi, jolla kuvataan todennäköisyyttä hylätä oikeassa oleva nollahypoteesi

MINIMAALINEN REITITYSALGORITMI

Reititysalgoritmi, jonka tehtävänä on etsiä lyhin reitti verkosta

MUURAHAISKOLONNAOPTIMOINTI

Tietojenkäsittelytieteessä käytetty termi muurahaisten lyhimmän reitin etsinnässä pesän ja ruoan välillä

MUURAHAISALGORITMI

Muurahaisten käyttäytymisen tutkimisen tuloksena syntynyt algoritmi, jonka tehtävänä on etsiä lyhintä polkua verkosta

OTOSKOKO

Tässä tutkielmassa: Tilastollisen kokeen suorituskertojen määrä samalle Kauppamatkustajan ilmentymälle

PARTIKKELI

Parvessa tai verkossa itsenäisesti ryhmän osana liikkuva agentti, muurahainen tai yksilö

PARVIÄLYKKYYS

Tutkimusalue, jossa tutkitaan partikkelien itseorganisoituvaa kollektiivista älykkyyttä

POSITIIVINEN PALAUTE

Parviälykkyydessä esiintyvä ilmiö, jonka pyrkimyksenä on vahvistaa muutosta

PROAKTIIVINEN

Ennakoiva

REAKTIIVINEN

Herkästi ulkoisiin ärsykkeisiin reagoiva

REYNOLDSIN SÄÄNNÖT

Parvikäyttäytymisen kolme sääntöä: välttäminen, suuntaus ja vetovoima. Säännöt pätevät kaikkiin parviin

STAATTINEN

Muuttumaton, ulkoisiin muutoksiin reagoimaton

SOLMU

Verkon osa, jolla yhdistetään verkon kaaret keskenään

TOPOLOGIA

Perusrakenne. Tässä tutkielmassa tarkoitetaan verkon perusrakennetta

USEIDEN EDESTAKAISTEN REITTIEN REITITYS

Multiple Round Trip Routing -algoritmi, joka on kehitetty datapakettien edestakaiseen reititykseen

VASTAHYPOTEESTI

Nollahypoteesin vastahypoteesi, jonka oikeellisuutta testataan tilastollisella testillä

VERKKO

Reititysongelmien kuvantamistapa ja verkko koostuu solmuista ja kaarista

VÄHITEN MERKITSEVÄ BITTI

Least Significant Bit. Luvun binääriesityksessä viimeinen eli oikeanpuoleisin bitti

Sisällysluettelo

Liitteet

1 Johdanto

Tässä tutkielmassa tutustutaan parviälykkyyteen ja kahteen muurahaispohjaiseen parviälykkyysalgoritmiin Ant System ja Ant Colony System. Parviälykkyyden tutkimisen inspiraationa on toiminut eläinkunta ja pääasiassa kulkusirkkojen, mehiläisten ja muurahaisten parvikäyttäytyminen. Tutkielmassa parviälykkyyttä tarkastellaan kulkusirkkojen, mehiläisten ja muurahaisten parvikäyttäytymisen avulla. Parviälykkyydessä on kysymys yksilöiden yhteistoiminnasta ongelman ratkaisemiseksi. Esimerkiksi muurahaiset käyttävät parviälykkyyttä ruoan etsimiseen ja mehiläiset uusien pesäpaikkojen etsimiseen. Toimintoihin käytetään yksinkertaisia sääntöjä, joiden avulla parvikäyttäytyminen ja sitä kautta parviälykkyys on mahdollista, mutta mistä säännöt johtavat juurensa ja miten ne syntyvät? Tässä tutkielmassa vastataan muun muassa tuohon kysymykseen.

Termi parviäly esiteltiin ensimmäisen kerran vuonna 1989 (Beni & Wang, 1989), jolloin parviälykkyyden tutkimus lähti todistetusti liikkeelle. Mehiläiset ja muurahaiset ovat olleet pääasiallinen innostuksen lähde parviälykkyyden tutkimuksessa (Fisher, 2009; Fleischer, 2003). Muurahaisten toimintaa on tutkittu eniten ja muurahaisten tutkimisen avulla on kehitetty parviälykkyysalgoritmeja, kuten Ant System ja Ant Colony System -algoritmit.

Eräitä tunnetuimpia parviälykkyyden tutkijoita ovat Eric Bonabeau, Marco Dorigo sekä Guy Theraulaz, jotka ovat myös kirjoittaneet kirjan Swarm Intelligence: From Natural to Artificial Systems (Bonabeau et al., 1999), johon tässä tutkielmassa viitataan useaan otteeseen. Lisäksi tutkielmassa käytetään muun muassa tutkijoiden Luca Maria Gambardellan, Vittorio Maniezzon ja Alberto Colornin tutkimustuloksia parviälykkyydestä (Di Caro et al., 1998; Dorigo et al., 1997).

Parviälykkyys ja muurahaispohjaiset algoritmit valittiin tutkielman aiheeksi, koska parviälykkyyttä voidaan implementoida jokapäiväiseen ongelmanratkaisuun, laskennallisista ongelmista liike-elämään (Bonabeau & Meyer, 2001). Tutkielmassa perehdytään parviälykkyyteen parvikäyttäytymisen kautta ja analysoidaan Ant System ja Ant Colony System -algoritmien suoritustehokkuutta Kauppamatkustajan ongelman ratkaisemisessa. Kauppamatkustajan ongelma on NP-täydellinen ongelma

(Cook, 2007), joten siihen ei ole löytynyt yleispätevää ratkaisua. Parviälykkyyden sovellukset, kuten Ant System ja Ant Colony System -algoritmit tarjoavat uusia ratkaisumahdollisuuksia Kauppamatkustajan ongelmaan. Kauppamatkustajan ongelmaa pidetään benchmarkina laskennallisessa optimoinnissa (Bonabeau et al., 1999), joten tässä tutkielmassa esiteltyjen algoritmien suoritustehokkuuden testaus Kauppamatkustajan ongelman ilmentymillä oli luonnollinen vaihtoehto.

Tutkielman kokeellisessa osuudessa verrataan Ant System ja Ant Colony System -algoritmien suoritusaikoja ja lyhimpiä löydettyjä reittejä Kauppamatkustajan ongelman ilmentymissä keskenään. Suoritusaikoja verrataan keskenään tilastollisin menetelmin. Saatujen tulosten avulla on haluttu osoittaa, että Ant Colony System -algoritmi on kehittyneempi versio Ant System -algoritmista sekä että muurahaispohjaiset parviälykkyysalgoritmit Ant System ja Ant Colony System ovat käyttökelpoisia algoritmeja lyhimmän reitin etsinnässä Kauppamatkustajan ongelman ilmentymissä.

Tutkielmassa johdatellaan lukijaa aiheeseen lukujen 2 ja 3 avulla. Luvussa 2 kerrotaan parvikäyttäytymisestä ja sen yhteydestä luontoon. Luvussa 3 keskitytään parviälykkyyteen, joka liitetään läheisesti parvikäyttäytymiseen. Luvussa keskitytään enemmän parviälykkyyden hyötyihin ja muurahaisiin, koska tutkielman aiheena ovat parviälykkyys ja muurahaispohjaiset algoritmit. Luvussa 4 esitellään Kauppamatkustajan ongelma sekä Ant System että Ant Colony System -algoritmit matemaattisten kaavojen tarkkuudella. Luvussa esitellään myös Kauppamatkustajan ongelman yksi käytännön sovellus sekä Ant Colony System -algoritmin parantamiseksi seitsemän eri viestintästrategiaa. Luvussa 5 Ant System ja Ant Colony System -algoritmeja testataan kauppamatkustajan ongelman ilmentymien avulla sekä esitetään ja analysoidaan saadut tulokset. Luvussa 6 on tutkielman yhteenveto ja pohdinta. Pohdinnassa esitellään tutkielman pohjalta tehdyt johtopäätökset ja jatkotutkimusideat.

2 PARVIKÄYTTÄYTYMINEN

Tässä luvussa esitellään parvikäyttäytymistä. Esimerkkien avulla huomataan, että parvikäyttäytyminen on osa ihmisten ja eläinten jokapäiväistä elämää. Lisäksi on havaittavissa, että parvikäyttäytymisen ja parviälykkyyden välillä ei ole suurta eroa ja parvikäyttäytymistä on ymmärrettävä, jotta voidaan ymmärtää parviälykkyyttä.

Kappaleessa 2.1 kuvataan parviälykkyyttä ympärillämme. Kappaleessa 2.2 kuvataan parvikäyttäytymiselle tyypillisiä näkymättömiä johtajia, joiden passiivisella johtamisella parvet liikkuvat. Kappaleessa 2.3 perehdytään kulkusirkkoihin, joiden tutkiminen on auttanut ymmärtämään parvikäyttäytymistä. Samassa kappaleessa esitellään myös Reynoldsin säännöt, joita jokainen parven yksilö noudattaa huomaamattaan.

2.1 Parvikäyttäytyminen ympärillämme

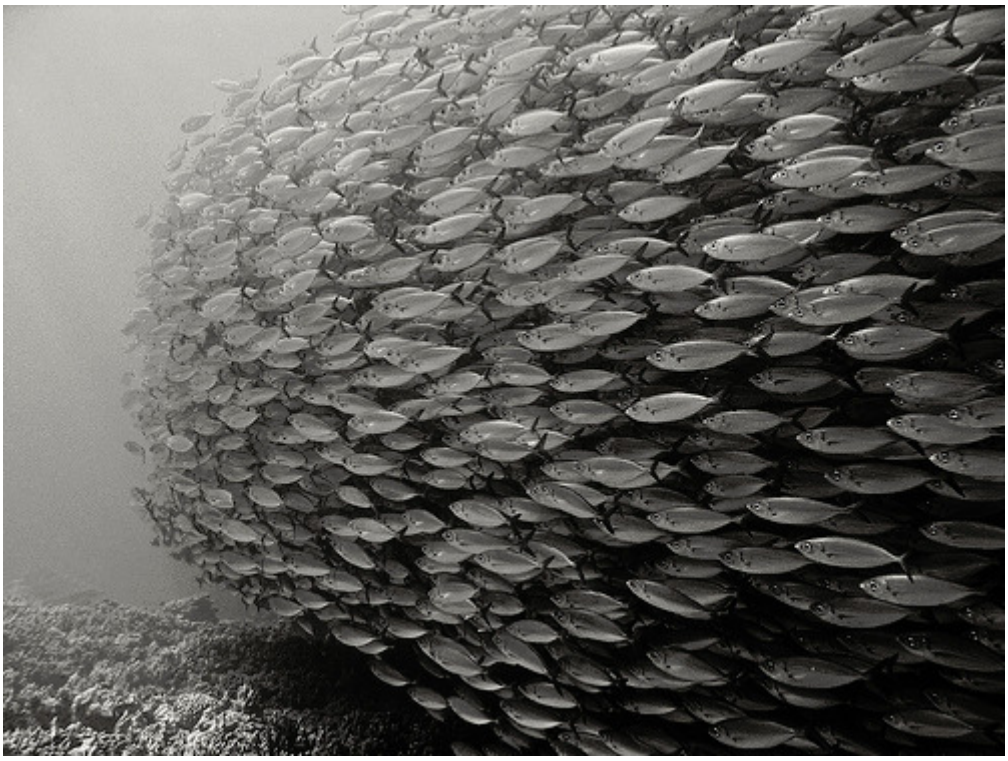
Parvikäyttäytyminen on eläinten käyttäytymispiirre, jossa eläimet hakeutuvat toisten samanlaisten yksilöiden läheisyyteen (Wootton, 1998). Parvikäyttäytyminen on ominaista kaikille suurina joukkoina esiintyville eläimille ja ihmisille.

Tutkimus (Fisher, 2009) on osoittanut, että parvikäyttäytyminen ei vaadi monimutkaisia käyttäytymismalleja, vaan yksinkertaisia vuorovaikutussääntöjä vierekkäisten yksilöiden kanssa. Esimerkiksi jalkapallopelissä ihmiset tekevät aaltoja yleisössä. Jokaisen yksilön kohdalla tehdään vuorollaan samat liikkeet, eli noustaan ylös, nostetaan kädet ilmaan ja lasketaan kädet alas. Seuraavana vuorossa olevat henkilöt tekevät edellisen esimerkin mukaan liikkeensä. Tällainen informaation välitys yksinkertaisilla menettelytavoilla on parvikäyttäytymisen perusperiaatteita.

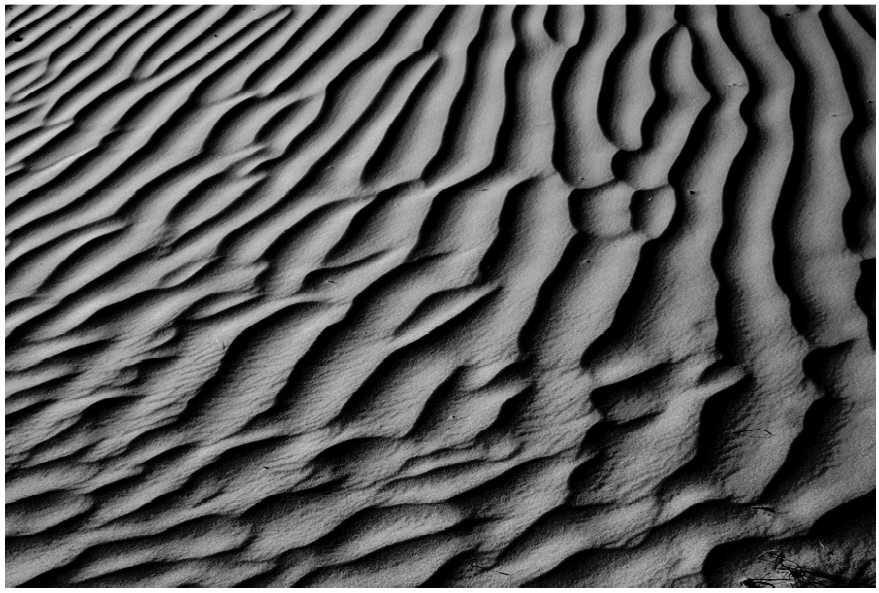
Eläimet, varsinkin hyönteiset ja ihmiset, osaavat liikkua sulavasti suurina joukkoina törmäilemättä toisiinsa säilyttäen tiiviin parvimaisen rakenteensa. Eläinten parvikäyttäytymistä on tutkittu paljon ja muun muassa muurahaisten käyttäytymisen tutkimisen (Dorigo et al., 1999) kautta on kehitetty parviälykkyyden sovelluksia

Parvikäyttäytymistä on kaikkialla elottomassa ja elollisessa luonnossa ja useat luontokappaleet ovat jatkuvasti tekemisissä parvikäyttäytymisen kanssa.

Noudattamalla yksinkertaisia sääntöjä luontokappaleet muodostavat monimutkaisia rakenteita ja kaavoja. Elottomat rakenteet muodostavat myös monimutkaisia rakenteita. Atomien molekyylit esimerkiksi muodostavat kiteitä ja hiekkadyynit muodostuvat tuulen puhaltaessa aavikon yli. Molemmissa tapahtumissa noudatetaan yksinkertaisia sääntöjä. Hiekkadyynit muodostuvat maan vetovoiman, tuulen ja viereisten hiekanjyvien kitkan vaikutuksesta. Atomit sekä molekyylit taas vetävät tai hylkivät toisiaan magneettikenttiensä mukaan muodostaen suuriakin kokonaisuuksia vetovoimiensa ansiosta. Esimerkki parvikäyttäytymisestä on esitetty kuvassa 1, jossa kalaparven kalat ovat lähellä toisiaan kuitenkaan törmäilemättä toisiinsa. Tiiviinä parvena kaloilla on paremmat mahdollisuudet välttyä saalistajilta. Kuvassa 2 on kuva aavikosta. Kuvalle on muodostunut säännöllinen kuvio tuulen ja hiekanjyvien kitkan vaikutuksesta.



Kuva 1: Kalaparvi, jonka yksilöt noudattavat vuorovaikutussääntöjä. Sääntöjä noudattamalla kalat pysyvät tiiviinä parvena. (Dunleavy, 2011.)



Kuva 2: Aavikolle muodostuneita kuvioita tuulen ja kitkan vaikutuksesta. (Yapp, 2005.)

Myös ihmiset ja eläimet hylkivät sekä vetävät toisiaan puoleensa vaikkakaan ei yhtä selkeästi kuin atomit ja molekyylit. Nämä ihmis- ja eläinjoukot vaikuttavat ulkopuolisen silmään olevan epäjärjestyksessä eli ulkoapäin tarkasteltuna joukot vaikuttavat olevan täydellisen epäjärjestyksen vallassa, mutta samalla kaikki ovat viereisiin yksilöihin nähden järjestyksessä ja noudattavat tiettyjä yksinkertaisia sääntöjä. Säännöt määrittelevät kokonaisia yhteisöjä, eikä niinkään vain sen yksittäisiä jäseniä. Tällaista järjestystä kutsutaan *dynaamiseksi järjestykseksi*, koska parven jäsenten väliset säännöt muodostavat dynaamisia, eli ajan mukaan muuttuvia kaavoja suurten kokonaisuuksien välille. (Fisher, 2009.)

Dynaamista järjestystä on kahdenlaista: sellaista, millä ei päädytä mihinkään päämäärään vaan kuljetaan ympäriinsä ikään kuin yrittäen löytää järjestys. Toinen dynaamisen järjestyksen muoto on olosuhteiden mukaan muuttuva järjestys. Tällaisesta järjestyksestä on hyvänä esimerkkinä kalaparvet, jotka mukautuvat kohdatessaan saalistajan ja palaavat sen jälkeen alkuperäiseen muotoonsa. Tällaiset parvet ovat dynaamisia parvia, koska ne pystyvät mukautumaan olosuhteiden muutoksiin. (Fisher, 2009.)

Dynaamiset parvet eivät tarvitse yksittäistä johtajaa, jotta muutokset tapahtuvat. Hyvänä esimerkkinä on yleisön yhtyminen yksittäisten ihmisten taputuksiin (Fisher, 2009). Aluksi yleisöstä alkavat taputtamaan muutamat ihmiset sieltä täältä ja

vähitellen taputuksiin on yhtynyt enemmän ja enemmän ihmisiä. Hetken kuluttua kaikki yleisöstä taputtavat synkronoidusti. Yleisössä kaikki toimivat itsensä johtajina eli niin sanottuina *näkymättöminä johtajina*. Aihetta käsitellään tarkemmin seuraavassa kappaleessa.

2.2 Näkymättömät johtajat

Näkymättömät johtajat (Fisher, 1999) ovat parvessa näkymättömiksi tekeytyviä suunnannäyttäjiä. Näkymätön johtaja on parhaimmillaan, kun sen olemassaoloa ei huomata. Näkymättömät johtajat pysyvät näkymättömissä parven muille jäsenille, koska parven käyttäytymissäännöt pitävät huolen, että parven jäsenet pysyvät parvessa. Partikkelien eksyessä reitiltä ne palaavat vähitellen takaisin parveen, koska niihin kohdistuu *negatiivinen palaute*, joka pyrkii korjaamaan virheitä. Negatiivinen palaute esitellään tarkemmin luvussa 3.

Ihmisten käyttäytymistä, johtajuutta ja päätöksentekoa tutkivassa tutkimuksessa (Dyer et al., 2009) henkilöt kävelivät tasaisella nopeudella huoneessa, jonka seinillä oli kirjaimia A:sta J:hin. Heidän piti kävellä noin kädenmitan päässä vähintään yhdestä henkilöstä ja pysähtyä, kun heitä käskettiin. Kenelläkään ei ollut omaa määränpäätään ja kaikkien oli pysyttävä ryhmässä, kuten mehiläiset parvessa. Muutamille henkilöille oli annettu ohjeeksi mennä tietyn kirjaimen kohdalle, mutta heidänkin piti kulkea koko ajan kädenmatkan päässä vähintään yhdestä henkilöstä. Ihmisten käskettyä pysähtyä suurin osa oli saman kirjaimen kohdalla, koska heidät oli johdatettu sinne. Muutamat henkilöt, joille oli annettu ohjeeksi mennä tietyn kirjaimen kohdalle toimivat johtajina. Samalla tavalla mehiläiset pystyvät lentämään parvena kohteeseensa vaikka suurin osa ei kohdetta tiedäkään.

Couzilin ja hänen kollegoidensa (Couzil et al., 2005) tekemä tietokonesimulaatio osoitti, että mitä isommasta parvesta oli kysymys, sitä vähemmän yksilöitä tarvittiin johdattamaan parvea. Edellisessä esimerkissä informoituja henkilöitä oli vain viisi prosenttia koko parvesta eli kymmenen henkilöä kahdestasadasta. Tietokonesimulaatiot osoittavat, että nämä viisi prosenttia informoiduista ihmisistä pystyvät johdattamaan muut määränpäähän yhdeksänkymmenen prosentin tarkkuudella (Fisher, 1999).

Muutamat informoidut yksilöt parvessa parantavat parven toimintaa huomattavasti, mutta ilman informoituja yksilöitä parvi reagoi vain ulkoisiin ärsykkeisiin. Esimerkiksi kalaparvet reagoivat, kun ne kohtaavat saalistajan ja kulkusirkat reagoivat tuuleen lentämällä myötätuulen suuntaisesti. Näissä esimerkeissä saalistaja sai kalaparven käyttäytymään tietyllä tavalla ja tuuli sai kulkusirkat käyttäytymään tietyllä tavalla. Parvikäyttäytymisellä pidetään parvi kasassa, mutta ilman päämäärää tai tavoitetta parven on mahdotonta olla *proaktiivinen*, eli ennakoida toimintaansa esimerkiksi saalistajien varalta.

2.3 Kulkusirkat ja Reynoldsin säännöt

Kulkusirkat ovat heinäsirkkoja, jotka muodostavat suuria yhtenäisiä parvia. Yksittäiset kulkusirkat vaeltelevat päämäärättömästi ympäriinsä, mutta tarpeeksi suuret kulkusirkkapopulaatiot käyttäytyvät kuin sotilaskolonna. Muiden sirkkojen välitön läheisyys saa kaikki sirkat kulkemaan joukkona eteenpäin etsien ruokaa ja syöden reitiltään kaiken mahdollisen ravinnoksi kelpaavan.

Kulkusirkkojen parvikäyttäytymisen tutkiminen on antanut arvokasta tietoa siitä, miten eläinparvet hyönteisistä ihmisiin liikkuvat, sekä siitä, minkälaisia sääntöjä parvet noudattavat. Kulkusirkat pysyvät parven sisällä organisoidussa muodostelmassaan välttääkseen tulemasta syödyksi takana kulkevan sirkan toimesta. Toinen, parven koossa pitävä voima on *positiivinen palaute*, eli muutaman kulkusirkan kulkeminen yhteen suuntaan saa muutkin kulkemaan samaan suuntaan[1]. Positiivinen palaute esitellään tarkemmin luvussa 3. (Fisher, 1999.)

Craig Reynolds kehitti (1995) parvisimulaation vuonna 1986 käyttäen lintujen sijaan pieniä kolmioita (*boids*). Boidit käyttäytyvät samaan tapaan, kuin lentävät linnut ja yksittäisten lintujen kadottaessa parvensa linnut löytävät takaisin muiden joukkoon hetken harhailtuaan päämäärättömästi ilmassa.

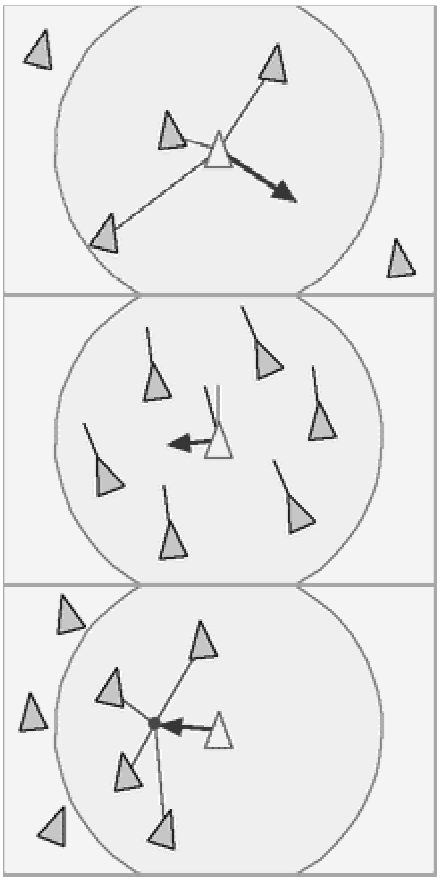
---

[1] Tällaisen monimutkaisen kollektiivisen käyttäytymisen parempaan ymmärtämiseen tarvitaan kuitenkin avuksi tietokonesimulaatioita (Fischer, 2009).

Reynoldsin kehittämät säännöt selittävät parvikäyttäytymistä, koska kaikki parven yksilöt noudattavat ko. sääntöjä. Säännön pätevät myös kaikkiin parviin eläinkunnassa sekä kaikkiin tietokonemalleihin, jotka simuloivat parvikäyttäytymistä. Tietokonesimulaation avulla on saatu selville, että myös kulkusirkat noudattavat kolmea *Reynoldsin sääntöä*. Reynolds havaitsi, että simulaation toteuttamiseen ei vaadita monimutkaista ohjelmointia, vaan boidit noudattavat kolmea yksinkertaista sääntöä. Säännöt ovat havainnollistettu kuvassa 3.

Määritelmä 1. Reynoldsin säännöt parvikäyttäytymisestä (Reynolds, 1995):

1. *Välttäminen* (*Avoidance*): Vältä törmäämästä muihin partikkeleihin
2. *Suuntaus* (*Alignment*): Liiku suunnilleen samaan suuntaan, kuin vieressä olevat partikkelit liikkuvat
3. *Vetovoima* (*Attraction*): Pidä etäisyys viereisiin partikkeleihin suurin piirtein samana



Kuva 3: Craig Reynoldsin lintuparvisimulaation kolme sääntöä. Ylhäällä sääntö 1. välttäminen, keskellä sääntö 2. suuntaus ja alhaalla sääntö 3. vetovoima. (Reynolds, 1995.)

3 PARVIÄLYKKYYS

Tässä luvussa esitellään *parviälykkyyttä*. Parviälykkyyden esimerkkeinä toimivat muurahaiset ja mehiläiset. Edellä mainitut hyönteiset ovat myös olleet pääasiallinen innostuksen lähde parviälykkyyden tutkimuksessa (Fisher, 2009; Fleischer, 2003). Hyönteisistä muurahaisten toimintaa on tutkittu eniten ja muurahaisten tutkimisen avulla on kehitetty parviälykkyysalgoritmeja, kuten Ant System ja Ant Colony System.

Kappaleessa 3.1 selitetään parviälykkyyden perusperiaatteita. Kappaleessa 3.2 tarkastellaan mehiläisiä ja kappaleessa 3.3 muurahaisia. Molemmat hyönteiset ovat olleet merkittävässä asemassa parviälykkyyden ymmärtämisessä ja kehittämisessä. Kappaleessa 3.4 tarkastellaan parviälykkyydet hyötyjä ongelmanratkaisussa sekä lyhyesti viittä parviälykkyyden sovellusta.

3.1 Parviälykkyyden perusperiaatteet

Parviälykkyys on parvikäyttäytymisen kehittyneempi muoto, missä parvet toimivat yhdessä ratkaistakseen ongelmia. Parviälykkyydessä on kyse *oppimisesta* ja se vaatii *kommunikaatiota* parven sisällä (Fleischer, 2003). *Parvikäyttäytymistä* kutsutaan parviälykkyydeksi, kun koko parvea käytetään *ongelmanratkaisussa*. Parvikäyttäytyminen on pelkästään parven jäsenten eli *partikkelien* yhteistoimintaa, ilman ongelmanratkaisua. Tietojenkäsittelytieteessä parven jäseniä kutsutaan partikkeleiksi.

Parviälykkyydessä on kyse suurten ja monimutkaisten kokonaisuuksien hallitsemisesta, vaikka kokonaisuuksien välinen kommunikaatio on minimaalista. Suurten kokonaisuuksien vuoksi parviälyllä toteutetut järjestelmät ovat hajautetusti ohjattuja, missä yhden osan hajoaminen ei kaada koko järjestelmää. (Fleisher, 2003.)

Parviälykkyyden taustalla on hyönteisten tutkiminen, hyönteisten käyttäytymisestä opitut asiat ja opittujen asioiden soveltamisesta tietojenkäsittelytieteeseen. Hyönteiset esiintyvät suurina, organisoituina joukkoina joiden selviytymiskyvyt ja ravinnonetsintäkyvyt ovat hyvät. Hyönteiset, kuten mehiläiset ja muurahaiset,

käyttävät selviytymiseen ja ravinnon etsimiseen työskentelytapoihin yksinkertaista *kommunikointia* ja organisoituun liikkumiseen yksinkertaisia *liikkumissääntöjä*. Hyönteiset käyttäytyvät ulkoisten ärsykkeiden mukaan ja nämä ärsykkeet saavat hyönteiset toimimaan *itseorganisoidusti* (Menzel & Giurfa, 2001). Yksittäinen hyönteinen ei löydä ratkaisua ongelmaan, mutta yhtenä parvena tai kolonnana ratkaisu löydetään jaetun informaation avulla (Camazine et al., 2001).

Parviälykkyydestä puhuttaessa esiintyvät usein termit *positiivinen ja negatiivinen palaute* (*positive and negative feedback*), jotka ovat keskeisiä termejä parvikäyttäytymisessä ja -älykkyydessä. Positiivisella palautteella vahvistetaan ilmiötä ja negatiivisella palautteella pyritään pitämään asiat ennallaan. Positiivinen ja negatiivinen palaute ohjailevat myös populaatioiden kasvua luonnossa, osakemarkkinoiden vaihtelua sekä parviälykkyydessä esiintyviä malleja. (Fisher, 2009.)

Positiivinen palaute synnyttää joukossa massakäyttäytymistä. Lähimmät yksilöt vaikuttavat mielipiteillään ja käyttäytymisellään muiden valintoihin. Tällä tavalla toimivat myös muurahaiset kolonnissaan: yksilöt pitävät valintaansa hyvänä, joten valintaa tekevät muurahaiset noudattavat lähimpien yksilöiden esimerkkiä.

Negatiivinen palaute pyrkii säilyttämään nykyisen tilan ja korjaamaan virheitä. Virheet aiheuttavat poikkeavuutta normaalitilasta, joten negatiivisella palautteella pyritään takaisin normaalitilaan. Esimerkiksi ajaessa autoa ja auton ajautuessa oikealle, negatiivisella palautteella pyritään saamaan ohjauspyörä kääntymään vasemmalle, jotta autolla palataan takaisin alkuperäiselle ajolinjalle. Positiivisella palautteella pyritään voimistamaan ohjauspyörän kääntymistä oikealle, jolloin auton pyörät vievät autoa enemmän oikealle pois ajolinjaltaan.

3.2 Mehiläiset

Reynoldsin säännöt eli liikkumissäännöt selittävät parvikäyttäytymistä, kuten kappaleessa 2.3 on todettu. Sen lisäksi mehiläisten käyttäytymistä tutkimalla ymmärretään Reynoldsin sääntöjen vaikutusta parven ongelmanratkaisussa, kun muutaman yksilön on johdatettava mehiläisparvi oikeaan paikkaan.

Yksittäiset mehiläiset noudattavat Reynoldsin kolmea sääntöä samoin kuin kulkusirkat. Kulkusirkoista poiketen yksittäiset mehiläiset osaavat viestittää toisilleen löytämänsä paikan sijainnin. Michiganin yliopistossa tehty Fred Dyerin tutkimus (Dyer, 2002) on osoittanut, että mehiläistiedustelijat välittävät tiedon muille mehiläisille tanssimalla niiden edessä kahdeksikkoa. Tanssin näkee kuitenkin vain murto-osa mehiläisistä, ja tanssin nähneet mehiläiset lentävät harvoin parven ensimmäisinä ja silti parvella päädytään oikeaan paikkaan. Martin Lindauer (1955) havaitsi ensimmäisenä, että muutamat parven mehiläisistä lentävät suuremmalla nopeudella kuin muut. Viisikymmentä vuotta myöhemmin sama ilmiö mehiläisparvessa vahvistettiin (Beekman et al. 2006). Näytti siltä, että suuremmalla nopeudella lentävät mehiläiset olivat suunnannäyttäjiä muille mehiläisille. Niiden avulla parvi löysi perille kohteeseensa. Tässä vaiheessa ei vielä tiedetty tietävätkö kaikki nopeimmin lentävät mehiläiset reittiä perille.

Tietokonesimulaatio (Couzil et al. 2005) osoitti, että mehiläisparvet löytävät perille, vaikkei kaikki nopeimmin lentävistä mehiläisistä tietäisikään reittiä perille. Tietokonesimulaatio osoitti myös, että parven johdattamiseen oikeaan kohteeseen tarvitaan vain muutama informoitu mehiläinen, koska kaikki noudattavat Reynoldsin kolmea sääntöä. Parven johdattamiseen tarvitaan vain muutama partikkeli, joilla on selvä määränpää. Muilla ei ole minkäänlaista määränpäätä, joten muut seuraavat informoituja mehiläisiä niin kauan, kun heillä ei ole omaa päämäärää tavoiteltavanaan.

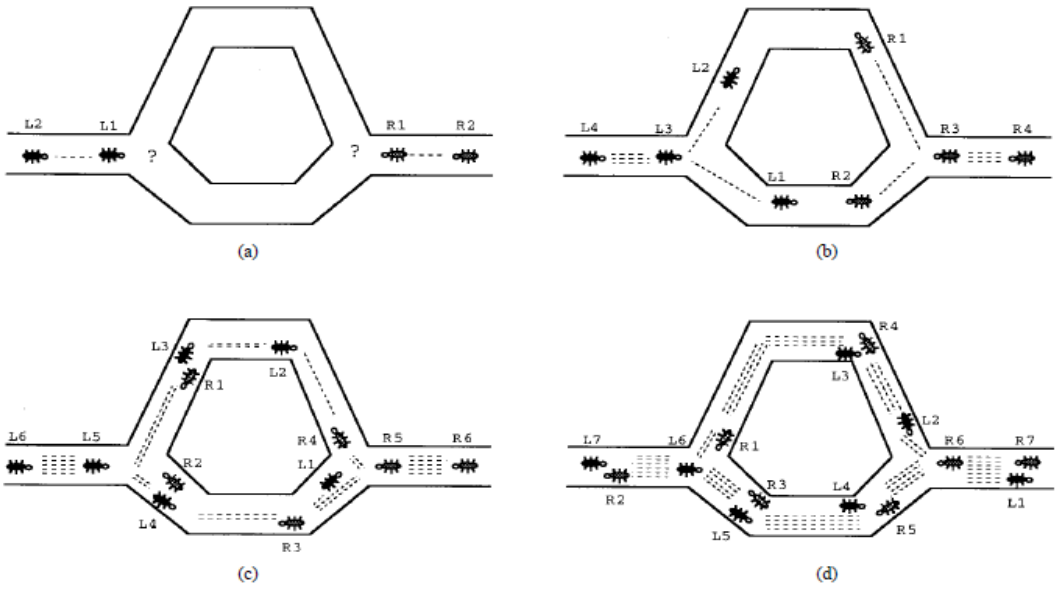
3.3 Muurahaiset

Kulkusirkat ja mehiläiset auttavat meitä ymmärtämään parvikäyttäytymistä ja osittain vastaamaan kysymykseen siitä mitä parviälykkyys on. Tutkimalla muurahaisten käyttäytymistä parviälykkyyttä ymmärretään paremmin. Muurahaisten tutkiminen onkin tarjonnut lukuisia algoritmeja parviälykkyyteen, joista kahta tutkitaan tässä tutkielmassa.

Muurahaiset löytävät aina lyhimmän reitin muurahaiskeon ja ruoan välillä ja ilmiön pohjalta on ryhdytty kehittämään muurahaispohjaisia parviälykkyysalgoritmeja (Dorigo et al., 1999). Brysselin yliopistossa ekologisen käyttäytymisen yksikössä

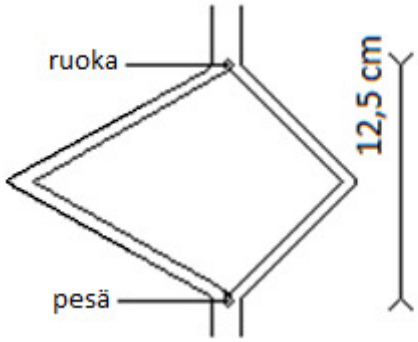
muurahaisten kyky lyhimmän reitin löytämiseksi muurahaiskeon ja ruoan välillä todistettiin kokeellisesti (Goss et a., 1989). Muurahaiskeon ja ruoan välille tehtiin silta, jota pitkin kulki kaksi eripituista reittiä kohteeseen. Ensin muurahaiset lähtivät sattumanvaraisesti kahta reittiä pitkin ruoan luokse. Muutaman minuutin kuluttua kaikki kulkivat lyhintä reittiä pitkin keolta ruoalle. Muurahaiset löysivät lyhimmän reitin levittämällä *feromoneja* reiteilleen. Feromonit ovat kemikaaleja, joiden hajua muurahaiset seuraavat (Fisher, 2009). Muurahaiset levittävät feromoneja kulkemilleen reiteilleen ja muurahaiset seuraavat hajun perusteella vahvinta feromonijälkeä. Muurahaiskeolla odottavat muurahaiset lähtevät seuraamana ensimmäisten muurahaisten jättämää feromonijälkeä ruoalle. Aikanaan pidemmiltä reiteiltä takaisin ehtineet muurahaiset lähtevät myös kulkemaan lyhintä reittiä, koska lyhimmällä reitillä on kulkenut enemmän muurahaisia kuin pidemmällä ja siihen on levitetty enemmän feromoneja.

Kuva 4 havainnollistaa vaiheittain kohtien a, b, c ja d avulla muurahaista hakeutumisen lyhimmälle reitille. Kohdassa a muurahaiset saapuvat kohtaan, jossa on tehtävä päätös reitistä. Kohta b osoittaa, että muurahaiset valitsevat sattumanvaraisesti jommankumman reitin. Kohdassa c osoitetaan, että lyhemmän reitin valinneet muurahaiset ovat ensimmäisinä perillä, koska kaikki muurahaiset liikkuvat samalla nopeudella. Kohdassa c lyhyempää reittiä on käytetty enemmän ja siihen on levitetty enemmän feromoneja, koska siitä on kulkenut enemmän muurahaisia kuin pidemmällä reitillä.



Kuva 4: Muurahaiset löytävät aina lyhimmän reitin. (Dorigo et al., 1997.)

Kuvassa 5 on havainnollistettu Brysselin yliopistossa tehdyn siltakokeen lähtöasetelma. Kuvassa osoitetaan miten muurahaiset löytävät lyhimmän reitin.



Kuva 5: Siltakoe. Kuvassa on siltakokeen lähtöasetelma, joka on lähteen (Goss et al., 1989) kuvan kaltainen.

Tietojenkäsittelytieteessä edellä kuvattu ongelma tunnetaan nimellä *muurahaiskolonnaoptimointi* (*Ant Colony Optimization*) (Bonabeau, 2000). Yksi tunnetuimpia optimointiongelmista on *Kauppamatkustajan ongelma*, jossa pyritään löytämään lyhin reitti kaupunkien välillä. Ongelman uskotaan olevan yksi vaikeimmista tunnetuista tietojenkäsittelytieteen *NP-täydellisistä ongelmista* (Gutin et al., 2002) eli ongelmista joihin ei ole keksitty polynomisessa ajassa toimivaa ratkaisua. Triviaali tapa on mitata jokaisen mahdollisen reitin pituus, mutta

kaupunkien määrän kasvaessa tämä tapa käy mahdottomaksi, koska vaihtoehtoisia reittejä on liian monta.

Muurahaiskolonnaoptimoinnin kehittynein sovellus on *hiukkasparvioptimointi* (*Particle Swarm Optimization*) (Kennedy & Eberhart, 1995), jossa muurahaiset korvataan *partikkeleilla*, jotka ovat itsenäisiä *agentteja* tai *yksilöitä*. Partikkeleilla on oma muistinsa, joten ne muistavat parhaat ratkaisunsa sekä pystyvät tarkkailemaan viereisten partikkelien toimintaa. Partikkelit noudattavat Reynoldsin sääntöjen kaltaisia toimintaperiaatteita eli välttävät törmäämästä viereisiin partikkeleihin, liikkuvat suunnilleen samaan suuntaan viereisten partikkelien kanssa ja pitävät etäisyyden viereisiin partikkeleihin suunnilleen vakiona.

3.4 Parviälykkyyden hyödyt ongelmanratkaisussa sekä parviälykkyyden sovellukset

Parviälykkyydessä partikkelit toimivat ryhmänä, jolla on yhteinen tavoite, joten parvessa on oltava yhteiset säännöt ja tavoitteet. Alun perin John Millerin ja Scott Pagen kokoamassa (Miller & Page, 2007) kahdeksan kohdan listassa esitellään asiat, jotka ryhmän partikkelit ottavat huomioon yhteistoiminnassa muiden partikkelien kanssa. Samankaltaisia listoja esitetään muun muassa Sumpterin (Sumpter, 2006) tutkielmassa, mutta tähän tutkielmaan valittiin Millerin ja Pagen lista, jonka avulla ymmärretään paremmin yksittäisen partikkelin huomioonotettavia asioita *kommunikoidessaan* muiden parven partikkelien kanssa.

Määritelmä 2. Parven yhteistoimintasäännöt (Miller & Page, 2007):

1) *Sopiva näkökulma*: Partikkelin on ymmärrettävä sille tuleva tieto muilta partikkeleilta ja ympäröivästä maailmasta.
2) *Sopivat tavoitteet*: Partikkelilla on oltava tavoite, jota kohti pyritään, esimerkiksi kalat välttävät syödyksi tulemista.
3) *Sopiva kommunikaatio*: Partikkelin on osattava jakaa tietoa keskenään, esimerkiksi lähettämällä kemiallisia viestejä toisilleen.
4) *Sopiva toiminta*: Partikkelin on pystyttävä vaikuttamaan toiminnallaan ympäröiviin partikkeleihin.

5) *Sopiva palaute*: Partikkelin on saatava toiminnastaan palautetta. Oikeasta toiminnasta partikkelit saavat hyvää palautetta ja väärästä toiminnasta rangaistus.
6) *Sopiva yritys*: Partikkelit tarvitsevat strategioita, jotta ne voivat ennakoida muiden partikkelien toimintaa.
7) *Sopiva tarkkaavaisuus*: Jokaisen partikkelin on osattava hyödyntää oikeanlaista rationaalisuutta.
8) *Sopiva keskittyminen*: Partikkelin on osattava keskittyä kerrallaan muutamiin sillä hetkellä tärkeisiin toimintoihin ja jättää huomiotta muut toiminnot.

Tietojenkäsittelytieteessä partikkeleille annetaan tehtäväksi ratkaista tietty tehtävä, esimerkiksi lyhimmän polun etsintä. Partikkelit löytävät ratkaisun yhdessä muiden partikkelien kanssa parvena. Ratkaisu on usein parempi kuin matemaatikon kehittämä ratkaisu samaan ongelmaan (Fisher, 2009), koska partikkelit toimivat itsenäisesti sekä pystyvät sopeutumaan ympäristöönsä ja sen muuttuviin olosuhteisiin. Muun muassa edellä kuvatut ominaisuudet kasvattavat parviälykkyysalgoritmien nopeutta ongelmanratkaisussa (Kassabalidis et al., 2001). Parviälykkyydessä on muitakin hyötyjä ongelmanratkaisuun:

Määritelmä 3. Parviälykkyyden merkittävimmät hyödyt ongelmanratkaisussa (Kassabalidis et al., 2001).

i) *Skaalautuvuus* (*Scalability*): Partikkelien määrä ja niiden työmäärä voidaan sopeuttaa ongelman kokoon nähden.
ii) *Virheensietokyky* (*Fault tolerance*): Muutamien partikkelien tai niiden välisten yhteyksien pettäminen ei pysäytä ongelman ratkaisua, vaan parven jäsenet pystyvät skaalautumaan uudestaan virheen mukaan.
iii) *Mukautuminen* (*Adaption*): Partikkelit pystyvät mukautumaan muutoksiin.
iv) *Nopeus* (*Speed*): Muutoksiin pystytään sopeutumaan nopeasti.
v) *Modulaarisuus* (*Modularity*): Partikkelit toimivat itsenäisesti muihin partikkeleihin nähden.
vi) *Itsenäisyys* (*Autonomy*): Partikkelit hallitsevat itse itseään.
vii) *Rinnakkaisuus* (*Parallelism*): Partikkelien toiminta on rinnakkaista.

Parviälykkyyden etuna on myös, että partikkelit ovat *proaktiivisia* eli pyrkivät ennakoimaan tulevaa. Partikkelit ovat myös *reaktiivisia* eli reagoivat herkästi ärsykkeisiin.

Määritelmässä 3 lueteltiin ominaisuuksia, joiden ansiosta parviälykkyys on suoraviivaisia menetelmiä parempi ja nopeampi tapa ratkaista tietojenkäsittelytieteen laskennallisia ongelmia. Partikkelit käyttävät päätöksenteossaan itse keräämäänsä informaatiota tietämättä partikkeliparvea koskevasta laajemmasta informaatiosta. Partikkelit tekevät päätöksensä noudattamalla kolmea Reynoldsin sääntöä (ks. *määritelmä 1 s. 8*). Kaikki partikkelit noudattavat myös parvessa parven yhteistoimintasääntöjä (ks. *määritelmä 2 s. 14*). Reynoldsin säännöt sekä parven yhteistoimintasäännöt implementoidaan parviälykkyysalgoritmeihin ja sovelluksiin: Tietojenkäsittelytieteessä yksittäiset partikkelit noudattavat aina Reynoldsin sääntöä liikkuessaan parvena, mutta parven yhteistoimintasäännöt ovat toteutettu algoritmista riippuen eri tavoin.

Parviälykkyys on tuonut uusia ratkaisumahdollisuuksia muun muassa laskennalliseen optimointiin ja tietoliikenteeseen (Ducatelle et al., 2010), koska kaikki partikkelit ratkovat ongelmaa *samanaikaisesti*. Taulukossa 1 on lueteltu sovellusalueita, joihin parviälykkyydestä löydetään hyviä ratkaisumahdollisuuksia. Jokaisen sovellusalueen perässä on lyhyt selitys parviälykkyyden käytön hyödyistä nimenomaisessa sovellusalueessa.

Taulukko 1: Parviälykkyyden hyötyjä eri sovellusalueilla.

Sovellusalue	**Kuvaus**
Ajoneuvojen reititysongelma	kuljetusalan kannattavuuden maksimointi (Bell et al., 2004).
Datan klusterointi	Tiedon ryhmittelyn nopeuttaminen parviälykkyyden avulla (Abraham et al., 2008).
Lentoyhtiöiden matkatavaroiden käsittely	Kustannustehokkuutta matkatavaroiden käsittelyssä pystytään parantamaan parviälykkyyden avulla (Bonabeau & Meyer, 2001).
Matkapuhelinten Ad Hoc -verkon säätely	Energiankulutuksen säätely ja sitä kautta verkon eliniän pidentäminen sekä verkon toiminnan nopeuttaminen (Gianni et al., 2005).
Tiedon louhinta	Tiedon etsinnän nopeutuminen parviälykkyysalgoritmien ansiosta (Monmarche, 1999).

4 MUURAHAISTEN PARVIÄLYKKYYS JA NIIHIN PERUSTUVAT ALGORITMIT

Hyönteisten itseorganisoituvuuden tutkiminen on parviälykkyyden perusta. Tietyt muurahaiset, esimerkiksi neotrooppiset armeijamuurahaiset muodostavat yli 200 000 muurahaisen metsästysjoukon ja muurahaiset keräävät tuhansia saaliita yhdessä päivässä (Garnier et al., 2007). Muurahaispohjaiset algoritmit tuovat tietojenkäsittelytieteeseen neotrooppisten armeijamuurahaisten toiminnan kaltaista tehokkuutta ongelmanratkaisuun.

Tässä luvussa keskitytään reititysalgoritmeihin. Kappaleessa 4.1 kerrotaan mitä ovat kommunikaatioverkot ja reititysalgoritmit. Kappaleessa 4.1.1 kerrotaan yleisimmistä reititysalgoritmien ongelmista ja kappaleessa 4.1.2 kerrotaan lyhyesti reititysalgoritmien testauksesta. Kappaleessa 4.2 esitellään Kauppamatkustajan ongelma, jota käytetään tässä tutkielmassa Ant System ja Ant Colony System -algoritmien suorituskyvyn testaukseen. Kappaleessa 4.3 esitellään Kauppamatkustajan ongelma käytännössä ajoneuvojen reititysongelmassa. Kappaleessa 4.4 esitellään muurahaispohjainen reititysalgoritmi Ant System ja selitetään kaikki algoritmin kaavat. Kappaleessa 4.5 esitellään muurahaispohjainen reititysalgoritmi Ant Colony System samalla tavalla, kun Ant System kappaleessa 4.4.

4.1 Muurahaispohjaisten algoritmien esitys tietojenkäsittelytieteessä

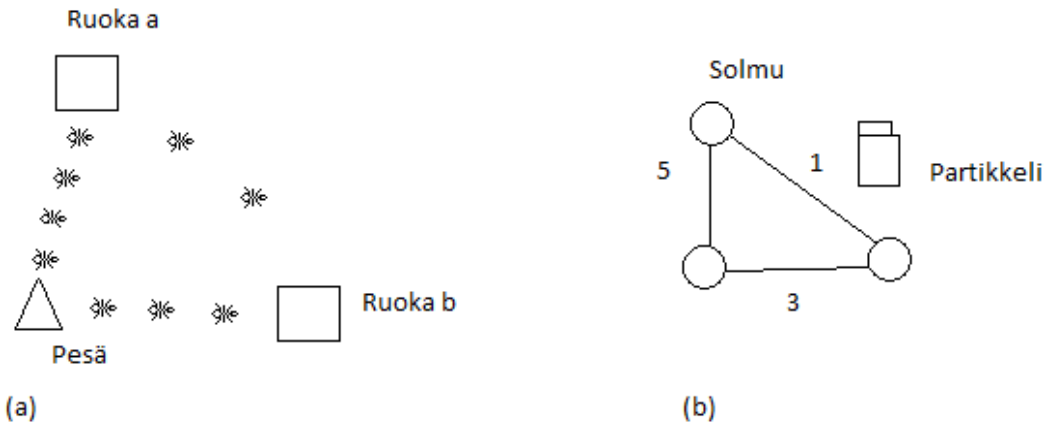
Tietojenkäsittelytieteessä muurahaispohjaisten reititysalgoritmien toimintaa mallinnetaan *verkkojen* avulla ja verkoilla pystytään mallintamaan monimutkaisia reititysongelmia (Di Caro et al., 1998). Reititysalgoritmit määrittelevät ja toteuttavat tiedonsiirron verkon *solmujen* välillä. Kirjallisuudessa verkkojen eri osille on monia eri termejä: verkosta käytetään lähteestä riippuen myös nimitystä *graafi*, mutta tässä tutkielmassa käytetään nimitystä verkko.

Verkko koostuu *solmuista* sekä *kaarista*, jotka yhdistävät solmuja (Di Caro et al., 1998). Kirjallisuudessa verkon solmuille käytetään nimityksiä *kärki* (*vertice*) ja

linkki (*link*). Muurahaisista, jotka kulkevat solmujen väliä, käytetään nimitystä partikkeli ja tätä nimitystä on käytetty jo aiemmin tässä tutkielmassa. Tietokonemaailman partikkelille on synonyymeinä eri lähteissä käytetty termejä *agentti* ja *yksilö*. Partikkelit kulkevat verkon kaaria pitkin lähtösolmusta päätesolmuun ja tätä reittiä kutsutaan *poluksi*.

Muurahaispohjaisissa algoritmeissa solmut edustavat esimerkiksi kaupunkeja (Bonabeau et al. 1999), kuten kauppamatkustajan ongelmassa, jossa kaaret kuvaavat reittejä kaupunkien välillä. Jokaisella kaarella on oma *painokerroin* ja esimerkiksi kauppamatkustajan ongelmassa painokertoimella kuvataan kaupunkien välistä etäisyyttä (Bonabeau et al., 1999). Pienellä painokertoimella tarkoitetaan lyhyttä etäisyyttä kaupunkien välillä. Muurahaispohjaiset parviälykkyysalgoritmit etsivät muun muassa lyhintä reittiä muurahaisten tapaan, mutta verkoissa feromonit ovat korvattu verkon kaarien painokertoimilla (Garnier et al., 2007).

Kuvassa 6 muurahaisten käyttäytyminen on kuvattu ja selitetty reaalimaailmassa ja tietojenkäsittelytieteessä. Kohdassa a on tilanne reaalimaailmasta, jossa muurahaiset etsivät lyhintä tietä ruokalähteelle. Ruualle a on lyhyempi matka, joten useampi muurahainen valitsee kyseisen reitin. Muurahaisten määrällä kuvataan feromonipitoisuutta reitillä. Kohdassa b on verkko, jonka avulla muurahaisten käyttäytyminen on mallinnettu tietojenkäsittelytieteessä. Tietojenkäsittelytieteessä puhutaan partikkeleista muurahaisten sijaan. Muurahaispohjaiset algoritmit etsivät ratkaisuja muun muassa kuvan b kaltaisista verkoista. Kaarilla on omat painokertoimensa, jotka kuvaavat feromonijäljen voimakkuutta ja samalla kaaren pituutta. Muurahaispohjaisissa algoritmeissa suurimman painokertoimen omaava kaari on lyhin. Painokerroin kuvastaa feromonien määrää kaarella eli kohdan a pesän ja ruoan a välinen kaari olisi kuvan 6 kohdassa b kaari, jonka painokerroin on viisi (5).



Kuva 6: Muurahaisten käyttäytymisen ja kommunikaatioverkot liittyvät läheisesti toisiinsa: Muurahaisten eniten käyttämällä reitillä on suurin painokerroin kommunikaatioverkossa ja kommunikaatioverkoissa muurahaiset ovat korvattu partikkeleilla.

Kuvan 6 kohdan b partikkeli voi kerätä ja kuljettaa verkossa erilaisia tietoja. Partikkelit keräävät verkosta informaatiota ja jakavat sitä muiden partikkelien kanssa aivan kuten muurahaiset jakavat tietoa epäsuorasti keskenään feromonien välityksellä.

4.2 Kommunikaatioverkot ja reititysalgoritmit

Kommunikaatioverkot ovat viestinvälityskanavia, joiden avulla jaetaan tietoa eri tahojen välillä. Verkot ovat useasta komponentista koostuvia laajoja hajautettuja verkkoja, joissa informaatiota välitetään verkon sisällä solmusta solmuun reititysalgoritmien avulla (Kassabalidis et al., 2001). Reititysongelma saadaan mallinnettua yksinkertaisessa muodossa kommunikaatioverkon avulla ja monia kommunikaatioverkkojen reititysongelmia pystytään ratkaisemaan muurahaispohjaisten reititysalgoritmien avulla (White & Pagurek, 1998). Reititysalgoritmit määrittelevät tiedonsiirron kommunikaatioverkoissa solmujen välillä. Parviälykkyyttä ja reititysalgoritmeja käytetään muuallakin kuin kommunikaatioverkoissa mutta tässä tutkielmassa keskitytään kommunikaatioverkkoihin.

Nykyiset reititysalgoritmit, kuten Dijkstran algoritmi (Dijkstra, 1959), eivät pysty tarjoamaan tarpeeksi tehokkaita ratkaisuja kommunikaatioverkkojen reititykseen kommunikaatioverkkojen laajuuden ja monimutkaisuuden vuoksi. Tämä johtuu siitä

että, hajautetut kommunikaatioverkot muuttavat jatkuvasti muotoaan, jolloin staattiset algoritmit eivät enää sovellu muuttuneeseen verkkoon (Bertsekas & Gallager, 1992). Staattiset algoritmit, kuten Dijkstranin algoritmi, eivät pysty *muuttumaan* verkon mukana. Parviälykkyysalgoritmeissa yksittäiset itseään hallinnoivat partikkelit pystyvät sopeutumaan muuttuvaan verkkoon, tekemään yhteistyötä toisten partikkelien kanssa sekä liikkumaan verkon sisällä mukautuen muuttuneeseen verkkoon (Bieszczad et al., 1998). Muurahaispohjaiset reititysalgoritmit ovat myös selvästi nopeampia rinnakkaisuutensa ansiosta kuin perinteiset reititysalgoritmit (Kassabalidis et al., 2001).

Reititysalgoritmit voidaan jaotella neljään kategoriaan: *staattisiin* eli muuttumattomiin, *adaptiivisiin* eli ajan mukaan muuttuviin sekä *keskitettyihin* ja *hajautettuihin* algoritmeihin (Kassabalidis et al., 2001). Reititysalgoritmit voidaan myös jaotella *minimaalisiin tai ei-minimaalisiin*. Minimaalisella reitityksellä etsitään lyhintä reittiä verkosta. Ei-minimaaliset ottavat huomioon muitakin *heuristiikoita* sopivan reitin etsimisessä. Reititysalgoritmien heuristiikkoihin voi tutustua yksityiskohtaisemmin Oidan ja Sekidon (Oida & Sekido, 1999) tutkimuksessa. Minimaalinen reititys jaetaan vielä optimaaliseen ja lyhimmän polun reititykseen (Bertsekas & Gallager, 1992; Di Caro et al., 1997). Optimaalisella reitityksellä etsitään optimaalisinta ratkaisua verkon kapasiteettiin nähden. Lyhimmän polun reitityksellä etsitään lyhintä reittiä verkon solmujen välillä välittämättä muista tekijöistä. Optimaalinen reititys optimoi valittavan reitin kustannustehokkuuden ja pakettien viivästymisen välillä ja lyhimmän polun reitityksellä valitaan reitti alku- ja päätesolmujen perusteella. Lyhimmän polun reitityksellä lasketaan kustannustehokkain reitti näiden kahden solmun välillä eikä oteta kantaa verkon muihin osiin.

Parviälykkyyteen perustuvia reititysalgoritmeja on monia erilaisia ja jokainen on räätälöity tiettyä ongelmaa varten. Seuraavaksi esitellään lyhyesti kolme algoritmia, jotta lukija ymmärtää, että muurahaispohjaisia algoritmeja on olemassa muitakin kuin tässä tutkielmassa tarkemmin Ant System ja Ant Colony System. Jokaisen alla esitellyn reititysalgoritmin tarkoituksena on ohjata liikenne lähtöpisteestä määränpäähän maksimoiden verkon suorituskyky ja minimoiden kustannukset (Kassabalidis, 2001).

Monissa puhelinverkoissa käytetään *Muurahaispohjaista ohjausta* (*Ant-Based Control, ABC*). Algoritmissa partikkelit lähetetään säännöllisin väliajoin verkkoon ja ne kulkevat sattumanvaraisesti niille annettua kohdetta kohti. Puhelinliikenne kulkee verkossa partikkelien paljastamia lyhimpiä reittejä pitkin. Lisätietoa algoritmista saadaan lähteestä (Schoonderwoerd, 1997).

Matkapuhelinten Ad Hoc -verkot (*MANET*) ovat puhelinverkkoja, jotka käyttävät langatonta kommunikaatiota. Yksi tähän tarkoitukseen kehitetty algoritmi on *AdHocNet* -algoritmi. Datapaketteja ohjataan oikeaan osoitteeseen ja partikkelit voivat liittyä tai kadota verkosta sattumanvaraisesti. Samoin verkon *topologia* voi muuttua milloin tahansa. Kaikki partikkelit ovat myös samanarvoisia ja verkon hallinta on hajautettu kaikille partikkeleille. Lisätietoa algoritmista saadaan lähteestä (Di Caro et al., 2005).

Useiden edestakaisten reittien reititys (*Multiple Round Trip Routing*) on kehitetty pakettien välitykseen. Algoritmissa on eteen- ja taaksepäin kulkevia partikkeleita. Eteenpäin kulkevat tutkivat ympäristöä ja taaksepäin kulkevat päivittävät kerättyä tietoa. Lisätietoa algoritmista saadaan lähteestä (Kassabalidis, 2001).

4.2.1 Reititysalgoritmien ongelmia

Reititysalgoritmeissa ilmenee monenlaisia ongelmia, mutta tunnistamalla ongelmat niihin pystytään varautumaan. Pelkästään solmujen tilaa tarkkailemalla ei pystytä ratkaisemaan kaikkia reititysalgoritmien ongelmia. Reitityksessä pitää ottaa huomioon myös suorituskyky. Suurimmat huolenaiheet ovat siirtonopeus ja verkossa oleva viive unohtamatta muistikapasiteettia ja kaistanleveyttä. Kommunikaatioverkkojen tulisi olla myös kustannustehokkaita, luotettavia ja aina toiminnassa. Kaikista kriittisimmät ongelmat ovat kuitenkin virheensietokyky ja verkon luotettavuus. (Di Caro et al., 1998.)

Keskitettyjen algoritmien ongelmana on skaalautuvuus ja tarve ulkoiselle valvonnalle (Raman, 1998). Ulkoinen valvonta tarkoittaa, että algoritmin hallinta ja valvonta tapahtuvat yhdestä lähteestä ja silloin algoritmin muut osat jäävät valvomatta. Toisin sanoen valvonnan ulkopuolella olevien osien ongelmiin ei pystytä reagoimaan tarpeeksi nopeasti. Käytännössä tätä mallia on mahdotonta käyttää,

koska keskityksestä johtuen verkossa on viivettä päätöksenteossa ja taulujen päivittämisessä. Keskitetyt algoritmit eivät pysty toipumaan nopeasti verkossa tapahtuvista informaatiokatkoksista (Kassabalidis et al., 2001).

Staattiset algoritmit olettavat, että verkon olosuhteet ovat aina ajasta riippuvaisia (Ahuja et al., 1993), eivätkä staattiset algoritmit käytä kaikkia verkon resursseja etsiessään lyhintä reittiä. Adaptiiviset eli mukautuvat algoritmit ovat ongelmissa verkon solmun pettäessä tai pelkästään liian nopeista verkossa tapahtuvista muutoksista (Bertsekas et al., 1992). Liian nopeat muutokset verkossa eivät ehdi päivittyä kaikkiin solmuihin ajallaan adaptiivisessa systeemissä.

Hajautetut reititysalgoritmit ovat tiukoilla reaaliajan rajoitteilla varustettuja algoritmeja, joissa tietokanta ja päätöksenteko ovat hajautettu pitkin verkon solmuja. Verkossa ilmeneviä vikoja ja viivästymisiä ei pystytä välttämään ja koko verkon tilaa ei pystytä kerralla määrittämään. Hajautus on tällä hetkellä eniten käytetty tekniikka reititysalgoritmeissa. (Di Caro et al., 1998.)

4.2.2 Reititysalgoritmien testaus

Algoritmeja voidaan testata erilaisilla tavoilla ja haasteellisinta on saada algoritmien välille vertailukelpoisia tuloksia. Tämän vuoksi on kehitetty testifunktioita, joita käytetään parviälykkyysalgoritmien ja edelleen muurahaispohjaisten reititysalgoritmien suorituskyvyn testaamiseen. Testifunktiot pysyvät jokaisen algoritmin kohdalla samoina, joten testauksesta saadaan keskenään vertailukelpoisia tuloksia. Testifunktio on kuvaus lähtöjoukosta maalijoukkoon, eli testattava yksittäinen syöte tuottaa täsmälleen yhden lopputuloksen. Testifunktioilla testataan muurahaispohjaisten algoritmien suorituskykyä ja tarkastelun kohteina ovat algoritmin virheettömyys ja nopeus eli kuinka nopeasti algoritmi pääsee lähimmäksi optimaalista ratkaisua.

Suorituskyvyn testaamista tietojenkäsittelytieteessä kutsutaan yleisesti *vertailuanalyysiksi* (*Benchmarking*) eli suorituskykyä verrataan toiseen samankaltaiseen algoritmiin. Yleisesti käytössä oleva Benchmark-testifunktio muurahaispohjaisille parviälykkyysalgoritmeille on Kauppamatkustajan ongelma.

4.3 Kauppamatkustajan ongelma (Travelling Salesman Problem), TSP

Kauppamatkustajan ongelma on laajalti tunnettu laskennallinen optimointiongelma, johon voidaan etsiä ratkaisua muurahaisalgoritmien avulla. Muurahaisalgoritmien soveltuvuutta eli kykyä ratkaista erilaisia optimointiongelmia testataan Kauppamatkustajan ongelman avulla, koska se on yksi eniten tutkituista NP-täydellisistä optimointiongelmista (Cook, 2007) eli ongelmista joihin ei ole löydetty polynomisessa ajassa toimivaa ratkaisua deterministisellä Turingin koneella (Herken, 1995). Kauppamatkustajan ongelmasta on kirjoitettu paljon tutkimustietoa muun muassa lähteistä (Applegate et al., 2006; Bonabeau et al., 1999).

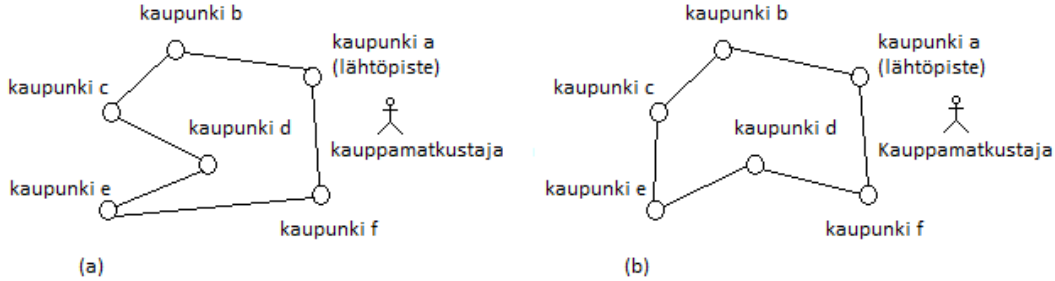
Kauppamatkustajan ongelmassa on kaupunkeja eli solmuja ja niitä yhdistäviä kulkureittejä eli kaaria. Kauppamatkustajan ongelmassa etsitään lyhintä reittiä solmujen välillä niin että kaikissa solmuissa vieraillaan vain kerran ja lopuksi palataan lähtösolmuun. (Cook, 2007). Tässä kappaleessa käytetään termejä solmu ja kaari, koska Kauppamatkustajan ongelmaa kuvataan kommunikaatioverkossa.

Kauppamatkustajan ongelmassa on n kappaletta solmuja, joiden välillä kauppamatkustaja pyrkii löytämään lyhimmän reitin niin, että jokaisessa solmussa vieraillaan yhden kerran. Löydetyn lyhimmän reitin on oltava suljettu eli viimeisestä solmusta palataan vielä lähtöpisteeseen. Suljettua reittiä kutsutaan *Hamiltonin poluksi*. (Iwamoto & Toussaint 1994.)

Kauppamatkustajan ongelma on NP-täydellinen ongelma, koska solmujen lukumäärän kasvaessa tarpeeksi suureksi ratkaisua ongelmaan ei pystytä löytämään polynomiaalisessa ajassa. Optimaaliseen ratkaisuun on laskettava kaikki mahdolliset kulkureitit ja kauppamatkustajan ongelmassa, jossa on n kappaletta solmuja, on $(n-1)!/2$ mahdollista reittiä. Ongelman aikavaativuus on eksponentiaalinen (2^n). (Cook, 2005.)

Kuvassa 7 on havainnollistettu kauppamatkustajan ongelman perusajatus. Kohdassa a kauppamatkustaja on löytänyt yhden mahdollisen reitin solmujen välille, mutta se ei ole lyhin reitti. Kohdassa b on löydetty lyhin reitti solmujen välille, joten se on

optimaalisin ratkaisu tähän ongelmaan. Muita reittejä ja solmuja yhdistäviä kaaria ei kuvassa 7 ole kuvattu.



Kuva 7: Kauppamatkustajan ongelman idea. Lyhin reitti kaupunkien välillä on kohdan b reitti.

Kauppamatkustajan ongelmaa kuvataan täydellisen ja painotetun *verkon* avulla, missä verkon *solmut* ovat *kaupunkeja* ja solmuja yhdistävät *kaaret* kaupunkeja yhdistäviä *kulkureittejä*. Täydellinen verkko tarkoittaa, että jokaisesta solmusta pääsee kaikkiin muihin solmuihin yhtä kaarta pitkin. Mikäli verkko ei ole täydellinen, pystytään solmujen väliin lisäämään kaari, mutta se ei saa vaikuttaa verkon optimaaliseen ratkaisuun. Jokaiselle kaarelle on laskettu painokerroin eli viereisten solmun välinen etäisyys on tiedossa. Kauppamatkustajan ongelmassa vierekkäisten solmujen etäisyys lasketaan Euklidisena etäisyytenä:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (x_i - y_j)^2}, \quad (1)$$

missä d_{ij} on solmujen i ja j välinen etäisyys ja x_i sekä y_i ovat solmun i koordinaatit ja vastaavasti x_j ja y_j ovat solmun j koordinaatit.

Kauppamatkustajan ongelmaan pystytään soveltamaan monia erilaisia ratkaisumenetelmiä, jotka kilpailevat nopeudellaan ongelman ratkaisussa, kun kaupunkien lukumäärä on tarpeeksi pieni. Ant System -algoritmin suorituskykyä testattiin ensimmäisen kerran Kauppamatkustajan ongelman avulla (Bonabeau et al., 1999). Kauppamatkustajan ongelma on myös käytössä eri aloilla yhteiskunnassamme, mm. logistiikassa (Christofides, 1976), kun ratkaistaan erilaisia reititysongelmia.

4.4 TSP käytännössä: ajoneuvojen reititysongelma

Ajoneuvojen reititysongelma on kauppamatkustajan ongelman erikoistapaus, mihin on lisätty muutama lisärajoite. Ajoneuvojen reititysongelma (*Vehicle Routing Problem, VRP*) on yleisnimitys kaikille ongelmille, joissa etsitään lyhintä reittiä kommunikaatioverkosta ja vieraillaan asiakkaan luona tai kaupungissa ajoneuvoilla (Christofides, 1976). Ongelmassa etsitään lyhintä ja kustannustehokkainta reittiä tavaran lastauspaikkojen ja asiakkaiden välillä. Tästä eteenpäin kappaleessa käytetään nimitystä kaupunki yhteisesti asiakkaasta sekä kaupungista. Ongelman kuvaus on esitetty seuraavassa listassa vaiheittain ja lista on lähteen (Laporte, 1992) listan kaltainen. Ajoneuvojen reititysongelmassa voi olla paljon muitakin rajoitteita, joita listassa ei ole mainittu. Niihin voi tutustua lähteessä (Golden et al., 1988).

1. Jokaisessa kaupungissa vieraillaan yhden kerran yhdellä ajoneuvolla
2. Kaikkien ajoneuvojen reitit alkavat ja päättyvät samaan lastauspaikkaan
3. Näiden rajoitteiden lisäksi ongelmassa voi olla muitakin rajoitteita
    - Tilavuusrajoite: Kaupunkiin liitetyn painon on oltava ei-negatiivinen ja painojen summan on oltava pienempi kuin ajoneuvon kokonaiskapasiteetti.
    - Reittien kaupunkien määrä on etukäteen tiedossa.
    - Aikarajoite: Reitillä kuluva aika on oltava alle reitille varattua aikaa. Aika sisältää matka-ajat kaupunkien välillä sekä kaupungeissa pysähtymiseen kuluvan ajan.
    - Kaupunkien välinen järjestys: kaupungissa j saatetaan vierailla ennen kaupunkia i.

Ajoneuvojen reititysongelma voidaan ratkaista Ant System -algoritmin avulla (Bullnheimer et al., 1997), jossa partikkelit etsivät parhaimman reitin kommunikaatioverkosta. Ajoneuvojen reititysongelmaa mallinnetaan kommunikaatioverkoilla samoin kuin Kauppamatkustajan ongelmaa.

Ajoneuvojen reititysongelmassa seuraava vierailtava kaupunki valitaan satunnaisen siirtymisen kaavalla, kuten kaupungista toiseen siirtyminen Ant System -

algoritmissa. Kaava on esitelty sivulla 30. Virtuaalisia feromoneja levitetään partikkelien kulkemille reiteille kaavan

$$\tau_{ij}^{new} = (1-\rho) * \tau_{ij}^{old} + \sum_{\mu=1}^{\sigma-1} \Delta\tau_{ij}^{\mu} + \sigma\Delta_{ij}^{*} \quad (2)$$

mukaan. Kaavassa 2 ρ on haihtumisen kerroin, τ_{ij}^{old} on reitin ij feromonipitoisuus ennen päivitystä ja τ_{ij}^{new} on reitin ij uusi feromonipitoisuus. Kaavassa Δ_{ij}^{*} on eliittipartikkelin löytämien reittien summa. Edellisessä kaavassa

$$\sum_{\mu=1}^{\sigma-1} \Delta\tau_{ij}^{\mu} = (\sigma-\mu)\, Q/L^{\mu}(t), \quad (3)$$

missä σ on eliittipartikkelien määrä ja μ on eliittipartikkelien sijoitus löydetyn reitin paremmuudessa mitattuna. Sijoitus määräytyy sen mukaan kuinka hyvä reitti on löydetty eli parempi reitti saa paremman sijoituksen. $L^{\mu}(t)$ on partikkelin μ kulkeman reitin pituus ja q on parametri, joka asetetaan tiettyyn arvoon parhaimman ratkaisun löytämiseksi.

Algoritmia saadaan muutettua lisäämällä siihen erilaisia rajoitteita ja parannuksia ongelmasta riippuen. Tärkeä huomioitava asia ajoneuvojen reititysongelmassa on kaupunkien etäisyys lastauspaikoista (Bullnheimer et al., 1997). Etäisyydet huomioonottaen kaupunkien välinen järjestys voi muuttua eli kaupungissa j vieraillaan ennen kaupunkia i. Kaupunkien vierailujärjestykseen vaikuttaa oleellisesti *säästöfunktio* (*saving function*), jolla lasketaan kaupunkien välistä matka-ajan mahdollisesti tuomaa säästöä:

$$\eta_{ij} = d_{i0} + d_{0j} - g * d_{ij} + f * \left|d_{i0} - d_{0j}\right|, \quad (4)$$

missä $d_{ij} \geq 0$ on kaaren paino, d_{i0} ja d_{0j} ovat kaupungeista lastattavien tavaroiden painot. Parametrit f ja g ovat säädettävissä optimaalisen ratkaisun löytämiseksi.

Toinen yleinen rajoite ongelmaan on aikaikkunarajoite eli tavaran toimitukset kaupunkeihin on tapahduttava tietyn ajan sisällä. Tässä mallissa otetaan huomioon myös odotusajat, koska kaupunkiin voidaan saapua liian aikaisin, mutta aika-ikkunarajoitteesta johtuen tavaroita ei voida toimittaa ajoissa. Reaalimaailman esimerkkejä aikaikkunarajoitteisista ajoneuvon reititysongelmista ovat muun muassa

pankkitoimitukset, postitoimitukset ja koulubussien reititykset. Lisätietoa aika-ikkunarajoitteisesta ajoneuvon reititysongelmasta (*Vehicle Routing Problem with Time Windows, VRPTW*) saadaan lähteestä (Solomon, 1987).

4.5 Ant System (AS)

Ant System on ensimmäinen muurahaispohjainen optimointialgoritmi (Bonabeau et al., 1999) ja algoritmissa käytetään hyödyksi *positiivista palautetta*, jotta löydetään paras mahdollisen ratkaisu erilaisiin optimointiongelmiin. Algoritmissa käytetään hyödyksi myös *negatiivista palautetta*, jottei ensimmäisellä löydetyllä ratkaisulla suljeta pois uusien parempien ratkaisujen löytämismahdollisuutta.

Ant System -algoritmin suorituskyvystä on saatu ensimmäiset tulokset implementoimalla algoritmi Kauppamatkustajan ongelmaan. Ant System -algoritmilla etsitään ratkaisua Kauppamatkustajan ongelmaan partikkeleilla, jotka liikkuvat kaupungista toiseen, kunnes kaikki kaupungit on käyty läpi (Bonabeau et al., 1999).

Partikkelin seuraavan solmun valinta noudattaa kolmea kriteeriä. Ensimmäinen kriteeri on partikkelin *oma muisti*, jossa on tieto vierailluista kaupungeista listana. Lista tyhjennetään täyden kierroksen päätyttyä, jotta verkko voidaan käydä uudestaan läpi. Täydellä kierroksella tarkoitetaan kierrosta, jossa verkon kaikissa solmuissa on vierailtu yhden kerran eli kaikki verkon solmut ovat käyty kertaalleen läpi. Listalla ylläpidetään partikkelille k solmujen joukkoa J, missä partikkeli ei ole vielä käynyt. Alkutilanteessa listassa ovat kaikki muut solmut, paitsi solmu, jossa partikkeli on kierroksen alussa. Käymällä listaa läpi partikkelit eivät käy samassa solmussa kahdesti.

Toinen kriteeri on *näkyvyys* eli etäisyyden käänteisluku $\eta_{ij} = 1/d_{ij}$, jolla kuvataan kuinka lähellä viereinen solmu on partikkelin tämänhetkistä solmua. Siinä käytetään ainoastaan partikkelin keräämää informaatiota sekä hyödynnetään kokemukseen perustuvaa eli heuristista lähestymistapaa seuraavan solmun valinnassa.

Kolmas kriteeri on *virtuaalisten feromonien* $\tau_{ij}(t)$ määrä solmuja i ja j yhdistävillä kaarilla tietyllä iteraatiokierroksella t. Virtuaalisten feromonien määrät muuttuvat algoritmin suorituksen aikana, koska kaikki muurahaiset levittävät uusia feromoneja kulkemilleen reitilleen. Partikkelien seuraava kaupunki valitaan satunaisen siirtymisen kaavan perusteella:

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in J_i^k} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}}, \tag{5}$$

missä $\tau_{ij}(t)$ on feromonien määrä kaarella ij ja η_{ij} on näkyvyys. Kaavassa α ja β ovat säädettävissä olevia parametreja. Kaavalla lasketaan seuraavan kaupungin valintatodennäköisyys $p_{ij}^{k}(t)$. Partikkelit valitsevat seuraavaksi kaupungiksi suurimman todennäköisyyden omaavan kaupungin. Partikkelit valitsevat todennäköisesti lähimmän solmun seuraavaksi kohteekseen, jos $\alpha = 0$ eli parametrin saadessa arvoja lähellä nollaa etäisyyden merkitystä korostetaan (Bonabeau et al., 1999). Mikäli $\beta = 0$ partikkelit noudattavat pelkästään vahvinta feromonien jälkeä solmun valinnassa ja kasvattaessa parametrin β arvoa solmujen etäisyyden tärkeyttä korostetaan feromonijälkeen nähden (Chu et al., 2004). Tyydyttävä ratkaisu eli reitti löydetään nopeasti, kun $\beta = 0$ mutta löydetty reitti ei todennäköisesti ole optimaalinen, koska se on löytynyt nopeasti. Edellä mainituista syistä molemmat, etäisyys sekä feromonien jättämä jälki, on otettava huomioon optimaalisinta reittiä etsiessä (Bonabeau et al. 1999). Tässä kappaleessa kaikki kaavat ovat lähteestä (Bonabeau et al. 1999).

Jokaisen täyden kierroksen jälkeen feromoneja levitetään partikkelin kulkemalle reitille kaavan

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} {}^{Q}\!/_{L^k(t)}, jos\ (i,j) \in T^k(t) \\ 0, \qquad\qquad muutoin \end{cases} \tag{6}$$

mukaan. Kaavassa $T^k(t)$ on partikkelin k tekemä täysi kierros solmujen välillä iteraatiokierroksella t ja $L^k(t)$ partikkelin k kulkema matka iteraatiokierroksella t. Q on parametri, jonka on hyvä olla samaa suuruusluokkaa kuin partikkelin kulkeman

optimaalisen reitin pituus (Bonabeau et al., 1999). Parametri Q ei kuitenkaan vaikuta oleellisesti levitettävien feromonien määrään.

Ant System -algoritmiin on lisättävä feromonien haihtuminen, koska ilman feromonien haihtumista partikkelit löytävät optimaalisen reitin vain sattumanvaraisesti. Feromonien jättämän jäljen on haihduttava ajan myötä, muuten kaikki partikkelit kulkisivat loppujen lopuksi samaa reittiä pitkin. Tätä varten tarvitaan kaava, jolla päivitetään feromonijälkeä:

$$\tau_{ij}(t) \leftarrow (1-\rho) * \tau_{ij}(t) + \Delta\tau_{ij}(t) + e\Delta\tau_{ij}^{e}(t), \quad (7)$$

missä ρ on *haihtumisen kerroin*, $\Delta\tau_{ij}(t)$ on partikkelien kulkemille kaarille jättämien feromonien summa tietyllä iteraatiokierroksella t, e on eliittipartikkeli ja $\Delta\tau_{ij}^{e}$ on eliittipartikkelien jättämien feromonien summa. Feromonijäljet haihtuvat haihtumisen kertoimen ρ mukaan ja ρ saavat arvoja väliltä $[0,1)$. Partikkelien lukumäärä on oltava sama koko suorituksen ajan. Liian monella partikkelilla löydetään nopea ja epätarkka ratkaisu. Liian pienellä määrällä partikkeleita ei ehditä käymään koko kommunikaatioverkkoa läpi tarpeeksi nopeasti. Feromonien jäljet ehtivät haihtua, jolloin optimaalista ratkaisua ei löydy. Dorigon ja hänen kollegoidensa mukaan (1996) partikkeleita ja kaupunkeja on oltava yhtä monta eli $m = n$. Partikkelit asetetaan joko sattumanvaraisesti solmuihin tai jokaiseen solmuun yksi partikkeli. Partikkelien asettelulla ei ole havaittu olevan lopputuloksen kannalta merkitystä (Bonabeau et al., 1999). Algoritmin suorituksen alussa, kun $t = 0$ feromonien määrä kaarilla on mahdollisimman pieni vakio τ_0. Partikkelit käyvät verkon läpi yhden iteraatiokierroksen aikana, jonka jälkeen algoritmissa suoritetaan uusi iteraatiokierros eli t kasvaa yhdellä.

Kaavassa käytetyt eliittipartikkelit $\Delta\tau_{ij}^{e}(t)$ parantavat algoritmin suoritustehokkuutta (Dorigo et al., 1996). Eliittipartikkelit e vahvistavat parhaita löydettyjä reittejä kaavan:

$$\Delta\tau_{ij}^{e}(t) = \begin{cases} {}^{Q}/_{L^{+}}, jos\ (i,j) \in T^{+}(t) \\ 0, \qquad\qquad muutoin \end{cases} \quad (8)$$

mukaan, missä L^+ on parhaimman löydetyn reitin pituus ja T^+ on paras löydetty reitti. Käytettäessä vain muutamia eliittipartikkeleita saadaan paras ratkaisu lyhimmäksi reitiksi (Bonaneau et al., 1999). Pienellä määrällä eliittipartikkeleita annetaan muille reiteille mahdollisuuden tulla löydetyksi. Liitteessä 1 on esitelty Ant System -algoritmin pseudokoodi.

4.6 Ant Colony System (ACS)

Ant Colony System -algoritmilla parannetaan Ant System -algoritmin suorituskykyä ja algoritmilla saavutetaan laskennallisesti parempia tuloksia kauppamatkustajan ongelman ilmentymissä (Bonaneau et al., 1999). Ant Colony System -algoritmiin on päivitetty siirtymissääntö kaupungista toiseen, feromonijäljen päivityssääntö, feromonipäivityksen käyttö sekä seuraavan solmun valinta.

Siirtymissääntö kaupungista toiseen on päivitetty tukemaan lyhimmän polun etsintää

$$j = \begin{cases} \arg max_{u \in J_i^k} \{[\tau_{iu}(t)] * [\eta_{iu}]^\beta\}, & kun\ q \leq q_0 \\ J, & kun\ q > q_0 \end{cases}, \quad (9)$$

missä q on satunnainen muuttuja väliltä $[0,1]$, q_0 on parametri väliltä $[0,1]$ ja $j \in J_i^k$ on kaavan

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)] * [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha * [\eta_{il}]^\beta} \quad (10)$$

mukaan valittu kaupunki. Parametrin q_0 arvojen vaihtelulla mahdollistetaan algoritmin keskittymisen parhaimpiin ratkaisuihin sen sijaan, että parasta ratkaisua etsittäisiin koko ajan kaikkien mahdollisten ratkaisujen joukosta. Kun muuttujan q_0 arvot ovat lähellä yhtä, algoritmilla keskitytään paikallisesti parhaimpiin ratkaisuihin ja käytetään hyödyksi tiedossa olevia solmujen välisiä etäisyyksiä ja reiteillä olevia feromonijälkiä. Kaavassa 9 merkinnällä $\arg max_{u \in J_i^k}$ tarkoitetaan, että valitaan paras kaari valittavissa olevista kaarista (Dorigo et al., 1997). Tässä kappaleessa kaikki kaavat ovat lähteestä (Bonabeau et al. 1999).

Paikallinen feromonijäljen päivitys suoritetaan aina kun partikkeli k on kaupungissa i ja valitsee seuraavan solmun joukosta $j \in J_i^k$. Feromonien päivitys reitillä (i, j) on kaavan

$$\tau_{ij}(t) \leftarrow (1 - \rho) * \tau_{ij}(t) + \rho * \tau_0 \quad (11)$$

mukainen. Gambardella ja Dorigo (Gambardella & Dorigo, 1995) ovat havainneet käytännön kautta, että asettamalla

$$\tau_0 = (n * L_{nn})^{-1} \quad (12)$$

tuottaa laskennallisesti hyvän tuloksen. Kaavassa 12 n on solmujen lukumäärä ja L_{nn} partikkelien kulkeman reitin pituus. Toisaalta Bonabeaun ja kumppanien (Bonabeau et al., 1999) mukaan parametrin τ_0 ollessa vakio, saadaan yhtä hyviä tuloksia, kuin asettamalla τ_0 kaavan 12 mukaan. Paikallista feromonien päivitystä ei voida jättää tekemättä, koska silloin kaikkia mahdollisia reittejä ei tutkittaisi ja optimaalisin ratkaisu jäisi löytämättä (Bonabeau et al., 1999).

Paikallisen feromonijäljen päivityssäännöllä haihdutetaan feromonijälkeä jo vierailluilta kaarilta, jotta lyhimmän reitin etsintä ohjataan vierailemattomille kaarille. Parempi reitti löydetään todennäköisemmin uusia kaaria pitkin kuin seuraamalla jo vierailtuja kaaria. Päivityksen johdosta samat partikkelit eivät kulje aikaisemmin kulkemiaan kaaria pitkin paitsi silloin, kun partikkelit eivät pääse eteenpäin vierailematomia kaaria pitkin. Reittiä, jossa on kuljettu kaari useampaan kertaan, ei kuitenkaan hyväksytä lyhimmäksi reitiksi. Ilman feromonien haihtumista kaarilta partikkelit kulkisivat samoja reittejä pitkin löytämättä uusia reittejä. (Bonabeau et al., 1999)

Feromonijäljen globaalilla päivityksellä päivitetään reittien feromonijälkeä sen jälkeen, kun partikkelit ovat löytäneet kommunikaatioverkosta *täyden kierroksen.* Toisin sanoen Ant Colony System -algoritmissa feromoneja levitetään vain parhaiksi havaituille reiteille, kun taas Ant System -algoritmissa feromoneja levitetään kaikille reiteille, jolloin optimaalisen ratkaisun löytäminen on hitaampaa. Feromonien levittämisellä vain parhaimmille reiteille rajoitetaan partikkelien kulkemaa matkaa ja

optimaalinen ratkaisu löydetään nopeammin, koska huonoja reittejä ei kuljeta (Bonabeau et al., 1999). Päivityskaava on:

$$\tau_{ij}(t) \leftarrow (1-\rho) * \tau_{ij}(t) + \rho * \Delta\tau_{ij}(t), \quad (13)$$

missä (i,j) on parhaalle reitille T^+ kuuluva kaari, ρ on parametri, jolla hallitaan feromonien haihtumista ja

$$\Delta\tau_{ij}(t) = {}^{1}/_{L^+}, \quad (14)$$

on kaava, missä L^+ on reitin T^+ pituus. Kaavassa 13 sallitaan feromonien jättäminen vain parhaimmille reiteille ja siitä johtuen Ant Colony System -algoritmi on parempi, kuin edeltäjänsä Ant System -algoritmin (Bonabeau et al., 1999).

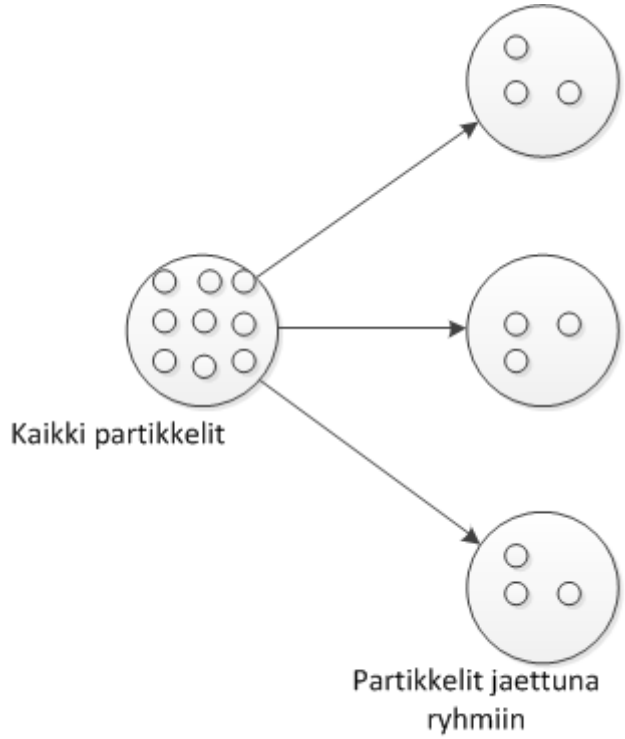
Seuraavan solmun valinnassa käytetään avuksi vierailtavien solmujen listaa J, joka on lista kaupungeista, joihin voidaan edetä sen hetkisestä kaupungista. Listassa on partikkelin sen hetkistä kaupunkia lähinnä olevat kaupungit etäisyysjärjestyksessä. Algoritmissa käydään listaa jatkuvasti läpi ja partikkelit siirtyvät seuraavaan kaupunkiin listan mukaan eli lähimmissä kaupungeissa käydään ensin ennen siirtymistä kauempiin kaupunkeihin. Vierailtava kaupunki valitaan aina kaavan 10 mukaan. (Bonabeau et al., 1999) Liitteessä 2 on esitetty Ant Colony System -algoritmin pseudokoodi.

Lähteessä (Bonabeau et al., 1999) on käytetty kandidaattilistaa seuraavan solmun valinnassa. Listan avulla pystytään vähentämään algoritmin suoritusaikaa (Gambardella & Dorigo, 1996). Kandidaattilistan avulla käydään läpi partikkelin solmua lähimpänä olevat solmut ja sen jälkeen muut solmut. Tässä tutkielmassa luvussa 5 Ant System ja Ant Colony System -algoritmien suorituskykyä testattaessa kandidaattilista on jätetty pois, koska algoritmin tehokkuudesta saadaan näyttöä ilman kandidaattilistaa.

4.6.1 Ant Colony System: Viestintästrategiat

Ant Colony System -algoritmilla pystytään tarjoamaan tehokkaampi ratkaisun kauppamatkustajan ongelmaan kuin Ant System -algoritmilla (Bonabeau et al., 1999) ja algoritmin suoritusaikaa pystytään parantamaan merkittävästi rinnakkaistamalla algoritmin toimintaa. Partikkelien luomisen jälkeen partikkelit jaetaan ryhmiin ja ryhmät toimivat kuten Ant Colony System -algoritmin ytksittäiset partikkelit. Ryhmien välillä on oltava vuorovaikutusta, jotta lyhin reitti löydetään eri ryhmien välillä. Tämä kappale on kirjoitettu mukaillen lähdettä (Chu et al., 2004).

Rinnakkaisen Ant Colony -algoritmin vaiheet ovat pääpiirteittäin saman, kuin Ant Colony -algoritmin. Algoritmin suorituksen alussa partikkelit jaetaan omiin ryhmiinsä ja ryhmien lukumäärä on korkeintaan yksi vähemmän, kuin partikkelien kokonaislukumäärä. Ryhmissä olevien partikkelien lukumäärä on sama, kuin ryhmien lukumäärä. Kuvassa 8 on havainnollistettu algoritmin suorituksen alussa tapahtuvaa ryhmiinjakoa. Kuvan 8 esimerkissä jokaisessa ryhmässä on kolme partikkelia, koska ryhmiä on kolme.



Kuva 8: Partikkelien jako ryhmiin. Kuva on lähteen (Chu et al., 2004) kuvan kaltainen.

Seuraavaksi esitellään seitsemän erilaista kommunikaatiostrategiaa *rinnakkaiseen Ant Colony System -algoritmiin* (*Parallel Ant Colony System*). Kaikki esiteltävät kommunikaatiostrategiat keskittyvät reittien fermonijälkien päivittämiseen. Päivittämiset tapahtuvat algoritmin eri vaiheissa, kuitenkin vasta paikallisten ja globaalien päivitysten jälkeen.

Kommunikaatiostrategiassa yksi feromonijäljet päivitetään jokaisen ryhmän osalta erikseen kaavan

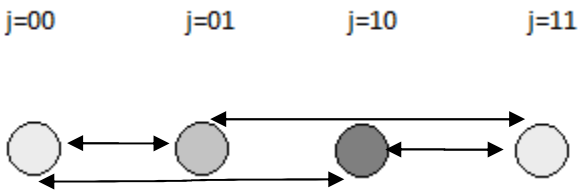
$$\tau_j(i,j) \leftarrow \tau_j(i,j) + \lambda * \Delta\tau_{best}(i,j) \tag{15}$$

mukaan, missä

$$\Delta\tau_{best}(i,j) = \begin{cases} (L_{gb})^{-1}, jos\ (i,j) \in \text{ó} \\ 0, \qquad\qquad muutoin \end{cases}, \tag{16}$$

missä symbolilla ó kuvataan joukkoa, jossa on kaikkien ryhmien löytämät reitit, L_{gb} on lyhimmän reitin pituus kaikkien ryhmien keskuudesta ja λ on *haihtumiskerroin*. Kaavalla $\tau_j(i,j)$ tarkoitetaan feromonijälkeä solmuja i ja j yhdistävällä kaarella. Tämän strategian päivitys suoritetaan algoritmin ensimmäisen kierroksen aikana eli partikkelien löydettyä ensimmäiset täydelliset kierrokset kommunikaatioverkosta.

Kommunikaatiostrategiassa kaksi feromonijälki päivitetään jokaisen ryhmän osalta viereisen ryhmän feromonijälkeen vertaamalla. Viereiset ryhmät määräytyvät ryhmien järjestysnumeron *vähiten merkitsevän bitin* (*least significant bit*) mukaan siten, että keskenään pareja ovat ryhmät joiden binääriluvut eroavat oikealta laskettuna vain yhden bitin verran toisistaan. Kuvassa 9 on havainnollistettu parien muodostusta järjestysluvun binääriesityksen perusteella. Pareja ovat 00 ja 01 sekä 00 ja 10. Pareja ovat myös 01 ja 10 sekä 10 ja 11. Paria ei muodostu 00:n ja 11:n välillä, koska luvut eroavat enemmän kuin yhden bitin verran toisistaan. Kuvan 9 esimerkissä pareja muodostui neljä.



Kuva 9: Feromonijäljen päivitys ryhmien mukaan. Kuva on lähteen (Chu et al., 2004) kuvan kaltainen. Kuvan tapauksessa nuolilla yhdistetyt ryhmät vertaavat feromonijälkiä keskenään. Järjestysluvut j vasemmalta oikealle järjestyksessä: 0, 1, 2, 3.

Jokaisella partikkeliryhmällä on vastaava pari, jonka feromonijäljen verraten feromonijälkeä päivitetään kaavan

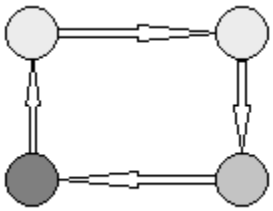
$$\tau_j(i,j) \leftarrow \tau_j(i,j) + \lambda * \Delta\tau_{ng}(i,j) \tag{17}$$

mukaan, missä

$$\Delta\tau_{ng}(i,j) = \begin{Bmatrix} (L_{ng})^{-1}, jos\ (i,j) \in ú \\ 0 \qquad\qquad muutoin \end{Bmatrix}, \tag{18}$$

missä ú on joukko mihin kuuluu kahden vierekkäisen ryhmän lyhimmät reitit ja L_{ng} on lyhin reitti partikkeliparin muodostamassa ryhmässä. Tällä strategialla päivitetään feromonijälkeä algoritmin toisella kierroksella eli kun partikkelit ovat löytäneet toisen kerran täydelliset reitit kommunikaatioverkosta.

Kommunikatiostrategiassa kolme feromonijälkeä päivitetään pareittain, kuten edellisessä kohdassa mutta parien muodostuksessa on merkittäviä eroja kuvan 9 strategiaan verrattuna. Parit muodostetaan kehärakenteen mukaan eli viereinen ryhmä määräytyy kehärakenteen mukaan eli kehän kulkusuunnassa viereinen ryhmä on edellisen pari. Kehärakenne on havainnollistettu kuvassa 10.



Kuva 10: Feromonijäljen päivitys kehärakenteen mukaan. Kuva on lähteen (Chu et al., 2004) kuvan kaltainen.

Neljännessä kommunikaatiostrategiassa käytetään samoja päivityssääntöjä kuin kolmannessa kommunikaatiostrategiassa, mutta parien muodostuksessa on eroja kolmanteen kommunikaatiostrategiaan verrattuna. Parit muodostetaan vähiten merkitsevien bittien mukaan, kuten kommunikaatiostrategiassa kaksi.

Kommunikaatiostrategiassa viisi feromonijälki päivitetään strategioiden yksi ja kaksi mukaan eli ensin päivitetään jokaisen ryhmän oma paras reitti ja sen jälkeen verrataan ryhmiä keskenään. Ryhmät ovat muodostettu kuvan 9 mukaan.

Kuudennessa kommunikaatiostrategiassa päivittämiseen käytetään strategioita yksi ja kolme eli ensin päivitetään kunkin ryhmän paras reitti ja sen jälkeen verrataan eri ryhmien reittejä keskenään. Ryhmät ovat muodostettu kehärakenteen eli kuvan 10 mukaan. Viimeisessä, seitsemännessä kommunikaatiostrategiassa päivittämiseen käytetään strategioita yksi ja neljä ja ryhmät ovat muodostettu kuvan 10 mukaan.

5 ANT SYSTEM JA ANT COLONY SYSTEM - ALGORITMIEN RATKAISUTEHOKKUUS KAUPPAMATKUSTAJAN ONGELMAN RATKAISUSSA

Tässä luvussa vertaillaan Ant System ja Ant Colony System -algoritmien suoritusnopeutta keskenään avoimen kirjaston (Reinelt, 2013) Kauppamatkustajan ongelman ilmentymillä berlin52, pr152 ja rd100. Berlin52 on 52 kaupunginosan kuvaus Berliinistä. Pr152 ja rd100 ovat satunnaisesti luotuja Kauppamatkustajan ongelman ilmentymiä (Reinelt, 2013). Numerolla Kauppamatkustajan ongelman nimessä viitataan kaupunkien tai kaupunginosien lukumäärään, esimerkiksi berlin52-ilmentymässä on 52 kaupunginosaa ja pr152-ilmentymässä 152 kaupunkia. Molemmissa algoritmeissa käytetään iteraatioiden määränä 1000:tta iteraatiota eli yksittäisellä partikkelilla etsitään lyhintä reittiä 1000 kertaa Kauppamatkustajan ongelman ilmentymästä. Iteraatioiden määrä on sama kuin McCaffreyn algoritmissa (McCaffrey, 2013), johon tämän tutkielman Ant System ja Ant Colony System -algoritmit perustuvat. McCaffreyn algoritmi on Microsoftin yleisen lisenssin alainen ja lisenssi on esitettävä algoritmin yhteydessä. Lisenssi on esitetty liitteessä 6.

Jokaisesta Kauppamatkustajan ongelman ilmentymästä etsitään lyhintä reittiä Ant System ja Ant Colony System -algoritmeilla kymmenen kertaa ($N = 10$) ja kaikkien ajokertojen suoritusajat mitataan. Mitattujen suoritusaikojen pohjalta halutaan testata lähteen (Bonabeau et al., 1999) väitettä, että Ant Colony System -algoritmi on paranneltu versio Ant System -algoritmista. Ant System ja Ant Colony System -algoritmien paremmuutta mitataan vertailemalla algoritmien suoritusaikoja keskenään. McCaffreyn algoritmissa Kauppamatkustajan ongelman ilmentymä luodaan itse, mutta tutkielmaa varten tehdyissä ratkaisussa käytetään valmiita Kauppamatkustajan ongelman ilmentymiä kaikille avoimesta tsp-kirjastosta (Reinelt, 2013). Tutkielmassa algoritmeissa käytetään tietorakenteena taulukkoja, kuten McCaffrey on käyttänyt omassa toteutuksessaan.

Kappaleessa 5.1 esitellään Ant System ja Ant Colony System -algoritmien päämetodit ja tietorakenteet. Kappaleessa 5.2 esitellään algoritmien muuttujat ja

metodit UML-kaavion avulla. Kappaleessa 5.3 kerrotaan toteutuksessa käytetyt työkalut sekä resurssit, joilla testit on ajettu. Kappaleessa 5.4 esitellään Kauppamatkustajan ongelman ilmentymillä saadut tulokset Ant System ja Ant Colony System -algoritmien suoritusaikojen eroista.

5.1 Ant System ja Ant Colony System -algoritmien tietorakenteet

Ant System ja Ant Colony System -algoritmit sisältävät neljä päämetodia, jotka luovat algoritmien tietorakenteet. Taulukossa 2 on listattu algoritmien päämetodit ja myöhemmin tässä kappaleessa metodien tietorakenteet on esitelty tarkemmin.

Taulukko 2: Ant System ja Ant Colony System -algoritmien päämetodit.

MakeGraphDistance
InitAnts
RandomTrail
InitPheromones

Ant System ja Ant Colony System -algoritmien suoritusten alussa metodilla *MakeGraphDistance* luodaan taulukko, johon tallennetaan kaikki Kauppamatkustajan ongelman ilmentymän kaupunginosien tai kaupunkien koordinaatit. Yleisimmin tunnetut Kauppamatkustajan ongelman ilmentymät eli kaupunkien tai kaupunginosien kuvaukset kommunikaatioverkossa on koottu tsp-tiedostoiksi, jotka ovat käytettävissä Heidelbergin yliopiston sivuilla (Reinelt, 2013). Taulukkoon tallennetaan kaikkien kaupunkien väliset etäisyydet. Taulukon indeksit vastaavat kaupunkeja eli taulukon solussa $[i, j]$ on tallennettu kaupunkien i ja j välinen etäisyys. Tässä luvussa käytetään nimitystä kaupunki kaupunginosan sijaan sekaannuksen välttämiseksi, vaikka taulukot 3, 4 ja 5 ovat Kauppamatkustajan ongelman ilmentymästä berlin52, jossa on kuvattu Berliinin kaupunginosia. Kaupungit ovat numeroitu ja taulukon indeksit vastaavat kaupungin numeroa. Yhtälailla voidaan käyttää kaupunkien nimiä numeroiden sijasta, mutta numeroinnilla säästetään muun muassa tietokoneiden muistia ja Heidelbergin yliopiston sivuilta (Reinelt, 2013) löytyvät symmetriset Kauppamatkustajan ongelmat käyttävät numerointia. Taulukko, johon kaupungin koordinaatit tallennetaan, on *etäisyysmatriisi*. Etäisyysmatriisi on symmetrinen diagonaalin

suhteen, eikä kaupungista itseensä voida mitata etäisyyttä. McCaffreyn toteutuksessa etäisyysmatriisissa kaikkien kaupunkien väliset etäisyydet saavat saman arvon eli kaikista kaupungeista on yhtä pitkä matka muihin kaupunkeihin. Tässä tutkielmassa kaupunkien koordinaatit luetaan tsp-tiedostosta Heidelbergin yliopiston sivuilta (Reinelt, 2013) Kauppamatkustajan ongelman ilmentymistä. Koordinaatit syötetään Euklidisen etäisyyden kaavaan, kaava 1, jolla lasketaan kaupunkien väliset Euklidiset etäisyydet. Kaavalla lasketut kaupunkien väliset etäisyydet tallennetaan etäisyysmatriisin soluihin, kuten taulukossa 3 on kuvattu.

Taulukko 3: Etäisyysmatriisin osa Kauppamatkustajan ongelman ilmentymästä berlin52. Taulukossa on kuvattu etäisyysmatriisin neljän ensimmäisen rivin ja sarakkeen muodostamat solut, jotka vastaavat kaupunkien *i* ja *j* välisiä etäisyyksiä.

	Sarakkeet				
Rivit	0	666	281	395	…
	666	0	1047	945	…
	281	1047	0	603	…
	395	945	603	0	…
	…	…	…	…	…

Etäisyysmatriisin luomisen jälkeen Ant System ja Ant Colony System -algoritmit konstruoivat satunnaiset reitit kaupunkien välille *InitAnts*- ja *RandomTrail*-metodien avulla. InitAnts-metodilla luodaan moniulotteinen taulukko eli taulukko, jonka kaikki solut sisältävät tietyn määrän soluja. Ensimmäisen taulukon ensimmäisen sarakkeen soluissa on partikkelia vastaava numero ja taulukon toisen sarakkeen soluissa on uusi taulukko, jossa on tallennettuna partikkelien reitit eli kaupunkien numerot. RandomTrail-metodilla arvotaan satunnaiset reitit partikkeleille. Taulukossa 4 on havainnollistettu metodeilla luotua tietorakennetta. Rivin ensimmäisessä solussa on partikkeleita vastaavat numerot ja seuraavassa solussa on partikkelin reitti. Solussa, jossa on tallennettuna partikkelin reitti, on yhtä monta solua kuin Kauppamatkustajan ongelman ilmentymässä on kaupunkeja. Soluihin on tallennettuna kaupunkeja vastaavat numerot.

Taulukko 4: Satunnainen reitti partikkelille Kauppamatkustajan ongelman ilmentymässä berlin52.

Partikkeli	Partikkelin reitti			
$\boldsymbol{p_0}$	1	3	2	…
$\boldsymbol{p_1}$	2	0	3	…
$\boldsymbol{p_2}$	0	2	1	…
$\boldsymbol{p_3}$	3	0	1	…
…	…	…	…	…

Kauppamatkustajan ongelmaan alustetaan Ant System ja Ant Colony System -algoritmien suorituksen alussa tietty määrä feromoneja jokaiselle kahta solmua yhdistävälle kaarelle *InitPheromones*-metodin avulla. Feromonien määrä kaarilla tallennetaan taulukkoon, jonka indeksit vastaavat etäisyysmatriisin indeksejä. Taulukossa 5 on esitetty algoritmin suorituksen alussa alustettu feromonipitoisuustaulukko. Taulukon arvot muuttuvat algoritmin suorituksen edetessä, koska algoritmin suorituksen edetessä päivitetään kaarien feromonipitoisuuksia. Feromonipitoisuustaulukon ja etäisyysmatriisin indeksit vastaavat toisiaan, jotta feromonien määrä pystytään liittämään yksinkertaisesti oikealle kaarelle eli solun i,j arvo on kaupunkeja i ja j yhdistävän kaaren feromonipitoisuus. Feromonipitoisuustaulukko on symmetrinen diagonaalinsa suhteen, kuten etäisyysmatriisikin.

Taulukko 5: Feromonipitoisuustaulukko algoritmin suorituksen alussa Kauppamatkustajan ongelman ilmentymässä berlin52. Taulukossa on kuvattu feromonipitoisuustaulukon neljän ensimmäisen rivin ja sarakkeen muodostamat solut, jotka vastaavat kaupunkeja $\boldsymbol{i}$ ja $\boldsymbol{j}$ yhdistävän kaaren feromonipitoisuutta.

	Sarakkeet				
Rivit	0	$1 * 10^{-5}$	$1 * 10^{-5}$	$1 * 10^{-5}$	…
	$1 * 10^{-5}$	0	$1 * 10^{-5}$	$1 * 10^{-5}$	…
	$1 * 10^{-5}$	$1 * 10^{-5}$	0	$1 * 10^{-5}$	…
	$1 * 10^{-5}$	$1 * 10^{-5}$	$1 * 10^{-5}$	0	…
	…	…	…	…	…

Feromonien määrää on pienennetty McCaffreyn käyttämästä alkuperäisestä arvosta 0,01 (McCaffrey, 2013) arvoon $1 * 10^{-5}$, koska kokeilemalla eri arvoja huomattiin, että pienennetyllä arvolla saadaan parempia tuloksia lyhimmäksi reitiksi kuin

käyttämällä arvoa 0,01. Kokeiltiin arvoja $1 * 10^{-1}$, $1 * 10^{-3}$, $1 * 10^{-4}$, $1 * 10^{-5}$, $1 * 10^{-6}$, $1 * 10^{-7}$, $1 * 10^{-8}$ ja $1 * 10^{-9}$. Lähteen (Bonabeau et al., 1999) mukaan vakioarvolla saadaan yhtä hyviä tuloksia, kuin asettamalla feromonipitoisuus kaavan 12 mukaan, joten päädyttiin käytännön testien kautta käyttämään arvoa $1 * 10^{-5}$. Arvolla $1 * 10^{-5}$ ja kaavan 12 mukaan tehtiin viisi koetta lyhimmän reitin löytämiseksi kullakin Kauppamatkustajan ongelman ilmentymillä rd100, berlin52 ja pr152. Koetuloksista huomattiin, että kaavan 12 mukaan Ant Colony System -algoritmi löysi lyhimmän reitin Kauppamatkustajan ongelman ilmentymistä rd100 ja berlin52 ja Ant System -algoritmi löysi lyhimmän reitin vakioarvolla Kauppamatkustajan ongelman ilmentymästä pr152. Tulokset ovat esitetty taulukossa 6.

Taulukko 6: Ant System ja Ant Colony System -algoritmien suoritusaikojen ja reittien pituukisen erotukset. Negatiivisella erotuksella tarkoitetaan, että Ant System -algoritmi on löytänyt lyhyemmän reitin tai reitin nopeammin kuin Ant Colony System -algoritmi.

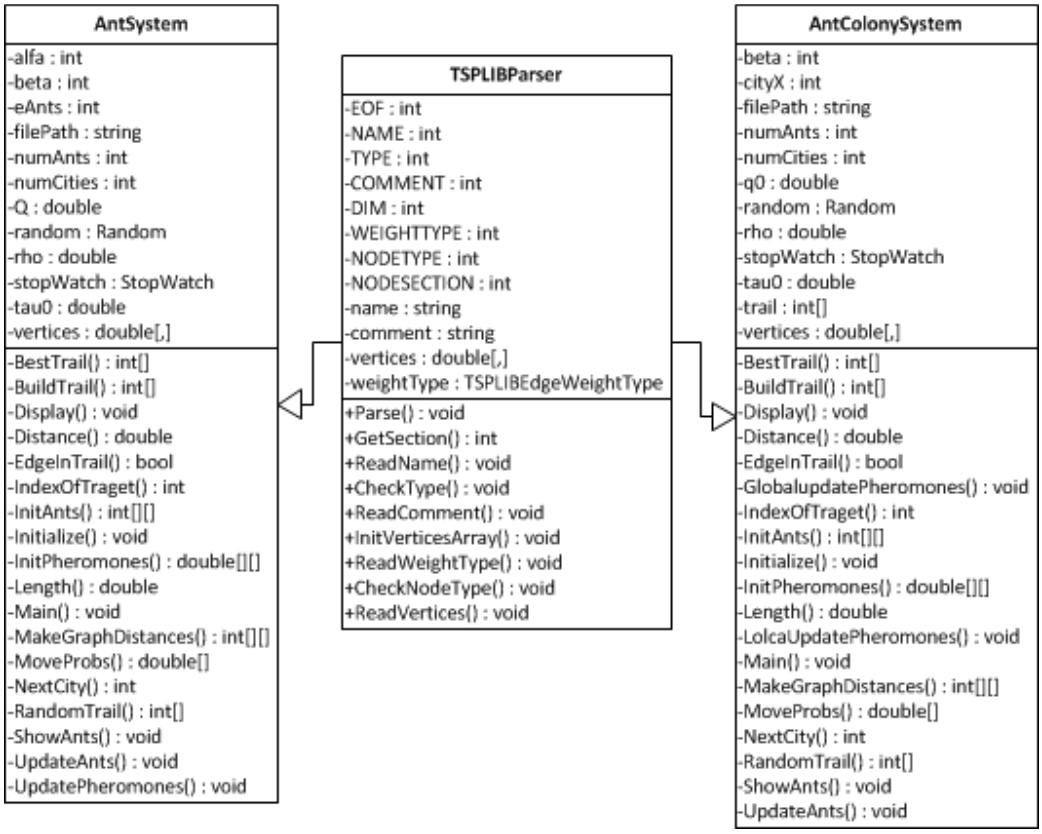
Selite	rd100	berlin52	pr152
Reitin pituus (AS) - Reitin pituus (ACS)	115,75	74,5	-289,95
Aika (AS) - Aika (ACS)	-19,3	162,25	3639,6

5.2 Ant System ja Ant Colony System -algoritmien muuttujat ja metodit

Ant System -algoritmiin on lisätty muuttujat *eAnts*, *tau0* ja *vertices*. Muuttujaan *eAnts* tallennetaan eliittipartikkelien määrä, muuttujaan *tau0* tallennetaan kaarille alustettavien feromonien määrän ja *vertices*-muuttujaan tallennetaan Kauppamatkustajan ongelman ilmentymän kaupunginosien koordinaatit, jotka ovat tallennettu tsp-tiedostoon.

Ant Colony System -algoritmiin on lisätty muuttujat *q0*, *tau0* sekä *vertices*. Muuttujaa $q0$ tarvitaan seuraavan kaupungin valintatodennäköisyyttä laskettaessa kaavan 9 mukaan. Muuttujat *tau0* ja *vertices* ovat samanlaiset kuin Ant System -algoritmissa eli *tau0*-muuttujaan tallennetaan kaarille alustettavien feromonien määrä ja *vertices*-muuttujaan tallennetaan Kauppamatkustajan ongelman ilmentymän kaupunginosien koordinaatit. Molemmissa algoritmeissa *filepath*-muuttujaan on

tallennettu tsp-tiedoston sijainti tietokoneella. Muuttujalla *stowatch* mitataan algoritmien suoritusaikaa. Ant System ja Ant Colony System -algoritmeissa olevat muuttujat ja metodit on listattu kuvan 10 UML-kaavioon. Ant Colony System -algoritmit kutsuvat luokkaa TSPLIBParser.cs, kuten kuvan 11 UML-kaaviossa on esitetty.



Kuva 7: Ant System ja Ant Colony System -algoritmien luokkakaaviot.

Molempiin algoritmeihin on lisätty metodi *Initialize*, jolla alustetaan ilmentymä TSPLIBParser.cs-luokasta (Wagner, 2012). Ilmentymän avulla tsp-tiedoston sisältö on käytössä ja tiedostosta saadaan luettua kaupunginosien lukumäärä sekä kaupunginosien koordinaatit. Ant System ja Ant Colony System -algoritmien metodin MakeGraphDistancen avulla tehdään etäisyysmatriisi Kauppamatkustajan ongelman ilmentymästä, kuten kappaleessa 5.1 on kuvattu. McCaffreyn algoritmissa ei käytetä TSPLIBParser.cs-luokkaa ja kaupunkien väliset etäisyydet ovat saaneet saman kokonaislukuarvon väliltä [1,8] (McCaffrey, 2013).

Seuraavan solmun valintatodennäköisyys lasketaan *MoveProbs*-metodissa ja McCaffreyn algoritmissa nimittäjälle asetetaan minimiarvo, koska seuraavan solmun valintatodennäköisyyden ollessa pieni algoritmia ei saada toimimaan oikein johtuen lukualueiden rajoituksista. (McCaffrey, 2013) Ilman minimiarvoa kaavojen 5 ja 10 nimittäjiin voidaan saada arvo nolla, kun solmujen välinen etäisyys on suuri. Ant System ja Ant Colony System -algoritmeissa käytetty nimittäjän minimiarvo seuraavan kaupungin valintatodennäköisyyttä laskettaessa on pienempi kuin alkuperäisessä algoritmissa käytetty arvo 0,0001 (McCaffrey, 2013). Tutkielmassa on päädytty käyttämään arvoa $1 * 10^{-30}$ kokeilemalla mielivaltaisesti valittuja arvoja minimiksi. Ant System ja Ant Colony System -algoritmeissa on kokeiltu nimittäjän minimiarvoksi arvoja 0,0001, $1 * 10^{-10}$, $1 * 10^{-20}$, $1 * 10^{-29}$, $1 * 10^{-30}$, $1 * 10^{-31}$, $1 * 10^{-35}$ sekä $1 * 10^{-40}$ ja todettu, että arvon $1 * 10^{-30}$ avulla löydetään keskimääräisesti lyhin reitti kauppamatkustajan ongelman ilmentymistä.

Ant System -algoritmin feromonien päivityssääntömetodia *UpdatePheromones* on muutettu kaavan 7 mukaiseksi. Ant Colony System -algoritmiin on lisätty paikallisen feromonijäljen päivityssääntömetodi *LocalUpdatePheromones*. Metodi on kaavan 11 mukainen. McCaffreyn metodia UpdatePheromones on muutettu kaavan 13 mukaiseksi ja metodi on nimetty GlobalUpdatePheromones-metodiksi. Metodien tarkemmat toteutukset esitetään liitteissä 3 ja 4. Liitteessä 3 esitetään Ant System -algoritmin C#-koodi ja liitteessä 4 esitetään Ant Colony System -algoritmin C#-koodi.

5.3 Toteutuksessa käytetyt ohjelmat ja resurssit

Molemmat Ant System ja Ant Colony System -algoritmit on toteutettu Microsoft Visual Studio 2010 Premium -ohjelmistokehitysympäristössä C# ohjelmointikielellä. Ohjelmointi ja algoritmien suorituskykyä mittaavat testit on tehty tietokoneella, jossa on 3.10 Ghz:n Intel Core i5-2400 -prosessori. Muistia tietokoneessa on 8 GB ja käyttöjärjestelmänä 64 bittinen Windows 7 Home Premium. Tulosten analysointiin on käytetty IBM SPSS Statistic -ohjelman versiota 19. Algoritmien UML-kaaviot on piirretty Microsoft Visio 2010 -ohjelmalla.

5.4 Algoritmien suoritustehokkuuksien analysointi kauppamatkustajan ongelmassa

Testit on suoritettu Ant System -algoritmilla Kauppamatkustajan ongelman ilmentymissä rd100, berlin52 ja pr152 parametreilla $\alpha = 1$, $\beta = 5$, $\rho = 0{,}5$, m ja $n = kaupunkien\ määrä$, $Q = 100$, $\tau_0 = 1 * 10^{-5}$ ja $e = 5$. Parametreja $\alpha = 1$ ja $\beta = 5$ on käytetty kaavassa 5. Parametria $\rho = 0{,}5$ on käytetty kaavoissa 11 ja 13. Parametri m on partikkelien määrä ja parametri n on kaupunkien määrä, joiden arvot saadaan tsp-tiedostosta. Parametria $Q = 100$ käytetään kaavoissa 6 ja 8 ja parametria $e = 5$ käytetään kaavassa 7. Parametri $\tau_0 = 1 * 10^{-5}$ on feromonipitoisuuden lähtöarvo, joten parametria ei käytetä feromonien päivityskaavoissa 5, 6 ja 7 kuin lähtöarvona. Parametrien arvot noudattavat lähteen (Bonabeau et al., 1999) arvoja. Ainut muutos lähteen arvoihin on parametrin τ_0 arvo, jonka valintaperusteet on esitetty kappaleessa 5.1.

Ant Colony System -algoritmin testit on suoritettu Kauppamatkustajan ongelman ilmentymissä rd100, berlin52 ja pr152 parametreilla $\beta = 2$, $q0 = 0{,}9$, $\tau_0 = 1 * 10^{-5}$, $\rho = 0{,}1$ ja m ja $n = kaupunkien\ määrä$. Parametria $\beta = 2$ käytetään kaavoissa 9 ja 10. Parametria $q0 = 0{,}9$ käytetään kaavassa 9 ja parametri $\tau_0 = 1 * 10^{-5}$ on käytössä kaavassa 11. Kaavoissa 9, 10 ja 11 parametri τ_0 on kaavojen alkuarvo. Parametria $\rho = 0{,}1$ käytetään kaavassa 11 ja 13. Parametri m on partikkelien määrä ja parametri n on kaupunkien määrä, joiden arvot saadaan tsp-tiedostosta, kuten Ant System -algoritmissa. Ant Colony System -algoritmin kohdalla noudatetaan lähteen (Bonabeau et al., 1999) parametrien arvoja. Poikkeuksena parametrin τ_0 arvo, joka on sama kuin Ant System -algoritmissa. Parametrin τ_0 arvo on vakio vastoin tutkielmassa esitettyä kaavaa, koska kokeilemalla eri arvoja todettiin että vakioarvolla saadaan keskimääräisesti yhtä hyviä tuloksia lyhimmäksi reitiksi kuin τ_0 ollessa kaavan 12 mukainen. Perustelut vakioarvon käyttöön on esitetty kappaleessa 5.1 ja liitteen 5 taulukossa 13.

Algoritmien suorituksista saaduista tuloksista kirjattiin ylös algoritmien suoritusaika ja algoritmien suorituksista saatuja tuloksia on verrattu keskenään tilastollisin menetelmin. Haluttiin todistaa, että Ant Colony System -algoritmi on tilastollisesti

merkittävästi nopeampi kuin Ant System löytämään lyhyempi tai likimain samanpituinen reitti Kauppamatkustajan ongelman ilmentymistä rd100, berlin52 ja pr152. Tilastolliset testit on tehty Mann-Whitneyn U-testillä (*Mann-Whitney U test*), ja riippumattomien otosten t-testillä (*Independent sample t-test*). Tutkielmassa käytetään kahta testiä, koska kaikki muuttujat eivät ole normaalijakautuneita. Normaalijakautuneisuus ja sen testaaminen selitetään myöhemmin tässä kappaleessa. Mann-Whitneyn U-testiä käytetään, kun kyseessä on kaksi toisistaan riippumatonta satunnaisesti valittua otosta ja kaikki muuttujat eivät ole normaalijakautuneita. Riippumattomien otosten t-testiä käytetään, kun kyseessä on riippumattomat otokset ja kaikki muuttujat ovat normaalijakautuneita. Tässä tutkielmassa muuttujana on suoritusaika.

Normaalijakautuneisuudella tarkoitetaan, että otokset, tässä tutkielmassa Kauppamatkustajan ongelman ilmentymät, ovat peräisin normaalijakautuneesta perusjoukosta. Tässä tutkielmassa perusjoukko on Heidelbergin yliopiston sivuilta löytyvä (Reinelt, 2013) alle 260 kaupungin Kauppamatkustajien ongelmien ilmentymien joukko. Kaupunkien määrä on rajoitettu, koska algoritmien suoritusajat ovat yli 260 kaupungin ilmentymissä kohtuuttoman suuria tutkielman laajuuteen nähden. Kaupunkien määrän rajoitus huomioituna perusjoukon koko on 42 Kauppamatkustajan ongelman ilmentymää. Ilmentymien joukosta on satunnaisesti valittu kolme Kauppamatkustajan ongelman ilmentymää eli otosta: Valitut Kauppamatkustajan ongelman ilmentymät olivat rd100, berlin52 ja pr152. Normaalijakautuneisuus on testattu IBM SPSS Statistic -ohjelmalla ja tulokset esitetty taulukossa 7. Normaalijakautuneisuuden testaamiseen on käytetty Shapiro-Wilkin testiä, jota käytetään kun *otoskoko* on alle 30. Tässä tutkielmassa otoskoko on kolme eli perusjoukosta valitut kolme Kauppamatkustajan ongelman ilmentymää. Kaikille kolmelle otokselle lyhintä reittiä on etsitty kymmenen kertaa Ant System ja Ant Colony System -algoritmeilla. Tämän tutkielman tilastolliset testit on tehty IBM SPSS Statistic -ohjelmalla, jonka valmiisiin testipohjiin on syötetty mitatut testitulokset. (Howell, 2010.)

Tilastollisessa testauksessa asetetaan ensin väittämät, joiden oikeellisuutta testataan eli *nollahypoteesi* H_0 ja *vastahypoteesi* H_1. Tilastollisissa testeissä joko hyväksytään nollahypoteesi H_0 tai hylätään H_0 ja hyväksytään vastahypoteesi H_1. Tilastotieteen

testeillä saadaan laskettua *p-arvo,* jonka mukaan hyväksytään tai hylätään nollahypoteesi ja vastaavasti hyväksytään tai hylätään vastahypoteesi. Mikäli testillä saadaan p-arvoksi pienempi kuin 0,01, nollahypoteesi hylätään ja vastahypoteesi hyväksytään. Arvoa 0,05 pidetään yleisesti hylkäämisrajana eli *merkitsevyystasona* nollahypoteesin hyväksymiselle tai hylkäämiselle (Howell, 2010), mutta tässä tutkielmassa halutaan saada tilastollisesti tarkempia tuloksia, joten käytetään merkitsevyystasona lukua 0,001. Hylkäämisrajaa pienemmillä p-arvoilla nollahypoteesi hyväksytään, koska p-arvo on todennäköisyys millä todennäköisyydellä vastahypoteesi on virheellinen. Virheen ollessa pienempi kuin merkitsevyystaso, vastahypoteesi hyväksytään. (Bergmann et al., 2000.)

Taulukosta 7 nähdään, että normaalijakautuneita ovat Kauppamatkustajan ilmentymistä berlin52 ja pr152 Ant Colony System -algoritmilla mitatut suoritusajat. Tasajakautuneita eli ei-normaalijakautuneita ovat Kauppamatkustajan ongelman ilmentymästä rd100 molempien algoritmien suoritusajat sekä ilmentymistä berlin52 ja pr152 Ant Colony System -algoritmin suoritusajat. Shapiro-Wilkin kokeessa nollahypoteesi on, että muuttujat noudattavat normaalijakaumaa. Vastahypoteesina on, että muuttujat eivät noudata normaalijakaumaa. Taulukon tuloksista nähdään että Kauppamatkustajan ongelman ilmentymän rd100 suoritusaikoja on verrattava riippumattomien ryhmien t-testillä ja ilmentymien berlin52 ja pr152 suoritusaikoja on verrattava Mann-Whitneyn U-testillä.

Taulukko 7: Normaalijakautuneisuuden testaus Shapiro-Wilkin testillä Kauppamatkustajan ongelman ilmentymistä rd100, berlin52ja pr152.. Taulukolla mukaillaan IBM SPSS Statistic -ohjelman taulukkoa.

Kauppamatkustajan ongelman ilmentymä	Algoritmi	Shapiro-Wilk
		P-arvo
Rd100	AS	,678
	ACS	,807
Berlin52	AS	,764
	ACS	,025
Pr152	AS	,468
	ACS	,000

Mann-Whitneyn U-testissä H_0:ksi asetetaan hypoteesi, että algoritmien suoritusajoilla ei ole tilastollistesti merkitsevää eroa. Toisin sanoen nollahypoteesin mukaan algoritmien suoritusajat ovat likimain samat. Vastahypoteesi H_1 mukaan erot ovat tilastollisesti merkitseviä eli algoritmien suoritusaikojen mediaanit eroavat toisistaan.

Verrattaessa Ant System -algoritmin ja Ant Colony System -algoritmin suoritusaikoja Kauppamatkustajan ongelman ilmentymällä rd100 riippumattomien otosten t-testillä saadaan taulukon 8 mukaisesti kaksisuuntaisen testin p-*arvoksi* (P-arvo) 0,000. P-arvon nojalla nollahypoteesi hylätään ja vastahypoteesi H_1 hyväksytään tarkoittaen, että suoritusajat eroavat toisistaan tilastollisesti merkitsevästi.

Taulukko 8: Riippumattomien otosten t-testillä saatu p-arvo Kauppamatkustajan ongelman ilmentymällä rd100. Suoritusaikojen vertailu. Taulukolla mukaillaan IBM SPSS Statistic -ohjelmalla suoritetun riippumattomien otosten t-testin tulostetta.

Testisuure	Suoritusaika rd100
P-arvo (2-suuntainen)	,000

Kauppamatkustajan ongelman ilmentymällä berlin52 Ant System ja Ant Colony System -algoritmien suoritusaikojen vertailussa Mann-Whitneyn U-tesillä saadaan taulukon 9 mukaisesti kaksisuuntaisen testin *p-arvoksi* (Tarkka p-arvo) 0,000. Arvo on pienempi, kuin raja-arvona käytetty 0,01 joten nollahypoteesi H_0 hylätään ja vastahypoteesi H_1 hyväksytään. Tässä tutkielmassa tilastollisessa testauksessa käytetään tarkkaa p-arvoa (Tarkka p-arvo), koska otoskoot ovat pieniä. Asymptoottista p-arvoa käytetään, kun testin otoskoko on suuri eli yli 30.

Taulukko 9: Mann-Whitneyn U-testillä saatu p-arvo Kauppamatkustajan ilmentymällä berlin52. Suoritusaikojen vertailu. Taulukolla mukaillaan SPSS Statistic -ohjelmalla suoritetun Mann-Whitneyn U-testin tulostetta.

Testisuure	Suoritusaika berlin52
Asymp. p-arvo (2-suuntainen)	,000
Tarkka p-arvo (2-suuntainen)	,000

Taulukon 10 mukaisesti Mann-Whitneyn U-testillä saadaan Kauppamatkustajan ongelman ilmentymällä pr152 kaksisuuntaisen testin *p-arvoksi* (Tarkka p-arvo) 0,000, joten vastahypoteesi H_1 hyväksytään ja nollahypoteesi H_0 hylätään. Vastahypoteesin hyväksymisellä tarkoitetaan että suoritusajat eroavat toisistaan tilastollisesti merkitsevästi.

Taulukko 10: Mann-Whitneyn U-testillä saatu p-arvo Kauppamatkustajan ongelman ilmentymällä pr152. Suoritusaikojen vertailu. Taulukolla mukaillaan IBM SPSS Statistic -ohjelmalla suoritetun Mann-Whiyneyn U-testin tulostetta.

Testisuure	Suoritusaika pr152
Asymp. p-arvo (2-suuntainen)	,000
Tarkka p-arvo (2-suuntainen)	,000

Taulukossa 11 on listattu algoritmien suoritusajat Kauppamatkustajan ongelman ilmentymissä rd100, berlin52 ja pr152. Riippumattomien otosten t-testillä ja Mann-Whitneyn U-testillä osoitettiin, että algoritmien suoritusajat eroavat tilastollisesti merkitsevästi toisistaan. Tarkastelemalla suoritusaikoja huomataan, että Ant Colony System suoriutui Kauppamatkustajan ongelman ilmentymistä nopeammin kuin Ant System.

Taulukko 11: Algoritmien suoritusajat millisekunteina Kauppamatkustajan ongelman ilmentymillä rd100, berlin52 ja pr152.

rd100		berlin52		pr152	
AS	ACS	AS	ACS	AS	ACS
1073817	83381	88802	11795	5591440	289151
1091442	84205	89664	11818	5589031	289639
1091086	83221	88711	11873	5489024	289213
1091442	83760	89387	11830	5548014	298417
1106163	84083	86454	11811	5598988	289014
1082829	83420	88152	11803	5501543	290518
1107998	83527	87083	11683	5698509	288457
1086480	83857	87380	11798	5683679	288520
1100128	83626	87634	11848	5448111	288465
1092115	83567	87362	11809	5489033	289442

Tässä luvussa testattiin Ant System ja Ant Colony System -algoritmien nopeutta kolmella eri Kauppamatkustajan ongelman ilmentymällä, rd100, berlin52 ja pr152. Saatuja tuloksia verrattiin keskenään riippumattomien otosten t-testillä sekä Mann-Whitneyn U-testillä. Tulokset olivat odotettuja eli algoritmien suoritusajat erosivat toisistaan tilastollisesti merkitsevästi. Ant Colony System -algoritmi löysi lyhimmän reitin Ant System -algoritmia nopeammin kaikista kolmesta ilmentymästä. Ant Colony System -algoritmi oli selvästi nopeampi kuin Ant System -algoritmi, joten voidaan todeta, että erot olivat tilastollisesti merkitseviä. Saadut tulokset tukivat lähdettä (Bonabeau et al., 1999), jonka mukaan Ant Colony System -algoritmi on paranneltu versio Ant System -algoritmista. Ant Colony System -algoritmin selvästi nopeampi suoritusaika johtui feromonien päivityssäännöistä. Ant Colony System -algoritmissa jokainen partikkeli päivittää feromonijäljen kulkemalleen kaarelle heti kaaren kuljettuaan. Tämän lisäksi algoritmin feromonien päivityssäännöllä 13, esitelty sivulla 33, päivitetään kaikki kaaret, jotka kuuluvat lyhimmälle reitille. Päivitysääntöjen avulla lyhimmän reitin etsintää ohjataan parhaimpien eli lyhimpien löydettyjen reittien suuntaan. Ant System -algoritmissa yhdellä feromonien päivityssäännöllä 7, esitelty sivulla 30, päivitetään feromonijälki kaikille kaarille jokaisen iteraatiokierroksen jälkeen. Ant System -algoritmissa eliittipartikkelit vahvistavat lyhimmän reitin feromonijälkeä, mutta eliittipartikkelit ovat vain pieni parannus algoritmiin verrattuna Ant Colony System -algoritmin päivitettyyn sääntöön 9, esitelty sivulla 31, ja kokonaan uuteen päivityssääntöön 11, joka on esitelty sivulla 32.

Tutkielmassa esitettyjen tulosten pohjalta voidaan todeta, että Ant Colony System -algoritmi on kehittyneempi muurahaispohjainen algoritmi kuin Ant System -algoritmi etsimään ratkaisua Kauppamatkustajan ongelman ilmentymistä rd100, berlin52 ja pr152. Ant Colony System -algoritmi löysi lyhimmän reitin Kauppamatkustajan ongelman ilmentymistä rd100, berlin52 ja pr152 tilastollisesti merkitsevästi nopeammin kuin Ant System -algoritmi. Ant Colony System -algoritmin löytämä lyhin reitti ei välttämättä ollut lyhyempi kuin Ant System -algoritmin löytämä lyhin reitti, mikä voidaan todeta taulukon 12 tuloksista.

Taulukko 12: Algoritmien löytämien lyhimpien reittien pituudet kokonaislukuina Kauppamatkustajan ongelman ilmentymillä rd100, berlin52 ja pr152.

rd100		berlin52		pr152	
AS	ACS	AS	ACS	AS	ACS
8839	8594	7240	7593	69487	69528
8746	8429	7278	7571	69212	69528
8900	8565	7295	7310	69212	69211
8641	8752	7293	7199	69487	69211
8756	8990	7741	7293	69246	70121
8945	8842	7217	7410	69342	69211
8652	8524	7277	7680	69312	69842
9047	8642	7199	7628	69421	69892
8724	8816	7254	7640	69212	69211
8993	9006	7564	7624	69453	69312

6 POHDINTA

Tässä tutkielmassa on perehdytty parviälykkyyteen ja kahteen muurahaispohjaiseen algoritmiin, jotka on kehitetty muurahaisten ravinnonetsintätaitoja tarkkailemalla. Tutkielma on aloitettu biologisesta lähtökohdasta ja edetty vähitellen kohti tieteellistä esitystapaa, jotta lukijan on helppo seurata ja ymmärtää tutkielman sisältö. Kaikki termit ja kaavat on pyritty selittämään tutkielmassa, jotta lukijalle ei jäisi epäselvyyksiä tutkielman lukemisen jälkeen.

Parviälykkyyden tutkiminen on saanut alkunsa eläinten ja ihmisten parvikäyttäytymisestä ja pääasiassa hyönteisten käyttäytymisen tutkimisesta. Hyönteiset toimivat tehokkaasti yhdessä ongelman ratkaisemiseksi ja erityisesti kulkusirkkojen, mehiläisten ja muurahaisten käyttäytymistä ja ravinnon etsintätapoja tutkimalla ollaan kehitetty muurahaispohjaisia parviälykkyysalgoritmeja lyhimmän polun etsintään. Muurahaisten parviälykkyyden ansiosta on syntynyt tutkielmassa tarkastellut muurahaispohjaiset algoritmit Ant System ja Ant Colony System.

Ant System ja Ant Colony System -algoritmien käyttömahdollisuudet eivät rajoitu pelkästään klassiseen kauppamatkustajan ongelmaan. Esimerkiksi joukkoliikenteessä saadaan aikaan merkittäviä kustannussäästöjä lyhimmällä reitillä, joten tutkielmassa esiteltyjä algoritmeja voidaan hyödyntää tähän tarkoitukseen. Algoritmeja voidaan käyttää monessa muussakin, kuten internetissä olevissa karttasovelluksissa, kuten Google Maps -sovelluksen reittihaussa. Käyttäjälle voidaan laskea lyhin reitti syöttämiensä kohteiden välille Ant Colony System -algoritmilla. Algoritmien käyttö- ja kehitysmahdollisuudet ovat erittäin laajat, joten toivottavasti muurahaispohjaisten algoritmien kehitys- ja tutkimustyötä jatketaan laajalla rintamalla. Tämän tutkielman tuloksista voidaan päätellä, että parviälykkyysalgoritmeilla on vielä potentiaalia kehittyä nopeammiksi. Ant System ja Ant Colony System -algoritmeissa on monia muuttujia, joille voidaan asettaa numeerisia arvoja ennen algoritmin suorituksen alkua. Algoritmit vaativat lisätutkimuksia, jotta voidaan olla varmoja, että algoritmit löytävät aina lyhimmän reitin mahdollisimman nopeasti.

Tutkielmassa molemmat algoritmit on ajettu jokaisen Kauppamatkustajan ongelman ilmentymän kohdalla kymmenen kertaa ja saatuja tuloksia on verrattu keskenään

riippumattomien otosten t-testillä ja Mann-Whitneyn U-testillä muuttujan normaalijakautuneisuuden mukaan. Tuloksista olisi saatu tilastollisesti tarkemman, jos otoskoko olisi ollut suurempi. Esimerkiksi, jos molemmat algoritmit olisi ajettu sata kertaa, tuloksista olisi saatu vieläkin tarkemmat. Algoritmien suoritusnopeutettua olisi saanut kasvatettua ohjelmoimalla algoritmit erilailla, mutta algoritmien nopeuden suhde pysyisi silti samana, joten tilastollinen analyysi tuottaisi saman tuloksen. Taulukkorannetta nopeampia tietorakenteita on muun muassa linkitetty lista, binääripuu ja hash-taulukko.

Algoritmien suoritusaikojen lisäksi kiinnostava tilastollisen testauksen kohde olisi lyhimmän löydetyn reitin pituus. Tässä tutkielmassa lyhimpien löydettyjen reittien pituuksia ei verrattu tilastollisesti keskenään. Löydettyjen reittien pituuksien tilastollinen vertailu jätettiin tämän tutkielman ulkopuolelle, koska Ant System ja Ant Colony System -algoritmien suoritusajat osoittivat Ant Colony System -algoritmin olevan paranneltu versio Ant System -algoritmista. Lyhimpien löydettyjen reittien pituudet eivät olleet selvästi paremmat kummallakaan algoritmilla, kuten taulukosta 12 voidaan todeta. Ant System ja Ant Colony System -algoritmien lyhimpien löydettyjen reittien tilastollinen testaus olisi seuraava luonteva tutkimuskohde tämän tutkielman jatkoksi. Samalla olisi mielenkiintoista testata lyhimmän löydetyn reitin ja algoritmin nopeuden suhteen tilastollista merkitsevyyttä toisiinsa eli löytääkö nopeampi Ant Colony Systen -algoritmi lyhyemmän reitin Kauppamatkustajan ongelman ilmentymästä kuin Ant System -algoritmi, kun iteraatioiden määrät ovat samat. Löydettyjen reittien pituudet vaihtelevat algoritmien kesken, kun algoritmien iteraatioiden määrä on riittävän pieni. Algoritmit löytävät eripituisia reittejä Kauppamatkustajan ongelman ilmentymistä iteraatioiden ollessa rajoitettuja, koska algoritmien feromonien päivityssäännöt ovat erilaisia. Mielestäni algoritmien pitäisi pystyä löytämään yhtä lyhyet reitit ilmentymistä iteraatioiden lähestyessä ääretöntä. Iteraatiokierrosten määrän lähestyessä ääretöntä eri päivityssääntöjen tuomat erot löydetyn reitin pituudessa pitäisi poistua.

Algoritmeissa on monia muuttujia, joiden optimaalisia arvoja olisi hyvä tutkia ja saada tieteellisiä perusteita arvojen käytölle. Tämän tutkielman aihepiiriin muuttujien arvojen tutkiminen ei kuulunut. Erityisesti Ant Colony System -algoritmin muuttujan τ_0 arvoa olisi mielenkiintoista tutkia. Tutkielmassa tehtiin testejä muuttujan τ_0

ollessa kaavan 12 mukainen ja sen ollessa vakio, mutta aiheeseen ei perehdytty syvemmin. Tulokset testeistä on nähtävissä liitteessä 5 taulukossa 13.

Algoritmien suorituskykyä voisi testata myös erittäin suurien Kauppamatkustajan ongelman ilmentymien avulla, eli ilmentymien, joissa solmujen määrät ovat suuria, esimerkiksi yli 500 ja siitä ylöspäin. Olen sitä mieltä, että algoritmien suoritusaikojen erojen pitäisi kasvaa todella suuriksi, koska Ant Colony System -algoritmi on kehitetty parantamaan Ant Systemin suorituskykyä -varsinkin suurille Kauppamatkustajan ongelman ilmentymille (Bonaneau et al., 1999) ja esimerkiksi Kauppamatkustajan ongelman ilmentymällä pr152 Ant Colony System löysi lyhimmän reitin noin 90 minuuttia nopeammin kuin Ant System -algoritmi.

Viitteet

Yapp J. (2005): *Waves*. *Everystockphoto.com*, http://www.everystockphoto.com/photo.php?imageId=10341&searchId=3fd6b6210e33bb046e69f256a138e28d&npos=11 (24.6.2013).

Abraham A., Das S., Roy S. (2008): Swarm Intelligence Algorithms for Data Clustering. *Soft Computing for Knowledge Discovery and Data Mining, part IV*, Lontoo, 279-314.

Ahuja R. K., Magnanti T. L., Orlin J. B. (1993): *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Inc., New Jersey.

Applegate D. L., Bixby R. E., Chvátal V., Cook W. J. (2006): *The Travelling Salesman Problem: A Computational Study*. Princeton University Press, New Jersey.

Di Caro, Gianni; Ducatelle, F.; Gambardella, L.M. (2005): "Swarm intelligence for routing in mobile ad hoc networks," *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE* , vol., no., pp.76,83, 8-10.

Beekman M., Fathke R. L., Seeley T. D. (2006): How does an informed minority of scouts guide a honeybee swarm as it flies to its new home? *Animal Behaviour*, **71**(1), 161-171.

Bell J. E., McMullen P. R. (2004): Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, **18**(1) 41-48.

Beni G., Wang J. (1989): Swarm Intelligence in Cellular Robotic Systems. *Proceedings of NATO Advanced Workshop on Robots and Biological Systems*, Berlin, 703-712.

Bergmann R., Ludbrook J., Spooren W. P. J. M., (2000): Different Outcomes of the Wilcoxon-Mann-Whitney Test from Different Statistics Packages. *The American Statistician*, **54**(1), 72-77.

Bertsekas D., Gallager R. (1992): *Data Networks*. Prentice-Hall Inc, New Jersey.

Bieszczad A., Pagurek B., White T. (1998): Mobile agents for network management. *IEEE Communication Surveys*, **1**(1), 2-9.

Bonabeau E., Meyer C. (2001): *Swarm Intelligence: A Whole New Way to Think About Business*. Harvard Business Review, Boston.

Bonabeau E., Dorigo M., Theraulaz G. (1999): *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York.

Bullnheimer B., Hartl R. F., Strauss C. (1997): *2nd International Conference on Metaheuristics - MIC97* (toim. Springer), Sophia Antipolis, 285-296.

Camazine S., Deneubourg J. L., Franks N. R., Sneyd J., Theraulaz G., Bonabeau E. (2001): Self-organization in biological systems. *Princeton University Press*. Princeton, New Jersey.

Chu S-C., Roddick J. F., Pan J-S. (2004): Ant colony system with communication strategies. *Information Science*, **167**(1-4), 63-76.

Cook W. (2005): *TSP: The Travelling Salesman Problem*. http://www.tsp.gatech.edu/problem/pcb3cnt.html (26.6.2013).

Cook W. (2007): TSP: The Travelling Salesman Problem. http://www.tsp.gatech.edu/problem/index.html (26.6.2013).

Couzil I. D., Krause J., Franks N. R., Levin S. A. (2005): Effective Leadership and Decision Making in Animal Groups on the Move. *Nature*, **455**(1), 513-516.

Di Caro G., Dorigo M. (1997): *AntNet: a mobile agents approach to adaptive routing*. IRIDIA/97-12, Université Libre de Bruxelles, Bryssel.

Di Caro G., Dorigo M. (1998): AntNet: Distributed Stigmergetic Control for Communication Networks. *Journal of Artificial Intelligence Research*, **9**(1), 317-365.

Di Caro, Gianni; Ducatelle, F.; Gambardella, L.M. (2005): "Swarm intelligence for routing in mobile ad hoc networks," *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE* , vol., no., pp.76,83, 8-10.

Dijkstra E. W. (1959): A note on two problems in connexion with graphs. *Numerische Mathematk*, **1**(1), 269-271.

Dorigo M., Caro G. D., Gambardella L. M. (1999): Ant Algorithms for Discrete Optimization. *Artifial Life*. **5**(2), 137-172.

Dorigo M., Gambardella L. M. (1997): Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *Evolutionary Computation, IEEE Transactions*, **1**(1), 53-66.

Dorigo, M.; Maniezzo, V.; Colorni, A. (1996): "Ant system: optimization by a colony of cooperating agents," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* , vol.26, no.1, pp.29,41.

Ducatelle F., Di Caro G. A., Gambardella L. M. (2010): Principles and applications of swarm intelligence for adaptive routing in telecommunications networks. *Swarm Intelligence*, **4**(3), 173-198.

Dunleavy S. (2011): *Big Eye Scad*. http://www.everystockphoto.com/photo.php?imageId=13633046 (26.6.2013).

Dyer F. C. (2002): The Biology of the Dance Language, *Annual Review of Entomology*, **47**(1), 917-949.

Dyer, J. R. G.; Johansson, A.; Helbing, D.; Couzin, I. D. & Krause, J. (2009): 'Leadership, consensus decision making and collective behaviour in humans', *Philosophical Transactions of the Royal Society B: Biological Sciences* **364** (1518), 781--789 .

Eberhart, R.C.; Yuhui Shi (2001): "Particle swarm optimization: developments, applications and resources," *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on* , vol.1, no., pp.81,86 vol. 1.

Fisher L. (2009): *The Perfect Swarm: The Science of Complexity in Everyday Life*. Basic Books, New York.

Fleischer M. (2003): *Foundation of Swarm Intelligence: From Principles to Practice*, Digital Repository at the University of Maryland, Maryland.

Gambardella L. M., Dorigo M. (1995): Ant-Q: A Reinforcement Learning approach to the traveling salesman problem, *Proceedings of Twelfth International Conference on Machine Learning*, pp. 252-260.

Gambardella, L.M.; Dorigo, M. (1996) : "Solving symmetric and asymmetric TSPs by ant colonies," *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on* , vol., no., pp.622,627, 20-22.

Garnier S., Gautrais J., Theraulaz G. (2007): The biological principles of swarm intelligence. Springer & Business Media. *Swarm Intelligence*, **1**(1), 3-31.

Goss S., Aaron S., Deneubourg J. L., Pasteels J. M. (1989): Self-organized Shortcuts in the Argentine Ant. *Naturwisenschaften*, **76**(12), 579-581.

Gutin G., Punnen A. (2002): *Combinatorial Optimization: The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, Boston/Dordrecht/Lontoo.

Herken R., (1995): *The Universal Turing Machine: A Half-century Survey*. Springer-Verlag Wien, New York.

Holland J. (1996): *Hidden Order: How Adaptation Builds Complexity*. Basic Books, New York.

Howell D. C. (2010): *Fundamental Statistics for the Behavioral Sciences*. Cengage Learning, California.

McCaffrey J. (2013): Test Run - Ant Colony Optimization. WWW-sivusto, http://msdn.microsoft.com/en-us/magazine/hh781027.aspx (26.6.2013)

Iwamoto C., Toussaint G. (1994): Finding Hamiltonian circuits in arrangements of Jordan curves is NP-complete. *Information Processing Letters*, **52**(4), 183-189.

Kassabalidis, I.; El-Sharkawi, M.A.; Marks, R.J., II; Arabshahi, P.; Gray, A.A. , (2001): "Swarm intelligence for routing in communication networks," *Global Tele-*

communications Conference, 2001. GLOBECOM '01. IEEE , vol.6, no., pp.3613,3617 vol.6.

Kennedy, J.; Eberhart, R. (1995): "Particle swarm optimization," *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol.4, no., pp.1942,1948 vol.4.

Laporte G. (1992): The Vehicle Routing Problem: An overview of exact and approximate algoritms. *European Journal of Operational Research*, **59**(3), 345-358.

Lindauer M. (1955): Schwarmbienen auf Wohnungssuche. *Zeitschrift für vergleischende Physiologie*, **37**(4), 263-324.

Menzel R., Giurfa M. (2001): Cognitive architecture of a mini-brain: the honeybee. *Trends in Congnitive Sciences*. **5**(2), 62-71.

Miller J. H., Page E. (2007): Complex Adaptive Systems: An Introduction to Computational Models of Social Life. *Princeton University Press*, New Jersey.

Monmarche N. (1999): On Data Clustering with Artificial Ants. *AAAI Technical Report WS-99-06*, AAAI, Palo Alto.

Oida, K.; Sekido, M. (1999): "An agent-based routing system for QoS guarantees," *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, vol.3, no., pp.833,838 vol.3.

Raman, L. (1998): "OSI systems and network management," *Communications Magazine, IEEE* , vol.36, no.3, pp.46,53.

Reinelt G. (2013): *Discrete and Combinatorial Optimization: TSPLIB*. WWW-sivusto, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/ (26.6.2013).

Reynolds C. (1995): *Boids : Background and Update*. http://www.red3d.com/cwr/boids/ (26.6.2013).

Schoonderwoerd R., Holland O., Brunet J., Rothkrantz L. (1997): Ant-based load balancing in telecommunications networks. *Adaptive Behavior*. **5**(2), 169-207.

Solomon M. M. (1987): Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, **35**(2), 254-265.

Sumpter D. J. T. (2006): The Principles of Collective Animal Behaviour. *Philosophical Transactions of the Royal Society B: Biological Sciences*, **361**(1465), 5-22.

Wagner S. (2012): TSPLIBParser.cs. *Heuristic and Evolutionary Algorithms Laboratory (HEAL),* http://dev.heuristiclab.com/trac/hl/core/browser/trunk/sources/HeuristicLab.Problems.TravelingSalesman/3.3/TSPLIBParser.cs?rev=7259 (26.6.2013).

White, T.; Pagurek, B. (1998): "Towards multi-swarm problem solving in networks," *Multi Agent Systems, 1998. Proceedings. International Conference on* , vol., no., pp.333,340, 3-7.

Liite 1: Ant System -algoritmin pseudokoodi

/*Alustukset*/

FOR jokaiselle kaarelle (i,j) DO

$\tau_{ij}(0) = \tau_0$

End FOR

FOR $k = 1,\ldots,m$ DO

Aseta partikkelit k sattumanvaraisesti kaupunkeihin 1–m

END FOR

Olkoon T^+ lyhin löydetty reitti ja L^+ lyhimmän löydetyn reitin pituus ja T_{max} iteraatioiden lukumäärä

/*Algoritmi*/

FOR $t = 1,\ldots,T_{max}$ DO

FOR $k = 1,\ldots,m$ DO

Rakenna kierros $T^k(t)$ $n-1$ kertaa kaavan

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in J_i^k} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}}$$

mukaan

END FOR

FOR $k = 1,\ldots,n$ DO

Laske partikkelin k kulkeman reitin $T^k(t)$ pituus $L^k(t)$

END FOR

IF Löydetty reitti on lyhyempi, kuin edellinen THEN

Päivitä T^+ ja L^+

END IF

FOR jokaiselle kaarelle (i,j) DO

Päivitä feromonijäljet kaarilla kaavan

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

mukaan, missä

$$\Delta\tau_{ij}^k(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^k(t),$$

$$\Delta\tau_{ij}^k(t) = \begin{cases} Q/L^k(t), jos\ (i,j) \in T^k(t) \\ 0, \qquad (i,j) \notin T^k(t) \end{cases}$$

ja

$$\Delta\tau_{ij}^{e}(t) = \begin{cases} {}^{Q}/_{L^{+}}, jos\ (i,j) \in T^{+}(t) \\ 0, \qquad\qquad (i,j) \notin T^{+}(t) \end{cases}$$

```
    END FOR
    FOR jokaiselle kaarelle (i,j) DO
        τ_ij(t + 1) = τ_ij(t)
    END FOR
END FOR
TULOSTA lyhin reitti T+ ja lyhimmän reitin pituus L+
/*Algoritmissa käytetyt vakiot*/
```

$$\alpha = 1, \beta = 5, \rho = 0{,}5, m = n, Q = 100,\ \tau_0 = 10^{-6}, e = 5$$

Liite 2: Ant Colony System -algoritmin pseudokoodi

/*Alustukset*/

FOR jokaiselle kaarelle (i,j) DO

$\tau_{ij}(0) = \tau_0$

End FOR

FOR $k = 1,\ldots,m$ DO

Aseta partikkelit k sattumanvaraisesti kaupunkeihin 1–m

END FOR

Olkoon T^+ lyhin löydetty reitti ja L^+ lyhimmän löydetyn reitin pituus ja T_{max} iteraatioiden lukumäärä

/*Algoritmi*/

FOR $t = 1,\ldots,T_{max}$ DO

FOR $k = 1,\ldots,m$ DO

Rakenna kierros $T^k(t)$ $n-1$ kertaa seuraavasti

IF Mikäli on olemassa ainakin yksi kaupunki j vierailtavien solmujen listassa J_i^k THEN

Valitse seuraava kaupunki j vierailtavien solmujen listasta $j \in J_i^k$, kun

$$j = \begin{cases} \arg max_{u \in J_i^k} \{[\tau_{iu}(t)] * [\eta_{iu}]^\beta\}, & kun\ q \leq q_0 \\ J, & kun\ q > q_0 \end{cases},$$

missä seuraava kaupunki j valitaan listasta $j \in J_i^k$ kaavan

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)] * [\eta_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha * [\eta_{il}]^\beta}$$

mukaan

END IF

Jokaisen siirtymän jälkeen suoritetaan paikallisen feromonijäljen päivitys:

$\tau_{ij}(t) = (1-\rho) * \tau_{ij}(t) + \rho * \tau_0$

END FOR

FOR $k = 1,\ldots,m$ DO

Laske partikkelin k kulkeman reitin $T^k(t)$ pituus $L^k(t)$

END FOR

IF Löydetty reitti on lyhyempi kuin edellinen THEN

Päivitä T^+ ja L^+

END IF

FOR jokaiselle kaarelle (i,j) DO

Päivitä feromonijäljet kaarilla kaavan

$\tau_{ij}(t) = (1-\rho) * \tau_{ij}(t) + \rho * \Delta\tau_{ij}(t)$

mukaan, missä

$\Delta\tau_{ij}(t) = {}^{1}/_{L^+}$

END FOR

FOR jokaiselle kaarelle (i,j) DO

$\tau_{ij}(t+1) = \tau_{ij}(t)$

END FOR

END FOR

TULOSTA lyhin reitti T^+ ja lyhimmän reitin pituus L^+

/*Algoritmissa käytetyt vakiot*/

$$\beta = 2, \rho = 0{,}1, q_0 = 0{,}9, m = n, \tau_0 = 10^{-6}$$

Liite 3: Ant System -algoritmin C#-lähdekoodi

```
using System;
using System.IO;
using System.Diagnostics;
using TSPParser;


namespace AntSystem
{
   /// <summary>
   /// Demo of Ant System (AS) solving a Traveling Salesman Problem (TSP).
   /// The core of this implementation is made by James McCaffreys. I have added
   /// few new funtionalities such as this impelentation uses knows TSP problems
   /// from the TSPLib95 library and removed all the min and max values of
   /// pheromones. There are free parameters alpha, beta, rho, and Q, tau0 and
   /// eAnts which are freely adjustable within this code. FilePath is the
   /// location of the TSP -instances in my computer.
   /// </summary>
   class AntSystem
   {
      /// <summary>
      /// Instance of a random class which uses current time to create randomness.
      /// </summary>
      static Random random = new Random(DateTime.UtcNow.Millisecond);

      /// <summary>
      /// Instance of a stopwatch to measure elapsed time of this algorithm.
      /// </summary>
      static Stopwatch stopWatch = new Stopwatch();

      /// <summary>
      /// Variable alpha.
      /// </summary>
      static int alpha = 1; // influence of pheromone on direction.
      /// <summary>
      /// Variable beta.
      /// </summary>
      static int beta = 5;  // influence of adjacent node distance.

      /// <summary>
      /// Pheromone decrease factor.
      /// </summary>
      static double rho = 0.5; // pheromone decrease factor
      /// <summary>
      /// Pheromone increase factor.
      /// </summary>
      static double Q = 100;
```

```
/// <summary>
/// Initial pheromone value for every edge.
/// </summary>
static double tau0 = Math.Pow(0.1, 5);

/// <summary>
/// Number of ants in the TSP problem are set in the Initialize method.
/// </summary>
static int numAnts = 0;

/// <summary>
/// Number of elitist ants
/// </summary>
static int eAnts = 5;

/// <summary>
/// Vertices of the TSP problem are set in the Initialize method.
/// </summary>
static double[,] vertices;
/// <summary>
/// Number of cities in the TSP problem are set in the Initialize method.
/// </summary>
static int numCities = 0;
/// <summary>
/// File path where the TSP problem is found.
/// </summary>
static string filePath = @"c:\temp\";

/*Main Program*/
/// <summary>
/// Console application for Ant System. User sees the results on
/// on a command promt.
/// </summary>
/// <param name="args">The args.</param>
static void Main(string[] args)
{
   try
   {
      Console.WriteLine("\nGive the TSP to solve:");
      Console.WriteLine("\tType 1 for rd100");
      Console.WriteLine("\tType 2 for berlin52");
      Console.WriteLine("\tType 3 for pr152");
      var input = Console.ReadKey();

      Initialize(input);

      stopWatch.Start();
      Console.WriteLine("\nBegin Ant System Optimization demo\n");
```

```
int maxTime = 1000;
double currLength = 0;
string elapsedTime = null;

Console.WriteLine("Number cities in problem = " + numCities);

Console.WriteLine("\nNumber ants = " + numAnts);
Console.WriteLine("Maximum time = " + maxTime);

Console.WriteLine("\nAlpha (pheromone influence) = " + alpha);
Console.WriteLine("Beta (local node influence) = " + beta);
Console.WriteLine("Rho (pheromone evaporation coefficient) = " +
 rho.ToString("F2"));
Console.WriteLine("Q (pheromone deposit factor) = " +
 Q.ToString("F2"));

Console.WriteLine("\nInitialing graph distances");
int[][] dists = MakeGraphDistances(numCities);

Console.WriteLine("\nInitialing ants to random trails\n");
int[][] ants = InitAnts(numAnts, numCities);
ShowAnts(ants, dists);

int[] bestTrail = BestTrail(ants, dists);
double bestLength = Length(bestTrail, dists);

Console.Write("\nBest initial trail length: " + bestLength.ToString("F1") +
 "\n");

Console.WriteLine("\nInitializing pheromones on trails");
double[][] pheromones = InitPheromones(numCities);

int time = 0;
Console.WriteLine("\nEntering UpdateAnts - UpdatePheromones loop\n");
while (time < maxTime)
{
   UpdateAnts(ants, pheromones, dists);

   int[] currBestTrail = BestTrail(ants, dists);
   currLength = Length(currBestTrail, dists);
   UpdatePheromones(pheromones, bestLength, currLength, ants, dists);

   if (currLength < bestLength)
   {
      bestLength = currLength;
      bestTrail = currBestTrail;
      Console.WriteLine("New best length of " +
      bestLength.ToString("F1") + " found at time " + time);
      elapsedTime = stopWatch.ElapsedMilliseconds.ToString();
```

```
            Console.WriteLine("Best tour was found in " + elapsedTime + " mil-
            liseconds");
          }

          ++time;
        }

        Console.WriteLine("\nTime complete");

        Console.WriteLine("\nBest trail found:");
        Display(bestTrail);
        Console.WriteLine("\nLength of best trail found: " +
         bestLength.ToString("F1"));

        Console.WriteLine("\nEnd Ant System Optimization demo\n");
        elapsedTime = stopWatch.ElapsedMilliseconds.ToString();
        Console.WriteLine("The whole thing took " + elapsedTime + "
         miliseconds");
        stopWatch.Stop();
        Console.ReadLine();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }

} // Main

// ------------------------------------------------------------------------------------
/// <summary>
/// Initialices an ant to a random starting node.
/// </summary>
/// <param name="numAnts">Number of ants in the problem.</param>
/// <param name="numCities">Number of the cities in the problem.</param>
/// <returns>A table where the trail for an ant is stored.</returns>
static int[][] InitAnts(int numAnts, int numCities)
{
    int[][] ants = new int[numAnts][];
    for (int k = 0; k < numAnts; ++k)
    {
        int start = random.Next(0, numCities);
        ants[k] = RandomTrail(start, numCities);
    }
    return ants;
}

/// <summary>
/// Randomizes the trail for an ant.
```

```
        /// </summary>
        /// <param name="start">A starting city for an ant.</param>
        /// <param name="numCities">Number of the cities in the problem.</param>
        /// <returns>Ant´s next target node.</returns>
        static int[] RandomTrail(int start, int numCities) // helper for InitAnts
        {
           int[] trail = new int[numCities];

           for (int i = 0; i < numCities; ++i) { trail[i] = i; } // sequential

           for (int i = 0; i < numCities; ++i) // Fisher-Yates shuffle
           {
              int r = random.Next(i, numCities);
              int tmp = trail[r]; trail[r] = trail[i]; trail[i] = tmp;
           }

           int idx = IndexOfTarget(trail, start); // put start at [0]
           int temp = trail[0];
           trail[0] = trail[idx];
           trail[idx] = temp;

           return trail;
        }

        /// <summary>
        /// Checks if the target(start) node is found on the first place from the
        /// ant´s trail list.
        /// </summary>
        /// <param name="trail">The trail of an ant.</param>
        /// <param name="target">The target(start) node of an ant.</param>
        /// <returns>The target node.</returns>
        /// <exception cref="System.Exception">Target not found in
IndexOfTarget</exception>
        static int IndexOfTarget(int[] trail, int target) // helper for RandomTrail
        {
           for (int i = 0; i < trail.Length; ++i)
           {
              if (trail[i] == target)
                 return i;
           }
           throw new Exception("Target not found in IndexOfTarget");
        }

        /// <summary>
        /// Calculates the total lenght of a trail
        /// </summary>
        /// <param name="trail">The trail of an ant.</param>
        /// <param name="dists">A distance matrix which contains are calculated
        /// all the distances between nodes.</param>
```

```
/// <returns>The total lenght of a trail.</returns>
static double Length(int[] trail, int[][] dists) // total length of a trail
{
   double result = 0.0;
   for (int i = 0; i < trail.Length - 1; ++i)
      result += Distance(trail[i], trail[i + 1], dists);
   return result;
}

// ------------------------------------------------------------------------------------------
/// <summary>
/// Compares the trails found by an ant and returns the shortest one.
/// </summary>
/// <param name="ants">A matrix which contains are an ant and a node
/// where the ant is situated.</param>
/// <param name="dists">A Distance matrix that contains all the
/// distances between nodes.</param>
/// <returns>The best trail found.</returns>
static int[] BestTrail(int[][] ants, int[][] dists)
{
   double bestLength = Length(ants[0], dists);
   int idxBestLength = 0;
   for (int k = 1; k < ants.Length; ++k)
   {
      double len = Length(ants[k], dists);
      if (len < bestLength)
      {
         bestLength = len;
         idxBestLength = k;
      }
   }
   int[] bestTrail = new int[numCities];
   ants[idxBestLength].CopyTo(bestTrail, 0);

   /*best trail has shortest total length*/
   return bestTrail;
}

// ------------------------------------------------------------------------------------------
/// <summary>
/// Initializes the pheromone value on all edges.
/// </summary>
/// <param name="numCities">Number of the cities in the problem.<</param>
/// <returns>A Matrix with initialized pheromone values.</returns>
static double[][] InitPheromones(int numCities)
{
   double[][] pheromones = new double[numCities][];
   for (int i = 0; i < numCities; ++i)
      pheromones[i] = new double[numCities];
```

```
   for (int i = 0; i < pheromones.Length; ++i)
      for (int j = 0; j < pheromones[i].Length; ++j)
         pheromones[i][j] = tau0;
   return pheromones;
}

// ------------------------------------------------------------------------------------
/*updates a new starting city for an ant to build a new trail*/
/// <summary>
/// Updates a ne trail for an ant.
/// </summary>
/// <param name="ants">A matrix which contains an ant and a node where
/// the ant is situated.</param>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.</param>
static void UpdateAnts(int[][] ants, double[][] pheromones, int[][] dists)
{
   for (int k = 0; k < ants.Length; ++k)
   {
      int start = random.Next(0, numCities);
      int[] newTrail = BuildTrail(start, pheromones, dists);
      ants[k] = newTrail;
   }
}

/// <summary>
/// Builds the trail fo an ant.
/// </summary>
/// <param name="start">A starting city for an ant.</param>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.</param>
/// <returns>A new trail for an ant.</returns>
static int[] BuildTrail(int start, double[][] pheromones, int[][] dists)
{
   int[] trail = new int[numCities];
   bool[] visited = new bool[numCities];
   trail[0] = start;
   visited[start] = true;
   for (int i = 0; i < numCities - 1; ++i)
   {
      int cityX = trail[i];
      int next = NextCity(cityX, visited, pheromones, dists);
      trail[i + 1] = next;
      visited[next] = true;
   }
```

```
    return trail;
}

/// <summary>
/// Calculates the next city for an ant to go to.
/// </summary>
/// <param name="cityX">Ant´s current city.</param>
/// <param name="visited">A Boolean value which tells if the city has
/// been visited or not.</param>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.<</param>
/// <returns>An index of a next target city.</returns>
/// <exception cref="System.Exception">Failure to return valid city
/// in NextCity</exception>
static int NextCity(int cityX, bool[] visited, double[][] pheromones, int[][] dists)
{
    // for ant k (with visited[]), at nodeX, what is next node in trail?
    double[] probs = MoveProbs(cityX, visited, pheromones, dists);

    double maxValue = probs[0];
    int index = 0;
    for (int i = 0; i < probs.Length; ++i)
        if (maxValue < probs[i])
        {
            maxValue = probs[i];
            index = i;
        }
    return index;
    throw new Exception("Failure to return valid city in NextCity");
}

/// <summary>
/// Calculates the probabilities to choose next city as a next target.
/// </summary>
/// <param name="cityX">Ant´s current city.</param>
/// <param name="visited">A Boolean value which tells if the city has
/// been visited or not.</param>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.</param>
/// <returns>A matrix containing the calculated probabilites.</returns>
static double[] MoveProbs(int cityX, bool[] visited, double[][] pheromones,
 int[][] dists)
{
    // for ant k, located at nodeX, with visited[], return the prob of moving to
    ///each city
```

```
    /* inclues cityX and visited cities*/
    double[] taueta = new double[numCities];
    /* table of probabilities*/
    double[] probs = new double[numCities];
    double sum = 0.0; // sum of all tauetas
    for (int i = 0; i < taueta.Length; ++i) // i is the adjacent city
    {
       if (i == cityX)
          taueta[i] = 0.0; // prob of moving to self is 0
       else if (visited[i] == true)
          taueta[i] = 0.0; // prob of moving to a visited city is 0
       else
       {
          taueta[i] = Math.Pow(pheromones[cityX][i], alpha) * Math.Pow((1.0 /
         Distance(cityX, i, dists)), beta);
          double minTaueta = Math.Pow(0.1, 30);
          if (taueta[i] < minTaueta)
             taueta[i] = minTaueta;
       }
       sum += taueta[i];
    }

    if (sum == 0)
       throw new Exception("Sum is zero");

    for (int i = 0; i < probs.Length; ++i)
       probs[i] = taueta[i] / sum;
    return probs;
 }

 // ------------------------------------------------------------------------------------
 /// <summary>
 /// Pheromone updating rule.
 /// </summary>
 /// <param name="pheromones">A matrix which contains pheromone values for
 /// all edges.</param>
 /// <param name="ants">A matrix which contains an ant and a node where the
 /// ant is situated.</param>
 /// <param name="dists">A distance matrix which contains all the distances
 /// between nodes.</param>
 static void UpdatePheromones(double[][] pheromones, double bestLength, dou
ble currLength, int[][] ants, int[][] dists)
 {
    for (int i = 0; i < pheromones.Length; ++i)
    {
       for (int j = i + 1; j < pheromones[i].Length; ++j)
       {
          for (int k = 0; k < ants.Length; ++k)
          {
```

```
            double length = Length(ants[k], dists); // length of ant k trail
            double decrease = (1.0 - rho) * pheromones[i][j];
            double increase = 0.0;
            double eliteIncrease = 0.0;
            if (EdgeInTrail(i, j, ants[k]) == true)
            {
               increase = (Q / length);
               if (currLength < bestLength)
                  eliteIncrease = eAnts * (Q / currLength);
            }
            pheromones[i][j] = decrease + increase + eliteIncrease;

            pheromones[j][i] = pheromones[i][j];
         }
      }
   }
}

/// <summary>
/// Checks if cityX and cityY are adjacent to each other on the ant´s trail.
/// </summary>
/// <param name="cityX">Ant´s current city.</param>
/// <param name="cityY">Ant´s target city.</param>
/// <param name="trail">The trail of an ant.</param>
/// <returns>True or false depending if cityX and cityY are adjacent to
///each other
/// on the ant´s trail.</returns>
static bool EdgeInTrail(int cityX, int cityY, int[] trail)
{
   // are cityX and cityY adjacent to each other in trail[]?
   int lastIndex = trail.Length - 1;
   int idx = IndexOfTarget(trail, cityX);

   if (idx == 0 && trail[1] == cityY) return true;
   else if (idx == 0 && trail[lastIndex] == cityY) return true;
   else if (idx == 0) return false;
   else if (idx == lastIndex && trail[lastIndex - 1] == cityY) return true;
   else if (idx == lastIndex && trail[0] == cityY) return true;
   else if (idx == lastIndex) return false;
   else if (trail[idx - 1] == cityY) return true;
   else if (trail[idx + 1] == cityY) return true;
   else return false;
}

// ----------------------------------------------------------------------------------------
/// <summary>
/// Creates a distance matrix from a TSPLIB95 text file coordinates.
/// </summary>
/// <param name="numCities">The number of cities in the problem.</param>
```

```
/// <returns>A distance matrix which contains all the distances between
///cities.</returns>
static int[][] MakeGraphDistances(int numCities)
{
   int[][] dists = new int[numCities][];
   for (int i = 0; i < dists.Length; ++i)
      dists[i] = new int[numCities];

   for (int i = 0; i < numCities; ++i)
      for (int j = 0; j < numCities; ++j)
         dists[i][j] = (int)Math.Sqrt(Math.Pow(vertices[i, 0] - vertices[j, 0], 2) +
        Math.Pow(vertices[i, 1] - vertices[j, 1], 2));
   return dists;
}

/// <summary>
/// Helper method which returns the distance of a given cities.
/// </summary>
/// <param name="cityX">The city X.</param>
/// <param name="cityY">The city Y.</param>
/// <param name="dists">A distance matrix which contains all the distances
/// between nodes.</param>
/// <returns>The distance between city X and city Y.</returns>
static double Distance(int cityX, int cityY, int[][] dists)
{
   return dists[cityX][cityY];
}

// ------------------------------------------------------------------------------------------
/// <summary>
/// Displays the specified trail in a command promt.
/// </summary>
/// <param name="trail">The trail of an ant.</param>
static void Display(int[] trail)
{
   for (int i = 0; i < trail.Length; ++i)
   {
      Console.Write(trail[i] + " ");
      if (i > 0 && i % 20 == 0) Console.WriteLine("");
   }
   Console.WriteLine("");
}


/// <summary>
/// Shows the ants in a command promt.
/// </summary>
/// <param name="ants">A matrix which contains an ant and a node where the
/// ant is situated.</param>
```

```
/// <param name="dists">A distance matrix which contains all the distances
/// between nodes.</param>
static void ShowAnts(int[][] ants, int[][] dists)
{
   for (int i = 0; i < ants.Length; ++i)
   {
      Console.Write(i + ": [ ");

      for (int j = 0; j < 4; ++j)
         Console.Write(ants[i][j] + " ");

      Console.Write(". . . ");

      for (int j = ants[i].Length - 4; j < ants[i].Length; ++j)
         Console.Write(ants[i][j] + " ");

      Console.Write("] len = ");
      double len = Length(ants[i], dists);
      Console.Write(len.ToString("F1"));
      Console.WriteLine("");
   }
}

/// <summary>
/// Reads an input from a keyboard and creates a corresponding filename.
/// Creates an instance of TSPLIBParser and parses the tsp file.
/// </summary>
/// <param name="input">The input from a keyboard.</param>
static void Initialize(ConsoleKeyInfo input)
{
   string fileName = "";

   switch (input.KeyChar)
   {
      case '1':
         fileName = filePath + "rd100.tsp";
         break;
      case '2':
         fileName = filePath + "berlin52.tsp";
         break;
      case '3':
         fileName = filePath + "pr152.tsp";
         break;
   }


   try
   {
      TSPLIBParser parser = new TSPLIBParser(fileName);
```

```
            parser.Parse();
            vertices = parser.Vertices;
            numCities = vertices.GetLength(0);
            numAnts = numCities;
        }
        catch (Exception e)
        {
            Console.Write("Exception: " + e + ". Probably because of file not found!");
        }
    }

  } // class AntSystem

} // ns
```

Liite 4: Ant Colony System -algoritmin C#-lähdekoodi

```
using System;
using System.IO;
using TSPParser;
using System.Diagnostics;

namespace AntSystem
{
   /// <summary>
   /// Demo of Ant Colony System (ACS) solving a Traveling Salesman
   ///Problem (TSP).
   /// The core of this implementation is made by James McCaffreys. I have added
   /// few new funtionalities such as this impelentation uses knows TSP problems
   /// from the TSPLib95 library and removed all the min and max values of
   /// pheromones. There are free parameters beta, rho, and q0 and tau0 which are
   /// freely adjustable within this code.  FilePath is the location
   /// of the TSP -instances in my computer.
   /// </summary>
   class AntSystemProgram
   {
      /// <summary>
      /// Instance of a Random class for randomness.
      /// </summary>
      static Random random = new Random(DateTime.UtcNow.Millisecond);

      /// <summary>
      /// Instance of a stopWatch class to measure running time of this algorithm.
      /// </summary>
      static Stopwatch stopWatch = new Stopwatch();

      /// <summary>
      /// Variable beta.
      /// </summary>
      static int beta = 2;  // influence of adjacent node distance.

      /// <summary>
      /// Pheromone decrease factor.
      /// </summary>
      static double rho = 0.1;

      /// <summary>
      /// A tunable parameter between [0,1] which is used in
      /// MoveProbs(int k, int cityX, bool[] visited, double[][] pheromones,  int[][]
      ///dists).
      /// </summary>
      static double q0 = 0.9;

      /// <summary>
```

```
/// Initial value of pheromone trails
/// </summary>
static double tau0 = Math.Pow(0.1, 5);

/// <summary>
/// Number of ants in the TSP problem are set in the Initialize method.
/// </summary>
static int numAnts = 0;

/// <summary>
/// Ant´s current city.
/// </summary>
static int cityX = 0;

/// <summary>
/// Ant´s trail in the TSP problem.
/// </summary>
static int[] trail;

/// <summary>
/// Vertices of the TSP problem are set in the Initialize method.
/// </summary>
static double[,] vertices;
/// <summary>
/// Number of cities in the TSP problem are set in the Initialize method.
/// </summary>
static int numCities = 0;

/// <summary>
/// File path where the TSP problem is found.
/// </summary>
static string filePath = @"c:\temp\";

/*Main Program*/
/// <summary>
/// Console application for Ant Colony System. User sees the results on on a
///command promt.
/// </summary>
/// <param name="args">The args.</param>
static void Main(string[] args)
{
   try
   {
      Console.WriteLine("\nGive the TSP to solve:");
      Console.WriteLine("\tType 1 for rd100");
      Console.WriteLine("\tType 2 for berlin52");
      Console.WriteLine("\tType 3 for pr152");
      var input = Console.ReadKey();
```

```
Initialize(input);
stopWatch.Start();
Console.WriteLine("\nBegin Ant Colony System Optimization demo\n");

int maxTime = 1000;
string elapsedTime = null;

Console.WriteLine("Number cities in problem = " + numCities);

Console.WriteLine("\nNumber ants = " + numAnts);
Console.WriteLine("Maximum time = " + maxTime);

Console.WriteLine("Beta (local node influence) = " + beta);
Console.WriteLine("Rho (pheromone evaporation coefficient) = " +
rho.ToString("F2"));

Console.WriteLine("\nInitialing graph distances");
int[][] dists = MakeGraphDistances(numCities);

Console.WriteLine("\nInitialing ants to random trails\n");
int[][] ants = InitAnts(numAnts, numCities);
ShowAnts(ants, dists);

int[] bestTrail = BestTrail(ants, dists);
double bestLength = Length(bestTrail, dists);

Console.Write("\nBest initial trail length: " + bestLength.ToString("F1") +
 "\n");

Console.WriteLine("\nInitializing pheromones on trails");
double[][] pheromones = InitPheromones(numCities);

int time = 0;
Console.WriteLine("\nEntering UpdateAnts - UpdatePheromones loop\n");
while (time < maxTime)
{
   UpdateAnts(ants, pheromones, dists);
   LocalUpdatePheromones(pheromones, trail, cityX);

   int[] currBestTrail = BestTrail(ants, dists);
   double currLength = Length(currBestTrail, dists);
   if (currLength < bestLength)
   {
      GlobalUpdatePheromones(pheromones, bestLength, currLength, ants,
      dists);
      bestLength = currLength;
      bestTrail = currBestTrail;
      Console.WriteLine("New best length of " +
      bestLength.ToString("F1") + " found at time " + time);
```

```
            elapsedTime = stopWatch.ElapsedMilliseconds.ToString();
            Console.WriteLine("Best tour was found in " + elapsedTime + "
            miliseconds");
          }
          ++time;
        }

        Console.WriteLine("\nTime complete");

        Console.WriteLine("\nBest trail found:");
        Display(bestTrail);
        Console.WriteLine("\nLength of best trail found: " +
         bestLength.ToString("F1"));

        Console.WriteLine("\nEnd Ant Colony System Optimization demo\n");
        elapsedTime = stopWatch.ElapsedMilliseconds.ToString();
        Console.WriteLine("The whole thing took " + elapsedTime + " millisec-
         onds");
        stopWatch.Stop();
        Console.ReadLine();
      }
      catch (Exception ex)
      {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
      }
    } // Main

    // ------------------------------------------------------------------------------------------
    /// <summary>
    /// Initialices an ant to a random starting node.
    /// </summary>
    /// <param name="numAnts">Number of ants in the problem.</param>
    /// <param name="numCities">Number of the cities in the problem.</param>
    /// <returns>A table where the trail for an ant is stored.</returns>
    static int[][] InitAnts(int numAnts, int numCities)
    {
      int[][] ants = new int[numAnts][];
      for (int k = 0; k < numAnts; ++k)
      {
        int start = random.Next(0, numCities);
        ants[k] = RandomTrail(start, numCities);
      }
      return ants;
    }

    /// <summary>
    /// Randomizes the trail for an ant.
    /// </summary>
```

```
/// <param name="start">A starting city for an ant.</param>
/// <param name="numCities">Number of the cities in the problem.</param>
/// <returns>Ant´s trail.</returns>
static int[] RandomTrail(int start, int numCities) // helper for InitAnts
{

   trail = new int[numCities];

   for (int i = 0; i < numCities; ++i) { trail[i] = i; }

   for (int i = 0; i < numCities; ++i) // Fisher-Yates shuffle
   {
      int r = random.Next(i, numCities);
      int tmp = trail[r]; trail[r] = trail[i]; trail[i] = tmp;
   }

   int idx = IndexOfTarget(trail, start); // put start at [0]
   int temp = trail[0];
   trail[0] = trail[idx];
   trail[idx] = temp;

   return trail;
}

/// <summary>
/// Checks if the targhet node is found from the ant´s trail list.
/// </summary>
/// <param name="trail">The trail of an ant.</param>
/// <param name="target">The target of an ant.</param>
/// <returns>The target node.</returns>
/// <exception cref="System.Exception">Target not found in
/// IndexOfTarget</exception>
static int IndexOfTarget(int[] trail, int target) // helper for RandomTrail
{
   for (int i = 0; i < trail.Length; ++i)
   {
      if (trail[i] == target)
         return i;
   }
   throw new Exception("Target not found in IndexOfTarget");
}

/// <summary>
/// Calculates the total lenght of a trail
/// </summary>
/// <param name="trail">The trail of an ant.</param>
/// <param name="dists">A distance matrix which contains all calculated the
/// distances between nodes.</param>
/// <returns>The total lenght of a trail.</returns>
```

```
static double Length(int[] trail, int[][] dists)
{
   double result = 0.0;
   for (int i = 0; i < trail.Length - 1; ++i)
      result += Distance(trail[i], trail[i + 1], dists);
   return result;
}

// ----------------------------------------------------------------------------------------
/// <summary>
/// Compares the trails found by an ant and returns the shortest one.
/// </summary>
/// <param name="ants">A matrix which contains are an ant and a node where
/// the ant is situated.</param>
/// <param name="dists">A Distance matrix that contains all the distances
/// between nodes.</param>
/// <returns>The best trail found.</returns>
static int[] BestTrail(int[][] ants, int[][] dists)
{
   double bestLength = Length(ants[0], dists);
   int idxBestLength = 0;
   for (int k = 1; k < ants.Length; ++k)
   {
      double len = Length(ants[k], dists);
      if (len < bestLength)
      {
         bestLength = len;
         idxBestLength = k;
      }
   }
   int numCities = ants[0].Length;
   int[] bestTrail = new int[numCities];
   ants[idxBestLength].CopyTo(bestTrail, 0);
   return bestTrail;
}

// ----------------------------------------------------------------------------------------
/// <summary>
/// Initializes the pheromone value on all edges.
/// </summary>
/// <param name="numCities">Number of the cities in the problem.<</param>
/// <returns>A Matrix with initialized pheromone values.</returns>
static double[][] InitPheromones(int numCities)
{
   double[][] pheromones = new double[numCities][];
   for (int i = 0; i < numCities; ++i)
      pheromones[i] = new double[numCities];
   for (int i = 0; i < pheromones.Length; ++i)
      for (int j = 0; j < pheromones[i].Length; ++j)
```

```
            pheromones[i][j] = tau0;
    return pheromones;
}

// ------------------------------------------------------------------------------------
/// <summary>
/// Updates a new trail for an ant.
/// </summary>
/// <param name="ants">A matrix which contains an ant and a node where
/// the ant is situated.</param>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.</param>
static void UpdateAnts(int[][] ants, double[][] pheromones, int[][] dists)
{
    for (int k = 0; k < ants.Length; ++k)
    {
        int start = random.Next(0, numCities);
        int[] newTrail = BuildTrail(start, pheromones, dists, cityX);
        ants[k] = newTrail;
    }
}

/// <summary>
/// Builds the trail fo an ant.
/// </summary>
/// <param name="start">A starting city for an ant.</param>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.</param>
/// <returns>A new trail for an ant.</returns>
static int[] BuildTrail(int start, double[][] pheromones, int[][] dists, int cityX)
{
    /*Initializes the trail*/
    for (int i = 0; i < numCities; ++i)
    {
        trail[i] = i;
    }

    int next = -1;
    bool[] visited = new bool[numCities];
    trail[0] = start;
    visited[start] = true;
    for (int i = 0; i < numCities - 1; ++i)
    {
        cityX = trail[i];
        next = NextCity(cityX, visited, pheromones, dists);
```

```
        trail[i + 1] = next;
        visited[next] = true;
    }
    return trail;
}

/// <summary>
/// Calculates the next city for an ant to go to.
/// </summary>
/// <param name="cityX">Ant´s current city.</param>
/// <param name="visited">A Boolean value which tells if the city has
/// been visited or not.</param>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.<</param>
/// <returns>An index of a next target city.</returns>
/// <exception cref="System.Exception">Failure to return valid city
/// in NextCity</exception>
static int NextCity(int cityX, bool[] visited, double[][] pheromones, int[][] dists)
{
    // for ant k (with visited[]), at nodeX, what is next node in trail?
    double[] probs = MoveProbs(cityX, visited, pheromones, dists);

    double maxValue = probs[0];
    int index = 0;
    for (int i = 0; i < probs.Length; ++i)
        if (maxValue < probs[i])
        {
            maxValue = probs[i];
            index = i;
        }
    return index;
    throw new Exception("Failure to return valid city in NextCity");
}




/// <summary>
/// Calculates the probabilities to choose next city as a next target.
/// <param name="cityX">Ant´s current city.</param>
/// <param name="visited">A Boolean value which tells if the city has
/// been visited or not.</param>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.</param>
/// <returns>A matrix containing the calculated probabilites.</returns>
```

```
static double[] MoveProbs(int cityX, bool[] visited, double[][] pheromones,
int[][] dists)
 {
    double q = random.NextDouble();

    int numCities = pheromones.Length;
    /* includes cityX and visited cities*/
    double[] taueta = new double[numCities];
    /* table of probabilities*/
    double[] probs = new double[numCities];
    double sum = 0.0; // sum of all tauetas


    if (q <= q0)
    {
       for (int i = 0; i < probs.Length; ++i) // i is the adjacent city
       {
          if (i == cityX)
             probs[i] = 0.0; // prob of moving to self is 0
          else if (visited[i] == true)
             probs[i] = 0.0; // prob of moving to a visited city is 0
          else
          {
             probs[i] = pheromones[cityX][i] * Math.Pow((1.0 / Distance(cityX, i,
             dists)), beta);
             double minProbs = Math.Pow(0.1, 30);
             if (probs[i] < minProbs)
                probs[i] = minProbs;
          }
       }

    }
    else
    {
       for (int i = 0; i < taueta.Length; ++i) // i is the adjacent city
       {
          if (i == cityX)
             taueta[i] = 0.0; // prob of moving to self is 0
          else if (visited[i] == true)
             taueta[i] = 0.0; // prob of moving to a visited city is 0
          else
          {
             taueta[i] = pheromones[cityX][i] * Math.Pow((1.0 / Distance(cityX, i,
             dists)), beta);
             double minTaueta = Math.Pow(0.1, 30);
             if (taueta[i] < minTaueta)
                taueta[i] = minTaueta;
          }
          sum += taueta[i];
```

```
        }

        if (sum == 0)
            throw new Exception("Sum is zero");

        for (int i = 0; i < probs.Length; ++i)
            probs[i] = taueta[i] / sum;

    }
    return probs;
}

// ------------------------------------------------------------------------------------
/// <summary>
/// Local pheromone update rule.
/// </summary>
/// <param name="pheromones">A matrix which contains pheromone values
/// for all edges.</param>
/// <param name="trail">A matrix which contains ant´s trail.</param>
/// <param name="cityX">Ant´s current city.</param>
static void LocalUpdatePheromones(double[][] pheromones, int[] trail, int
cityX)
{
    int[] antsTrail = trail;

    for (int i = 0; i < antsTrail.Length - 1; ++i)
        if (cityX == trail[i])
        {
            double decrease = (1.0 - rho) * pheromones[trail[i]][trail[i + 1]] + rho *
          tau0;

            pheromones[trail[i]][trail[i + 1]] = decrease;
            pheromones[trail[i + 1]][trail[i]] = pheromones[trail[i]][trail[i + 1]];
        }
}

/// <summary>
/// Global pheromone update rule.
/// </summary>
/// <param name="pheromones">A matrix which contains pheromone values
/// all edges.</param>
/// <param name="ants">A matrix which contains an ant and a node where
/// the ant is situated.</param>
/// <param name="dists">A distance matrix which contains all the distances
/// between nodes.</param>
static void GlobalUpdatePheromones(double[][] pheromones, double
bestLength, double currLength, int[][] ants, int[][] dists)
{
    for (int i = 0; i < pheromones.Length; ++i)
```

```
    {
      for (int j = i + 1; j < pheromones[i].Length; ++j)
      {
        for (int k = 0; k < ants.Length; ++k)
        {
          if (EdgeInTrail(i, j, ants[k]) == true)
            if (currLength < bestLength)
            {
              /* length of ant k trail */
              double length = Length(ants[k], dists);
              double decrease = (1.0 - rho) * pheromones[i][j];
              double increase = 0.0;

              increase = (1 / currLength);

              pheromones[i][j] = decrease + rho * increase;
              pheromones[j][i] = pheromones[i][j];
            }
        }
      }
    }
}

/// <summary>
/// Checks if cityX and cityY are adjacent to each other on the ant´s trail.
/// </summary>
/// <param name="cityX">Ant´s current city.</param>
/// <param name="cityY">Ant´s target city.</param>
/// <param name="trail">The trail of an ant.</param>
/// <returns>True or false depending if cityX and cityY are adjacent to each
/// other on the ant´s trail.</returns>
static bool EdgeInTrail(int cityX, int cityY, int[] trail)
{
  int lastIndex = trail.Length - 1;
  int idx = IndexOfTarget(trail, cityX);

  if (idx == 0 && trail[1] == cityY) return true;
  else if (idx == 0 && trail[lastIndex] == cityY) return true;
  else if (idx == 0) return false;
  else if (idx == lastIndex && trail[lastIndex - 1] == cityY) return true;
  else if (idx == lastIndex && trail[0] == cityY) return true;
  else if (idx == lastIndex) return false;
  else if (trail[idx - 1] == cityY) return true;
  else if (trail[idx + 1] == cityY) return true;
  else return false;
}

// ------------------------------------------------------------------------------------
/// <summary>
```

```
/// Creates a distance matrix from a TSPLIB95 text file coordinates.
/// </summary>
/// <param name="numCities">The number of cities in the problem.</param>
/// <returns>A distance matrix which contains all the distances between
///cities.</returns>
static int[][] MakeGraphDistances(int numCities)
{
   int[][] dists = new int[numCities][];
   for (int i = 0; i < dists.Length; ++i)
      dists[i] = new int[numCities];

   for (int i = 0; i < numCities; ++i)
      for (int j = 0; j < numCities; ++j)
         dists[i][j] = (int)Math.Sqrt(Math.Pow(vertices[i, 0] - vertices[j, 0], 2) +
        Math.Pow(vertices[i, 1] - vertices[j, 1], 2));
   return dists;
}

/// <summary>
/// Helper method which returns the distance of a given cities.
/// </summary>
/// <param name="cityX">Ant´s current city.</param>
/// <param name="cityY">Ant´s target city.</param>
/// <param name="dists">A distance matrix which contains all the distances
/// between nodes.</param>
/// <returns>The distance between city X and city Y.</returns>
static double Distance(int cityX, int cityY, int[][] dists)
{
   return dists[cityX][cityY];
}

// ----------------------------------------------------------------------------------------
/// <summary>
/// Displays the specified trail in a command promt.
/// </summary>
/// <param name="trail">The trail of an ant.</param>
static void Display(int[] trail)
{
   for (int i = 0; i < trail.Length; ++i)
   {
      Console.Write(trail[i] + " ");
      if (i > 0 && i % 20 == 0) Console.WriteLine("");
   }
   Console.WriteLine("");
}


/// <summary>
/// Shows the ants in a command promt.
```

```
/// </summary>
/// <param name="ants">A matrix which contains an ant and a node where
/// the ant is situated.</param>
/// <param name="dists">A distance matrix which contains all the
/// distances between nodes.</param>
static void ShowAnts(int[][] ants, int[][] dists)
{
   for (int i = 0; i < ants.Length; ++i)
   {
      Console.Write(i + ": [ ");

      for (int j = 0; j < 4; ++j)
         Console.Write(ants[i][j] + " ");

      Console.Write(". . . ");

      for (int j = ants[i].Length - 4; j < ants[i].Length; ++j)
         Console.Write(ants[i][j] + " ");

      Console.Write("] len = ");
      double len = Length(ants[i], dists);
      Console.Write(len.ToString("F1"));
      Console.WriteLine("");
   }
}


/// <summary>
/// Reads an input from a keyboard and creates a corresponding filename.
/// Creates an instance of TSPLIBParser and parses the tsp file.
/// </summary>
/// <param name="input">The input from a keyboard.</param>
static void Initialize(ConsoleKeyInfo input)
{
   string fileName = "";

   switch (input.KeyChar)
   {
      case '1':
         fileName = filePath + "rd100.tsp";
         break;
      case '2':
         fileName = filePath + "berlin52.tsp";
         break;
      case '3':
         fileName = filePath + "pr152.tsp";
         break;
   }
```

```
            try
            {
               TSPLIBParser parser = new TSPLIBParser(fileName);
               parser.Parse();
               vertices = parser.Vertices;
               numCities = vertices.GetLength(0);
               numAnts = numCities;
               trail = new int[numCities];
            }
            catch (Exception e)
            {
               Console.Write("Exception: " + e + ". Probably because of file not found!");
            }
}
   } // class AntColonyProgram
} // ns
```

Liite 5: Ant Colony System -algoritmin suoritusajat ja löydettyjen reittien pituudet muuttujan τ_0 arvojen vaihdellessa

Taulukko 13: Ant Colony System -algoritmin suoritusaikojen ja reittien pituuksien vaihtelu muuttujan τ_0 vaihdellessa.

Reitin pituus ja aika (ms)	Mittaustulokset						
	$\tau_0 = (n * L_{nn})^{-1}$				$\tau_0 = 1 * 10^{-5}$		
	rd100	berlin52	pr152		rd100	berlin52	pr152
Reitin pituus	8906	7859	68587		8986	7622	69211
Aika	83707	11818	295162		89038	11993	300724
Reitin pituus	8310	7199	69745		8833	8021	69892
Aika	84227	11811	299907		85526	12008	300887
Reitin pituus	8918	7816	70121		8904	7366	69029
Aika	84416	11940	293355		85226	12068	300057
Reitin pituus	9106	7619	69178		9086	7551	68871
Aika	89445	12006	297266		85221	12256	300813
Reitin pituus	8885	7915	68591		8859	7909	68591
Aika	86689	12229	300977		86737	12303	302384
Reitin pituus (keskiarvo)	8804,7	7637,2	69408,7		8920,5	7711,8	69118,8
Aika (keskiarvo)	85696,8	11996,5	297333,4		85677,5	12159	300973

Liite 6: Microsoft Public Licence

MICROSOFT PUBLIC LICENSE (Ms-PL)

This license governs use of the accompanying software. If you use the software, you accept this license. If you do not accept the license, do not use the software.

1. Definitions

The terms "reproduce," "reproduction," "derivative works," and "distribution" have the same meaning here as under U.S. copyright law.

A "contribution" is the original software, or any additions or changes to the software.

A "contributor" is any person that distributes its contribution under this license.

"Licensed patents" are a contributor's patent claims that read directly on its contribution.

2. Grant of Rights

(A) Copyright Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free copyright license to reproduce its contribution, prepare derivative works of its contribution, and distribute its contribution or any derivative works that you create.

(B) Patent Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free license under its licensed patents to make, have made, use, sell, offer for sale, import, and/or otherwise dispose of its contribution in the software or derivative works of the contribution in the software.

3. Conditions and Limitations

(A) No Trademark License- This license does not grant you rights to use any contributors' name, logo, or trademarks.

(B) If you bring a patent claim against any contributor over patents that you claim are infringed by the software, your patent license from such contributor to the software ends automatically.

(C) If you distribute any portion of the software, you must retain all copyright, patent, trademark, and attribution notices that are present in the software.

(D) If you distribute any portion of the software in source code form, you may do so only under this license by including a complete copy of this license with your distribution. If you distribute any portion of the software in compiled or object code form, you may only do so under a license that complies with this license.

(E) The software is licensed "as-is." You bear the risk of using it. The contributors give no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this license cannot change. To the extent permitted under your local laws, the contributors exclude the implied warranties of merchantability, fitness for a particular purpose and non-infringement.