RICOCHET ROBOTS PELIN ANALYSOINTI JA RATKAISUALGORITMI

Lappeenrannan-Lahden teknillinen yliopisto LUT

Laskennallisen tekniikan koulutusohjelma

2022

Ville-Petteri Manninen

Ohjaaja: TkT Jouni Sampo

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Laskennallinen tekniikka

Ville-Petteri Manninen

Ricochet Robots pelin analysointi ja ratkaisualgoritmi

Kandidaatintyö
2022
35 sivua, 14 kuvaa, 8 taulukkoa, 2 liitettä
Ohjaaja: TkT Jouni Sampo

Avainsanat: Algoritmit, Ongelmanratkaisu, ASP, Tilastollinen analyysi, Lautapelit

Ricochet Robots on vuonna 1999 Alex Randolphin kehittämä lautapeli, jossa tavoitteena on neljää pelinappulaa hyödyntämällä siirtää niistä tietty maaliruutuun. Ratkaisua vaikeuttaa pelinappuloiden liikkeen pysähtyminen vasta laudalla olevaan seinään tai toiseen nappulaan, jolloin niiden on toimittava yhdessä toisilleen seininä ratkaisun saavuttamiseksi. Ongelma on osoitettu olevan NP-vaikea.

Työssä rakennettiin leveyshakuun perustuva algoritmi, joka ratkaisee sille syötetyn lautapelin tilanteen optimaalisesti. Algoritmin toimintaa tehostettiin rajaamalla tarkasteltavia tilanteita muun muassa poistamalla turhia siirtoja ja implementoimalla heuristiikkaa. Samalla algoritmin tekemiä operaatioita optimoitiin koodin ja toteutuksen puolella.

Lopullinen algoritmi sisältää lähes kaikki testatut menetelmät, joista suurin osa nopeutti ratkaisun löytymistä sadoilla prosenteilla. Lisäyksien kannattavuudesta varmistuttiin parittaisella t-testillä. Algoritmi pystyy ratkaisemaan tapaukset rajattuun 15:een siirtoon asti, kattaen 99.94% lautapelin tilanteista alle kolmessa minuutissa. Regressioanalyysin avulla luotiin ennuste, jonka mukaan tapaukset 24:ään siirtoon asti ratkeavat päivässä. Kerättyjen 100 000:n ratkaisun perusteella saatiin optimaalisen ratkaisun pituuden keskiarvon 99.9% luottamusväliksi $[6.28, 6.33]$. Lautapelin tilanteista 34.07% on myös ratkaistavissa siirtämällä vain yhtä pelinappulaa, jotka painottuvat pienille siirtojen määrille.

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Computational Engineering

Ville-Petteri Manninen

Analysis of and solver for the game Ricochet Robots

Bachelor's Thesis
2022
35 pages, 14 figures, 8 tables, 2 appendices
Supervisor: D.Sc. (Tech.) Jouni Sampo

Keywords: Algorithms, Problem solving, ASP, Statistical analysis, board games

Ricochet Robots is a game designed by Alex Randolph in 1999, in which the goal is to move a specific one of the four game pieces to the goal cell. Finding a solution is made more difficult by limiting the movement of the pieces, since they stop only when colliding with a wall on the board or another piece. This way the pieces must be used as walls for one another to find a solution. Solving the problem has been proved to be NP-hard.

The objective of this thesis was to build a breadth-first search algorithm which could solve the case of the game provided to it optimally. The algorithm was made more efficient by limiting the moves and positions it must consider and by implementing heuristics. Also the operations used by the algorithm were optimized both in the code and implementation.

The final algorithm included almost all the tested methods, most of which allowed the algorithm to find solutions hundreds of percent's faster. Paired t-test was used to determine whether a method was worthwhile to implements or not. The algorithm could solve the cases to the limit of 15 moves covering 99.94% of the board games cases in three minutes. Using regression analysis, the algorithm could solve cases up to 24 moves in a day. Using the collected 100 000 solutions the 99.9% confidence interval for the optimal solutions length was calculated to be $[6.28, 6.33]$. From the board games cases, 34.07% can also be solved using only one game piece, which are weighted towards lower number of moves.

Symboli- ja lyhenneluettelo

L	Yleinen ratkaisun pituus (siirtojen määrä)
N	Kappalemäärä löydetyille ratkaisuille
X_i	Algoritmin versiolla i ratkaisun löytymiseen kulunut aika

Sisällys

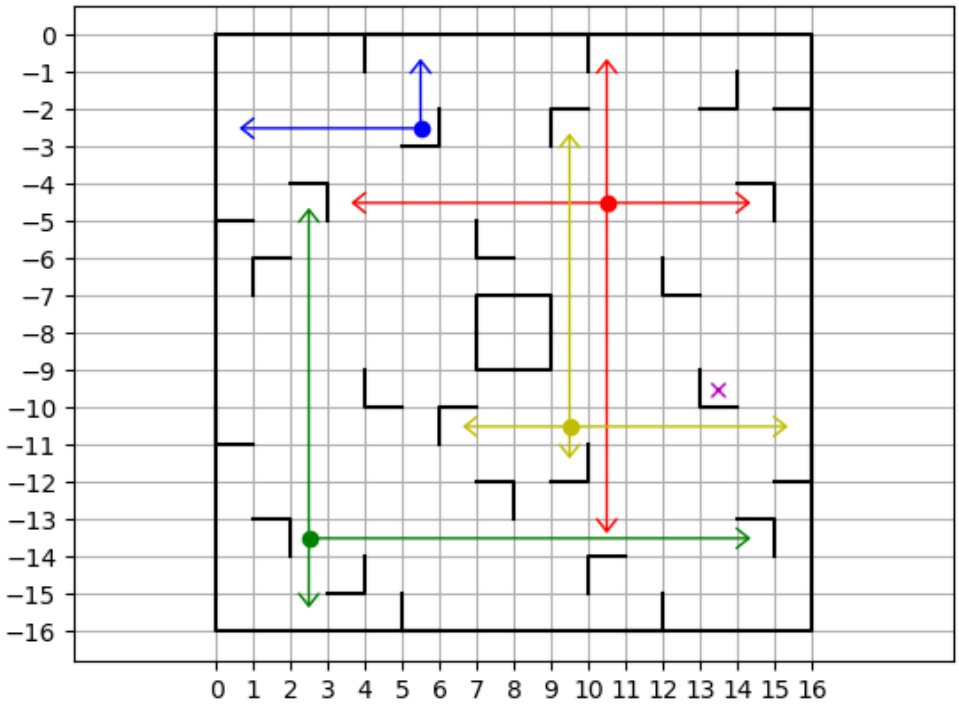
1 JOHDANTO

1.1 Tausta

Ricochet Robots on vuonna 1999 Alex Randolphin kehittämä kilpailulliseen ongelmanratkaisuun perustuva lautapeli (Alkuperäinen saksankielinen nimi: 'Rasende Roboter'). Peli koostuu pelilaudasta, joka on muodoltaan 16 x 16 ruudukko, jossa sijaitsee eriväriseä kuvioituja maaleja ja L:n muotoisia seiniä. Laudalle asetetaan yleensä 4 tai 5 eriväristä pelinappulaa (robottia), jotka toimivat yleisinä pelivälineinä jokaiselle pelaajalle. Pelin eteneminen koostuu kierroksista, joissa tavoitteena on 'ratkaista' eli keksiä kombinaatio erilaisille siirroille, joilla tietyn värinen robotti saadaan siirrettyä maaliruutuun. Kyseiseen pelinappulaan viitataan työssä nimellä 'aktiivinen robotti'. Esimerkkitilanne ja -ratkaisu on esitettynä kuvassa 1. Tilannetta hankaloittaa se, että robottia siirtäessä sen on liikuttava, kunnes se törmää toiseen robottiin tai seinään. Näillä säännöillä jokainen peliin osallistuja yrittää ensin keksiä ratkaisun itsenäisesti tekemättä siirtoja laudalla, kun pelaaja luulee löytäneensä ratkaisun ilmoittaa hän muille sen vaatiman siirtojen määrän ja aloittaa yhden minuutin ajastimen. Tuon minuutin kuluttua lyhyimmän ratkaisun löytänyt pelaaja aloittaa esittämällä ratkaisunsa laudalla. Ratkaisun ollessa oikein, saa hän pisteen kierroksen voittamisesta. Muuten vuoro siirtyy seuraavaksi lyhyimmän ratkaisun löytäneelle. Kierroksia pelataan kunnes yksi pelaaja saavuttaa tavoitepistemäärän, jolloin hän on voittanut kyseisen pelin [1].

Kuva 1. Yhdeksän siirron esimerkkiratkaisu esitettynä graafisesti. Etsinnässä oli ratkaisu vihreällä robotilla, joka löydettiin hyödyntämällä punaista robottia seinänä ja lopuksi siirtämällä keltainen robotti pois edestä. Ratkaisun löytymiseen algoritmin viimeisellä versiolla kului aikaa viisi sekuntia.

Pelikierroksella vastaan tulevan yksittäistapauksen ratkaisu on osoitettu NP-vaikeaksi ongelmaksi [2]. Kokonaisuudessaan jokaisesta tilanteesta on mahdollista tehdä 16 erilaista siirtoa, mikäli yhtään rajoittavaa tekijää ei huomioida. Tällöin tarkasteltavien tilanteiden määrä olisi jo 10:n siirron jälkeen suuruusluokaltaan 10^8 - 10^{10}. Todellisuudessa mahdollisia siirtoja on yleensä 6 - 10 kappaletta, kun lähtötilanteesta on edetty joitain siirtoja eteenpäin. Tämä johtuu siitä, että robottien sijainnit ovat aluksi satunnaisia ja jokaisen siirron jälkeen siitä eteenpäin on vähintään samansuuntainen siirto estetty. Toisaalta lähtötilanteessa siirtojen määrä on myös yleensä lähempänä 12:ta kuin tuota pahinta 16:ta. Mahdollisista siirroista on esimerkki kuvassa 2. Pelin ollessa näin kompleksinen, on se silti ratkaistavissa algoritmillisesti hyödyntäen heuristiikkaa ja siirtojen rajaamista loogisesti. Ratkaisun kompleksisuuden lisäksi lautapelissä on myös mahdollista muodostaa erilaisia lähtötilanteita noin 5.4 biljoonaa, joka lisää aivan uuden ulottuvuuden kokonaisuudessaan pelin ratkaisemiselle.



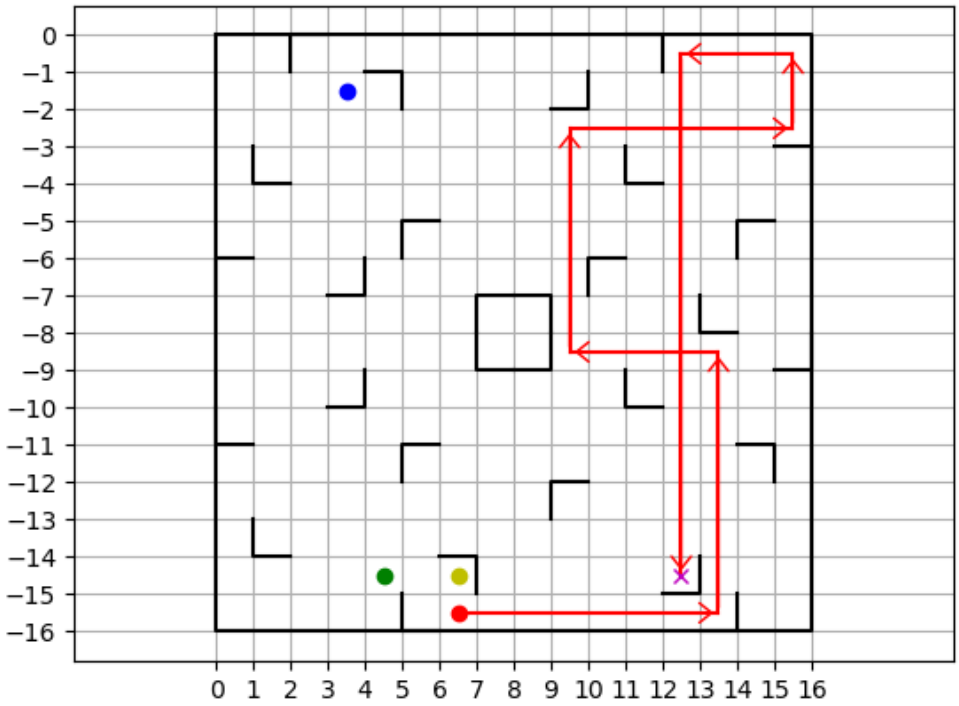
Kuva 2. Lähtötilanne yksittäistapauksen ratkaisuun, jossa mahdollisia siirtoja on 13 kappaletta.

Ongelman yleisempänä kontekstina on usean toimijan yhteisen tavoitteen saavuttaminen. Tässä tapauksessa neljän samanlaisen robotin on yhdessä saatava yksi siirrettyä tiettyyn ruutuun hyödyntäen ympäristöä ja toisiaan seininä kiertäen yksilölliset rajoitteet liikkumisen suhteen, jolloin parempi ratkaisu voidaan löytää. Ongelmaa on tämän ja kompleksisuutensa ansiosta hyödynnetty tutkiessa ihmisläheistä ongelmanratkaisua [3] ja 'Answer Set Programming' esimerkkejä [4].

1.2 Tavoitteet ja rajaus

Työn tavoitteena on rakentaa algoritmi, joka pystyy ratkaisemaan sille syötetyn yksittäistapauksen. Annettuna informaationa toimii lähtötilanne, johon kuuluu seinien, maalin ja robottien sijainnit. Algoritmin tulisi löytää nopeasti optimaalinen ratkaisu tilanteeseen, koska pelin tavoitteena on keksiä mahdollisimman lyhyt ratkaisu ennen muita pelaajia. Toteutettavan algoritmin tulisi toimia mahdollisimman yleisesti, jolloin sille voidaan syöttää mikä tahansa sääntöjen ja rajauksien sisällä oleva tapaus.

Yksinkertaisen algoritmin toteuttamisen lisäksi tarkoituksena on kehittää sitä nopeammaksi ja testata implementoitujen menetelmien kannattavuutta algoritmin toiminnan nopeuden suhteen. Lisäksi tarkoitus on tutkia peliä tilastollisesti muodostamalla jakauma satunnaisen ratkaisun pituudelle, verrata onko hyödynnetyillä laudan paloilla merkitystä jakaumaan ja tutkimalla kuinka suuri osa tilanteista on ratkaistavissa optimaalisesti siirtämällä vain yhtä robottia, josta esimerkki on esitettynä kuvassa 3.



Kuva 3. Kahdeksan siirtoa pitkä ratkaisu, jossa on käytetty vain punaista robottia.

Tarkasteltavat tilanteet on rajattu pelin perinteisimpään muotoon eli lautana toimii 16 x 16 ruudukko ja robottien määräksi on valittu neljä. Samalla mahdolliset laudan konfiguraatiot on rajattu lautapelin sisältämiin ja sääntöihin perustuviin vaihtoehtoihin, joissa suurin osa tilanteista on mahdollista ratkaista alle 15 siirrolla täysin satunnaisesta robottien asetelmasta. Tämän perusteella siirtojen ylärajaksi on asetettu 15 siirtoa, koska sitä pidempien ratkaisujen esiintyminen on oletettu epätodennäköiseksi ja niiden ratkaisemiseen kuluva aika voi kasvaa suureksi. Myös erityistapaus, jossa maaliin saatava robotti on jo yhden siirron päässä tai jo valmiiksi siellä jätetään tarkastelematta. Tämä perustuu säännöissä mainittuun tapaukseen, jonka mukaan robotin on tehtävä vähintään yksi 90°

käännös ennen maaliin saapumista [1]. Rajauksien yli menevien tapausten tutkimisessa hyödynnetään jakauman sovitusta kerättyyn dataan tilanteiden esiintyvyyden arviointiin ja regressioanalyysiä ratkaisuun kuluvan ajan ennustamiseen.

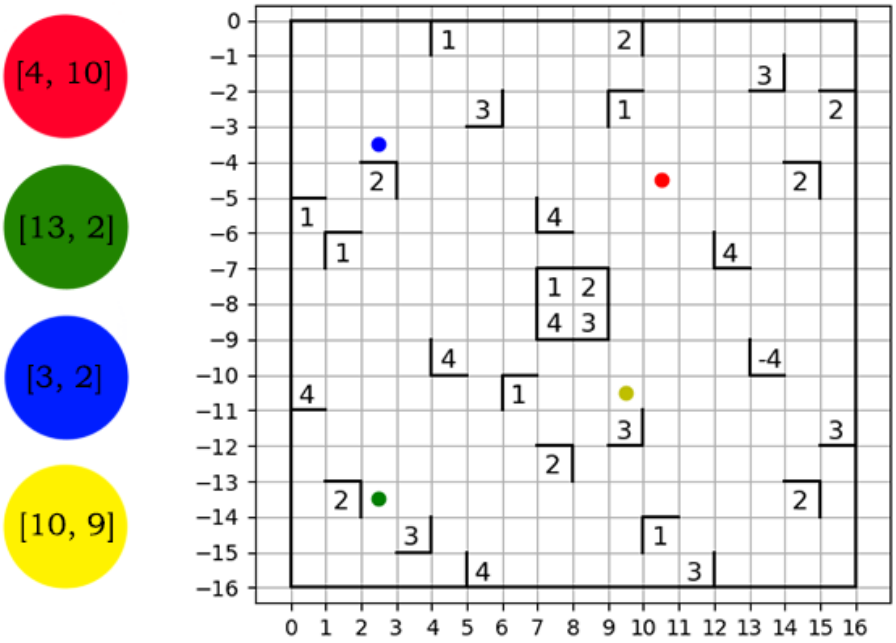
1.3 Työn rakenne

Työn käsittely on jaettu kuuteen kappaleeseen, joista ensimmäisessä esitellään pelin mallintamisen, algoritmin toteutuksen ja luotujen heurististen menetelmien teoria. Toisessa kappaleessa esitellään lyhyesti hyödynnetyt ohjelmistot, kirjastot ja perusta kerätylle datalle. Kolmas kappale koostuu itse peliin liittyvän datan analysoinnista, jossa esitetään muun muassa jakauma optimaalisen ratkaisun pituudelle ja yhden robotin ratkaisuiden todennäköisyydet. Neljäs kappale sisältää algoritmin versioiden vertailua ja luodun algoritmin toimivuuden analysointia kyseisen ongelman ratkaisemiseen ja erilaisien lisäysten vaikutusta sen tehokkuuteen. Kahdessa viimeisessä kappaleessa kootaan tehty työ ja esitetään johtopäätökset tuloksien pohjalta algoritmin lopulliselle versiolle ja pohditaan aiheeseen liittyen mahdollista jatkotutkimusta.

2 MALLIN TAUSTA JA TEORIA

2.1 Pelin mallinnus

Kyseisen ongelman ollessa visuaalinen (lautapeli), on se muunnettava muotoon, jossa sitä voidaan käsitellä tietokoneella. Mallinnus perustuu pohjimmiltaan matriiseihin ja erilaisen informaation ilmaisemiseen lukuarvoilla. Lauta on syötteessä 16 x 16 matriisi, jossa jokainen alkio kuvastaa sen ruudun sisältämää informaatiota. Mikäli ruudun ympärillä on L:n muotoinen seinä, asetetaan sen ruudun arvoksi luku joukosta $\{1, 2, 3, 4\}$, kuvaten seinän kulman sijaintia. Tarkasteltavan tapauksen maalin ollessa myös kyseisessä ruudussa, vaihdetaan sen arvo negatiiviseksi. Muiden ruutujen ja laudan reunojen arvot jätetään nolliksi. Tässä muodossa lauta on helppo muodostaa käsin tai satunnaisesti generoiden ja syöttää algoritmille ratkaistavaksi. Sen pohjalta pystytään luomaan matriisit, joiden käsittely on tehokkaampaa algoritmin ajon aikana ja piirtää graafinen esitys pelilaudalle. Robottien sijainnit tallennetaan toiseen matriisiin, jossa jokainen rivi vastaa robotin indeksejä laudalla. Esimerkki syötteistä on kuvassa 4.



Kuva 4. Esimerkkilauta syötematriisin informaatiolla ja robottien sijainneilla. Tyhjä ruutu vastaa arvoa 0 ja rivien järjestys on vaihdettu negatiiviseksi graafisen esityksen kohdalla, jotta matriisin käsittely ja kuva olisivat samoin päin.

Syötematriisi muutetaan 16 x 16 x 4 matriisiksi, jossa jokainen alkio on binäärisessä muodossa ja kuvaa seinän olemassaoloa $(0/1)$ kyseiseen suuntaan, joka ilmoitetaan syvyyden indeksinä. Esimerkiksi tilanteessa, jossa ruudun arvo on syötteessä 3, tulee tässä matriisissa olemaan syvyyksillä 2 ja 3 (indeksit 1 ja 2) kyseisessä ruudussa arvot 1. Muodostetulla matriisilla voidaan yksittäisellä alkioon viittauksella tarkistaa onko robotin liikkeen edessä seinä. Toisten robottien kanssa tapahtuvat törmäykset tarkistetaan niiden sijaintien avulla eli onko liikutettavan robotin seuraavalla ruudulla jo valmiiksi toinen.

2.2 Algoritmin idea

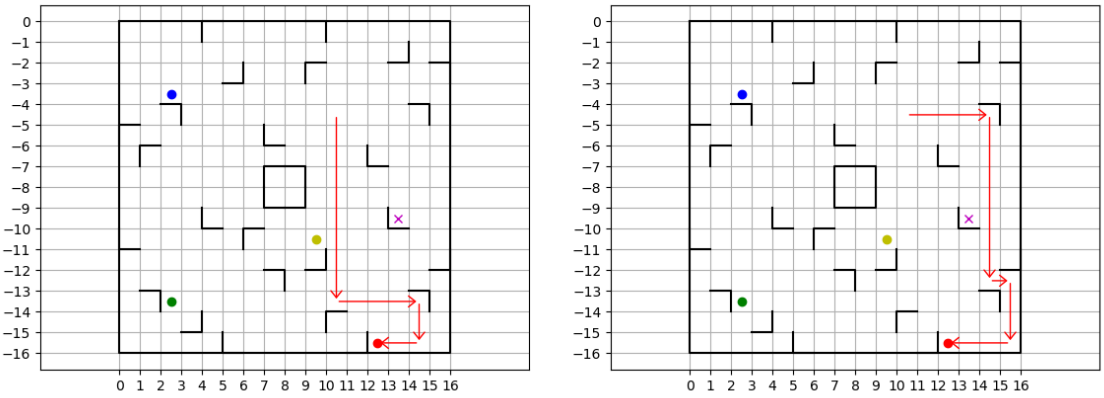
Pohjimmiltaan algoritmi toteutetaan leveyshakupuuna, jossa syvyytenä toimii tehtyjen siirtojen määrä ja vastaavasti haarautuvana tekijänä mahdolliset siirrot kustakin tilanteesta. Alustavasti algoritmi tarkastelee tilanteita järjestyksessä yksi kerrallaan ottaen jonosta seuraavan edellisen ollessa tutkittu kokonaan. Jokaisesta yksittäisestä tilanteesta muodostetaan jokaista siirtoa kohden solmulle uusi lapsi, joka laitetaan seuraavan syvyyden jonoon tarkastelua varten. Tällaista silmukkaa toistetaan kunnes aktiivinen robotti saavuttaa jossain haarassa maaliruudun, jonka jälkeen ohjelma kulkee maalin saavuttaneesta solmusta omaa haaraansa takaisin puun juureen keräämällä robottien sijainnit ja tehdyt siirrot ratkaisun esittämistä varten.

Kuten kompleksisuutta esitellessä kävi ilmi, että tarkasteltavien tilanteiden määrä kasvaa nopeasti todella suureksi. Voidaan olettaa pelkän syvyyshaun hyödyntämisen olevan hidas tapa löytää ratkaisu kyseiseen ongelmaan. Mikäli mahdollisia siirtoja ja vastaan tulevia tilanteita ei rajata ollenkaan, algoritmilla kestäisi useita vuosia löytää ratkaisu vaikeampiin tapauksiin.

2.3 Tarkasteltavien tilanteiden rajaus

Ensimmäinen askel algoritmin kehittämisessä otetaan lisäämällä siirtojen tekemiseen liittyvää logiikkaa, joiden tarkoitus on rajata tarkasteltavia tilanteita. Alustava algoritmi on hyvin rajaamaton tarkastelun suhteen, koska esimerkiksi sama tilanne voi tulla vastaan useaan kertaan. Tuon pohjalta rajaaminen aloitetaan hylkäämällä toiseen kertaan vastaan tulevat sijainnit, kuten kuvan 5 esimerkissä on esitetty. Rajauksen toteuttamiseksi muodostetaan kahdeksan ulotteinen matriisi, joka tallentaa kaikki tarkastellut tilanteet ykkösinä ja säilyttää muut tilanteet nollina. Tämän avulla matriisialkioon voidaan viitata

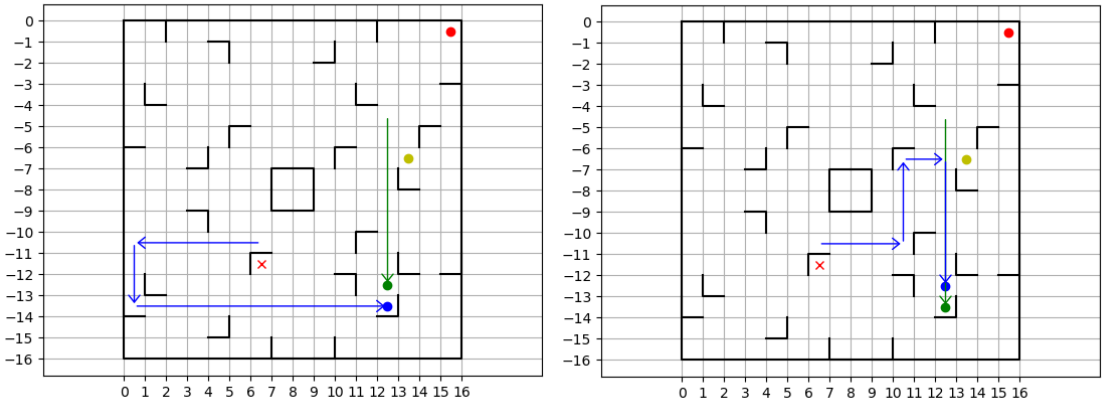
käyttämällä robottien sijainteja indeksinä, jolloin voidaan nopeasti tarkistaa onko kyseinen kombinaatio tarkasteltu jo aiemmin. Mikäli matriisista löytyvä alkio on saanut arvon 1, voidaan kyseinen tapaus pudottaa kokonaan pois pienentäen runsaasti tarkastelua vaativien tilanteiden määrää jatkossa.



Kuva 5. Esimerkki kahdesta siirtosarjasta, jotka päätyvät identtisiin sijainteihin eri määrällä siirtoja. Vasemmassa kuvassa punainen robotti saadaan sijaintiin $[15, 12]$ neljällä siirrolla ja oikean puoleisessa viidellä, jolloin oikean puoleisesta tilanteesta ei jatkettaisi siirtojen tekemistä eteenpäin.

Toinen pelin logiikkaan liittyvä tekijä, joka otetaan huomioon löytyy myös pelin säännöistä. Sen mukaan seuraavaa siirtoa tehdessä ei edellisellä siirrolla siirrettyä robottia voi siirtää takaisin [1]. Tämän avulla pystytään suoraan tiputtamaan jokaisesta tilanteesta yksi mahdollinen siirto kokonaan pois. Ennen mainittua muutosta algoritmi tarkisti siirtoa tehdessä vain, oliko siirrettävä robotti liikkunut yhtään ruutua.

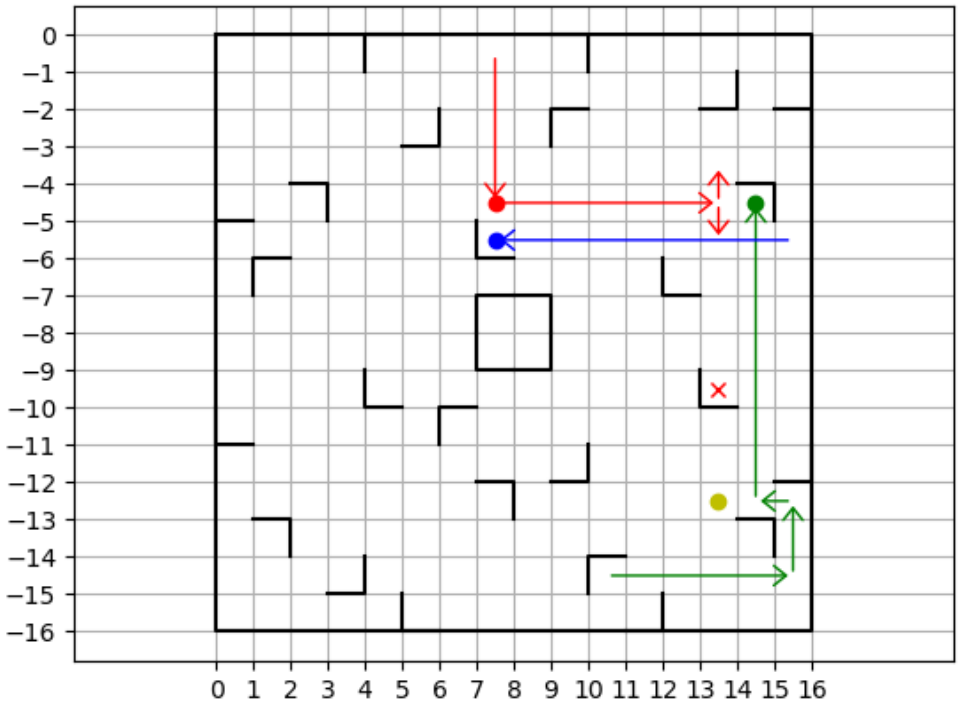
Kahden ylemmän menetelmän lisäksi myöhemmässä vaiheessa algoritmin kehitystä kaksinkertaisten sijaintien käsittely tehdään vielä tarkemmin. Koska neljän robotin joukossa vain aktiivisen robotin värillä on merkitys ratkaisun kannalta, voidaan läpikäytyjen sijaintien joukkoon lisätä myös kaikki viisi muuta kombinaatiota. Niissä muiden robottien sijainnit ovat vastaavat, mutta väriltään ne ovat asettuneet eri tavalla kuten kuvassa 6. Kyseinen muutos on kokeellisempi, koska sen vaatima laskenta hidastaa algoritmia hiukan aiempaa enemmän, jolloin mainitun rajauksen vaikutusta verrataan vain varmasti nopeimpaan algoritmiin. Tämän menetelmän rajaamien tapausten määrä on aiempaan verrattuna huomattavasti pienempi, koska kyseiset tilanteet vaativat suuremman määrän siirtoja tapahtuakseen. Käytännössä kahden ei-aktiivisen robotin paikkaa on vaihdettava.



Kuva 6. Esimerkki kahdesta siirtosarjasta, jotka päätyvät vastaaviin sijainteihin eri määrällä siirtoja, kun ei-aktiivisia robotteja käsitellään yhdenvertaisina. Vasemmassa kuvassa sininen ja vihreä robotti saadaan neljällä siirrolla sijainteihin, jotka mahdollistavat punaisen robotin pääsyn maaliin ja oikean puoleisessa puolestaan viidellä siirrolla, jolloin oikean puoleisesta tilanteesta ei jatkettaisi siirtojen tekemistä eteenpäin.

2.4 Heuristiikka

Algoritmille yksi helppo tapa pienentää suorittaessa läpi käytävien tapausten määrää, on tunnistaa mahdollisuus löytää ratkaisu siirtämällä robottia tiettyyn suuntaan heti edellisen siirron jälkeen. Tämän toteuttamiseksi muodostetaan matriisi, johon merkitään ruudut, joista on mahdollista päästä yhdellä siirrolla suoraan maaliin. Implementoimalla alkion tarkistus algoritmiin pystyy se keskeyttämään syvyyden läpikäynnin, kun sopiva tilanne tulee vastaan kuten kuvassa 7. Lopullinen ratkaisu löytyisi viimeistään tällä hetkellä tarkasteltavasta tilanteesta seuraavalla syvyydellä. Menetelmällä saatu ratkaisu pysyy edelleen optimaalisena, koska samalla syvyydellä jäljellä olevien tilanteiden ei ole mahdollista ratketa pienemmällä määrällä siirtoja, sillä nämä olisivat löytyneet jo edellistä syvyyttä tarkasteltaessa.



Kuva 7. Esitetystä tilanteessa siirrettäessä punaista robottia oikealle saavuttaa se ruudun, josta ratkaisu on yhden siirron päässä. Seuraavaksi tutkittaisiin punaisella robotilla siirrot ylös- ja alaspäin ennen seuraavaan tilanteen ottamista tarkasteluun.

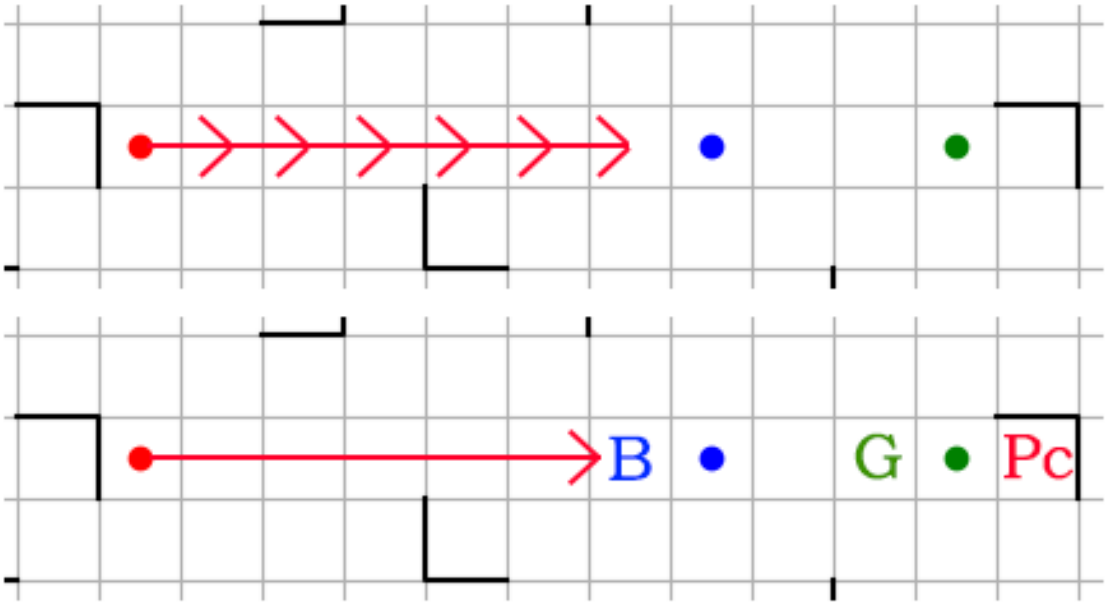
Toinen heuristinen menetelmä, jonka tavoitteena on nopeuttaa algoritmin toimintaa hyödyntämällä tarkastelun muokkausta, on robottien siirtojen järjestyksen vaihtaminen. Algoritmin aiemmat versiot käsittelivät aina siirrot järjestyksessä: punainen, vihreä, sininen ja keltainen, jolloin useita siirtoja keltaiselta robotilta vaativat ratkaisut löytyvät tarkastelusta myöhemmin. Tämän pohjalta tehdään algoritmiin muokkaus, joka siirtää aluksi aktiivisen robotin tarkasteluun ensimmäiseksi. Muutoksen ansiosta tutkittavien tilanteiden määrän tulisi pienetä varsinkin aiemman järjestyksen myöhempien värien kohdalla ja erityisesti ratkaisun koostuessa pääosin vain aktiivisen robotin siirroista.

Kokeellisempana versiona aiemmasta siirtojärjestyksen muokkauksesta tehtiin uusi versio, johon lisättiin dynaamisuus, jossa aktiivisen robotin jälkeen siirretään viimeksi siirrettyä robottia ennen kahta muuta. Esimerkiksi tilanteessa, jossa edellinen siirto oli sinisellä robotilla ylöspäin, olisi robottien käsittelyjärjestys seuraava: keltainen, sininen, punainen ja vihreä, etsiessä keltaiselle robotille ratkaisua. Kuten aiempi ei-aktiivisien robottien yhdenvertainen käsittely, tämäkin muokkaus vaatii enemmän aikaa algoritmilta suorittaa ja

sen vaikutus on pienempi, jolloin sen kannattavuutta verrataan vastaavasti.

2.5 Siirtojen määrittämisen tehostaminen

Keskeisimpänä operaationa algoritmin toiminnassa on robottien siirtäminen. Alunperin algoritmin kaikki versiot tarkistivat voiko siirrettävä robotti edetä seuraavalle ruudulle haluttuun suuntaan. Siirron ollessa mahdollinen siirretään se seuraavaan ruutuun ja tarkastelu tehdään uudestaan kunnes liike on estetty. Toimintamalli on tehoton, koska ohjelma joutuu toistamaan samaa tarkistusta useaan kertaan. Muutos siirtojen tekemiseen toteutetaan muodostetamalla 16 x 16 x 4 matriisi, joka sisältää uuden rivin tai sarakkeen, johon robotti pysähtyy lähtiessään liikkeelle ruudusta suuntaan, joka ilmoitettu syvyyden indeksinä. Esimerkiksi [15, 15, 0] alkio antaa uuden rivin robotille, jota siirretään ylöspäin pelilaudan oikeasta alakulmasta. Matriisin sisältämät arvot määritetään ennen ensimmäisen siirron tarkastelua. Aiemman lisäksi muiden robottien kanssa tapahtuvien törmäyksien määrittäminen on tehtävä uudestaan, jotta hyöty saataisiin siirrettyä niihinkin tilanteisiin mahdollisimman tehokkaasti. Tämä tarkastelu toteutetaan tutkimalla onko siirtosuunnassa ainakin yksi toinen robotti, jolloin niiden aiheuttamaa pysähtymisruutua verrataan ennalta määrättyyn, joista liikutettavan robotin uudeksi sijainniksi asetetaan rajoittavampi. Esimerkki siirtojen tarkastelusta on kuvassa 8.



Kuva 8. Esitetyssä tilanteessa punaisen robotin siirtäminen oikealle olisi alunperin edennyt yksi ruutu kerrallaan, kunnes liike ei ollut enää mahdollista. Esilaskennan avulla alkioon viittaus antaisi ruudun ’Pc’ kyseiselle siirrolle, mutta sinisen ja vihreän robotin sijaitseminen samalla rivillä lisää vertailuun niiden asettamat uudet sijainnit ’B’ ja ’G’. Tällöin punainen robotti siirretään ruutuun ’B’, koska sitä vastaava sarake tulee suunnassa ensimmäisenä vastaan.

3 AINEISTO JA OHJELMISTOT

Algoritmi toteutetaan Python-ohjelmointikielellä (versio 3.7.2) ja koodin kirjoittamisessa ja ajamisessa käytetään Pythonin IDLE-editoria. Itse algoritmi toimii pääosin hyödyntämällä NumPy-kirjastoa, sillä suurin osa algoritmin käyttämistä tietorakenteista on NumPy-matriiseja. Graafiset esitykset tehdään MatPlotLib-kirjastolla. Algoritmille syötetään informaatio pelitilanteesta kappaleen 2.1 mukaisesti ja ratkaisun löytyessä se palauttaa ratkaisun pituuden, siihen kuluneen ajan, robottien siirrot ja sijainnit. Algoritmin versiot tallennetaan erillisinä kokonaisuuksina menetelmien tuoman nopeuden kasvun tutkimista varten.

Työn aineistona hyödynnetään Z-Man Gamesin vuonna 2013 julkaiseman version lautoja, joista käyttöön valittiin puolet A-C, joissa on vain normaaleja seiniä esteinä. Lautojen D-puolella on näiden lisäksi myös "peilejä", jotka vaikuttavat liikkeeseen eritavalla riippuen robotin väristä, jonka takia ne rajataan tarkastelun ulkopuolelle. Laudan palaset ovat kooltaan 8 x 8 ruudukoita, joista yhdistämällä neljä saadaan rakennettua kokonainen pelilauta. Tämä yhdistäminen on toteutettava käyttämällä yksi kappale jokaista tarjolla olevaa palan väriä (punainen, sininen, keltainen ja vihreä), jolloin saadaan muodostettua sallittu kombinaatio laudasta. Vastaavan kombinaation erilaiset permutaatiot tiputettiin pois lukitsemalla punainen pala aina laudan vasempaan yläkulmaan. Hyödynnetyt laudan palat ovat esitettynä liitteessä 1.

Poikkeuksena normaaliin lautapeliin verrattuna maalin generointi suoritettiin aina yksivärisenä ja satunnaisella sijainnilla L:n muotoisen seinän sisällä, joka ei ole laudan keskellä tai reunassa. Menettely ei tuota aina suoraan lautapelissä esiintyvää maalia, sillä fyysisillä laudoilla maalien väri ja sijainti kombinaatio on ennalta määrätty. Tästä johtuen mahdollisia maaleja normaaliin peliin verrattuna tulee olemaan noin nelinkertainen määrä. Vastaavasti robottien sijainti kunkin tapauksen alussa on satunnainen, joka vastaa ensimmäisen pelikierroksen tilannetta, koska myöhemmin robotit alottaisivat edellisen kierroksen loppupisteistä. Näistä johtuen ratkaisuiden jakaumassa voi olla pientä poikkeavuutta aidon pelitilanteen tarkasteluun verrattuna. Ohjelmoidun satunnaisgeneroinnin avulla pystytään nopeasti luomaan useita erilaisia tilanteita, joissa luotuja algoritmeja voidaan testata ja ratkaisusta saatua dataa kerätä suurissa määrin tarkempaa analyysiä varten.

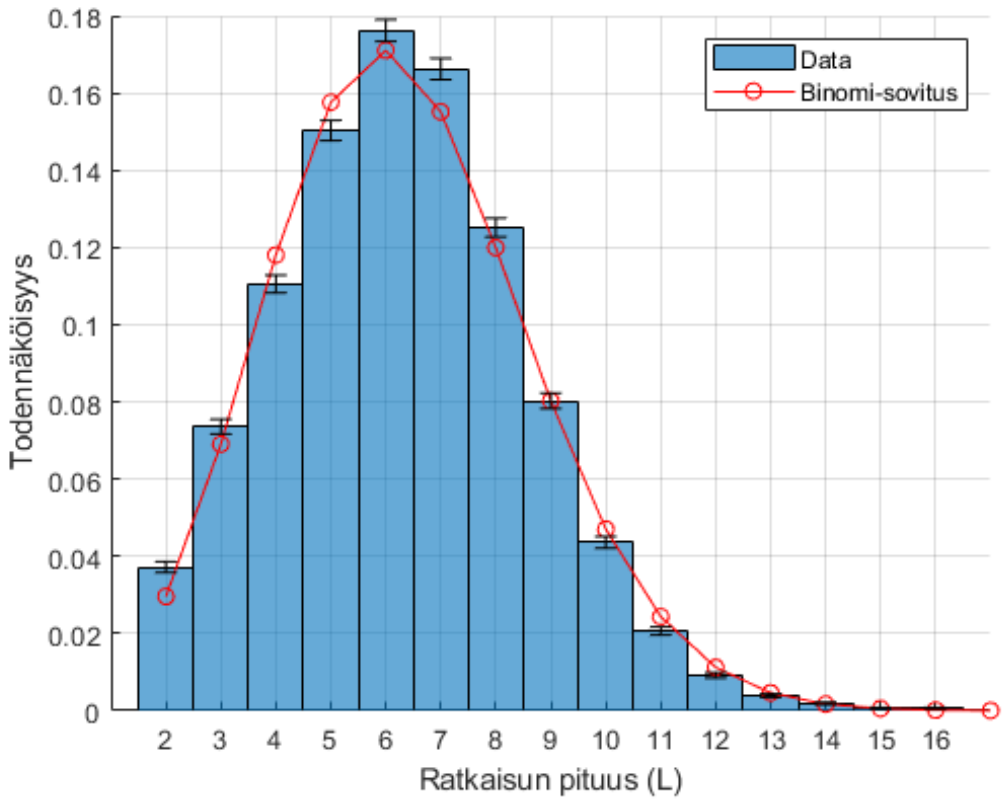
Analyysin datana käytetään algoritmien keräämää informaatiota, jota hankitaan ratkomalla satunnaisgeneroinnin tuottamia tapauksia useilla algoritmin versioilla. Riippuen verrattavista algoritmeista ja halutusta tarkkuudesta ratkaisevat ne joko saman tai eri yksittäistapauksen. Siirtojen ylärajaa muutetaan algoritmin versiosta riippuen, jotta ajat pysyivät

järkevinä. Tällä tavalla kerättyä dataa hyödynnetään tilastollisen analyysin tekemiseen, joka toteutetaan MATLAB-ohjelmistolla (Versio: R2019b).

4 PELIN TILASTOLLINEN ANALYYSI

4.1 Pelin tapaukset

Aluksi tehdään satunnaiseen pelitilanteeseen liittyvä analyysi eli tutkitaan optimaalisen ratkaisun siirtojen määrän jakaumaa. Kappaleessa esitetty data on kerätty algoritmin versioilla, jotka pystyvät ratkaisemaan tapaukset 15:een siirtoon asti minuuteissa. Hyödynnetty data on koottu kaikista vertailutapauksista huomioimalla vain löydetyn ratkaisun pituus. Yhteensä tapauksia kertyi 100 000 kappaletta, joiden pohjalta luotiin jakauma niiden pituudesta kuvaan 9 ja data itsessään on esitettynä liitteessä 2.



Kuva 9. Optimaalisen ratkaisun pituuden jakauma, sovitettu binomijakauma ja todennäköisyyksien 99% luottamusvälit.

Kerätyn datan perusteella rajaamalla tarkastelu 15:een siirtoon on saatu ratkaistua 99.94% satunnaisista tilanteista. Liitteessä 2 esitettyjen tulosten perusteella voidaan myös sanoa, että noin 99% tapauksista on ratkaistavissa alle 12 siirrolla ja vastaavasti noin 95% on alle

10 siirrolla. Ratkaisun pituuden odotusarvon 99.9% luottamusväliksi saadaan $[6.28, 6.33]$ siirtoa. Kokonaisuudessaan rajatun 15 siirron yli päätyi 63 tapausta.

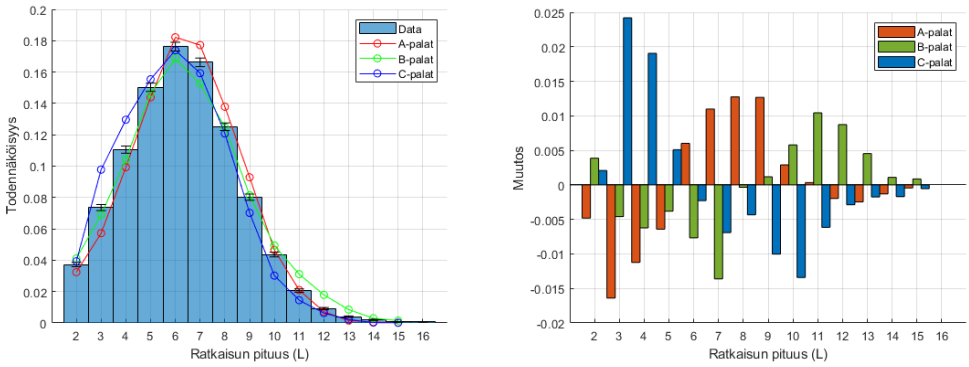
Graafisen esityksen perusteella data vaikuttaisi olevan binomijakautunut, joten dataan sovitettiin binomijakauma, jonka avulla arvioidaan pidempien ratkaisuiden esiintyvyyttä. Sovitteeksi saatiin $L \sim Bin(45, 0.14)$ ja ratkaisun pituuden vastaavaksi luottamusväliksi $[6.34, 6.38]$. Jakaumalla estimoidut 16-20 pituisien ratkaisuiden todennäköisyydet ovat esitettynä taulukossa 1. Datan ja jakauman yhdenvertaisuutta testataan kahden näytteen Kolmogorov-Smirnovin testillä [5]. Toiseksi populaatioksi otetaan 100 000 näytettä sovitetusta jakaumasta, josta poistetaan kaikki 0 ja 1 arvot. Testin tuloksena nollahypoteesi hylätään P-arvolla 0.00002, jolloin näytteet eivät ole samasta jakaumasta. Tämä osoittaa ettei data noudata sovitettua binomijakaumaa ja sen avulla luotuihin arvioihin kannattaa suhtautua varauksella.

Taulukko 1. Sovitetulla jakaumalla estimoidut todennäköisyydet ja määrät 63 tapauksesta 16-20 pituisille ratkaisuille.

L	16	17	18	19	20
P (%)	0.018	0.0049	0.0012	0.00029	0.000061
N	46.4	12.6	3.1	0.7	0.2

4.2 Laudan palojen vertailu

Käytettyjen laudan palojen vaikutusta ratkaisuiden pituuden jakaumaan tutkitaan generoimalla 10 000 tapausta, joissa lauta kasataan satunnaisessa järjestyksestä vain tietyillä laudan puolilla A-C liitteen 1 mukaisesti. Näistä kombinaatioista kerättyjen ratkaisuiden pituuksien jakaumaa verrataan täysin satunnaiseen tilanteeseen, jonka tulokset on kuvassa 10. Jakaumien yhdenvertaisuutta tutkitaan kahden näytteen Kolmogorov-Smirnovin testillä [5] ottamalla alkuperäisestä datasta satunnaisesti 10 000 näytettä vertailuun, joista nollahypoteesin (H_0) hylkäämisen totuusarvot ja P-arvot ovat taulukossa 2.



(a) (b)

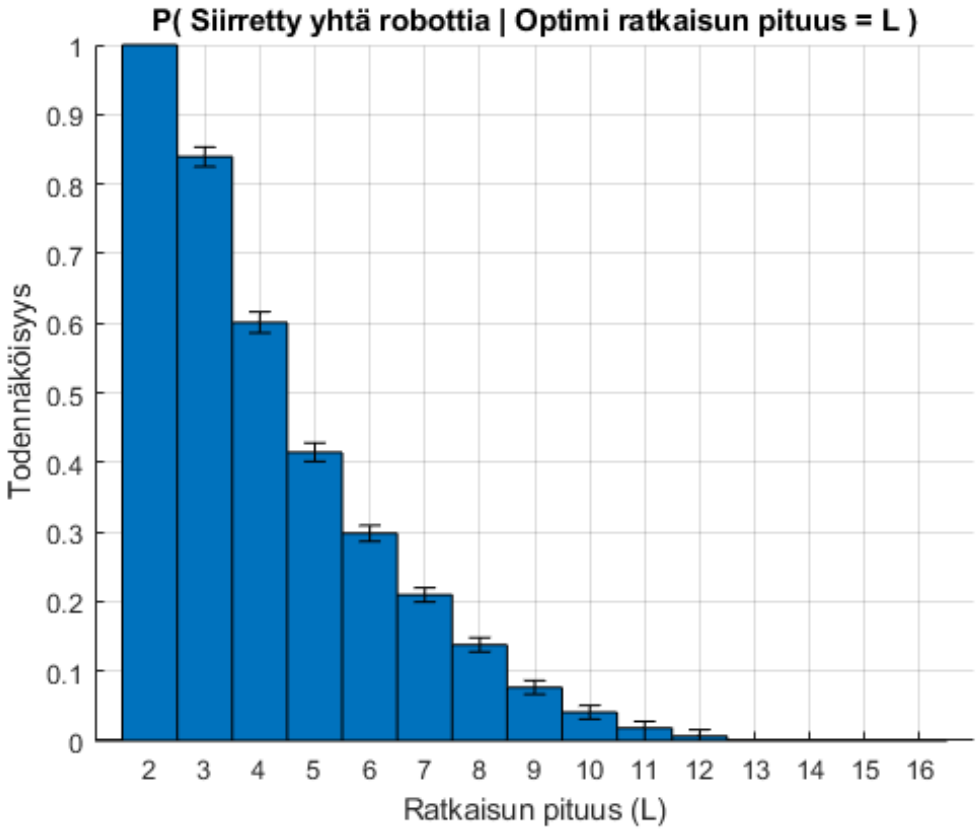
Kuva 10. Saadut jakaumat (a) ja muutokset (b) täysin satunnaiseen tilanteeseen verrattuna, kun lauta on muodostettu vain tietyistä palojen puolista.

Taulukko 2. Ratkaisun pituuden odotusarvon 99.9% luottamusvälit ja Kolmogorov-Smirnovin testien tulokset käytettäviä paloja rajatessa. H_0-rivillä arvo 1 vastaa nollhypoteesin hylkäämistä ja vastaavasti 0 hyväksymistä. Mikäli nollahypoteesi hyväksytään, näytteet ovat testin perusteella samasta jakaumasta.

Palat	A	B	C
E(L)	$[6.38, 6.49]$	$[6.42, 6.54]$	$[5.95, 6.05]$
H_0	1	1	1
P	$1.7 \cdot 10^{-6}$	$3.6 \cdot 10^{-4}$	$3.3 \cdot 10^{-12}$

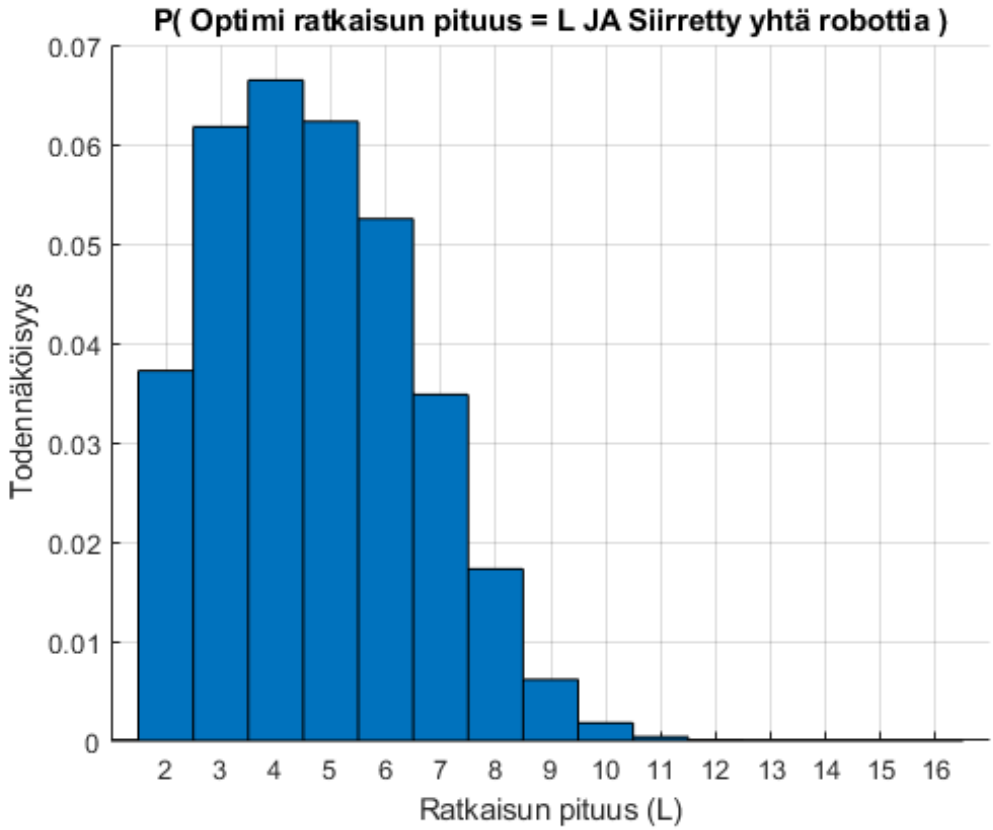
4.3 Yhden robotin ratkaisut

Seuraavaksi tutkitaan kuinka usein on mahdollista löytää optimaalinen ratkaisu siirtämällä vain yhtä robottia kuten kuvassa 3 on esitetty. Datan keräämiseksi algoritmista tehdään versio, joka löytää kaikki erilaiset optimaaliset ratkaisut, joiden joukosta tunnistetaan ratkaisut yhdellä robotilla. Tapauksesta kerätään tieto optimaalisen ratkaisun pituudesta ja löytyikö ainakin yksi optimaalinen ratkaisu vain yhtä robottia siirtämällä. Kyseinen algoritmi toimii ongelman rajauksen mukaisesti eli ratkaisuja etsitään 15:een siirtoon asti. Yhteensä ratkaisuja kerätään 50 000 kappaletta, joista data on esitettynä liitteessä 2 ja kuvassa 11.



Kuva 11. Ehdollinen todennäköisyys yhdellä robotilla löydetyn ratkaisun optimaalisuudelle. Todennäköisyyksille laskettiin 99% luottamusvälit, jotka ovat esitettynä kuvassa mustilla viivoilla.

Hyödyntämällä yleisen ratkaisun dataa pystymme ehdollisen todennäköisyyden kaavalla laskemaan todennäköisyydet tilanteelle, jossa satunnaisen vastaantulevan tapauksen ratkaiseminen onnistuu yhtä robottia siirtämällä. Todennäköisyydet ovat esitettynä histogrammina kuvassa 12 ja tarkemmat arvot liitteessä 2. Summaamalla saadut arvot eri pituisille tapauksille saadaan kokonaisuudessaan todennäköisyydeksi 34.07% löytää ratkaisu yhdellä robotilla. Suurin osa näistä tapauksista on kuitenkin painottunut pienille siirtojen määrille.



Kuva 12. Todennäköisyys täysin satunnaiselle tilanteelle, jossa optimaalinen ratkaisu on pituudeltaan L ja se on löydettävissä siirtämällä vain yhtä robottia.

5 ALGORITMIN ANALYSOINTI

Seuraavissa kappaleissa algoritmin kehytyksen vaiheita ja muutoksien kannattavuutta analysoidaan tilastollisesti. Varmemmin algoritmin toimintaa kehittävien muutoksien analyysissä hyödynnetäään vain prosentuaalista parannusta edelliseen tai verrattavaan versioon, joka lasketaan kaavalla: $SM(X_i, X_j) = (\frac{E(X_i)}{E(X_j)} - 1) \cdot 100\%$, jossa X_i vastaa vanhan algoritmin ratkaisuaikaa ja X_j vastaavasti uuden. Kaavan pohjalta saaduilla arvoilla voidaan tarkastella yleisellä tasolla kuinka monta prosenttia nopeampi uusi versio on edelliseen verrattuna. Kyseiset vertailuarvot lasketaan erikseen jokaiselle syvyydelle.

Vaikeammin havaittavien tai algoritmin toimintaa enemmän hidastavien muutoksien kannattavuutta puolestaan tutkitaan kahden otoksen yksisuuntaisella parittaisella t-testillä. Menettely on valittu siltä pohjalta, että muutoksen implementoinnin kannattavuus ei ole välttämättä suoraan selvä kerätystä datasta ja siitä halutaan varmistua. Hyödynnetyn testin tulokset ilmoitetaan taulukoiden H_0-rivillä totuusarvolla hylkäämiselle eli arvo 0 vastaa nollahypoteesin hyväksymistä ja 1 puolestaan hylkäystä. Testin P-arvo on ilmoitettu vastaavan H_0-rivin alla.

Käytetyt hypoteesit:

$$\begin{aligned} H_0 : &\quad E(X_i - X_j) \leq 0 \\ H_1 : &\quad E(X_i - X_j) > 0 \end{aligned}$$

Hypoteeseissa esiintyvät arvot vastaavat siis vanhan (X_i) ja uuden (X_j) algoritmin ratkaisun löytämiseen kuluneita aikoja samalle yksittäistapaukselle. Testin merkitsevyystasoksi on valittu $\alpha = 0.05$, jolloin lasketun testisuureen t ja sitä vastaavan P-arvon avulla voidaan varmistua H_0:n hylättäessä siitä, että uusi algoritmi on tilastollisesti nopeampi aiempaan versioon verrattuna. Näille vertailuille lasketaan myös edellä mainittu prosentuaalinen muutos vastaavalla tavalla, kuten se myös hypoteesitesti suoritetaan erikseen jokaiselle syvyydelle.

5.1 Ensimmäiset versiot

Ensimmäisen kolmen version vertaaminen tehdään yksinkertaisesti tarkastelemalla eri pituisten ratkaisuiden löytymiseen kuluneiden aikojen keskiarvoja. Muutoksien ollessa lähtökohtaisesti tehokkaita ja niiden vaikutuksen huomasi jo yksittäisiä tapauksia ratkoessa,

niin tarkempaa tilastollista analyysiä ei näiden välillä tehdä. Samalla yksittäiset ratkaisut ovat algoritmeillä erilaisesta tilanteesta, jolloin poikkeavuutta todelliseen vaikutukseen voi olla, mutta pääpiirteiltään muutoksen suuruus ja algoritmien nopeudellinen suhde pysyy samanlaisena. Taulukoissa 3-5 esitetyssä datassa on 500 ratkaisun keskiarvot ja algoritmin nopeuden kasvu aiempaan ja ensimmäiseen versioon verrattuna. Nämä ratkaisut on rajattu algoritmille sopivilla pituuksilla, koska niiden löytämisen ajat kasvavat korkeiksi suuremmilla siirtojen määrillä. Taulukossa puuttuva data kuvastaakin tuota rajausta. Kaikki käsitellyt algoritmit toimivat siirtojen esilaskennan avulla.

Vertauksessa tässä kappaleessa ovat seuraavat algoritmin versiot, joissa uusi versio sisältää myös aiemmat lisäykset:

1: Syvyyshaku ilman heuristiikkaa (Kappale 2.2)

2: Siirtojen rajaaminen (Kappale 2.3)

3: Tarkistus seuraavalla siirrolla löytyvälle ratkaisulle (Kappale 2.4)

Taulukko 3. Ratkaisun löytämiseen kuluvien aikojen keskiarvot pelkkää syvyyshakua hyödyntäen. Algoritmin versio 1.

L	2	3	4	5	6
$E(X_1)$ (s)	0.0015	0.017	0.21	1.75	19.14

Algoritmin versiolla 1 etsittiin ratkaisuja 6 siirtoon asti. Datan pohjalta siirtojen määrän kasvaessa, algorimin tarvitsema aika noin kymmenkertaistuu jokaista siirtoa kohden, joten ratkaisun löytymiseen syvyydeltä 7 kuluisi jo yli 3 minuuttia. Kappaleen 4.1 perusteella voidaan sanoa, että tämä versio algoritmista pystyy ratkaisemaan 55% lautapelin tapauksista alle minuutissa.

Taulukko 4. Tilanteiden rajaus pelin logiikalla, versio 2

L	2	3	4	5	6	7	8	9	10	11
$E(X_2)$ (s)	0.0017	0.014	0.082	0.27	0.73	1.96	4.21	11.16	18.37	36.02
$SM(X_1, X_2)$	-10.12	24.67	153.85	545.88	2504.10	-	-	-	-	-

Algoritmin versioon 2 implementoidut menetelmät saivat aikaan keskimäärin 644% no-

peammin toimivan algoritmin. Muutosten jälkeen algoritmilla pystytään löytämään 10 siirtoa pitkä ratkaisu samassa ajassa, kuin kuuden siirron ratkaisut algoritmin ensimmäisellä versiolla. Kappaleen 4.1 pohjalta voidaan vastaavasti sanoa, että kyseinen algoritmi pystyy ratkaisemaan 98% lautapelin tilanteista alle minuutissa.

Taulukko 5. Tarkistus ratkaisun löytymiselle seuraavalla siirrolla, versio 3

L	2	3	4	5	6	7	8	9	10	11	12
$E(X_3)$ (s)	0.0001	0.0014	0.0096	0.050	0.18	0.50	1.45	3.31	7.44	12.58	41.08
$SM(X_1, X_3)$	1129.73	1123.80	2070.36	3426.85	10350.15	-	-	-	-	-	-
$SM(X_2, X_3)$	1268.17	881.67	754.99	446.05	301.30	290.04	191.06	237.48	147.03	186.21	-

Algoritmin versiolla 3 löydetään ratkaisuja keskimääräisesti 470% nopeammin versioon 2 verrattuna. Suurimmat parannukset näiden versioiden välillä saavutetaan lyhyillä siirtojen määrillä, kun puolestaan pidempien ratkaisuiden kanssa parannus on noin 200%. Algoritmin ensimmäiseen versioon verrattuna nopeuden kasvun osalta puhutaan 3620%:sta. Tällä algoritmin versiolla löydetään jo 12 siirron mittaisia ratkaisuja minuutissa.

5.2 Värien järjestyksen muuttaminen

Muutoksien ollessa tästä eteenpäin ajallisesti pienempiä, niiden kannattavuutta tutkitaan hypoteesitestillä. Tässä kappaleessa vertailuun otetaan kappaleessa 2.4 mainittu värien tarkastelujärjestyksen muuttaminen. Algoritmin kaksi versiota ratkaisevat saman satunnaistilanteen, joka arvotaan 25000 kertaa ja niistä kerätään sama data kuin muistakin tilanteista. Tämän datan avulla suoritetaan hypoteesitesti, jonka tulokset on esitetty taulukossa 6.

Verrataan edellisen kappaleen algoritmia 3 uuteen versioon:

4: Vastaava algoritmi, jossa värien tarkastelujärjestys muutettu (Kappale 2.4)

Taulukko 6. Värien järjestyksen muuntamisen vaikutus algoritmin toimintaan, hypoteesitestin tulokset.

L	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$H_{0,(3,4)}$	1	1	1	1	1	1	1	1	1	1	1	1	1	0
$P_{(3,4)}$	0	0	0	0	0	0	0	0	0	0	0	0.0002	0.021	0.26

Datan perusteella pystymme sanomaan, että värien tarkastelujärjestyksen muuntaminen on algroritmin toimintaa merkittävästi nopeuttava menetelmä. Testien P- ja t-arvojen perusteella nollahypoteesi hylättiin valitulla merkitsevyystasolla syvyyttä 15 lukuunottamatta, joka voi johtua pienestä tapauksien määrästä (15 kappaletta). Tämän perusteella on siis oikeutettua ottaa järjestyksen muuntaminen uudeksi algoritmin pohjatasoksi, kun seuraavia muokkauksia tehdään ja testataan.

5.3 Siirtojen esilaskenta

Kappaleessa 2.5 mainitun siirtojen tarkastelun muuttamisen tehokkuutta tutkitaan algoritmin neljän ensimmäisen version avulla, joista kolme ensimmäistä on käsitelty edellisissä kappaleissa ja viimeinen tulee mukaan vertailuun seuraavassa. Nämä versiot ratkaisivat 500 erilaista tilannetta sekä esilaskennan että alkuperäisen siirtojen määrityksen avulla. Esilaskennasta saatua hyötyä tutkitaan laskemalla vastaavat suhteelliset muutokset, hyödyntäen näiden kahden ratkaisun löytämiseen kuluneita aikoja.

Taulukko 7. Siirtojen tarkastelun muuttamisesta saatu hyöty.

L	2	3	4	5	6	7	8	9	10	11	12	13	μ
$SM(X_{1,0}, X_{1,1})$	568.52	592.29	612.94	594.57	578.09	-	-	-	-	-	-	-	589.28
$SM(X_{2,0}, X_{2,1})$	393.09	442.87	474.03	494.54	485.51	485.17	483.24	481.34	480.15	481.37	-	-	470.13
$SM(X_{3,0}, X_{3,1})$	436.39	543.45	621.89	666.88	674.23	654.79	659.95	661.95	645.82	642.03	644.89	606.29	621.55
$SM(X_{4,0}, X_{4,1})$	327.78	515.03	614.41	632.82	662.90	677.41	660.55	653.86	639.09	650.62	654.44	638.59	610.63

Algoritmien välisenä erona on hyvä mainita kyseisen version painotteisuus siirtojen määrittämiselle, koska yhden syvyyden päässä olevan ratkaisun etsiminen vaatii useamman siirron tekemistä tietyissä tilanteissa. Kuitenkin siirtojen tekemisen muuttaminen yksittäisen ruudun tarkastelusta esilaskentaan ja vertailuun on tehnyt algoritmista noin 570% nopeamman eli prosessointinopeus on lähes kuusinkertainen alkuperäiseen verrattuna.

5.4 Viimeiset heuristiset menetelmät

Kappaleiden 2.3 ja 2.4 loppupuolella mainitut menetelmät ovat rajauksiltaan heikompia ja vaadittavalta käsittelyltä ajon aikana vaativampia. Tästä syystä näiden kahden lisäyksen implementoinnin kannatavuudesta yhdessä ja erikseen varmistutaan hypoteesitestillä, jossa vertauskohtana toimii kappaleessa 5.2 nopeammaksi todettu algoritmi. Jokainen algoritmin versio ratkaisee saman tilanteen, joka arvotaan 25000 kertaa. Testien tulokset ovat esitettynä taulukossa 8 ja ratkaisuaikojen vertailu kuvassa 13. Aikojen keskiarvot ja tapausmäärät löytyvät liitteestä 2.

Uudet lisäykset, joita verrataan algoritmin versioon 4 ovat:

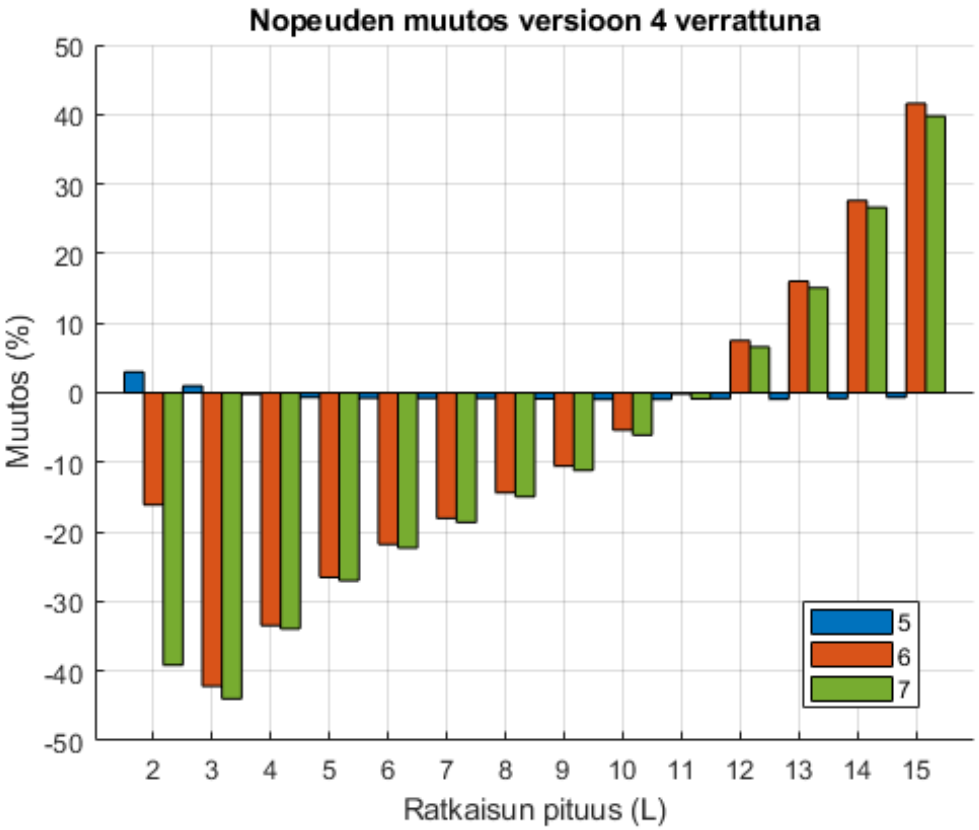
5: Dynaaminen toisen robotin tarkastelu (Kappale 2.3)

6: Ei-aktiivisien robottien yhdenvertaisuus (Kappale 2.4)

7: Molemmat yllä mainitut menetelmät

Taulukko 8. Neljän viimeisen algoritmin version hypoteesitestin tulokset.

L	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$H_{0,(4,5)}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$P_{(4,5)}$	0.034	0.20	0.94	1	1	1	1	1	1	1	1	1	1	0.99
$H_{0,(4,6)}$	0	0	0	0	0	0	0	0	0	1	1	1	1	1
$P_{(4,6)}$	1	1	1	1	1	1	1	1	1	0.016	0.000	0.000	0.000	0.0001
$H_{0,(4,7)}$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
$P_{(4,7)}$	1	1	1	1	1	1	1	1	1	0.24	0.000	0.000	0.000	0.0001



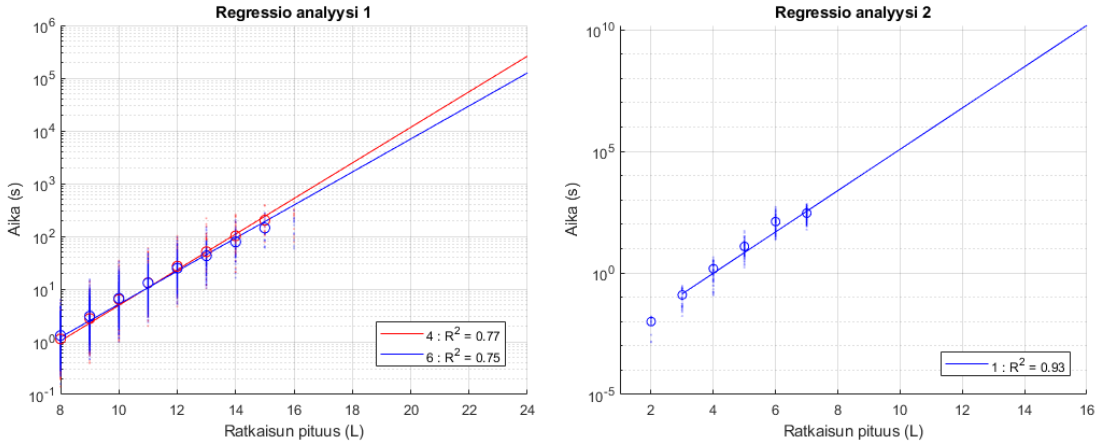
Kuva 13. Esitettynä kuinka paljon nopeammin erimittaiset ratkaisut löytyvät kappaleessa käsiteltyjen algoritmien avulla lähtökohtaan 4 verrattuna. Lukuarvot on laskettu kaavalla $SM(X_4, X_j)$.

Kuten taulukosta 8 huomataan, ei-aktiivisien robottien käsittely yhdenvertaisina on nopeampi yli 10 siirron ratkaisuissa testien perusteella. Tämä vaikutus ratkaisun löytymiseen nähdään myös kuvasta 13, jossa syvyyden 11 jälkeen algoritmit 6 ja 7 ovat kymmeniä prosentteja nopeampia, mutta kaikissa tapauksissa ennen syvyyttä yksitoista 10- 45% hitaampia. Dynaamisen toisen robotin tarkastelun testeissä puolestaan H_0 hylättiin vain syvyydellä 2.

5.5 Regressioanalyysi

Olemassa on vielä yli 16 siirron ratkaisuja, jotka kattavat kappaleen 4.1 perusteella 0.06% kaikista vastaan tulevista tilanteista. Nuo olivat kuitenkin rajattu keskeisimmän tutkimusalueen ulkopuolelle, mutta regressioanalyysin avulla voimme arvioida näiden ratkaisuiden löytymiseen kuluvaa aikaa. Ennusteen laskeminen toteutetaan sovittamalla PNS-mentelmällä lineaarinen malli datan viimeisen kahdeksan pituuden ratkaisuaiko-

jen kymmenkantaiseen logaritmiin. Tällöin kuluvan ajan kymmenkantainen logaritmi $\log_{10}(t) = \beta_1 + \beta_2 \cdot x$, jossa x on siirtojen määrä ja sovitettavat parametrit ovat β_1 ja β_2. Mallin avulla saamme käsityksen algoritmin toiminnasta rajauksia pidemmillä ratkaisuilla. Kuvassa 14 esitetyissä graafeissa mallin arvot on muunnettu takaisin exponentiaaliseen muotoon, jolloin matemaattinen muoto on $t = 10^{(\beta_1+\beta_2 \cdot x)}$. Esitetty malli sovitettiin versiolle 1 ilman kappaleessa 2.5 mainittua esilaskentaa algoritmin ihan ensimmäisen version ratkaisuaikojen tutkimista varten. Vastaavasti malli sovitettiin versioille 4 ja 6 esilaskennan kanssa lopullisen tilanteen tutkimista varten.



Kuva 14. Regressioanalyysin kuvaajat versiolle 1 ilman esilaskentaa ja versioille 4 ja 6 esilaskennan kanssa.

Kuten kuvasta 14 voimme huomata, että 24 siirron ratkaisuiden löytyminen kestäisi algoritmilla 6 suurusluokaltaan 10^5 sekuntia, joka vastaa noin 28 tuntia eli kaikki ratkaisut tuohon siirtomäärään asti löytyisi päivän aikana. Puolestaan algoritmin ensimmäisellä versiolla kestäisi 15 siirron ratkaisuiden löytäminen suuruusluokaltaan 10^9 sekuntia, joka vastaa noin 32 vuotta. Sisällyttämällä esilaskenta ensimmäiseen versioon ratkaisu löytyisi noin $\frac{1}{6}$ ajassa taulukossa 7 esitettyjen tulosten perusteella.

6 JOHTOPÄÄTÖKSET

Algoritmiin sisälletyistä menetelmistä kaikki dynaamista toisen robotin käsittelyä lukuunottamatta olivat testien perusteella kannattavia taulukoiden 4-6 ja 8 perusteella. Niillä saatiin rajattua tarkasteltavia tapauksia riittävästi, jotta algoritmi saatiin toimimaan riittävässä ajassa rajauksien osalta. Yleisesti eniten hyötyä tuottanut muutos oli siirtojen tarkastelun muuttaminen yhdestä ruudusta kerrallaan esilaskettuun muotoon, jolla algoritmin prosessointinopeus kasvoi noin kuusinkertaiseksi, kuten taulukkosta 7 voidaan päätellä. Muidenkin menetelmien hyödyt olivat korkeita kuten datan perusteella huomattiin. Mutta niiden vaikutuksesta toisiinsa nähden on vaikea lähteä sanomaan tarkemmin, koska tarkastelua tehtiin pitkälti kumulatiivisesti. Kuitenkin implementoitujen menetelmien tuottama nopeuden kasvu oli sadoista jopa tuhansiin prosentteihin. Kokonaisuudessaan algoritmi on kehittynyt merkittävästi, sillä kuvan 14 perusteella algoritmin ensimmäinen versio ei löydä vuosiin yli 40% lautapelin tilanteista, kun puolestaan viimeisellä versiolla 99.94% tapauksista ratkeaa parissa minuutissa.

Lopullisen parhaan algoritmin valitsemiseen jää vaihtoehdoiksi versiot 4 ja 6, jolloin ratkaiseva muutos niiden välillä on ei-aktiivisien värien yhdenvertaisuus, sillä toisen robotin käsittely dynaamisesti ei ole kannattavaa taulukon 8 perusteella. Molempien algoritmien puolesta voi löytää perusteluja, miksi juuri kyseinen on nopeampi ratkaisuiden löytämisessä. Mikäli ei-aktiivisien robottien yhdenvertainen käsittely jätetään pois, löydetään versiolla 4 noin 96%:iin tapauksista ratkaisu nopeammin taulukossa 8 ja liitteessä 2 esitettyjen tulosten perusteella. Tuon osuuden pohjalta versio 4 on yleisesti nopein. Kuitenkin ratkaisuiden löytymiseen kuluvat ajat ovat yleensä alle 10 sekuntia noissa tapauksissa niiden lyhyyden takia ja algoritmi 6 häviääkin versiolle 4 keskimäärin vain 0 - 0.2 sekuntia, jota on vaikea huomata ilman tarkkaa numeerista tarkastelua. Vastaavasti pidemmissä tapauksissa yhdenvertainen käsittely säästää jo 13 siirron kohdalla noin kymmenen sekuntia ja sitä pidemmissä ratkaisuissa vielä enemmän. Käytännöllisyyden kannalta voidaan sanoa version 6 olevan tehokkain tässä työssä toteutettu algoritmi.

Rajauksen ja samalla tavoitteen asettaminen 15:een siirtoon osoittautui hyväksi, sillä sen kattavuus yltää 99.94% lautapelissä esiintyvistä tilanteista, kuten kappaleessa 4.1 todettiin. Tämä tarkoittaa käytännössä sitä, että valmistettu algoritmi pystyy ratkaisemaan tuon osuuden tapauksista keskimäärin alle kolmessa minuutissa liitteessä 2 esitettyjen tulosten perusteella. Ratkaisun pituuden odotusarvon 99.9% luottamusvälin ollessa $[6.28, 6.33]$ siirtoa voidaan sanoa suurimman osan ratkaisuista painottuvan pienille siirtojen määrille. Rajauksen yli menevien tapausten prosentuaalinen osuus 0.06% vastaa käytännössä, että yksi tapaus 1666:sta ratkeaa optimaalisesti yli 15 siirrolla. Binomijakauman avulla

estimoituna näistä ylimenevistä tapauksista suurin osa jää tuohon 16 tai 17 siirtoon kuten taulukossa 1 esitetyistä arvoista nähdään. Samalla 99.96% tapauksista olisi jakaumalla estimoituna alle 20 siirtoa, jotka ratkeaisivat algoritmilla parissa tunnissa kappaleessa 5.5 esitettyjen ennusteiden perusteella. Kuitenkin binomijakaumalla tehtyihin ennusteisiin on suhtauduttava varauksella, koska kappaleessa 4.1 tehdyn Kolmogorov-Smirnovin testin tuloksen perusteella data ei noudata sovitettua jakaumaa.

Hyödynnetyillä laudan paloilla näyttäisi myös olevan kappaleen 4.2 perusteella merkitys ratkaisun pituuden jakaumaan. Vertailluista kombinaatioista kuvan 10 ja taulukon 2 perusteella laudan C-puolia hyödyntämällä ratkaisut ovat painottuneet lyhyempiin ratkaisuihin, jolloin palojen voidaan sanoa olevan helpompia. Puolet A ja B sen sijaan pidensivät ratkaisun odotusarvoa tuottamalla vaikeampia tilanteita, mutta A:n jakauma on painottunut keskipitkille ratkaisuille vähentäen lyhyitä ratkaisuja, kun taas B-puolet tuottavat enemmän pidempiä ratkaisuja vähentäen keskipitkien määrää. Hyödynnetyistä paloista riippumatta jakauman suhteelliset osuudet pysyvät samanlaisina eli kuusi siirtoa pitkät ratkaisut ovat yleisimpiä ja muuten arvot käyttäytyvät toisiinsa nähden vastaavalla tavalla.

Pelaamiseen liittyvänä strategiana vain yhden robotin siirtäminen vaikuttaa toimivalta, sillä yksi tapaus kolmesta on ratkaistavassa optimaalisesti yhden robotin avulla kappaleen 4.3 mukaisesti. Lähtökohtaisesti tilanteen tullessa vastaan on siirrettävä robotti jo tiedossa, jolloin optimaalinen ratkaisu voidaan löytää käytännössä heti tarkistamalla onko reitin löytäminen mahdollista. Toisinaan tuo ratkaisu voi olla erittäin kompleksinen, jolloin sen keksiminen ei välttämättä ole helppoa, kuten kuvassa 3 esitetyn tapauksen kuvittelisi olevan. Mikäli ratkaisu ei tuolla kerralla onnistu vain yhdellä robotilla, on mahdollista, että parilla muutoksella siirtojärjestykseen saadaan aktiivinen robotti liikutettua maaliin toisien robottien avulla.

Liitteessä 2 esitettyjen ratkaisuaikojen keskiarvojen perusteella algoritmi löytää optimaalisen ratkaisun tilanteeseen nopeammin kuin useat ihmispelaajat ehtivät löytää jonkin ratkaisun. Monessa tapauksessa, jossa ratkaisun pituus on väliltä $[4, 7]$, ihminen ei välttämättä kerkeä aloittaa mahdollisen ratkaisun kokeilua, kun algoritmi on jo löytänyt optimaalisen ratkaisun.

7 KESKUSTELU

Johtopäätöksien perusteella voidaan sanoa, että nopein algoritmin versio hoitaa tehtävänsä mallikkaasti, vaikka runsaasti nopeampi algoritmi on mahdollista toteuttaa. Sen osalta optimaalisempi toteutus toisi jo hiukan lisää tehoa prosessointiin, mutta rinnakkaislaskenta olisi varmasti yksi siirtojen tekemistä ja tilanteiden tarkistamista nopeuttava menetelmä. Algoritmin toimiessa pohjimmiltaan toistorakenteilla voitaisiin Pythonin salliessa hyödyntää esimerkiksi Joblib-kirjastoa, jolla näiden toistorakenteiden rinnakkaistaminen on mahdollista. Tällöin teoreettisesti prosessointinopeus voitaisiin saada heti monikertaistettua, mikäli käytössä oleva prosessori sen sallisi. Toinen vaihtoehto olisi aloittaa uudestaan toisella ohjelmointikielellä, joka voisi soveltua kehitetyille algoritmeille vielä Pythonia paremmin.

Algoritmia kehittäessä ja dataa kerätessä olisi voinut heuristisien menetelmien lisäykset tehdä myös erilaisessa järjestyksessä, jolloin yksittäisten menetelmien tuoma hyöty olisi ollut helpommin vertailtavissa, kuten viimeisen kahden lisäyksen kanssa oli toimittu. Ensimmäisten lisäysten kumulatiivisuus kumosi siis niiden osalta mahdollisuuden tarkempaan vertailuun.

Itse lautapelin osalta on muistettava, että maalien valinta ja sääntöjen pohjalta pois rajatut tapaukset vaikuttavat dataan. Verrattaessa pelin etenemiseen robottien sijainnit ovat aina arvottu satunnaisesti, joka vastaa vain pelin alkua. Normaalisti seuraavaa ratkaisua etsittäessä robotit aloittavat edellisen kierroksen lopetuspaikoillaan, jolloin ratkaisuiden jakauma voi muuttua merkittävästi. Lautapelin osalta ottamalla nuo tapaukset huomioon ja sisällyttämällä aidot kombinaatiot maaleille, ovat tapoja, joilla ongelman tutkimista voitaisiin viedä eteenpäin ja saada datasta yhä luotettavampaa.

LÄHTEET

[1] Alex Randolph. Ricochet robots rules eng. Along the game 'Ricochet Robots' in 2017, https://images.zmangames.com/filer_public/c0/b4/c0b482f1-ad3e-4e5d-ae48-0c11aa7c317a/en-ricochet_robot-rules.pdf, 2017. 3rd edition.

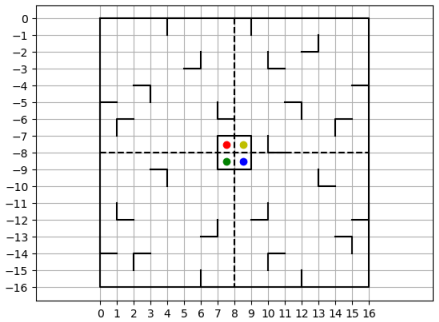
[2] Samuel Masseport, Benoit Darties, Rodolphe Giroudeau, and Jorick Lartigau. Ricochet robots game: complexity analysis technical report. HAL Archives, https://hal.archives-ouvertes.fr/hal-02191102, 7 2019. hal-02191102.

[3] Nicolas Butko, Katharina A. Lehmann, and Veronica Ramenzoni. Ricochet robots - a case study for human complex problem solving. *Proceedings of the Annual Santa Fe Institute Summer School on Complex Systems (CSSS'05)*, 2005.

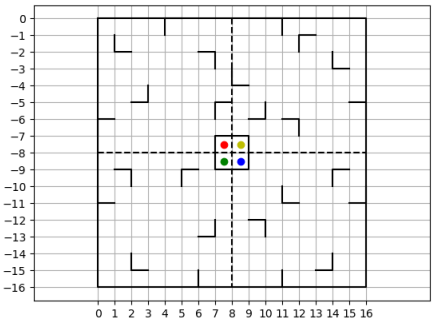
[4] Gebser Martin, Jost Holger, Kaminski Roland, Obermeier Philipp, Sabuncu Orkunt, Schaub Torsten, and Schneider Marius. Ricochet robots: A transverse asp benchmark. *Logic Programming and Nonmonotonic Reasoning*, pages 348–360, 2013.

[5] Frank J. Massey. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

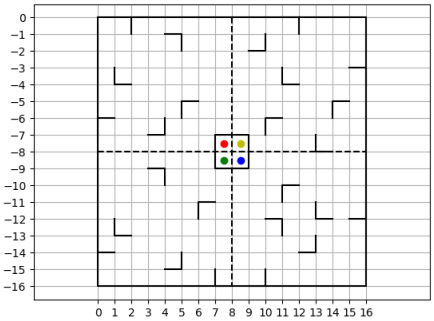
Liite 1. Pelilaudan palat



(a) 1. palat



(b) 2. palat



(c) 3. palat

Kuva A1.1. Hyödynnetyt lautojen palat esitettynä kokonaisina lautoina. Palat ovat erotettu toisistaan katkoviivoilla ja väri esitettynä laudan keskustassa. Erilaisia asetelmia paloista voidaan muodostaa vaihtamalla värien järjestystä ja kääntämällä palat sijainnin vaatimalla tavalla. Lautapelistä vastaavat palat löytää Z-Man Gamesin 2013 vuonna julkaisemasta versiosta, jossa laudan paloja vastaavat värit ovat 1-4 järjestyksessä: vihreä, sininen, punainen ja keltainen ja vastaavasti 1-3 puolet ovat pelissä A-C.

Liite 2. Pelin analysoinnin ja nopeimpien algoritmien data

Taulukko A2.1. Optimaalisen ratkaisun jakauman data. Taulukossa on esitettynä tapausmäärät ja todennäköisyydet eri pituisille tilanteille ja todennäköisyyksien kumulatiivinen summa.

L	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16+
N	3722	7360	11054	15031	17618	16621	12514	8023	4361	2066	917	396	190	64	63
P (%)	3.72	7.36	11.05	15.03	17.62	16.62	12.51	8.02	4.36	2.07	0.92	0.40	0.19	0.06	0.06
Σ P (%)	3.72	11.08	22.14	37.17	54.78	71.41	83.92	91.94	96.30	98.37	99.29	99.68	99.87	99.94	1.00

Taulukko A2.2. Yhdellä robotilla löydettyjen optimaalisien ratkaisujen data. N-rivillä on esitettynä tuktittujen tapausten määrä, joista yhdellä robotilla ratkesi optimaalisesti N_s-rivillä esitetty määrä. Todennäköisyydet ovat esitettynä ehdolliselle ja satunnaiselle tilanteelle ja kumulatiivinen summa on laskettu satunnaisen tilanteen avulla.

L	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16+
N	1828	3712	5578	7551	8794	8389	6207	3986	2160	998	448	187	87	33	42
N_s	1828	3115	3353	3130	2620	1758	856	305	88	18	3	0	0	0	0
Cond P (%)	100	83.92	60.11	41.45	29.79	20.96	13.79	7.65	4.07	1.80	0.67	0	0	0	0
P (%)	3.72	6.18	6.64	6.23	5.25	3.48	1.73	0.61	0.18	0.04	0.01	0	0	0	0
Σ P (%)	3.72	9.90	16.54	22.77	28.02	31.51	33.23	33.85	34.02	34.06	34.07	34.07	34.07	34.07	34.07

Taulukko A2.3. Ratkaisun löytymiseen kuluneiden aikojen keskiarvot kullakin pituudella, neljä nopeinta algoritmia (kappale 5.4). 12 ratkaisua oli pituudeltaan yli 15 siirtoa.

L	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N	959	1797	2745	3719	4395	4209	3127	2018	1093	527	238	97	48	16
$E(X_4)$ (s)	0.0001	0.0007	0.006	0.033	0.13	0.41	1.1	2.8	6.4	13.2	27.1	51.1	102.0	201.5
$E(X_5)$ (s)	0.0001	0.0007	0.006	0.033	0.13	0.42	1.1	2.8	6.4	13.3	27.3	51.6	102.9	202.8
$E(X_6)$ (s)	0.0001	0.0011	0.009	0.044	0.17	0.50	1.3	3.1	6.6	13.0	24.8	42.6	78.3	143.4
$E(X_7)$ (s)	0.0001	0.0012	0.009	0.045	0.17	0.51	1.3	3.1	6.7	13.1	25.0	43.0	79.0	145.0