Tampereen yliopisto

Olli Inkinen

USEAN TOIMIJAN REITITYSONGELMA KONFLIKTIPERUSTAISELLA HAKUALGORITMILLA KAIVOSYMPÄRISTÖSSÄ

Kandidaatintutkielma
Tekniikan ja luonnontieteiden tiedekunta
Tarkastaja: Teijo Juntunen
Joulukuu 2023

TIIVISTELMÄ

Olli Inkinen: Usean toimijan reititysongelma konfliktiperustaisella hakualgoritmilla kaivosympäristössä
Kandidaatintutkielma
Tampereen yliopisto
Teknisten tieteiden kandidaatin tutkinto-ohjelma
Joulukuu 2023

Työn tavoitteena oli tutkia konfliktiperustaisen hakualgoritmin suoriutumista kaivosympäristöön pohjautuvissa kartoissa usealla samanaikaisella toimijalla. Karttoja tutkimuksessa käytettiin kolmea erilaista: yksi suonilouhintaa kuvaavaa ja kaksi pilarilouhintaa kuvaavaa. Työssä ideana on, että toimijat liikkuvat kartalla samanaikaisesti toisiaan väistellen pyrkien kohti maalipistettään. Näiden kyseisten toimijoiden reitit algoritmi pyrkii ratkaisemaan. Algoritmin saama ratkaisu ei luultavasti ollut optimaalisin ratkaisu ongelmaan varsinkaan suurilla toimijamäärillä, koska ongelma on kompleksisuudeltaan NP-vaikea tehtävä.

Työ suoritettiin tunnin suoritusajoin algoritmia käyttämällä. Algoritmiin oli lisätty oma määräysfunktio, joka määritteli toimijoille lähtö- ja maalipisteet halutulla tavalla. Käytetyt kartat täytyi myös lisätä algoritmin käytettäviksi. Karttojen pohjana toimi ruudukko, missä toimijat pystyvät liikkumaan tai odottamaan rajoitteita noudattaen. Liikkumissuuntia toimijoilla oli käytössä ylös, alas, oikealle ja vasemmalle. Toimijoiden lähtö- ja maalipisteiden sijoituskriteereinä oli, että ne loisivat toimijalle reitin, jossa aiheutuu konflikteja muiden toimijoiden reittien kanssa, ja että koko karttaa käytettäisiin. Onnistuneessa ratkaisussa jokainen toimija pääsee maalipisteeseensä aikarajan sisällä ilman yhteentörmäyksiä. Jokaisen onnistuneen ratkaisun jälkeen ongelmaa vaikeutettiin lisäämällä karttaan yksi toimija lisää. Tätä toteutettiin, kunnes ongelmaa ei pystytty ratkaisemaan alle tunnissa.

Tutkimuksessa kerättiin dataa ratkaisuajoista ja algoritmin ylä- ja alatason iteraatiomääristä. Kerätystä datasta luotiin kuvaajat havainnollistamaan tuloksia. Tuloksista saatiin selville, että ahtaiden karttojen ratkaisu vaikeutuu huomattavasti jokaisen lisätyn toimijan jälkeen. Hieman avoimempi kartta pärjäsi varsinkin alatason iteraatiomäärissä ahtaampia karttoja paremmin. Lisäksi jokaisen toimijan lisäyksen yhteydessä ratkaisuun vaadittavat iteraatiomäärät moninkertaistuvat, mikä vaatii enemmän prosessointitehoa, jotta ratkaisu löydetään ajallaan.

Avainsanat: monitoimija-reitinhaku, MAPF, reititysongelma, konfliktiperustainen hakualgoritmi, Conflict Based Search, kaivosympäristö

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

CVRP	Kapasiteettirajoitettu ajoneuvon reititysongelma
KPH	Konfliktiperustainen hakualgoritmi
MAPF	Monitoimija-reitinhakuongelma
TAA*	Taso-aika-A*-algoritmi
VRPTW	Aikaikkunoihin rajoitettu reititysongelma

1. JOHDANTO

Reittioptimointi on tärkeä osa nykypäivän yhteiskuntaa, kuten esimerkiksi logistiikassa, kuljetustoimissa, tietoverkoissa ja internetin reitityksessä [1]. Se antaa mahdollisuuden kehittää maailmaa energiatehokkaammaksi ja ajankäytöllisesti optimaalisemmaksi [2]. Reittioptimointia käytetään useissa eri tarkoituksissa, kuten määränpään saavuttamiseen mahdollisimman nopeasti (esimerkiksi Google Maps -ohjelmiston parhaimman reitin määritys) tai varastorobottien liikkumisessa [3, s. 13155]. Reittioptimointi toteutetaan yleensä erilaisten algoritmien avulla [4, s. 263].

Energiatehokkuus on hyödyksi sekä liiketulokselle että ympäristölle. Optimoimalla sekä robottien ajankäytön että erilaiset reitit pystytään parantamaan energiatehokkuutta. Kaivoskoneet ovat yleensä suurikokoisia ja hitaita, minkä lisäksi ne kuluttavat paljon energiaa. Esimerkiksi CAT AD30 kaivoskuormaajan polttoaineen kulutus on 27,0–56,0 litraa tunnissa riippuen kuormien painosta ja teiden laadusta [5]. Maan alla liikkuvien automaattisten koneiden reitit ovat ahtaita, eikä ohituspaikkoja ole kaikkialla [6]. Optimoimalla ajoneuvojen reitit pystytään pitämään liikenne kapeissa tunneleissa mahdollisimman sujuvana ja tuottavana.

Tässä työssä tutkitaan, milloin pullonkauloja sisältävä kartta tukkeutuu *toimijoiden* määrän ja ohituspaikkojen vähyyden takia. Työssä on tarkoitus luoda kolme erilaista pullonkauloja sisältävää karttaa. Karttaan sijoitetaan toimijoita, jotka pyrkivät pääsemään omasta lähtöpisteestään omaan maalipisteeseen ilman toisiin toimijoihin törmäämistä. Ratkaisuista kerätään iteraatiomäärät ja kulunut ratkaisuaika talteen päätelmiä varten. Konfliktiperustaisella hakualgoritmilla (engl. Conflict Based Search) pyritään tuottamaan onnistunut ratkaisu *monitoimija-reitinhakuongelmaan* (engl. multi-agent pathfinding problem, MAPF). Konfliktiperustainen hakualgoritmi (KPH) suoriutuu pullonkaulakartoissa yleensä hyvin verrattuna muihin erilaisiin algoritmeihin [7, s. 52]. Onnistuneen ratkaisun jälkeen lisätään yksi toimija lisää omilla lähtö- ja maalipisteillään hankaloittamaan ongelmaa. Iteraatioita toteutetaan, kunnes ratkaiseminen epäonnistuu tai kestää liian kauan.

Toinen luku alustaa ongelman teoriaa. Luvussa tarkastellaan ajoneuvojen reititysongelmien taustaa, graafiteorian osia, joita käytetään työssä reittejä toteuttaessa, monitoimija-reitinhakuongelman haasteet ja mahdollisuudet, A*-algoritmi, KPH-algoritmin toimintaperiaate ja lisäksi kaivosympäristön ominaisuuksia. Kolmannessa luvussa esitellään

käytetyt tutkimusmenetelmät. Luvussa kerrotaan myös käytetyistä työkaluista ja tutkimuksen suorittamisesta. Luvussa neljä esitellään algoritmin tuottamat tulokset, koostetaan niistä kuvaajia ja tehdään päätelmiä. Viimeisessä luvussa tehdään yhteenveto saaduista tuloksista ja pohditaan mahdollisia jatkotutkimuskohteita.

2. REITTIOPTIMOINTI USEAN TOIMIJAN REITITYKSEN NÄKÖKULMASTA

Reittioptimointi ongelmana on paljon tutkittu aihe [8]. Sen sovelluksia on esimerkiksi tilattujen tuotteiden etsintä ja haku varastoista mahdollisimman optimaalisesti [9]. Tutkimuksissa [4, 7] tarkoituksena on tutkia, miten tietynlainen algoritmi tai ratkaisuprosessi onnistuu tietyntyyppisessä ympäristössä. Algoritmien ratkaisutavat eroavat toisistaan, joten on tarvetta tutkia optimaalisimpia ja nopeimpia reititystapoja sekä lisäksi kehittää uusia tapoja. Optimoinnin perusongelmana on: halutaanko mahdollisimman lyhyt ratkaisu pituudeltaan vai nopeiten löytynyt ratkaisu. Suurissa reititysongelmissa optimaalisen ratkaisun löytäminen voi olla todella hidasta.

2.1 Ajoneuvojen reititysongelmat

Reititysongelmat ja ajoneuvojen reititysongelmat ovat lähtöisin 1950-luvulta, kun George Dantzig ja John Ramser pyrkivät ratkaisemaan bensiinin toimitusongelman huoltoasemille käyttämällä matemaattista ohjelmointimuotoilua ja algoritmista lähestymistapaa ongelmaan. Optimaalisen reitin määrittämistä annetussa reittiverkostossa joukolle toimijoita, jotka liikkuvat paikasta A paikkaan B jotain reitistöä pitkin, kutsutaan reititysongelmaksi [10, s. VII]. Ongelmaa ratkaistaessa yleensä on jokin tavoite. Osa tavoitteista voivat olla vastakkaisia ja mahdollisia tavoitteita ovat esimerkiksi: kuljetuksen kokonaiskustannuksen minimointi, kokonaisajan minimointi, tarvittavan ajoneuvomäärän minimointi, reittien tasapainotus ajallisesti tai ajoneuvon kuorman mukaan [10, s. VII; 11, s. 4]. Reititysongelmia pystytään jakamaan useisiin alalajeihin, kuten kapasiteettirajoitettuihin ajoneuvon reititysongelmiin (engl. Capacitated Vehicle Routing Problem, CVRP) tai tiettyihin aikaikkunoihin rajoitettuihin reititysongelmiin (engl. Vehicle Routing Problem with Time Windows, VRPTW) [10, s. VII].

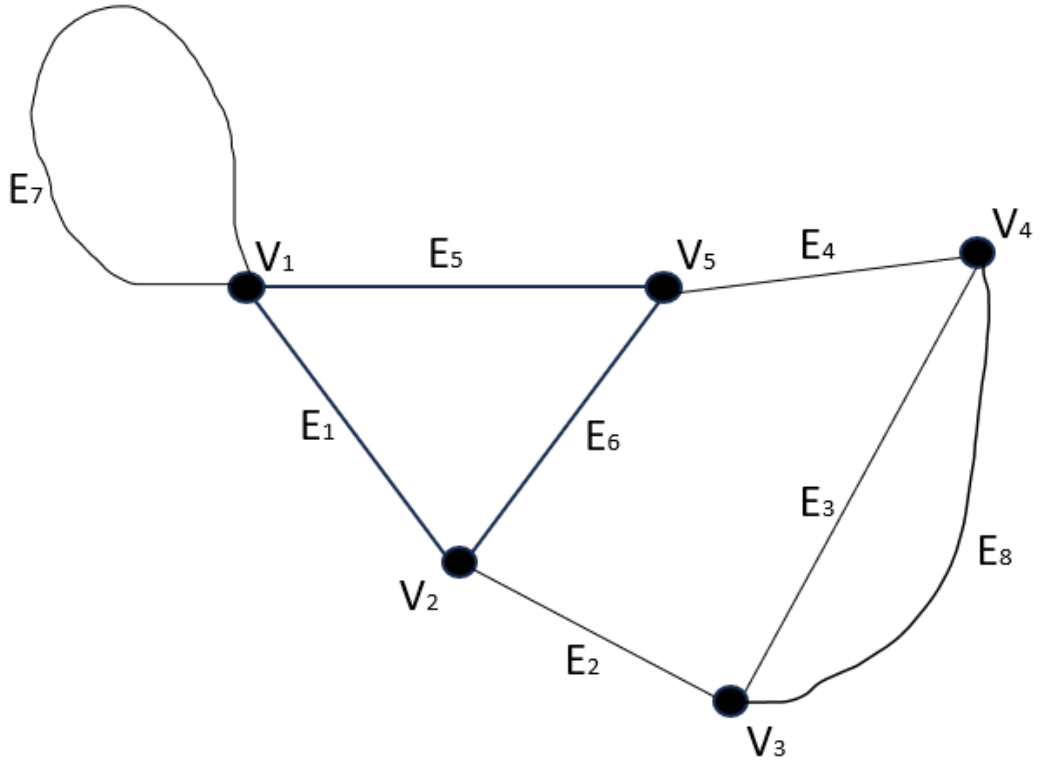
Ajoneuvojen reititysongelmien mallinnukseen on esitetty kolmea lähestymistapaa. Ne ovat ajoneuvovirtausformulointi, jota käytetään yksinkertaisiin reititysongelmiin, tuotevirtausformulointi, jolla pystytään löytämään täsmällinen ratkaisu kapasiteettia sisältäviin ongelmiin ja kolmannen tyyppiset mallit sisältävät eksponentiaalisen määrän binäärisiä muuttujia [11, s. 11]. Reititysongelmia on myös pyritty ratkaisemaan metaheuristisilla menetelmillä. Metaheurististen menetelmien tarkoituksena on tutkia kaikista lupaavimpia alueita ratkaisuavaruudessa. Ne koostuvat monimutkaisista naapureiden tutkimissäännöistä, muistirakenteista ja yhdistetyistä ratkaisumahdollisuuksista [11, s. 109]. Saadut

ratkaisut ovat huomattavasti parempia kuin perinteiset heuristisilla tavoilla saadut ratkaisut, mutta niiden saamiseen kuluu enemmän laskentatehoa [11, s. 109]. Metaheuristisia menetelmiä ovat esimerkiksi simuloitu hehkutus (engl. simulated annealing) ja perinnölliset algoritmit (engl. genetic algorithms) [11, s. 129].

2.2 Graafiteoria

Graafiteoria on toimiva apuväline mallintamaan erilaisia reititysongelmia. Sen avulla pystytään määrittelemään kartta, jossa toimijat liikkuvat. Graafi koostuu solmupisteistä ja sivuista eri solmupisteiden välillä. Yksinkertainen graafi voidaan esittää järjestettynä parina $G = (V, E)$, joka koostuu joukosta *V* solmupisteitä ja joukosta *E* sivuja [l12, s. 2]. *V* on siis rajattu joukko solmupisteitä ja *E* on joukko sivuja, jotka sijoittuvat kahden tietyn solmupisteen välille. Kaariksi kutsutaan sivuja, joita pitkin pystyy liikkumaan vain tiettyyn suuntaan [13, s. 25].

Graafit tulkitaan Jungnickelin [13, s. 2] mukaan 2D-olioina tasossa. Solmupisteet kuvataan pisteinä ja sivut kuvataan mieluiten suorina viivoina pisteiden välille. Kuvassa 2.1 esitetään viiden solmupisteen ja kahdeksan erilaisen sivun graafi. Solmupisteisiin voi kytkeytyä useita sivuja tai vain yksittäinen. E_1 on sivu, joka lähtee solmusta V_1 ja loppuu pisteeseen V_1. Tällaista sivua kutsutaan Kumarin ja Prasantin [12, s. 2] mukaan *itsenäiseksi silmukaksi*. Kahta solmupistettä, joita yhdistää sivu, kutsutaan *naapureiksi* [13, s. 2]. Esimerkiksi pisteet V_2 ja V_5 ovat naapureita, koska niiden välillä on sivu E_6. Kahta erillistä sivua samojen kahden solmupisteen välillä kutsutaan *rinnakkaisiksi sivuiksi*. Graafia ilman itsenäistä silmukkaa tai rinnakkaisia sivuja kutsutaan yksinkertaiseksi graafiksi. Jos graafi sisältää jommankumman tai molemmat, kutsutaan sitä *yleiseksi graafiksi* [12, s. 2].



Kuva 2.1. Graafi, joka sisältää viisi solmupistettä ja kahdeksan sivua.

Laajentamalla solmupisteiden määrää ja lisäämällä niiden välille sivuja pystytään luomaan laajempia kokonaisuuksia, joita voidaan esimerkiksi käyttää karttoina. Tässä työssä on tarkoituksena esittää algoritmin käyttämät pullonkaulakartat ruudukon tapaisina graafeina Sharonin et al. [7] tapaan.

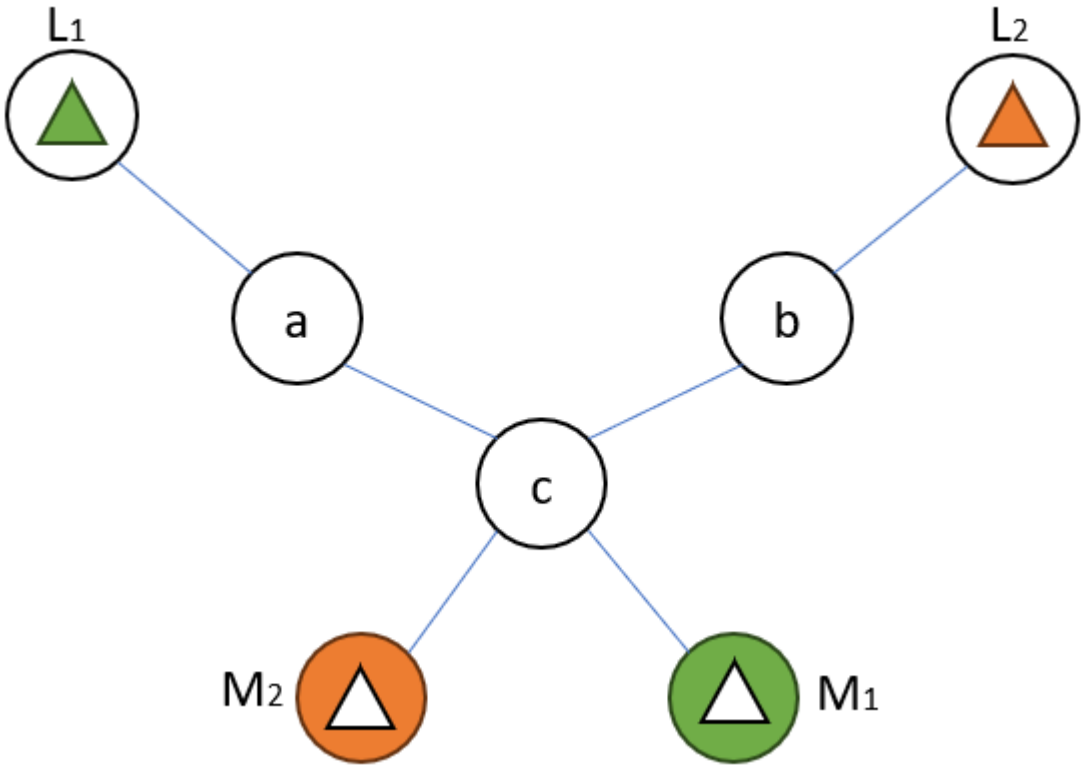
2.3 Usean toimijan reititysongelma

MAPF-ongelma on laajennus yksittäisen toimijan reititysongelmasta usealle toimijalle [7, s. 41]. Laajennettu ongelma koostuu graafista, joka luo karttamaisen rakenteen ongelmalle, ja toimijoiden määrästä. MAPF-ongelmiin optimaalisen ratkaisun löytäminen on laskennalliselta kompleksisuudeltaan NP-vaikea (engl. NP-hard) tehtävä, sillä mahdolliset toimijoiden liikkeet kasvavat eksponentiaalisesti toimijoiden määrän kasvaessa [7, s.41; 14].

Jokaiselle toimijalle on annettu yksilöllinen lähtöpiste ja yksilöllinen maalipiste tavoiteltavaksi. Jokaisella aika-askeleella toimijoilla on mahdollisuus joko liikkua vapaana olevaan viereiseen solmupisteeseen tai odottaa tämänhetkisessä solmupisteessä [15, s. 563]. Tavoitteena on saada ratkaisu, jossa jokainen toimija pääsee maaliinsa ilman, että toimijat törmäävät käyttämällä samaa graafin sivua tai olemalla samassa solmupisteessä

samalla aika-askeleella. Toimijat saavat käyttää jo aikaisemmin käytettyjä sivuja tai solmupisteitä noudattaen edellä mainittuja rajoitteita. Sharonin et al. [15, s. 563] mukaan kokonaiskustannus (engl. sum-of-costs) toiminnalle on jokaisen toimijan ottaman aika-askeleen summa, jonka toimijat ovat ottaneet saavuttaakseen tavoitteensa. Toimijoiden käyttämät aika-askeleet siis summataan yhteen ja summaa kutsutaan kokonaiskustannukseksi. Näin ollen sekä liikkumisen että odottamisen arvot ovat yhtä suuret.

Kuvassa 2.2 ovat lähtöpisteet L_1 ja L_2 sekä maalipisteet M_1 ja M_2. Vihreän kolmion pitää kulkea solmupisteiden a ja c kautta päästäkseen maaliinsa. Oranssin kolmion reitti on pisteiden b ja c kautta. Koska reitit ovat yhtä pitkät, törmäisivät kolmiot pisteessä c, mutta esimerkiksi odotuttamalla oranssia kolmiota yhden aika-askeleen pisteessä b, kulkevat kolmiot törmäyksittä. Näin pystytään laskemaan kokonaiskustannus, joka kuluu tehtävän ratkaisuun. Vihreä kolmio käyttää kolme aika-askelta ja oranssi kolmio neljä aika-askelta, joten optimaalinen kokonaiskustannus sum-of-costs-kustannusfunktiolla tehtävälle on seitsemän.



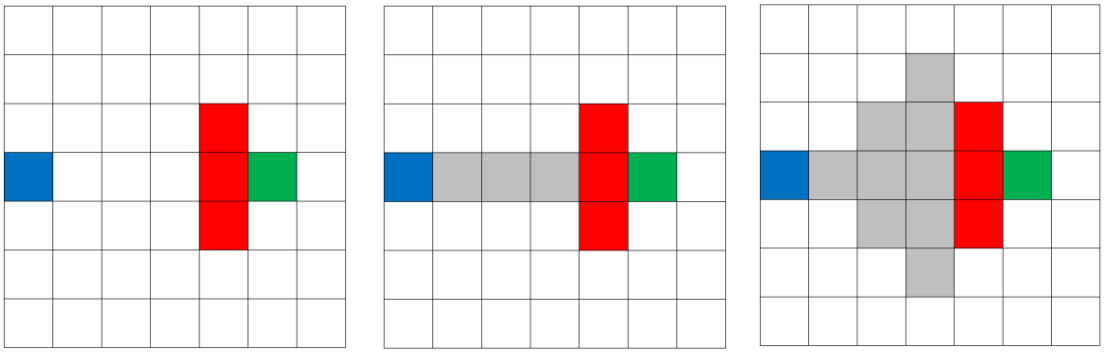
Kuva 2.2. MAPF-ongelma kahdella toimijalla. Vihreän kolmion tavoite on vihreä alue ja oranssilla kolmiolla oranssi alue [7, s. 43].

MAPF-ongelmia löytyy useista eri tieteen ja teollisuuden aloista. Käytännöllisiä käyttötarkoituksia ovat esimerkiksi ajoneuvojen reititys [16], videopeleissä hahmojen liikkuminen [17] tai varastorobotiikka [18].

2.4 A*-algoritmi

A*-algoritmi (A star algoritmi) on monikäyttöinen hakualgoritmi, jota voidaan käyttää lyhimmän reitin etsimiseen muun muassa tietokonepelien hahmoille [19]. A*-algoritmille määritetään lähtö- ja maalipiste. Se käyttää *heuristista arviointia h(x)*, joka antaa ehdotuksen parhaasta maalipisteeseen kulkevasta reitistä. A*-algoritmi käy läpi pisteitä arvion antamassa järjestyksessä, kunnes maalipiste on saavutettu [20]. Cuin ja Shin [19] mukaan A* tutkii jatkuvasti lupaavimpia tutkimattomia pisteitä, jotka se on nähnyt (tutkittujen pisteiden tuntemattomat naapuripisteet). Kun sijainti tutkitaan, algoritmi tarkistaa, onko kyseinen piste maalipiste, jos ei niin algoritmi jatkaa tutkimista pidemmällä kartassa [19].

Kuvassa 2.3 esitetään, kuinka A*-algoritmi toimii heuristista arviointia käyttäen. Toimija lähtee alkupisteestään (sininen ruutu) pyrkimään kohti maalipistettä (vihreä ruutu). Toimija tutkii soluja niiden välillä (harmaa ruutu) ja törmää esteeseen (punainen ruutu). Tämän jälkeen se pyrkii kiertämään esteen päästäkseen maaliin. Algoritmi ei tutki väärässä suunnassa olevia soluja ennen kuin este havaitaan.



Kuva 2.3. Heuristinen arviointi A*-algoritmilla vaiheittain.

A*-algoritmin haittapuoli on se, että se säilyttää kaikki tutkitut pisteet listattuna muistissaan, jonka takia sillä on vaikeuksia suurten ongelmien kanssa. Lisäksi suurten toimijamäärien liikkeiden laskemisesta tulee laskennallisesti mahdotonta [7, s. 45]

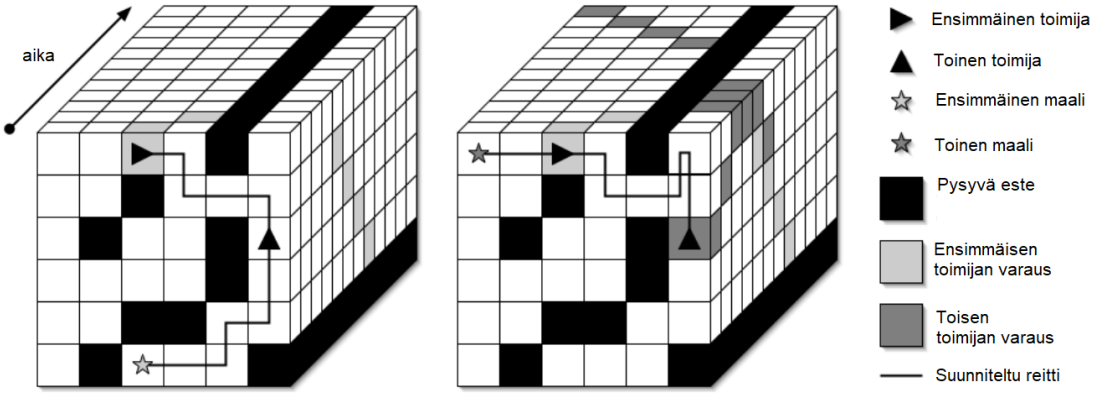
2.5 A*-algoritmi taso-aikakartassa

A*-algoritmin tutkimaan karttaan saadaan kolmas ulottuvuus lisäämällä ongelmaan aika. Normaalin A*-algoritmin karttaa voidaan kutsua *tasokartaksi* ja laajennettua versiota *taso-aikakartaksi* [21]. Silver [21] toteaa, että taso-aikakartta koostuu kolmiulotteisesta

ruudukosta soluja: *Solu(x, y, t)*. Kirjaimet x ja y tarkoittavat solun koordinaatteja 2D-karttapohjassa ja kirjain t merkitsee aika-askelta. Taso-aika-karttaan sijoittuvaa A*-algoritmia voidaan kutsua taso-aika-A* (TAA*).

Kartalla voidaan kuvitella olevan neljä ilmansuuntaa: pohjoinen, itä, etelä ja länsi. Toimija voi joko liikkua yhteen näistä suunnista tai pysyä paikallaan aika-askeleen verran. Tungoksen syntyessä kartalla *odottaminen* voi olla paras ratkaisu, kunnes pullonkaulassa oleva ruuhka vähenee [21]. Esimerkiksi *liikkumalla etelää kohti* yhden askeleen verran toimija siirtyy solusta Solu(x, y, t) soluun Solu(x, y-1, t+1). Jos toimija *odottaa* aika-askeleen verran, siirtyy toimija solusta Solu(x, y, t) soluun Solu(x, y, t+1).

Kuvassa 2.4. esitetään kahdessa osassa kahden toimijan reitinhakutoiminnot. Vasemmanpuoleisessa taso-aikakartassa ensimmäinen toimija varaa itselleen optimaalisen reitin maalipisteeseensä. Samalla ensimmäinen toimija varaa itselleen tiettyinä aika-askeleina käyttämänsä solut, jotka ovat rajoitteita toiselle toimijalla. Toinen toimija etsii itselleen reitin maalipisteeseensä aiheuttamatta konfliktia rajoitteiden kanssa. Oikeanpuoleisesta taso-aikakartasta nähdään, että toinen toimija odottaa kartan oikeassa yläkulmassa kaksi ylimääräistä aika-askelta välttääkseen törmäyksen.



Kuva 2.4. Taso-aikakartta havainnollistettuna [21].

Työssä käytetään konfliktiperustaisen hakualgoritmin alatasolla TAA*-algoritmia. TAA*-algoritmilla pystytään selvittämään yksittäisen toimijan reitti ajan ollessa mukana ongelmassa.

2.6 Konfliktiperustainen hakualgoritmi

Konfliktiperustainen hakualgoritmi on kaksitasoinen algoritmi. Korkeammalla tasolla pyritään välttämään törmäyksiä *konfliktipuussa* eri toimijoiden välillä. Jokainen tietosolu konfliktipuussa koostuu joukosta rajoitteita toimijoiden liikkeille [7, s. 40]. Rajoitteita ovat toisen toimijan sijainti ja liikkuminen tiettyinä ajan hetkinä ja esteet kartalla. Alemmalla

tasolla suoritetaan nopeata yksittäisten toimijoiden reitinetsintää ylemmän tason rajoitteiden sisällä. Useissa tapauksissa kaksitasoinen rakenne mahdollistaa KPH:n tutkimaan vähemmän toimintatiloja kuin yksinkertaisempi A*-algoritmi samalla säilyttäen optimaalisuuden [7, s.40].

KPH ratkaisee usean toimijan reititysongelman jakamalla sen suuremmaksi määräksi pienempiä yhden toimijan reititysongelmia [15, s. 564]. Pääideana on luoda yksilöllinen rajoitteiden verkko jokaiselle toimijalle ja löytää toimijoiden omat reitit, jotka sopivat yhteen rajoitteiden kanssa [15, s. 565].

Ohjelma 1 sisältää tutkimuksessa käytetyn algoritmin pseudokoodin. Algoritmi ensin etsii toimijoille reitit ilman rajoituksia. Se lisää reitit rajoitteina konfliktipuuhun. Tämän jälkeen se etsii konflikteja toimijoiden väliltä. Sitten algoritmi luo kaksi uutta vaihtoehtoista reittiä toimijoille, joissa toimijat väistävät toisiaan eri tavoin. Toimintaa toteutetaan, kunnes konflikteja ei enää tapahdu.

```
    noodi = etsi reitit yksittäisille toimijoille ilman rajoituk-
2 sia.
    Lisää noodi konfliktipuuhun.
4
    while konfliktipuu ei ole tyhjä:
6       paras = noodi, jolla on alin kustannus konfliktipuussa

8       Tarkista paras ratkaisu, kunnes konflikti tapahtuu.
        if konfliktia ei tapahdu:
10          return paras

12      konflikti = Etsi ensimmäiset 2 toimijaa, joilla on risteävät
  reitit
14
        uusi_noodi1 = noodi, jossa ensimmäinen toimija väistää
16 toista toimijaa
        Lisää uusi_noodi1 konfliktipuuhun.
18
        uusi_noodi2 = noodi, jossa toinen toimija väistää ensim-
20 mäistä toimijaa
        Lisää uusi_noodi2 konfliktipuuhun.
22
```

Ohjelma 1: Tutkimuksessa käytetyn algoritmin pseudokoodi [22].

Koska KPH sisältää useita toimijoita samanaikaisesti, toimii KPH:n alatasolla *taso-aika-A** (TAA*). Eri toimijoiden liikkeet pystytään näin ollen tallentamaan tietyille ajan hetkille rajoitteina. Voidaan kuvitella, että jokainen toimija luo kohdalleen samanlaisen esteen kuin karttapohjan seinämät. Tällainen este on kuitenkin vain niinä aika-askeleina, kun toimija kyseisessä pisteessä on.

2.7 Kaivosympäristön ominaisuudet

Kaivosympäristöjä on monenlaisia, kuten maanpäällisiä avolouhoksia [23] tai maanalaisia kaivoksia, jotka käyttävät kaivoskuiluja tai kaltevia tunneleita. Jälkimmäisiä käytetään, jos mineraaliesiintymät ovat syvällä ja niitä ei pystytä louhimaan maanpäällisiä kaivoksia käyttäen [24]. Lisäksi jos malmiesiintymä on jyrkkä, on kustannustehokkaampaa louhia tunneleiden avulla [6].

Koska tunneleiden kaivaminen maan alla on erittäin kallista, ei kaivoksissa ole yleensä resursseja luoda suuria liikennekäytäviä maan alle. Lisäksi suuremman tunnelin rakentaminen vaatii tukirakenteita enemmän verrattuna pienempään tunneliin. Kaivuu ei ole taloudellisesti niin kannattavaa, jos kaivuun yhteydessä ei louhita haluttua malmia. Maanalaisten kaivosten tunnelit ovat poikkileikkaukseltaan pieniä [24] ja sen takia ohituspaikkoja on harvassa. Kaivoksissa louhiessa osa työkoneista tuo louhittua kiviainesta maan pinnalle tyhjennystä varten ennen kuin ne palaavat takaisin hakemaan uutta lastia. Tällöin kaivoksen kulkutunneleihin syntyy pullonkauloja, jotka hidastavat toimintaa ja tuotantoa.

Louhimistapoja on monia erilaisia ja tässä esitetään niistä olennaisimmat tähän tutkimukseen liittyen. Pilarilouhinta on louhimistapa, jota yleensä käytetään hiilen louhimiseen [6, 25]. Siinä ideana on kaivaa suuria tiloja hiiliesiintymään ja jättää suuria hiilipylväitä tukemaan rakennetta [6]. Toinen louhimistapa on suonilouhinta, jota käytetään pienikokoisten suoniesiintymien louhimiseen. Tarkoituksena on kaivaa käytävä suonta pitkin, jolloin mineraaliköyhän materiaalin määrä on mahdollisimman pieni [25].

3. TUTKIMUSMENETELMÄT JA -AINEISTO

Tutkimuksessa on käytetty kapeaa muutaman ohituspaikan sisältävää käytävää, jonka päässä on kääntymiseen tarkoitettu silmukka. Kyseisen kartan on tarkoitus luoda pullonkaulaosioita ja kuvata näin suonilouhinnan erityispiirteitä kuten ahtautta.

Lisäksi tutkimuksessa käytettiin karttaa, jossa on pyritty mallintamaan suuria tiloja ja näiden välille hiilipylväitä, mitkä ovat yleisiä pilari-louhimismenetelmässä. Kolmas kartta on täsmälleen samanlainen kuin aikaisempi, mutta siitä on poistettu toinen kulkureitti sisään luolastoon. Alla olevissa luvuissa kartat esitellään tarkemmin.

Oletuksena on, että toimijoiden määrän kasvattamien kasvattaa ratkaisuun kulunutta aikaa sekä ylemmän ja alemman tason algoritmien iteraatiomääriä. Lisäksi ahtaus ja pullonkaulojen määrä hidastavat ratkaisua ja lisäävät iteraatiomääriä. Tarkoituksena tutkimuksessa ei ole löytää optimaalista ratkaisua ongelmalle vaan pelkkä ratkaisu riittää.

3.1 Ongelman ratkaisun perusteet

Tässä työssä tutkittiin monitoimijareitityksen suppenemista erilaisilla kaivoskartoilla toimijamäärän funktiona. Toimijat liikkuvat kartalla samanaikaisesti, mikä voi aiheuttaa tukoksia karttaan ja vaikeuksia ratkaisualgoritmille. Tukoksilla tarkoitetaan, että toimijoiden liike muuttuu enenemissä määrin odottamiseksi liikkumisen sijaan. Tällöin algoritmi ei pysty saavuttamaan ratkaisua välttämättä kyllin nopeasti halutussa ajassa. Tutkimuksessa algoritmille annettiin tunti aikaa ratkaista ongelma. Käytettävissä ollut aika valittiin kahdesta syystä. Useiden suoritusmäärien takia aikaa täytyi rajata, mutta aikaa haluttiin riittävästi käyttöön, että algoritmin suorituskykyä pystyttiin arvioimaan uskottavasti. Tutkimuksessa vertaillaan sekä ylä-, että alatason algoritmien iteraatioiden määrien oletettua kasvua toimijoiden määrän lisääntyessä. Ongelman ratkaisuun käytetään muokattua versiota käyttäjän ”GavinPHR” [22] koodipohjasta.

Koodipohjaan lisättiin uusi määräysfunktio, joka määrittää listassa olevan ensimmäisen lähtöpisteen ja ensimmäisen maalipisteen pariksi tietylle toimijalle. Tämän jälkeen toinen lähtöpiste ja toinen maalipiste pariksi eri toimijalle. Tätä toistetaan, kunnes jokainen listassa ollut lähtö- ja maalipiste on määrätty toimijoille. Koodipohjaan luotiin myös uusia karttoja tutkimusta varten.

3.2 Tutkimuksessa käytetyt kartat

Algoritmissa jokaisessa kartassa karttapohjana toimii ruudukko, jonka yksi neliö on 100*100 pikseliä. Neliön keskipiste toimii siis solmupisteenä. Toimijat siis liikkuvat vapaissa neliöissä ja sijoittuvat niiden keskelle. Toimijoiden koko on yhtä suuri kuin yksittäisen ruudun koko. Vaakasuunnassa kartassa ruutuja oli 19 kappaletta. Pystysuunnassa ruutuja oli 11 kappaletta. Kartan koko oli 1920*1080, minkä takia kyseiset ruutumäärät.

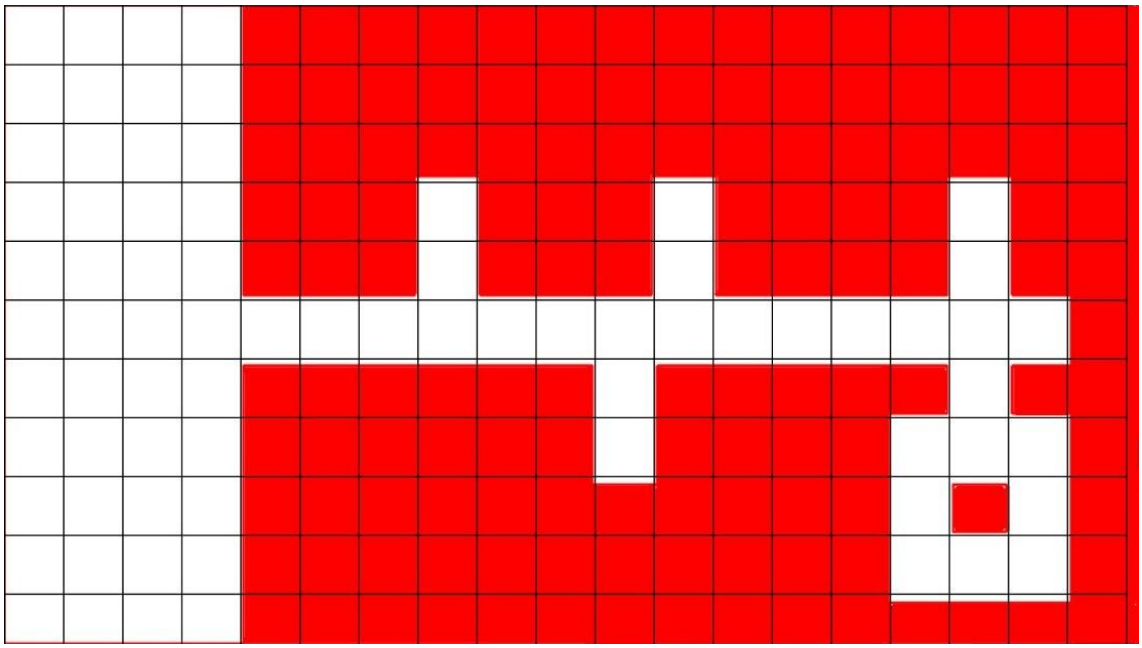
Kartoissa punaisella on merkitty seinämät, joihin toimijat eivät pääse liikkumaan. Valkoisella pohjalla toimijat voivat vapaasti liikkua väistellen toisiaan ruudukossa.

Maalien ja lähtöjen paikoitusperiaatteena oli toimijoille kriteereinä koko kartan käyttö ja konfliktien varmistaminen toimijoiden reiteillä. Koko kartan käytöllä tarkoitetaan, että tietyn toimijan lähtö- ja maalipiste sijoitettiin eri puolille karttaa.

3.2.1 Kartta 1

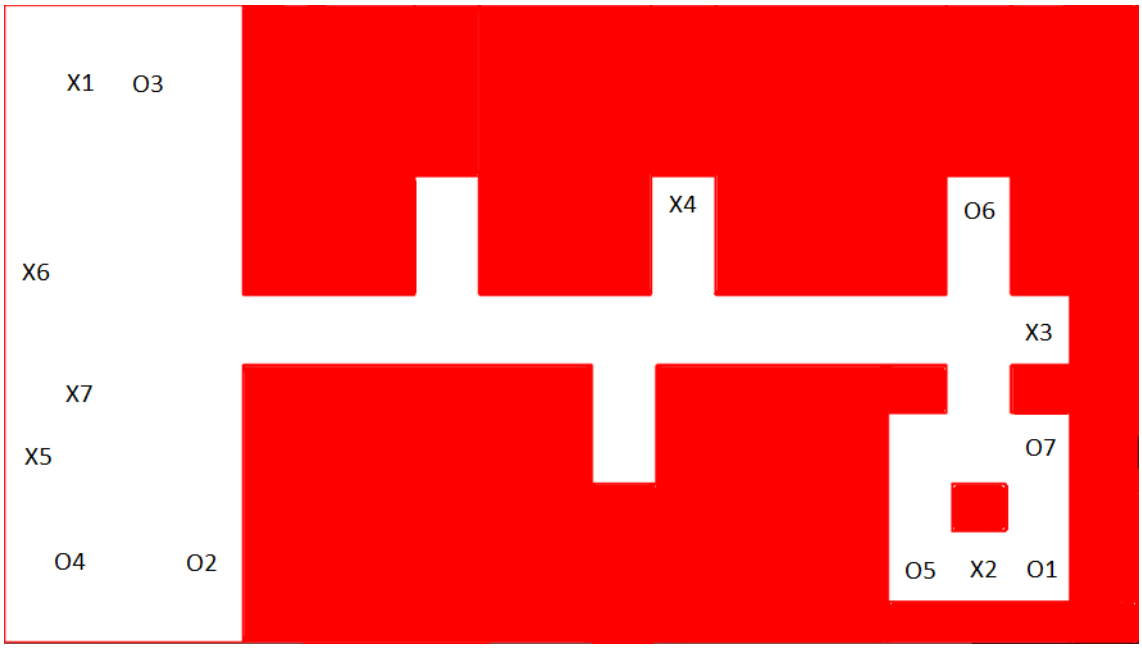
Ensimmäinen kartta sisälsi suonilouhimistapaa esittävän kartan pitkällä ja kapealla käytävällä, joka sisälsi muutamia ohituspaikkoja ja päätysilmukan käytävän päässä. Kartta 1 oli kolmesta kartasta kaikista ahtain.

Kuvassa 3.1 on esitettynä karttapohjana käytetty 19*11 ruudukko kartan 1 päällä. Kuvan 3.1 oikeassa reunassa eivät ruudukot kata koko karttaa, sillä pohjassa oli 20 pikseliä ylimääräistä. Sama asia tapahtuu alareunassa, mutta toisinpäin: pohja oli 20 pikseliä vajaa, mutta pyöristyy silti kokonaiseksi ruuduksi. Kartassa näkyy myös pieniä valkoisia osuuksia seinämien käyttämissä ruuduissa. Seinämät antavat käytävälle periksi 5–10 pikseliä riippuen kohdasta. Tällä pyrittiin varmistamaan toimijoiden varma liikkuminen ilman koodissa tulevia häiriöitä tai vikailmoituksia.



Kuva 3.1. Kartta 1, jossa on esitettynä karttapohjan ruudukko.

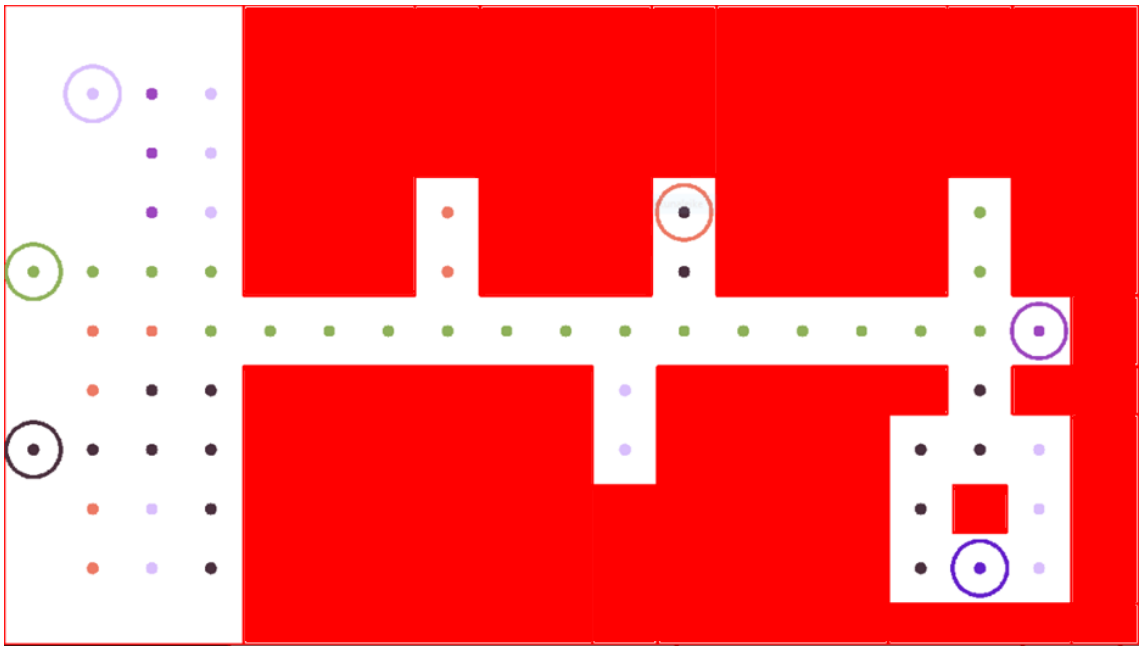
Kuvassa 3.2 punaisella merkatut osat ovat seinämiä, joissa toimijat eivät pysty liikkumaan. Kuvassa X-alkuiset merkinnät ovat numeroituja toimijoiden lähtöpisteitä ja O-alkuiset ovat maalipisteitä. Esimerkiksi toimija, joka lähtee pisteestä X1, etsii itselleen reitin pisteeseen O1 samalla välttäen törmäyksiä muiden toimijoiden tai seinämien kanssa.



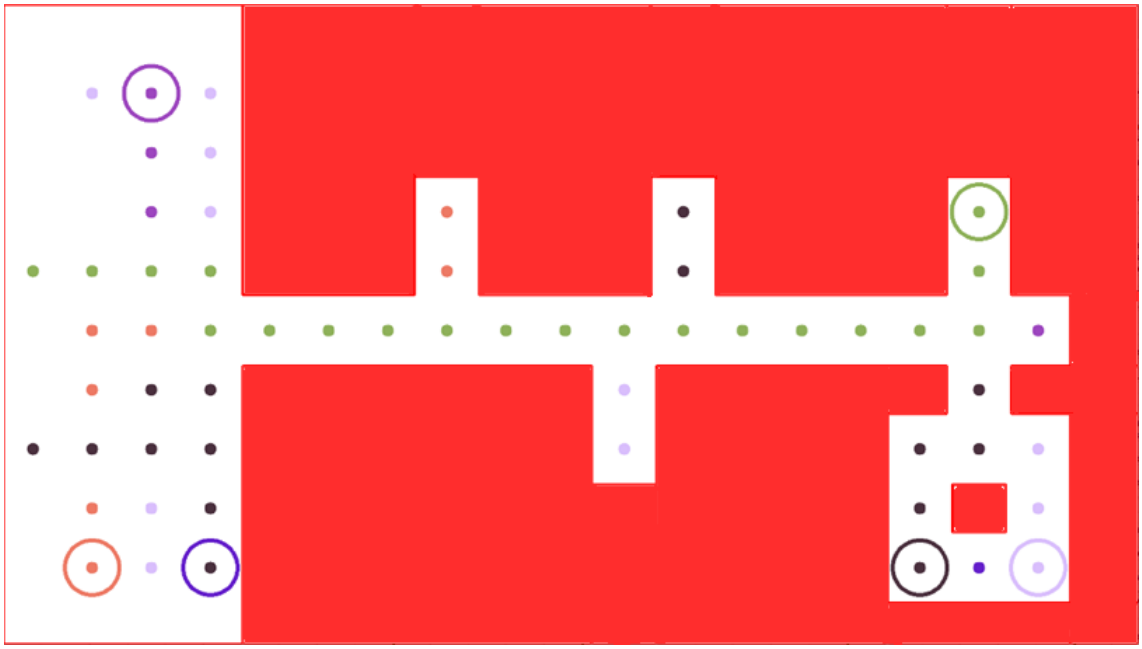
Kuva 3.2. Ensimmäisen kartan pohjapiirustus.

Kuvissa 3.3 ja 3.4 esitetään kuuden toimijan samanaikainen lähtö- ja maalitilanne. Toimijat ovat liikkuneet värjäämiään ruutuja pitkin maaleihinsa vältellen konflikteja. Toimijat värjäävät reittinsä edellisen päälle, joten jälkimmäisestä kuvasta 3.4 voidaan tarkastella,

missä mikäkin toimija on liikkunut viimeisimpänä. Kyseisen ongelman ratkaisussa, osa toimijoista väistää muita toimijoita keskellä näkyviin ohituspaikkoihin, kunnes heidän oma vuoronsa tulee. Kuvasta 3.4 nähdään, että algoritmi on ratkaissut ongelman niin, että viimeisimpänä ohituspaikkoja käyttäneet ovat olleet musta, oranssi ja vaaleanvioletti toimija.



Kuva 3.3. Kuuden toimijan reiteistä lähtötilanteessa.



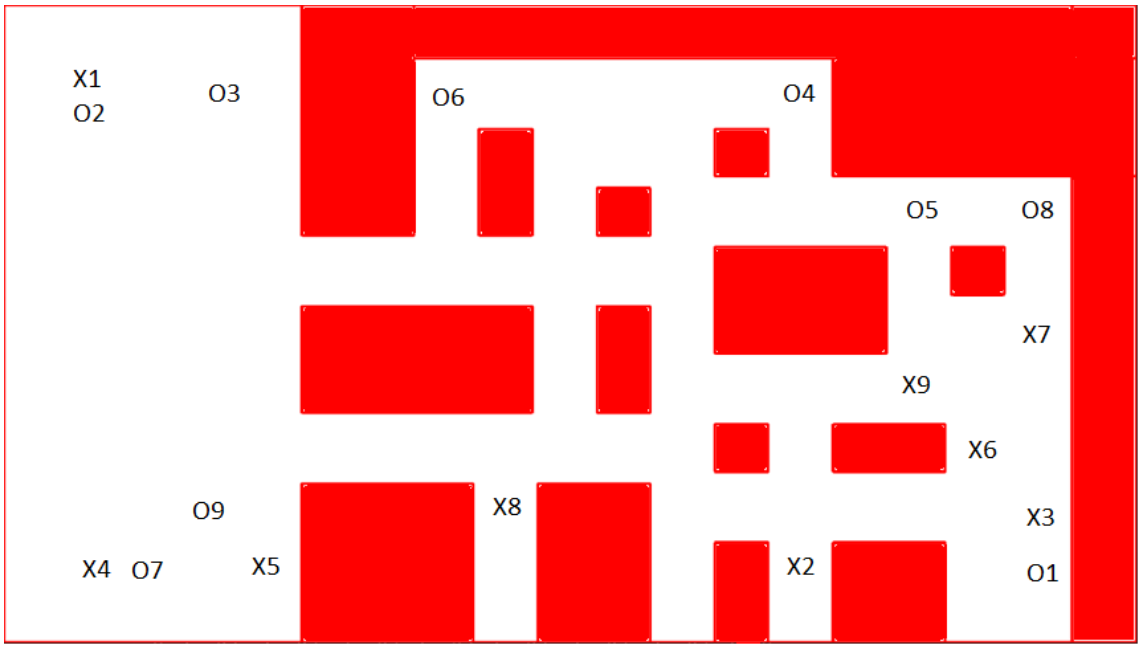
Kuva 3.4. Kuuden toimijan reiteistä maalitilanteessa.

Ensimmäisessä kartassa algoritmin oletetaan pärjäävän huonoiten. Tämä johtuu kartan ahtaudesta verrattuna muihin avoimempiin karttoihin sekä siitä, että kartassa on käytössä vähemmän ruutuja.

3.2.2 Kartta 2

Toinen kartta on luotu kuvaamaan huone ja pilari louhimistapaa. Kartta sisältää kyseiseen louhimistapaan liittyviä avaria tiloja sekä tilaa tukevia pilareita.

Kuvassa 3.5 on esitettynä kartassa 2 käytetyt lähtö- ja maalipisteet karttapohjassa. Kuvassa 3.5 on enemmän lähtö- ja maalipisteitä kuin kuvassa 3.2, koska algoritmi pärjäsi paremmin toisessa karttapohjassa. Huomautuksena pisteet O2 ja X1 sijaitsevat samoissa koordinaateissa. Tämä tarkoittaa, että pisteestä X1 lähtevän toimijan on lähdettävä liikkeelle ennen kuin toimijan, jonka maalipiste on O2 saapuu kyseiseen ruutuun.

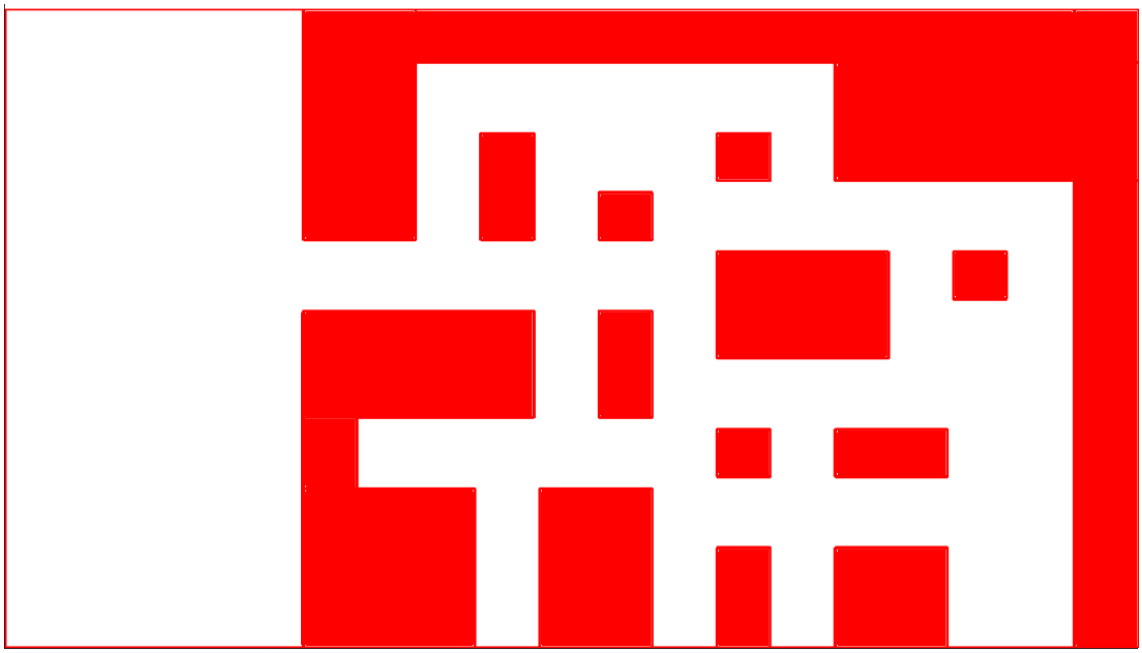


Kuva 3.5. Toisen kartan pohjapiirustus lähtö- ja maalipisteillä.

Toisen kartan oletetaan pärjäävän tutkimuksessa parhaiten, koska kartta on avonaisin käytetyistä kartoista. Lisäksi kartassa on kaksi sisäänkäyntiä luolastoon vähentämässä toimijoiden välisiä konflikteja.

3.2.3 Kartta 3

Kuvassa 3.6 on esitettynä kolmas karttapohja. Karttapohja on täsmälleen samanlainen kuin kuvan 3.5 kartan, mutta alempi sisäänkäynti luolastoon on suljettuna. Tällä muutoksella pyrittiin luomaan enemmän pullonkauloja melko avonaiseen karttaan, mikä lisäsi konfliktien määrää toimijoilla. Vaikka kuvassa 3.6 ei näy lähtö- tai maalipisteitä, ne olivat täsmälleen samat kuin kuvan 3.5. Pisteiden sijainnit pidettiin samana, että kartat olisivat vertailtavia.



Kuva 3.6. Kolmannen kartan pohjapiirustus.

Kolmannessa kartassa pyrittiin vertailemaan toisen sisäänkäynnin poistamisen vaikutusta karttaan 2 verrattuna. Haluttiin tutkia, kuinka suuri vaikutus algoritmin toimintaan edellä mainitulla muutoksella saataisiin.

3.3 Käytetyt työkalut

Työssä on käytetty ohjelmointiympäristönä Visual Studio Codea ja ohjelmointikielenä Pythonia. Tutkimus on suoritettu käyttäen henkilökohtaista pöytätietokonetta. Pöytätietokoneen käyttöjärjestelmänä toimii Windows 10 Home. Tietokoneen prosessori on Intel 13600-K ja käytössä on 16 GB DDR5 RAM-muistia.

Käytetty Pythonin versio oli 3.11. Visual Studio Coden käytetty versio oli 1.84.2. Lisäksi moduuleja, joita täytyi ladata algoritmin käyttöä varten, oli: NumPy 1.26.2, PyYAML 6.0.1 ja opencv-contrib-python 4.8.1.78.

3.4 Kokeellisen tutkimuksen suoritustapa

Kokeellinen tutkimus suoritetaan käyttämällä algoritmia halutussa kartassa tietyllä toimijamäärällä yhden tunnin ajan. Tunnin aikana algoritmin tavoitteena on ratkaista ongelma ja pysäyttää ohjelma. Ratkaisun löytyessä kirjataan ylös algoritmin käyttämät iteraatiomäärät ylä- ja alatasoilla. Samalla merkitään, missä kartassa saatu tulos tehtiin ja millä toimijamäärällä. Jos algoritmi ei saavuta tunnin aikana ratkaisua ongelmaan, se pysäytetään ja tulokseksi kirjataan ”ei ratkaisua”.

Suoritus aloitetaan käyttäen yhtä toimijaa kartassa 1. Onnistuneen suorituksen jälkeen lisätään yksi toimija lisää ja ratkaisu aloitetaan alusta. Aikaisemman toimijan lähtö- ja maalipisteet pysyvät aina samoina, että tuloksia pystytään vertailemaan. Toimijoiden määrää kasvatetaan jokaisella iteraatiolla samalla kuin aikaisemmat toimijat pitävät omat asemansa, kunnes algoritmi ei pysty ratkaisemaan karttaa halutussa aikamääreessä. Testi toistetaan muissa kartoissa samalla tavalla.

Kokeen aikana tietokoneelta suljetaan muut ohjelmat, jotka kuluttavat prosessointitehoa tai muistia. Näin pyritään luomaan mahdollisimman tasalaatuinen ja vertailtava tutkimusympäristö.

4. TULOKSET JA NIIDEN TARKASTELU

Algoritmia suoritettiin kolmessa eri karttatyypissä, kunnes algoritmi ei saanut ratkaistua reititysongelmaa yhden tunnin aikana. Tulokset kirjattiin ylös ja ne ovat esitettyinä tässä luvussa. Toimijoita lisättiin yksi jokaisen onnistuneen ratkaisukerran jälkeen. Kun ratkaisua ei saatu, kirjattiin tulokseksi *ei ratkaisua*. Tutkimuksessa saatiin kerättyä hyödyllistä dataa, josta on luotu erilaisia kuvaajia. Taulukkoihin 1–3 kerättiin eri kartoista saadut tulokset, joiden avulla alla olevat kuvaajat on luotu.

Taulukkoon 1 on kerätty ensimmäisen kartan tulokset. Se sisältää tiedot ratkaisuajasta, ja ylä- ja alatason iteraatiomääristä. Ylä- ja alatason sarakkeet kuvaavat algoritmin eri osien iteraatiomääriä, mitä käytetään myös taulukoissa 2 ja 3.

Taulukko 1. Ensimmäisen kartan tulokset

Toimija määrä	Aika (s)	Ylätaso (KPH)	Alataso (TAA*)
1	0,4975	1	26
2	9,0355	4	29 628
3	55,8642	8	313 335
4	121,2352	11	747 057
5	277,5805	18	1 279 043
6	2476,4742	112	17 902 001
7	Ei ratkaisua	Ei ratkaisua	Ei ratkaisua

Toisen kartan tulokset on koottu taulukkoon 2. Taulukosta 2 voidaan lukea, että ongelman ratkaiseminen epäonnistui yhdeksännen toimijan jälkeen.

Taulukko 2. Toisen kartan tulokset

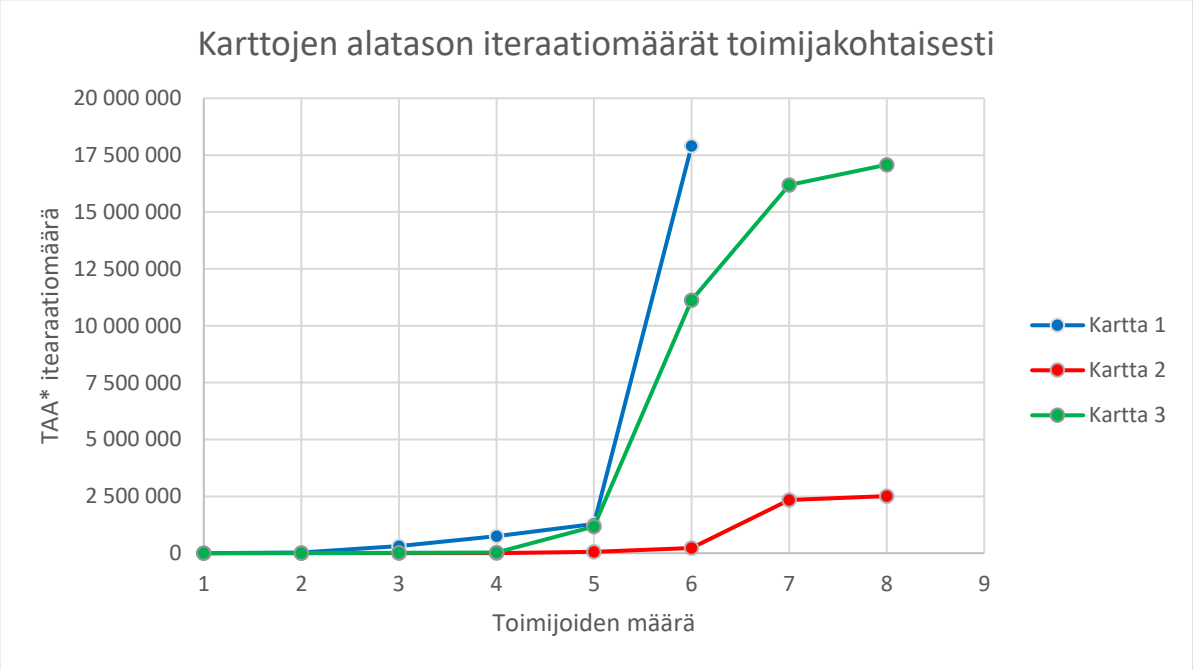
Toimija määrä	Aika (s)	Ylätaso (KPH)	Alataso (TAA*)
1	0,4978	1	25
2	0,5176	1	46
3	0,4944	1	68
4	2,6639	3	2934
5	30,2595	7	61 062
6	98,1339	10	225 394
7	1103,5166	57	2 339 112
8	1141,6029	59	2 508 880
9	Ei ratkaisua	Ei ratkaisua	Ei ratkaisua

Taulukkoon 3 on kerätty kolmannesta kartasta saatu data. Kolmannessa kartassa algoritmi epäonnistui ratkaisussa yhdeksällä toimijalla kuten toisessa kartassakin.

Taulukko 3. Kolmannen kartan tulokset

Toimija määrä	Aika (s)	Ylätaso (KPH)	Alataso (TAA*)
1	0,5297	1	25
2	5,3748	2	4946
3	13,8489	4	20 370
4	14,0578	4	23 176
5	219,0382	10	1 174 400
6	1886,9854	36	11 122 191
7	2046,9923	58	16 185 294
8	2173,862	61	17 078 214
9	Ei ratkaisua	Ei ratkaisua	Ei ratkaisua

Kuvan 4.7 kuvaajassa on esitetty jokaisen kartan alatason algoritmin iteraatiomäärät lineaarisesti interpoloituna. Kuvasta 4.7 voidaan todeta, että ensimmäisessä kartassa iteraatiomäärät kasvoivat yli kymmenkertaisesti toimijoiden määrän lisääntyessä viidestä kuuteen. Samalla taulukosta 1 voidaan lukea ratkaisuajan kasvu melkein kymmenkertaiseksi.

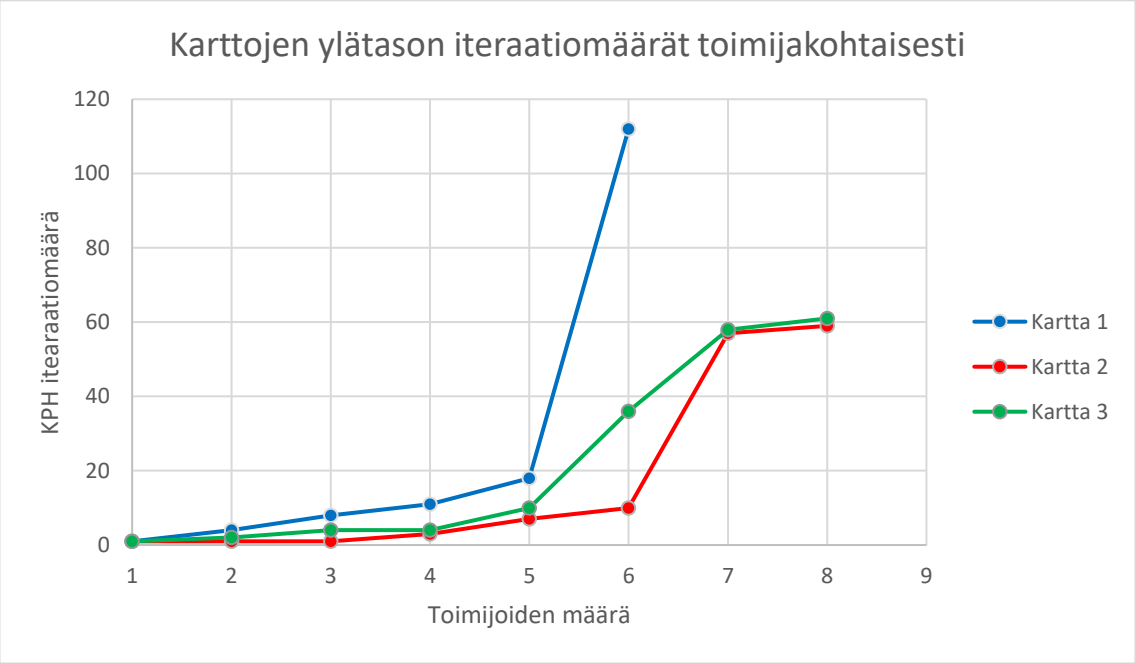


Kuva 4.1. Karttojen alatason algoritmin iteraatiomäärät kuvaajassa esitettynä.

Kuvasta 4.7 havaitsee hyvin, kuinka yksi sisäänkäynti vähemmän vaikuttaa suuresti kuuden tai suuremman toimijamäärän ongelmissa kartoissa 2 ja 3. Kun sisäänkäyntejä on vain yksi, kartoissa 1 ja 3 huomaa selvän yhteneväisyyden. Ongelman muuttuessa vaikeammaksi kuudennen toimijan lisäyksen yhteydessä nousivat iteraatiomäärät kummassakin kartassa. Taas kartta 2 pystyttiin ratkaisemaan lähes 50-kertaa pienemmällä iteraatiomäärällä verrattuna karttaan 3.

Kartassa 1 algoritmi ei pystynyt ratkaisemaan ongelmaa seitsemällä toimijalla. Kartoissa 2 ja 3 KPH pystyi ratkaisemaan ongelman ja kummankin iteraatiomäärät kasvoivat. Seitsemännen ja kahdeksannen toimijan välillä ero ei kuitenkaan kummassakaan kartassa ollut kovin suuri prosentuaalisesti. Yllättävää oli, että kartta 2 ei onnistunut saamaan ratkaisua yhdeksällä toimijalla, vaikka se oli aiemmilla toimijamäärillä pärjännyt muita karttoja huomattavasti paremmin. Tämän voi selittää mahdollisesti yhdeksännen toimijan luomilla konflikteilla, joita se jo valmiiksi monimutkaiseen ongelmaan lisäsi.

Kuvan 4.8 kuvaajassa on esitettynä ylätason algoritmin iteraatiomäärät lineaarisesti interpoloituna. Pienillä toimijamäärillä ensimmäisen kartan KPH:n iteraatiomäärät nousivat tasaisesti, kunnes kuudes toimija moninkertaisti iteraatiomäärän.



Kuva 4.2. Karttojen ylätason algoritmin iteraatiomäärät kuvaajassa esitettynä.

Taulukosta 2 pystyy selvemmin lukemaan kartan 2 pienen toimijamäärän iteraatiomäärät, kuten sen että konflikteja ei tapahtunut ensimmäisen kolmen toimijan välillä lainkaan. Kartan 3 iteraatiomäärät nousivat melko tasaisesti viiteen toimijaan saakka.

Kuudes toimija kasvatti kartan 3 ylätason algoritmin iteraatiomäärää yli kolminkertaiseksi. Kartassa 2 ei vielä kuudennen toimijan lisäys vaikuttanut aikaisempaa enempää. Odottamattomasti kartat 2 ja 3 olivat melkein tasan iteraatiomäärissä sekä seitsemän että kahdeksan toimijan reititysongelmissa.

Toimijoiden määrän kasvattaminen yhdelläkin toimijalla moninkertaisti varsinkin alatason iteraatiomäärät. Iteraatiomäärät kasvoivat lähes eksponentiaalisesti alatasolla. Ylätа-

solla toimijoiden määrän kasvattamisen sijaan toimijoiden lähtö- ja maalipisteet vaikuttivat iteraatiomääriin enemmän. Jos jo valmiiksi ruuhkautuneelle reitille laittaa lisää toimijoita kulkemaan, aiheuttaa se enemmän työtä ylätason KPH:lle. Jos toimijan olisi sijoittanut sivumpaan, olisi se kulkenut omin toimin, ilman muiden toimijoiden reitteihin vaikuttamista. Alatasolla iteraatiomääriin vaikuttaa myös sijainti, mutta myös määrä, sillä jokainen toimija lisää eksponentiaalisesti tutkittavia solmupisteitä alatason TAA*-algoritmille. Sijainti vaikuttaa myös rajoitteiden määrään, mikä ylätasolla kerätään ja annetaan alatason ratkaistavaksi, millä on oma vaikutuksensa kasvavaan iteraatiomäärään. Lähtö- ja maalipisteiden satunnaisuuden tutkimista on myös seuraavassa luvussa ehdotettu jatkotutkimuskohteeksi.

5. YHTEENVETO

Työn tavoitteena oli selvittää konfliktiperustaisen hakualgoritmin toiminnan tehokkuus erilaisissa kaivosympäristöön perustuvissa kartoissa eri toimijamäärillä. Algoritmia suoritettiin kartoissa yksitellen, kunnes ratkaisua ei pystytty enää saamaan halutussa aikamääreessä. Onnistuneiden yritysten jälkeen toimijamäärää kyseisessä kartassa lisättiin ja aiemman suorituksen data kerättiin talteen myöhempää tarkastelua varten.

Työssä käytettiin sitä varten muokattua versiota konfliktiperustaisesta hakualgoritmista usean toimijan reititysongelman ratkaisuun kaivosympäristössä. Konfliktiperustaisen hakualgoritmin käyttö on toimiva ratkaisu pullonkaulakartoissa, joita kaivosympäristössä esiintyy, verrattuna avoimissa ympäristöissä paremmin pärjääviin muunlaisiin ratkaisualgoritmeihin.

Algoritmin toimintaa täytyi muuttaa lisäämällä oma toimijoiden lähtö- ja maalipisteiden määrittävä funktio koodipohjaan. Lisäksi käyttöä varten täytyi luoda omia toimivia karttapohjia järkevällä pohjaruudukon koolla, jotta tilaa oli tarpeeksi toimijoiden liikkeille, muttei myöskään liikaa, mikä hidastaisi algoritmin suoritusta liikaa. Kohtalaisen kokoisen ruudukon valinta varmisti riittävien tulosten saannin karttojen monimutkaisuutta poistamatta.

Työssä saatiin selville erilaisten karttapohjien vaikutusta ongelman ratkaisun vaikeuteen. Pystyttiin myös päättelemään, että yhdenkin toimijan lisäys karttaan yleensä moninkertaistaa iteraatiomäärät sekä ylä- että alatason algoritmeissa. Kun karttapohja oli ahtaampi, aiheutti se huomattavasti enemmän työtä algoritmille. Yhden lisäväylän poistaminen kartasta vaikutti suuresti ratkaisutehokkuuteen yli viisi toimijaa sisältävissä testeissä.

Yleisesti ottaen tutkimuksessa saatiin kerättyä laadukasta vertailukykyistä dataa konfliktiperustaisen hakualgoritmin iteraatiomääristä ja käytetyllä prosessointi- ja muistiteholla saavutetuista ratkaisuajoista. Tutkimus antaa pohjaa alla mainituille jatkotutkimuskohteille.

Mahdollisia jatkotutkimuskohteita ovat konfliktiperustaisen hakualgoritmin toiminnan tehostaminen tai kokonaan uudenlaisen algoritmin kehittäminen ongelman ratkaisuun. Jos tarkoituksena on käyttää kyseistä algoritmia oikeiden autonomisten kaivosajoneuvojen reitittämiseen, täytyy algoritmia parannella, että toimijat käyttäytyvät enemmän ajoneuvomaisesti. Tällä tarkoitetaan, että toimijat kääntyisivät pehmeämmin käyrämäisesti solusta soluun kuin nykyisen algoritmin suorakulmaisilla hypyillä. Myös aikatason muutta-

minen diskreetistä jatkuvaksi olisi erittäin hyvä jatkokehityskohde. Lisäksi lähtö- ja maalipisteiden muuttamisella satunnaisiksi pystyttäisiin tutkimaan, kuinka paljon merkitystä sillä olisi tuloksiin.

LÄHTEET

[1] P. Jaillet, J. Qi, M. Sim. "Routing optimization under uncertainty." *Operations research* 64.1 2016: 186-200.

[2] C. Liu, J. Wang, W. Cai, Y. Zhang. An energy-efficient dynamic route optimization algorithm for connected and automated vehicles using velocity-space-time networks. *IEEE access*, *7*, 2019, 108866-108877.

[3] M. Ng, Y. Chong, K. Ko, Y. Park, Y. Leau. Adaptive path finding algorithm in dynamic environment for warehouse robot. Neural computing & applications. 2020;32(17):13155–71.https://browzine.com/libraries/2233/journals/3390/issues/352231721

[4] R. Sharda, et al. *The Vehicle Routing Problem: Latest Advances and New Challenges*. 1. Aufl. Vol. 43. Boston, MA: Springer-Verlag, 2008. https://link-springer-com.libproxy.tuni.fi/book/10.1007/978-0-387-77778-8

[5] Caterpillar Performance Handbook. forty-fifth ed. Caterpillar. Inc. 2015. https://www.wagnerusedequipment.com/wp-content/uploads/2021/09/Performance_Handbook.pdf

[6] H. Harraz. Underground mining Methods. 10.13140/RG.2.1.2881.1124. (2010) https://www.researchgate.net/publication/301833045_Underground_mining_Methods?channel=doi&linkId=5729f56808ae2efbfdbc10c2&showFulltext=true

[7] G. Sharon, R. Stern, A. Felner, NR. Sturtevant, “Conflict-Based Search for Optimal Multi-Agent Pathfinding.” *Artificial intelligence* 219 2015: 40–66. https://browzine.com/libraries/2233/journals/2266/issues/7529997

[8] G. Konstantakopoulos, S. Gayialis, E. Kechagias. Vehicle routing problem and related algorithms for logistics distribution: a literature review and classification. Operational research. 2022;22(3):2033–62.

[9] N. Shetty, B. Sah, SH. Chung. Route optimization for warehouse order picking operations via vehicle routing and simulation. SN applied sciences. 2020;2(2):311-.

https://link-springer-com.libproxy.tuni.fi/article/10.1007/s42452-020-2076-x

[10] T. Caric. Vehicle routing problem. Gold H, editor. Rijeka, Croatia: IntechOpen; 2008. https://directory.doabooks.org/handle/20.500.12854/64674

[11] P. Toth, D. Vigo, eds. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, 2002. https://industri.fatek.unpatti.ac.id/wp-content/uploads/2019/03/002-The-Vehicle-Routing-Problem-Monograf-on-discrete-mathematics-and-applications-Paolo-Toth-Daniele-Vigo-Edisi-1-2002.pdf

[12] R. Kumar, PK. Pattnaik. Graph Theory. Laxmi Publications Pvt Ltd; 2018.

[13] D. Jungnickel. *Graphs, Networks and Algorithms*. 3rd ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. https://link-springer-com.libproxy.tuni.fi/book/10.1007/978-3-540-72780-4

[14] Y. Jingjin, S. LaValle. "Structure and intractability of optimal multi-robot path planning on graphs." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 27. No. 1. 2013.

[15] G. Sharon, R. Stern, A. Felner, NR. Sturtevant. “Conflict-Based Search For Optimal Multi-Agent Path Finding.” Proceedings of the ... AAAI Conference on Artificial Intelligence 26.1 2021: 563–569. https://ojs.aaai.org/index.php/AAAI/article/view/8140

[16] K. Dresner, P. Stone. “A Multiagent Approach to Autonomous Intersection Management.” The Journal of artificial intelligence research 31 2008: 591–656. https://jair.org/index.php/jair/article/view/10542

[17] D. Silver. “Cooperative Pathfinding.” Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 1.1 2021: 117–122. Web.

[18] W. Honig, S. Kiesel, A. Tinka, J. Durham, N. Ayanian. “Persistent and Robust Execution of MAPF Schedules in Warehouses.” IEEE robotics and automation letters 4.2 2019: 1125–1131. https://ieeexplore.ieee.org/document/8620328

[19] X. Cui, H. Shi. A*-based Pathfinding in Modern Computer Games. 2010:11 https://www.researchgate.net/publication/267809499_A-based_Pathfinding_in_Modern_Computer_Games

[20] D. Rachmawati, L. Gustin. “Analysis of Dijkstra’s Algorithm and A Algorithm in Shortest Path Problem.” *Journal of Physics: Conference Series* 1566.1 2020: 12061-. https://iopscience.iop.org/article/10.1088/1742-6596/1566/1/012061/pdf

[21] D. Silver. Cooperative Pathfinding. University of Alberta 2020 https://www.davidsilver.uk/wp-content/uploads/2020/03/coop-path-AIWisdom.pdf

[22] H. Peng. ”Multi-Agent Path Finding”. GitHub. 2021. https://github.com/GavinPHR/Multi-Agent-Path-Finding (Viitattu 11/2023)

[23] Mine layout. Vol. 184, The Mining magazine (London). 2001. 150–152 p.

[24] R.R. Tatiya. *Surface and underground excavations: methods, techniques and equipment*. CRC Press, 2005.

[25] H. Hamrin, W. Hustrulid, R. Bullock. "Underground mining methods and applications." *Underground mining methods: Engineering fundamentals and international case studies* 2001: 3-14.