

Jyry Uitto

MUURAHAIKYHDYSKUNNAT HAJAUTETUN LASKENNAN ALGORITMIEN POHJANA

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Huhtikuu 2020

TIIVISTELMÄ

Jyry Uitto: Muurahaisyhdyskunnat hajautetun laskennan algoritmien pohjana

Kandidaattitutkielma

Tampereen yliopisto

Tietojenkäsittelytieteiden tutkinto-ohjelma

Huhtikuu 2020

Muurahaisyhdyskunnat toimivat hajautetun laskennan inspiraationa, niistä on otettu mallia niin tasolta pisteen optimaaliseen etsimiseen kuin laskentaverkkojen optimointiin. Tässä tutkielmassa perehdytään tason tutkimisen ongelmiin ja sen käsittelyyn luotuun ANTS-ongelmaan ja sen ratkaisuväriatioihin. Lisäksi perehdytään laskentaverkkojen optimointiin käytettyihin ratkaisuihin. Tarkempaan tarkasteluun tulee ANTS-ongelman aikavaatimukset, rajoitteet, ratkaisujen virheensietokyky, sekä yksittäisten agenttien rakennevaatimukset niin oraakkelin johdolla, että ilman. Tutkielmassa käydään läpi myös muurahaisyhdyskuntien tuomia ratkaisuja verkkolaskentaympäristöjen optimointiin, myös sen ratkaisuväriatioihin ja virheensietokykyyn. Tutkielma tuo yhteen hajautetun laskennan tutkimustuloksia.

Avainsanat: Hajautettu laskenta, algoritmit, taso, ANTS, laskentaverkko

1. Johdanto	1
2. Tason tutkiminen	1
2.1 Aikavaatimuksia erilaisten kohteiden löytämiselle	2
2.2 Tason tutkimisen ANTS-ongelma ja sen variaatioiden ratkaisuja	3
2.3 Muita tason tutkimuksia	9
3. Muurahaisyhdyskunta-algoritmi	10
3.1 Muurahaisyhdyskunta-algoritmi verkkolaskentaympäristössä	11
4. Yhteenveto	14
Lähdeluettelo	14

Tutkielmassa käytettyjä termejä:

- **Taso:** Ruuduista koostuva taso, koordinaatisto.
- **Agentti:** Yksittäinen laskentayksikkö, joka käyttää sille annettua algoritmia tai automaattia. Esimerkiksi muurahainen.
- **Automaatti:** Yksinkertainen algoritmin mukaan tiloissa siirtyvä kone.
- **Äärellinen automaatti:** Automaatti, jolla on äärellinen määrä tiloja.
- **Deterministinen automaatti:** Kaikki automaatin algoritmin tilasiirtymät ovat yksiselitteisiä.
- **Epädeterministinen automaatti:** Osa valinnoista on moniselitteisiä ja automaatti joutuu valitsemaan vaihtoehtojen välillä.
- **Satunnainen epädeterministinen automaatti:** Valinnat tehdään satunnaisuuteen nojautuen, esimerkiksi heittämällä kolikkoa.
- **Pinoautomaatti:** Automaatilla on mahdollisuus käyttää pino-tietorakennetta tilasiirtymisessä.
- **Hajautettu laskenta:** Laskenta tehdään monella laskentayksiköllä.
- **Synkronoitu:** Kaikki laskenta tapahtuu samanaikaisesti ja samalla nopeudella.
- **Epäsynkronoitu:** Kaikki laskenta ei välttämättä tapahdu samanaikaisesti ja samalla nopeudella.
- **Ants Nearby Treasure Search (ANTS)-ongelma:** Ongelmassa kuvataan, kuinka löydetään kohde äärettömästä tasosta. Käytännön esimerkistä nimensäkin saanut ongelma: Kuinka muurahaiset löytävät ruuan tuntemattomasta pesäänsä ympäröivästä maastosta, mikä kuvautuu helposti tasoksi.
- **Oraakkeli:** Kaikkietävä entiteetti, joka ohjastaa agenteja antamalla agenteille tutkittavia kohteita.
- **Verkkolaskenta (Grid computing):** Laskentaa suorittava monen laskentayksikön suljettu ympäristö.

1. Johdanto

Tason tutkimiseen on luotu jo lukuisia algoritmeja. Sen tutkimiseen on luotu niin laskentateholtaan rajattomia ratkaisuja kuin laskentatehollisesti minimaalisia ratkaisuja. Tässä tutkielmassa perehdyttävään minimaalista laskentatehoa vaativiin ratkaisuihin ja yleisiin ja perinteisiin algoritmeihin, jotka näillä premiseillä on luotu. Tason tutkimisen algoritmit ovat tietojenkäsittelyn kannalta hyvin olennaisia, sillä niille löytyy myös suoria reaallimaailman käyttökohteita. Tason tutkimisen ratkaisuja voi käyttää esimerkiksi pelastustehtävissä, muun muassa meripelastus on tason tutkimista parhaimmillaan. Variaatioita meripelastusongelmallekin on huomattavia määriä, sillä kohde voi liikkua tai käytettävissä olevat resurssit ovat äärimmäisen rajatut. Muita käyttökohteita tason tutkimisen algoritmeille on äärettömästi ja niiden tutkiminen ei tule vähenemään tulevaisuudessa.

Tämän tutkielman tarkoituksena on tuoda esille, mistä hajautettu laskenta hakee inspiraatioita ongelmiinsa ja esitellä tehtyjä algoritmeja ja niiden ideoita tason tutkimiseen ja verkkolaskentaympäristön optimoimiseen. Tutkielmassa läpikäytyissä algoritmeissa on pohjana muurahaisyhdyskunnat, joiden toimintaa seuraamalla on kehitetty varsin optimaalisia ratkaisuja monimutkaisiin ongelmiin. Algoritmeissa on kehitetty myös virheensietokykyisiä variaatioita, sillä on varsin relevantti kysymys, mitä tapahtuu jos yksi agentti yhdyskunnassa meneekin epäkuntoon, ja miten tällaisesta ongelmasta selvittää ilman koko algoritmin suoriutumisen pysähtymistä.

Tutkielman luvussa 2 tutustutaan ensin Bayezatesin ja muiden (1995) tekemään tutkimukseen, jossa perehdytään tason tutkimisen ongelmiin ja esitetään niille tuloksia. Feinerman ja muut (2012) jatkavat Bayezatesin ja muiden tutkimuksesta ja luovat tason tutkimisen ANTS-ongelman. Ongelmaan perehdytään esittämällä tutkimuksia ja tuloksia, joita ovat kehittäneet ongelman ratkaisuksi niin Feinerman ja muut (2012), sekä eri tutkijat erilaisilla tavoitteilla ja rajoitteilla. Luvussa 3 perehdytään muurahaisyhdyskunta-algoritmiin ja siitä verkkolaskentaympäristöön kehitettyihin algoritmeihin Ludwigin ja muiden (2011) johdolla ja sitten lisäämällä Idrisin ja muiden (2017) kehittämä virheensietokykyinen algoritmi tutkittavien lähestymistapojen joukkoon. Luvussa 4 tehdään yhteenvedo materiaalista.

2. Tason tutkiminen

Tason tutkimisesta on tietojenkäsittelytieteissä monia julkaisuja. Sen tutkimiseksi on tehty jo paljon ratkaisuja ja menetelmiä, sekä perehdytty erinäisiin vaatimuksiin, mitä ratkaisuvariaatioilta voidaan olettaa. Tutkimuksia on tehty niin tarvittavien agenttien määrästä, tarvittavan tiedon määrästä, kuin myös suoritusajan muuttumisesta, kun parametrit vaihtuvat kysymystä ratkaistaessa. Omalla tavallaan tason tutkiminen onkin jo ratkaistu: On olemassa monta menetelmää, jotka tutkivat tason varmasti alusta loppuun.

Miksemme kuitenkin pakkaa tavaroitamme ja lopeta tämän kysymyksen mielettöntä pohtimista? Mielekkäitä tutkimuskysymyksiä löytyy jatkuvasti, sillä on eri asia, tutkiaanko tasoa optimaalisesti kahdella robotilla vai miljoonalla, onko niillä rajaton laskentateho vai pystyvätkö ne vain yksinkertaisiin päätöksiin, osaavatko ne kommunikoida rajattomasti vai pitääkö niiden olla samassa pisteessä.

Tason tutkimisessa on yleensä perusoletuksena origolähtöinen ratkaisumalli eli $(0,0)$ koordinaatit, josta lähtöisin tutkiminen suoritetaan. Taso tavataan mieltää koordinaatiksi algoritmien ja tutkimisen helpottamiseksi. Tällöin pätee lainalaisuudet Manhattanin etäisyydestä, jolloin agenttien etenemistä on helpompi hahmottaa. Tutkimukset käyttävät kohteen etäisyyden merkintänä D (distance) ja aikayksikkönä t (time), joka kuvaa aikaa, mikä kuluu yhden ruudun liikkumiseen. Tällöin pätee minimiaikavaatimus, että suoritus-aika on Manhattan-etäisyys origosta kohteeseen D kertaa aikayksikkö t . Kaikki ratkaisuun kykenevät algoritmit myös sisältävät jonkinlaisen ratkaisun siihen, kuinka kaikki agentit saadaan palautettua alkuasetelmaan. Ei olisi kovinkaan mielekäästä, jos agentit kaatoisivat kohteen löydettyään.

2.1 Aikavaatimuksia erilaisten kohteiden löytämiselle

Baezayates ja muut (1993) tutkivat minkälaisilla aikavaatimuksilla yksittäinen äärettömän tehokas agentti voi löytää kätketyn kohteen äärettömästä tasosta. Heillä oli ongelmaan kolme eri variaatiota, joissa pohdittiin tarjolla olevan tiedon määrän (knowledge) vaikutusta etsintää tekeväälle agentille. Mikäli agentti tiesi suunnan (direction) mistä etsiä, oli suoritus-aika jokaisessa tutkitussa ongelmassa (problem) erinomainen, mutta jos tiedoksi jäi kohteen etäisyys (distance), niin suoritus-aika heikentyi huomattavasti. Jos agentti ei tiennyt kohteesta mitään (nothing), niin etsintään meni moninkertainen aika aiempaan verrattuna. Taulukkoon 1 on koottu suoritus-aikoja eri määrillä agenteille saatavilla olevaa tietoa.

Baezayatesin ja muiden (1993) tutkimat ongelmat, joiden tulokset esitetään taulukossa 1, ovat piste suoralla (point in line), piste satunnaisessa säteessä m (point on m -rays), piste tasossa (point in lattice), piste tasossa, jonka etäisyyden parillisuus tiedetään (point in lattice with parity), vaaka- tai pystysuora tasossa (orthogonal line in plane) ja suora tasossa (line in plane).

The Advantage of Knowing Where Things Are

Problem	Knowledge		
	Direction	Distance	Nothing
Point on line	n	$3n$	$9n$
Point on m -rays	n	$(2m - 1)n$	$(1 + 2m^m / (m - 1)^{m-1})n$
Point in lattice	n	$9n - 2$	$\leq 2n^2 + 5n + 2, \geq 2n^2 + 4n - 1$
Point in lattice with parity	n	$\leq 2n^2 + 4n + n \bmod 2$	$\leq 2n^2 + 4n + n \bmod 2$
Orthogonal line in plane	n	$4.24 \dots n$	$\leq 13.02n, \geq 12.74n$
Line in plane	n	$6.39 \dots n$	$\leq 13.81n, \geq 12.74n$

Taulukko 1. Agentin suoritus aika suhteessa tiedon määrään (Baezayates et al. 1993).

Baezayates ja muut (1995) Jatkoivat aiempaa tutkimusta perehtymällä optimaalisiin suoritus aikoihin, mikäli agentteja oli monta ja keskittyen erityisesti suorien etsimiseen. Mikäli tiedossa oli kohteen suunta, niin yksi robotti oli yhtä tehokas kuin äärettömän monta. Jos tiedossa oli etäisyys, niin suoritusnopeus lähenee kohteen etäisyyttä verrannollisesti agenttien lukumäärään. Baezayates ja muut (1995) löysivät ongelmilleen raja-arvon, jonka jälkeen lisäagentit eivät enää huomattavasti parantaneet tulosta. He myös ratkaisivat raja-arvot logaritmiselle spiraalihauille kahdella agentilla.

2.2 Tason tutkimisen ANTS-ongelma ja sen variaatioiden ratkaisuja

Feinerman ja Korman (2012) rakensivat näiden tutkimusten varaan ANTS-ongelman, jossa ratkaistiin optimaalisia suoritus aikoja origokeskeisestä hausta katseltuna ja etsittäessä pisteessä olevaa kohdetta ja erityisesti tarkastellen agentteja, jotka eivät voi kommunikoida keskenään haun alettua. ANTS-ongelmassa tasosta muodostettiin ruudukko, jonka ruudut olivat aikayksikön pituisia. He käänsivät tutkimuksen varsin yleismaailmalliseen rinnastukseen, muurahaisyhdyskuntiin. Haun aloituspiste on muurahaispesä ja haku tasossa on muurahaisten pesää ympäröivän maaston tutkimista. Tavoitteena oli löytää kohde, joka miellettiin ruuaksi mahdollisimman nopeasti.

Feinerman ja Korman (2012) kehittivät tutkimuksessaan satunnaisuuteen perustuvaa ratkaisua. He aloittivat tutkimalla algoritmia (algoritmi 1), joka tiesi kuinka monta agenttia oli suorittamassa hakua ja optimoi hakua sen pohjalta. Tämä algoritmi sai optimaalisen suoritusajan (Sipser 1997) $O(D + D^2 / k)$, jossa k = agenttien lukumäärä ja D kohteen etäisyys. Algoritmi 1 koostuu kolmesta sisäkkäisestä silmukasta. Sisimän silmukan mihin kukin agentti lähetetään suorittamaan spiraalihakua tasolla. Sisimmän silmukan määrän päätetään muiden silmukoiden indeksoinnilla, jotta haku alkaa tutki-

malla kohteita lähempänä origoa ja vasta etäännyttäen hakua indeksien kasvaessa. Algoritmin ongelmana on sen kompleksisuus, joka myös Feinermanin ja Kormanin (2012) mukaan saattaa vaatia liikaa muurahaisten kaltaisilta agenteilta.

Oletettavasti algoritmin 1 kompleksisuudesta motivoituneina Feinerman ja Korman (2012) esittävät vaihtoehdoisen algoritmin, joka ei saavuta täydellistä suoritusaikaa, mutta on huomattavan yksinkertainen, eikä vaadi edes tietoa agenttien lukumäärästä. Tämä vaihtoehtoinen algoritmi (algoritmi 2) arpoo kaikille agenteille satunnaisen pisteen tasossa (vaihe 1), ja lähettää agentin suorittamaan spiraalihakua määrätyksi ajaksi kohteeseen (vaihe 2), jonka jälkeen se pelaa takaisin origoon (vaihe 3).

```

begin
  Each agent performs the following;
  for  $\ell$  from 0 to  $\infty$  do the big-stage  $\ell$ 
    for  $i$  from 0 to  $\ell$  do the stage  $i$ 
      for  $j$  from 0 to  $i$  do the phase  $j$ 
        • Set  $k_j \leftarrow 2^j$  and  $D_{i,j} \leftarrow \sqrt{2^{(i+j)}/j^{(1+\varepsilon)}}$ ;
        • Go to node  $u \in B(D_{i,j})$  chosen uniformly at random among the nodes in  $B(D_{i,j})$ ;
        • Perform a spiral search starting at  $u$  for  $t_{i,j} = 2^{i+2}/j^{1+\varepsilon}$  time;
        • Return to the source
      end
    end
  end
end

```

Algorithm 1: The uniform algorithm $\mathcal{A}_{\text{uniform}}$.

Algoritmi 1. Optimaaliseen suoritus aikaan kykenevä kompleksinen algoritmi (Feinerman and Korman 2012).

```

begin
  Each agent performs the following three actions;
  1. Go to a node  $u \in V(G)$  with probability  $p(u)$ ;
  2. Perform a spiral search for  $t(u) = d(u)^{2+\delta}$  time;
  3. Return to the source
end

```

Algorithm 2: The harmonic search algorithm.

Algoritmi 2. Yksinkertainen algoritmi, joka kykenee vain lähes optimaaliseen suoritus aikaan (Feinerman and Korman 2012).

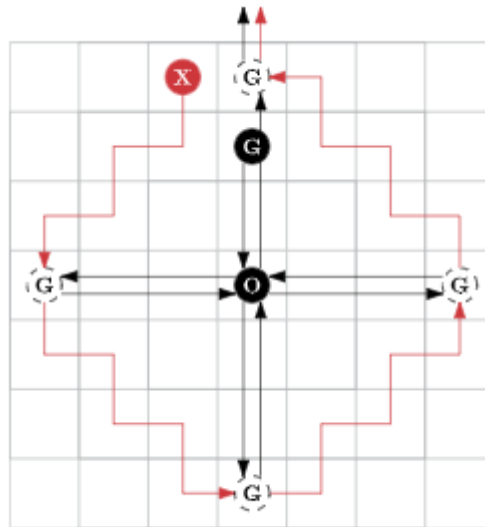
Feinerman ja Korman (2017) jatkoivat saman ongelman ratkaisujen parissa. He formalisoivat kysymykselle siitä käytettävän nimen ANTS ja tutkivat optimaalisia suoritus aikoja ja raja-arvoja erilaisille hauille ja algoritmeille. Uutena näkökulmana he tuovat oraakkelin, joka antaa neuvoja agenteille. Oraakkelille on variaatioita niin sille annettavan tiedon määrässä kuin sen käyttämässä neuvonantomenetelmässä. Algoritmit, mitä Feinerman ja

Korman (2017) luovat ovat hyvin lähellä heidän aikaisempiaan, mutta keskittävät tehon yksittäisiltä agenteilta keskeiselle entiteetille oraakkelille. Tutkimus toi tuloksia hakuaikojen optimaalisuuteen ja sen parantumiseen suhteessa tiedon määrään. Feinerman ja Korman (2017) jättävät auki kysymyksen siitä, onko satunnaisuuteen perustuva oraakkeli yksinkertaisesti tehokkaampi kuin deterministinen vastaparinsa.

Emek ja muut (2015) tutkivat myös Feinermanin ja Kormanin (2012) luomaa ANTS-ongelmaa, mutta varsin eri näkökulmasta. Emek ja muut (2015) tutkivat kuinka taso voidaan tutkia optimaalisesti mahdollisimman vähäisellä määrällä agenteja. He poistivat vaatimuksen, että agentit ovat identtisiä keskenään, mutta samalla he poistivat algoritmiltaan vaatimuksen oraakkelista, joka ohjaisi agenttien toimintaa. Tilalle otettiin deterministisyyteen tai satunnaisuuteen pohjautuvat äärelliset tai pinorakenteiset automaatit. Lisäksi agenteille annettiin kyky kommunikoida, mikäli ne kohtasivat toisensa origon ulkopuolella. Emek ja muut (2015) toteavat myös tutkimuksessaan, että algoritmin optimaaliseen agenttien lukumäärään vaikuttaa ympäristön ajallinen rakenne. Mikäli agentit toimivat hieman eri nopeudella eli epäsynkronoidusti niin tarvittu agenttien määrä kasvaa verrattuna samanaikaisesti toimiviin synkronoituihin agenteihin.

Algoritmit, joita Emek ja muut (2015) tutkivat olivat neljän agentin spiraalihaku epäsynkronoidussa ympäristössä, kolmen agentin spiraalihaku synkronoidussa ympäristössä, kolmen deterministisen agentin satunnaisuuteen perustuva geometrinen haku, sekä kahden agentin geometrinen satunnaishaku.

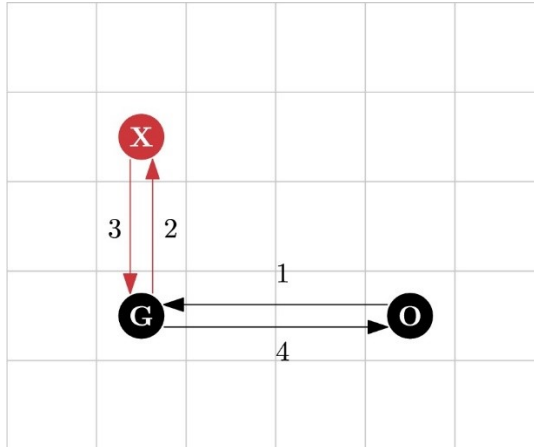
Neljän agentin epäsynkronoidussa spiraalihaussa, kolme agenttia neljästä toimii maamerkkeinä yhdelle agentille. Tason tutkintaa suorittava agentti (X) suorittaa origon ympäri tehtävää spiraalihakua siirtäen maamerkkejään: pohjois-agenttia (N), itä-agenttia (E) ja länsi-agenttia (W) reitillään yhä kauemmaksi origosta (kuva 1). Kaikki agentit ovat deterministisiä automaatteja, joilla on oman tehtävänsä mukainen algoritmi, jolla ne siirtyvät kauemmaksi aina kohdattuaan toisen agentin eli niiden välillä kiertävän tutkimusta suorittavan agentin. Kohteen löytämisen jälkeen algoritmi pääsee alkuasetelmaan suorittamalla algoritmia käänteisesti. (Emek et al. 2015)



Kuva 2. Kolmen agentin haku synkronoidussa ympäristössä (Emek et al. 2015).

Emekin ja muiden (2015) algoritmien luonne muuttuu, kun annetut parametrit tiukentuvat. Mikäli agentit joutuvat epäsynkronoituun ympäristöön tai niiden lukumäärä laskee alle kolmen, niin Emekin ja muiden (2015) esittämä kolmen agentin spiraalihaku ei enää toimi. Tällöin he ratkaisevat tason etsintäongelman algoritmeilla, jotka perustuvat geometriseen hakuun ja satunnaisuuteen. Satunnaisuus tuo ongelmaan mutkan, sillä deterministiset automaattit eivät niitä tue, joten ratkaisuun tulee satunnaiset epädeterministiset rajalliset automaattit. Geometrisen haun hyvänä ominaisuutena voidaan pitää sen toimivuutta niin ajallisesti synkronoidussa, että epäsynkronoidussa ympäristössä.

Kolmen agentin algoritmi muuttuu epäsynkronoidussa ympäristössä satunnaisuuteen perustuvaan geometriseen hakuun, johon vaaditaan satunnaisuuteen kykenevä automaatti. Algoritmin iteraatio muodostuu automaattien kolikonheitosta. Yksi automaateista on vain origon merkki, toinen ja kolmas suorittavat geometrisen haun osat heittämällä kolikkoa, kunnes päätyvät pisteeseen x. Palautuminen origoon tapahtuu, kun heitetään väärä kolikon arvo ja palataan vastakkaiseen suuntaan, kunnes törmätään toiseen agenttiin. Tämän algoritmin vaiheet on havainnollistettu kuvassa 3. (Emek et al. 2015)



Kuva 3. Satunnaisuuteen perustuvan geometrisen haun iteraatio (Emek et al. 2015).

Tässä algoritmossa sekä satunnaisissa algoritmeissa yleensä on mielenkiintoinen kysymys, kuinka voidaan todistaa, että algoritmit suoriutuvat äärellisessä ajassa, vaikka haku tehdään satunnaisuudella. Emek ja muut (2015) todistavat kuinka geometrisen haun suoritus-aika pysyy äärellisenä, vaikka käytetään kolikkoa heittäviä agenteja, ja kuinka ne pystyvät oletus-suoritusajallisesti jokseenkin heikkoon suoritusajaan $O(2^D)$.

Emek ja muut (2015) saavat ratkaisun, myös kahdella sekä yhdellä automaatilla, jotka nojautuvat samaan geometriseen hakuun. Kääntöpuolena agenttien lukumäärän vähenemiselle on niiden muodostuminen monimutkaisemmiksi ja vaativammiksi pinoautomaateiksi. Pinolla agentit suunnistavat aina satunnaisuudella muodostettuun päämääräänsä lisäämällä pinon liikkumisensa suunnan ja palaavat pinosta poistamalla merkin-töjään ja palaamalla vastakkaisella liikkeellä takaisin origoon.

Emek ja muut (2015) todistavat tutkimuksessaan, että yksittäinen agentti ilman pinorakennetta ei pysty ratkaisemaan ANTS-ongelmaa. He jättävät samalla auki kysymyksen siitä kykeneekö kaksi satunnaisuuteen nojautuvaa agenttia tai kolme determinististä agenttia ratkaisemaan ANTS-ongelman epäsynkronoidussa ympäristössä, sekä kysymyksen voiko kaksi satunnaisuuteen nojautuvaa automaattia ratkaista ongelman synkronoidussa ympäristössä.

ANTS-ongelman ratkaisua mahdollisimman pienellä määrällä agenteja voidaan pitää vastakohtana Langnerin ja muiden (2014) tutkimukselle virheensietokykyisestä algoritmista ANTS-ongelman ratkaisuun. Heidän premisseillään heidän luomansa algoritmin pitäisi sietää n kappaletta kohdennettuja virheitä ja silti kyetä optimaaliseen suoritusajaan. Langner ja muut (2014) loivat algoritmin, jonka virheensietokyky perustuu suureen määrään agenteja ja niistä luotuihin agenttirykelmiin, mitkä toimivat algoritmin tukipisteinä ja jotka eivät tuhoudu, vaikka niihin kohdistettaisiin kaikki n virhettä. Tämä algoritmi vaatii kymmenen kyseistä agenttirykelmää, sekä kaksinkertaisen määrän tutkintaa

suorittavia agenteja verrattuna maksimaaliseen virheiden määrään. Yhteensä heidän luomansa algoritmi siis vaatii kolmetoistakertaisen määrän agenteja virheisiin verrattuna, mikä virheiden lukumäärän kasvaessa nousee nopeasti huomattavan suureksi lukemaksi.

Langnerin ja muiden (2014) luoma algoritmi toimii spiraalihakuun pohjautuen. Algoritmissa agenttilukumäärällisesti kalliit agenttirykelmät toimivat tienviittoina yksittäisille agenteille, jotka suorittavat neljään sektoriin jaettua spiraalihakua. Agenttirykelmät etäännyvät origosta aina onnistuneen spiraalihakuiteraation jälkeen. Algoritmi tarkistaa aina ennen iteraation alkua, onko viimeisimmässä iteraatiossa tapahtunut suoritusta haittaava virhe. Mikäli tällainen virhe todetaan, niin algoritmi suorittaa uudestaan viimeisimmän iteraation paikaten virheellisen kohdan jäljellä olevilla hakua suorittavilla agenteilla. Tämä koko iteraation toistaminen aina virheen sattuessa on ajallisesti kallis operaatio ja täten algoritmin suoritusaika jääkin aikaan $O(D+D^2/n+Df)$, jossa f on toteutuneiden virheiden määrä, n agenttien lukumäärä ja D kohteen etäisyys.

2.3 Muita tason tutkimuksia

ANTS-ongelman variaatioita on myös ratkaisseet esimerkiksi Brandt ja muut (2018) ja Lenzen ja muut (2014). Brandt ja muut (2018) perehtyvät Emekin ja muiden (2015) avoimeksi jättämään kysymykseen kolmesta synkronoidusta äärellisestä automaatista ja niiden kyvystä, tai pikemminkin kyvyttömyydestä, tutkia taso äärellisessä ajassa. Lenzen ja muut (2014) puolestaan perehtyvät ANTS-ongelman aikavaatimusten lisäksi valintojen monimutkaisuuden kysymykseen. Heidän ajatuksenaan on se, että jos tutkimme yksinkertaisia biologisia eliöitä, niin algoritmi, jota ne suorittavat ei voi olla järin monimutkainen tai se ei koskaan ilmenisi reaali maailmassa.

Tason tutkimisen ongelmaa on aikaisempien tutkimusten lisäksi tutkittu myös monen muun ongelman perspektiivistä. Cohen ja muut (2017) tutkivat äärettömien yksi- ja kaksikulotteisten tasojen minimiagenttimäärien vaatimuksia, jos agentit ovat satunnaisia äärellisiä automaatteja. Altshuler ja muut (2011) puolestaan tutkivat, minkälainen vaikutus tason laajenemisella on tason täydellisen tutkimisen vaatimukseen, niin agenttien lukumäärässä kuin suoritusajassa.

Hazra ja muut (2017a, 2017b) perehtyvät vastaavaan ongelmaan peliteoreettisesta näkökulmasta. Hazra ja muut (2017a) esittävät, kuinka voidaan mallintaa tasolla etsintää tekevää ja etsijää välttelevää agenttia, aikaisempaan ANTS-ongelmaan verrattuna etsivällä agentilla on huomattavasti enemmän laskennallista tehoa ja kohde on liikkuva. Lisänä on myös mahdolliset esteet liikkumiseen tasolla ja vain toiselle agentille sallitut reitit. Peliteoreettinen näkökulma mallintamiseen tulee siitä, että etsivälle ja pakenevalle agentille annetaan suorituksista pisteitä. Etsivälle sen mukaan, kuinka nopeasti se löytää kohteen ja kohteelle sen mukaan, kuinka kauan se kykenee välttelemään etsijää. Näiden

pisteitysten perusteella yritetään saada mahdollisimman optimaaliset algoritmit molemmille agenteille. Hazra ja muut (2017b) tutkivat, kuinka voidaan optimaalisesti tutkia kaikki tason pisteet, mikäli agenteilla on polttoainerajoitus sekä tankkauspisteet. Peliteoreettinen lähestymistapa ongelmaan tuo algoritmille pisteytysmenetelmän iteraatioiden määrän perusteella. Mitä vähemmän iteraatioita, niin sitä enemmän pisteitä algoritmi saa. Variaatioita algoritmien parametreissa on agenttien yhteistyökyvyssä, polttoainemäärissä sekä polttoainepisteiden lukumäärässä.

3. Muurahaisyhdyskunta-algoritmi

Muurahaisyhdyskunta-algoritmi (Ant colony algorithm) on, kuten nimestäkin voi päätellä saanut inspiraationsa muurahaisyhdyskunnista. Se perustuu tapaan millä muurahaiset etsivät ruokaa ja merkkavat feromonilla reitin löytämäänsä ruuan lähteeseen, jotta muut muurahaiset voivat seurata feromonijälkeä kohteen luokse. Lähtöasetelmassa kaikki muurahaiset lähtevät satunnaisesti kulkemaan ympäriinsä, jos muurahainen löytää ruokaa, niin se vie ruokaa kotiin ja merkkaa reitin feromonilla. Mikäli toinen muurahainen löytää feromonin, niin satunnaisuuteen ja feromonin vaikutusasteeseen pohjautuen se valitsee, seuraako se feromonin merkkamaa reittiä vai jatkaako se omaa etsintäänsä. Mikäli monta muurahaista löytää saman ruuan lähteen, niin feromonimerkkaus reitille kopioituu lisäten reitin painoarvoa. Tämä menetelmä myös samalla optimoi lyhyemmän ruuan lähteen yli kauemman ruuan lähteen. Lähempänä olevaan ruuan lähteeseen pääsee nopeammin, joten siihen myös hyvin todennäköisesti päättyy useampi muurahainen ladaten polun feromonilatausta entistä vahvemmaksi. Algoritmi myös itsessään estää mahdollista turhaa ehtyneelle ruuan lähteelle kulkemista sillä, kun ruuan lähde ehtyy, niin feromoneja ei kyseiselle reitille enää laiteta ja sen arvo laskee muihin polkuihin verrattuna. Feromonit myös haihtuvat ajan kuluessa. Mikäli näin ei olisi, niin kerran merkattu, mutta turhaksi käynyt polku johtaisi muurahaiset jatkuvasti harhaan tuhlaten resursseja ja mahdollisesti lamauttaen koko etsinnän. (Macura 2020)

Muurahaisyhdyskunta-algoritmia käytetään useissa haku- ja optimointiongelmien ratkaisuisissa. Tasojen tutkimisen yhteydessä, sitä on esimerkiksi käyttänyt Xiaojing ja muut (2019) kolmiulotteisen tason tutkimiseen. He kehittivät perinteiselle muurahaisyhdyskunta-algoritmille parempaa versiota, kun kysymyksenä on reaali maailman maa-alueelta tunnetun pisteen etsimisongelma. Aikaisempaan tason tutkimiseen verrattuna, kolmas ulottuvuus monimutkaistaa algoritmia ja luo esteitä agenttien kululle. Tämä vaatii algoritmilta kykyä sopeutua maaston esteisiin sekä hankaloittaa reitin etsintää, sillä linnuntie ei ole välttämättä paras eikä edes mahdollinen optio. Lisäksi algoritmin tulisi kyetä saamaan kyseinen reitti mahdollisimman optimaaliseksi esteistä huolimatta. Xiaojing ja

muut (2019) ovat hienosäätäneet perinteisen muurahaisyhdyskunta-algoritmin heuristiikka parantaakseen sen suorituskykyä ja tehostaneet lopputuloksen optimaalisuutta, lisäksi he ovat antaneet kyvyn säätää globaalia feromonitasoa lisäoptimointia varten. Tämä tuottaa varsin tehokkaan algoritmin kohteen etsimistä varten, jos on tiedossa jo määränpää ja lähtöpiste kolmiulotteisella tasolla.

Muurahaisyhdyskunta-algoritmi on tehokas algoritmi kohteen ja reitin etsintään, mikäli tarjolla on agenteja, joilla on vähäinen laskentateho. Se on saanut inspiraationsa evoluution muovaamasta muurahaisten ruuanhaketavasta, ja tästäkin voidaan päätellä, että se on varsin optimaalinen ratkaisu reitin- ja kohteenhakualgoritmiksi.

3.1 Muurahaisyhdyskunta-algoritmi verkkolaskentaympäristössä

Algoritmia hieman muokkaamalla ja laittamalla se etsimään kohdetta verkkolaskentaympäristössä, jolla on eniten vapaata laskentatehoa, saadaan varsin järkevä ja tehokas tapa jakaa laskentatehtäviä verkkolaskentaympäristössä.

Ludwig ja muut (2011) tutkivat miten hyönteisyhdyskuntia voidaan käyttää verkkolaskentaympäristössä ja kuinka tehokkaasti. He loivat AntZ-algoritmin, jonka tavoitteena on jakaa verkkolaskentaympäristön laskentatehtävät mahdollisimman tasaisesti ja tehokkaasti. AntZ toimii, kuten tavallinenkin muurahaisyhdyskunta-algoritmi, sillä kun laskentaverkkoon sijoitetaan uusi laskutehtävä, niin luodaan uusi agentti, joka kiertää satunnaisesti verkkolaskentaympäristössä. Sen tehtävä on etsiä laskentayksikkö, jolla on vähiten tehtäviä suoritettavana mahdollisimman nopeasti ja vähäisellä laskennallisten tehtävien määrällä. Agentti pitää muistissa käymänsä laskentayksiköt ja niiden laskentatehtävien määrän, sekä sijoittaa aina laskentayksikköön tiedon käymistään laskentayksiköistä, joka toimii käytännössä feromonipolkuna muille agenteille. Lopuksi agentti sijoittaa laskentatehtävän siihen laskentayksikköön, jolla on vähiten laskentatehtäviä. AntZ-algoritmi on havainnollistettu algoritmina 3.1 ja sen metodin chooseNextStep tarkennus on esitetty algoritmina 3.2.

Algorithm 3.1: ANTZALGORITHM(*MutRate*,
MaxSteps, *DecayRate*)

```
step ← 1
initialize()
while step < MaxSteps
do {
  currentLoad ← getNodeLoadInformation()
  AntHistory.add(currentLoad)
  localLoadTable.update()
  if random() < MutRate
  then nextNode = RandomlyChosenStep()
  else nextNode = chooseNextStep()
  MutRate ← MutRate – DecayRate
  step ← step + 1
  moveTo(nextNode)
deliverJobToNode()
```

Algoritmi 3.1. AntZ-algoritmin yksittäisen agentin algoritmi (Ludwig et al. 2011).

Algorithm 3.2: CHOOSENEXTSTEP()

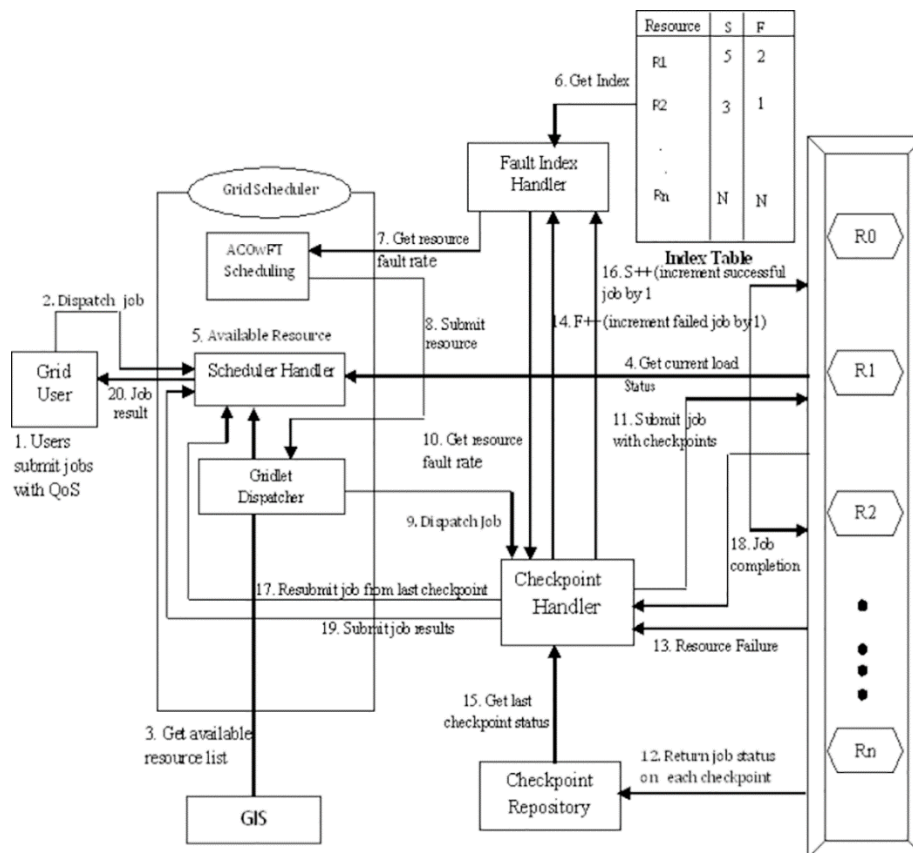
```
bestNode ← currentNode
bestLoad ← currentLoad
for entry ← 1 to n
{
  if entry.load < bestLoad
  then bestNode ← entry.node
  else if entry.load = bestLoad
  {
    if random.next < probability
    then bestNode ← entry.node
```

Algoritmi 3.2. Agentin chooseNextStep() metodin tarkennus (Ludwig et al. 2011).

Idris ja muut (2017) jatkoivat Ludwigin ja muiden (2011) luoman AntZ-algoritmin kehittämistä. He perehtyivät siihen, kuinka algoritmista saadaan virheensietokykyinen, sillä monen laskentayksikön systeemeissä virheet ovat yleisiä ja vievät huomattavasti resursseja. Täten optimaalinen virheiden käsittely verkkolaskentaympäristön algoritmissa on oleellista, jopa pakollista. Yleinen menetelmä virheensietokyvyn saavuttamiseksi on tarkistuspisteiden luonti mahdollisten virheiden varalta ja järjestelmän palauttaminen niihin,

mikäli virhe suorittaessa tapahtuu. Tähän ratkaisuun myös Idris ja muut (2017) päätyivät lisäten algoritmiin virheiden käsittelyyn kykenevän komponentin ja virhealttiuden tiedon säilömiseen.

Agentit AntZ-algoritmissa veivät mukanaan tiedon tutkimistaan laskentayksiköistä päivittäen reitillään olevien laskentayksiköiden taulukot aikaisempien varatusta kapasiteetistä. Idris ja muut (2017) lisäävät tähän operaatioon vain laskentayksikön virhetodennäköisyyden. Vaikka muutos aikaisempaan AntZ-algoritmiin on pieni, niin kokonaisuudesta tulee jo varsin monimutkainen. Yhden verkkolaskentaympäristöön laitettavan laskentatoimenpiteen läpikäymisessä on jo kahdenkymmenen askeleen prosessi, jotta laskentatehtävä saadaan käsiteltyksi. Kuvassa 4 esitetään yksittäisen laskentatehtävän läpikulku virheitä käsittelevässä verkkolaskentaympäristössä. Virheiden käsitteleminen tuo prosessiin paljon lisää laskentatehtäviä, mutta silti Idrisin ja muiden (2017) parantaman AntZ-algoritmin suoritustehokkuus pysyi melkein samana kuin alkuperäisen, jos verkkolaskentaympäristö oli pieni. Verkkolaskentaympäristön kasvaessa paranneltu algoritmi pärjasi huomattavasti paremmin, sillä käsiteltävien virheiden määrä kasvoi systeemissä ja niiden käsittelyssä se oli huomattavasti tehokkaampi.



Kuva 4. Virheensietokykyisen AntZ-algoritmin kuvaus kaaviona (Idris et al. 2017).

4. Yhteenveto

Kuten tässä tutkimuksessa esitellyistä algoritmeista voidaan todeta, on evoluution prosesseilla vielä annettavaa tieteelliselle tutkimukselle. Muurahaisyhdyskunnan ratkaisut niiden arkisiin ongelmiin ovat edelleen inspiraation lähteenä tieteelle. Ympäriinsä vaeltavat muurahaiset kykenevät luomaan optimaalisen ratkaisun, niin kohteen etsintään, kuin suuriin verkkolaskentaympäristöihin. Muurahaisyhdyskunnilla on varmasti myös vielä annettavaa esimerkiksi laskentatehtävien jakamisen optimoinnin kanssa, sillä ne ovat tehokkaita kokonaisuuksia, joissa on suuri määrä laskentayksiköitä suorittamasta toimintaa. Tähänkin näkökulmaan ovat perehtyneet esimerkiksi Cornejo ja muut (2014) analysoimalla muurahaisyhdyskuntien työnjakoa.

Tässä tutkielmassa käytiin läpi tason tutkimista ja muurahaisyhdyskunta-algoritmia verkkolaskentaympäristössä hajautetun laskennan näkökulmasta ja tuotiin esille muutamia yleisiä ongelma-asetelmia kuten ANTS ja AntZ. Näihin ongelmiin on rakennettu varsin optimaalisia muurahaisyhdyskuntien inspiroimia ratkaisuja, mutta edelleen kuten niihin perehtyneissä tutkimuksissakin todetaan, on huomattavasti avoimia kysymyksiä, jos näkökulmaa tai ongelmaa lähestytään eri näkökulmasta.

Lähdeluettelo

- Altshuler, Y. and Bruckstein, A. (2011). Static and expanding grid coverage with ant robots: Complexity results, *Theoretical Computer Science*, Volume 412, Issue 35, Pages 4661-4674, ISSN 0304-3975, <https://doi.org/10.1016/j.tcs.2011.05.001>.
- Baezayates, R. Culberson, J. and Rawlins, G. (1993). Searching in the Plane, *Information and Computation*, Volume 106, Issue 2, 1993, Pages 234-252, ISSN 0890-5401, <https://doi.org/10.1006/inco.1993.1054>.
- Baezayates, R. and Schott, R. (1995). Parallel searching in the plane, *Computational Geometry*, Volume 5, Issue 3, 1995, Pages 143-154, ISSN 0925-7721, [https://doi.org/10.1016/0925-7721\(95\)00003-R](https://doi.org/10.1016/0925-7721(95)00003-R).
- Brandt, S. Uitto, J. and Wattenhofer, R. (2018). Tight Bounds for Asynchronous Collaborative Grid Exploration. Doi:<https://arxiv.org/abs/1205.2170>

- Cohen, L. Emek, Y. Louidor, O. and Uitto, J. (2017). Exploring an Infinite Space with Finite Memory Scouts, Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. 2017, 207-224
- Cornejo, A. Dornhaus, A. Lynch, N. and Nagpal, R. (2014). Task Allocation in Ant Colonies. In: Kuhn F. (eds) Distributed Computing. DISC 2014. Lecture Notes in Computer Science, vol 8784. Springer, Berlin, Heidelberg
- Emek, Y. Langner, T. Stolz, D. Uitto, J. and Wattenhofer, R. (2015). How many ants does it take to find the food?, Theoretical Computer Science, Volume 608, Part 3, Pages 255-267, ISSN 0304-3975, <https://doi.org/10.1016/j.tcs.2015.05.054>.
- Feinerman, O. and Korman, A. (2012). Memory Lower Bounds for Randomized Collaborative Search and Applications to Biology. Doi: <https://arxiv.org/abs/1205.4545>
- Feinerman, O. and Korman, A. (2017). The ANTS problem. *Distributed Computing* 30(3), 149-168. doi:<http://dx.doi.org.libproxy.tuni.fi/10.1007/s00446-016-0285-8>
- Feinerman, O. Korman, A. Lotker, Z. and Sereni, J. (2012). Collaborative Search on the Plane Without Communication. In Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing. ACM, 2012. 77–86.
- Hazra, T. Kumar, C. and Nene, M. (2017a). Modeling and Analysis of Grid-Based Target Searching Problems in a Mobile Sensor Network. *Wireless Pers Commun* 95, 4717–4732 <https://doi-org.libproxy.tuni.fi/10.1007/s11277-017-4115-5>
- Hazra, T. Kumar, C. and Nene, M. (2017b). Multi-agent target searching with time constraints using game-theoretic approaches. *Kybernetes*, 46(8), 1278-1302. doi:<http://dx.doi.org.libproxy.tuni.fi/10.1108/K-01-2017-0039>
- Idris, H. Ezugwu, A. Junaidu, S. and Adewumi, A. (2017). An improved ant colony optimization algorithm with fault tolerance for job scheduling in grid computing systems. *PLoS One*, 12(5) doi:<http://dx.doi.org.libproxy.tuni.fi/10.1371/journal.pone.0177567>

- Langner T. Uitto J. Stolz D. and Wattenhofer R. (2014). Fault-Tolerant ANTS. In: Kuhn F. (eds) Distributed Computing. DISC 2014. Lecture Notes in Computer Science, vol 8784. Springer, Berlin, Heidelberg
- Lenzen, C. Lynch, N. Newport, C and Radeva T. (2014). Trade-offs between selection complexity and performance when searching the plane without communication. In Proceedings of the 2014 ACM symposium on Principles of distributed computing (PODC '14). Association for Computing Machinery, New York, NY, USA, 252–261.
DOI:<https://doi.org/10.1145/2611462.2611463>
- Ludwig, S. and Moallem, A. (2011). Swarm Intelligence Approaches for Grid Load Balancing. *J Grid Computing* 9, 279–301 <https://doi-org.libproxy.tuni.fi/10.1007/s10723-011-9180-5>
- Macura, W. (2020). Ant Colony Algorithm.
<http://mathworld.wolfram.com/AntColonyAlgorithm.html>
- Sipser, M. (1997). Introduction to the Theory of Computation . Boston (Mass.): PWS.
- Xiaojing, L. and Dongman, Y. (2019). Study on an Optimal Path Planning for a Robot Based on an Improved ANT Colony Algorithm. *Aut. Control Comp.Sci.* **53**, 236–243.
<https://doi-org.libproxy.tuni.fi/10.3103/S0146411619030064>