

Eternity II -reunatäsmäysoin heuristiikoista

Tuomas Kujala

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Erkki Mäkinen
Toukokuu 2011

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi (Algoritmiikka)
Eternity II -reunatäsmäyksen heuristiikoista
Tuomas Kujala
Pro gradu -tutkielma, 78 sivua
Toukokuu 2011

Reunatäsmäyset ovat lautapelejä, jotka näennäisestä helppoudestaan huolimatta ovat NP-täydellisiä. Niiden ratkaisemiseen ei siis ole tiedossa tehokasta algoritmia. Reunatäsmäysopeleille voidaan kuitenkin kehittää erilaisia heuristiikkoja, jotka pyrkivät ratkaisemaan pelin mahdollisimman hyvin.

Tässä tutkielmassa esitellään neljä erilaista heuristiikkaa, joista kaksi perustuu evoluutioalgoritmeihin, yksi hill climbing -tekniikkaan ja yksi simuloituun jäähtymiseen. Kaikilla esitellyillä heuristiikoilla pyritään ratkaisemaan Eternity II -reunatäsmäysopelele mahdollisimman hyvin.

Tuloksista selvisi, että molemmat evoluutioalgoritmit tuottivat mainituista heuristiikoista tässä tutkielmassa parhaan yksittäisen tuloksen, 407/480 pistettä. Tulos on vielä melko kaukana Eternity II:n täydellisestä ratkaisusta, mutta algoritmeja edelleen kehittämällä paremmatkin pistemäärät lienevät mahdollisia.

Avainsanat ja -sanonnat: reunatäsmäyset, metaheuristiikat, geneettiset algoritmit, hill climbing, simuloitu jäähtymys, kombinatorinen optimointi

Alkusanat

Suuri työ on nyt vihdoin takana, kun Pro Gradu -tutkielmani on valmis ja maisteriksi valmistuminen hämmöttää edessä. Vielä vuosi sitten minulla ei ollut mitään tietoa siitä, mikä tutkielmani aihe voisi olla ja kuinka kauan työn kirjoittamisessa menisi.

Tähän tilanteeseen on kuitenkin nyt saavuttu, joten haluaisinkin kiittää ohjaajaani Erkki Mäkistä tutkielman ideoinnin avustamisesta, oikeaan suuntaan ohjaamisesta sekä arvokkaista kommentteista, joita sain aina todella nopeasti ja usein myös virka-aikojen ulkopuolella. Kiitokset myös tutkielmani toiselle tarkastajalle Timo Poraselle hyvistä kommentteista.

Suurin kiitos kuuluu kuitenkin vaimolleni Maaritille, joka väsymättömästi jaksoi kannustaa minua graduni tekoon aina silloinkin, kun oma motivaationi oli hieman hukassa. Omalla tahdillani tämä työ ei olisi varmastikaan valmistunut tähän päivämäärään mennessä. Kiitokset myös Maaritille hyvistä tarjoiluista, joiden avulla jaksoin puurtaa pitkän kirjoitusprosessin läpi.

Kangasalla, 24. toukokuuta 2011

Tuomas Kujala

Sisällysluettelo

1.	Johdanto	1
1.1.	Motiivi	2
1.2.	Reunatäsmäyspelit kirjallisuudessa.....	2
2.	Reunatäsmäyspelit.....	4
2.1.	Rakenne.....	5
2.2.	Reunatäsmäyspelien aikavaativuus ja approksimoitavuus	7
2.2.1.	Päätösongelmat ja aikavaativuusluokat.....	8
2.2.2.	Reunatäsmäyspelien approksimointi	10
3.	Etsintäalgoritmit.....	13
3.1.	Geneettiset algoritmit.....	13
3.1.1.	Rakenne ja toimintaperiaate.....	13
3.1.2.	Populaation muodostaminen.....	16
3.2.	Monitavoitteiset evolutiiviset algoritmit.....	21
3.3.	Hill climbing -algoritmit.....	23
3.4.	Simuloitu jäähdytys	24
4.	Algoritmien esittely	26
4.1.	Algoritmien yleinen rakenne	26
4.2.	Algoritmi 1 (geneettinen algoritmi)	28
4.3.	Algoritmi 2 (monitavoitteinen evolutiivinen algoritmi).....	31
4.4.	Algoritmi 3 (hill climbing -algoritmi).....	32
4.5.	Algoritmi 4 (simuloitu jäähdytys).....	33
5.	Tulokset	35
5.1.	Algoritmin 1 vertailua Muñozin ja muiden tuloksiin	35
5.2.	Algoritmin 1 tuloksia	38
5.3.	Algoritmin 2 vertailua Muñozin ja muiden tuloksiin	46
5.4.	Algoritmin 2 tuloksia	47
5.5.	Algoritmin 3 tuloksia	54
5.6.	Algoritmin 4 tuloksia	63
6.	Yhteenvedo	75
	Viiteluettelo.....	76

1. Johdanto

Reunatäsmäyspelit ovat perinteisten palapelien tyyllisiä lautapelejä, joissa pelilaudalle asetellaan yleensä jonkin säännöllisen monikulmion (tässä tutkielmassa neliön) muotoisia paloja. Palojen kaikilla sivuilla on tietty tunniste, ja palat tulee asettaa pelilaudalle siten, että jokaisen palan jokaisella sivulla on vastassaan samanlainen tunniste kuin sivulla itsellään. Tässä tutkielmassa pelilaudan reunaan ja kulmiin tulevilla paloilla on myös erityinen reunatunniste. Tällaiset palat tulee sijoittaa pelilaudan reunaan tai kulmaan siten, että minäkään palan reunatunnistella ei ole vastassaan muita paloja.

Reunatäsmäyspelit ovat mielenkiintoisia, mutta myös todella haastavia pelejä, jotka erottuvat tavallisista palapeleistä huomattavalla vaikeudellaan. Pelin säännöt on helppo oppia ja jopa pieni lapsi voi osata asettaa paloja pelilaudalle sääntöjen mukaisesti. Kuitenkin oikean ratkaisun löytyminen reunatäsmäyspeliin voi pelin suhteellisen pienestäkin koosta huolimatta olla todellisen työn ja tuskan takana. Reunatäsmäyspelien ratkaisemisen vaikeus perustuu niiden NP-täydellisyyteen [Savelsbergh and van Embe Boas 1984; Takenaga and Walsh, 2006; Demaine and Demaine, 2007]. Tästä seuraa, että reunatäsmäyspeleille ei ole olemassa tehokasta ratkaisualgoritmia. Vaikkei tietylle reunatäsmäyspelin ilmentymälle voidakaan välttämättä löytää täydellistä ratkaisua, voidaan erilaisilla heuristiikoilla löytää kuitenkin melko hyviä osittaisratkaisuja.

Tässä tutkielmassa esitellään neljä erilaista reunatäsmäyspelien ratkaisemiseen tarkoitettua heuristiikkaa, joista ensimmäinen on geneettinen algoritmi, toinen puolestaan perustuu erääseen monitavoitteiseen evolutiiviseen algoritmiin, kolmas on hill climbing -algoritmi ja neljäs algoritmi perustuu simuloituun jäähdytykseen. Tutkielmassa käytetty geneettinen algoritmi ja monitavoitteinen evolutiivinen algoritmi perustuvat Muñozin ja muiden [2009] kehittämiin lähdekoodeihin, joita on muunnettu toimimaan tätä tutkielmaa varten kehittämäni ohjelmarungon yhteydessä. Muuntotyön yhteydessä algoritmien toiminnallisuus on pyritty pitämään mahdollisimman samanlaisena. Muut tutkielmassa käsiteltävät algoritmit ovat omaa tuotantoani.

Seuraavassa luvussa käsitellään tarkemmin reunatäsmäyspelien rakennetta ja ominaisuuksia. Luvussa esitellään myös tämän tutkielman kannalta tärkeä reunatäsmäyspeli Eternity II. Lisäksi luvussa 2 käsitellään reunatäsmäyspelien aikavaativuutta ja approksimoituvuutta. Luku 3 kertoo tutkielmassa käytettyjen algoritmien taustaa ja esittelee myös algoritmien yleisiä toimintaperiaatteita. Luvussa 4 käydään tutkielmassa käytetyt algoritmit ja niihin liittyvä ohjelmarunko tarkemmin läpi. Luku 5 sisältää algoritmeilla tehdyt testit, testien tulokset ja tulosten analyysia. Lopuksi luvussa 6 tehdään yhteenveto tutkielman tuloksista.

1.1. Motiivi

Erilaiset loogista päättelyä vaativat pelit ja ongelmat ovat aina kiinnostaneet minua. Nuorempana kulutin mieluusti aikaani erilaisia ongelmia ratkoen, mutta ennen pitkää aloin pohtia myös yksittäisten ratkaisujen takana olevaa syvempää logiikkaa ja rakennetta. Loogista päättelyä vaativilla peleillä ja ongelmilla on usein sellainen ominaisuus, että ne ovat idealtaan yksinkertaisia ja niiden säännöt ovat melko helposti omaksuttavissa. Yksittäisistä pelitapauksista voi kuitenkin tehdä mielivaltaisen helppoja tai vaikeita muuntelemalla sopivalla tavalla pelin rakenteeseen vaikuttavia parametreja. Yleisesti ottaen pelitapauksen koko vaikuttaa oleellisesti ratkaisun vaativuuteen (tai sen työläyteen). Tämän takia loogiset pelit säilyttävät mielenkiintonsa usein pitkään, sillä jokaiselle pelaajalle löytynee hänelle sopiva vaikeustaso ja pitkään uutta opittavaa.

Tutustuin reunatäsmäyspeleihin tarkemmin nelisen vuotta sitten. Näin silloin uutisen uudesta Eternity II -lautapelistä, jonka ratkaisijalle luvattiin kahden miljoonan dollarin palkinto [Eternity II, 2007]. Ennen kuin perehdyin asiaan tarkemmin, innostuin kovasti tästä pelistä ja kuvittelin jo mielessäni kuinka helposti pelin voisikaan ratkaista tietokoneella. Kirjoitin aiheesta hieman myöhemmin kandidaatintyöni [Kujala, 2008], joka palauttikin minut maan pinnalle. Kandidaatintyötäni varten kehittämälläni pelilaudan rajoitteet huomioivalla yksinkertaisella peruutustekniikkaan (engl. *backtracking*) perustuvalla raa'an voiman algoritmeilla Eternity II:n ratkaisun löytyminen oli käytännössä katsoen mahdollonta massiivisen hakuavaruuden takia.

Eternity II ja muut reunatäsmäyspelit jäivät silti kiinnostamaan vielä kandidaatintutkielmani jälkeen, ja edellisestä työstäni hieman viisastuneena keskityn tässä tutkielmassa Eternity II:n mahdollisimman hyviin osittaisratkaisuihin täydellisen ratkaisun hakemisen sijaan.

1.2. Reunatäsmäyspelit kirjallisuudessa

Reunatäsmäyspelejä on tutkittu kirjallisuudessa melko paljon eri yhteyksissä (ks. [Antoniadis and Lingas, 2010]). Jo vuonna 1966 Berger [1966] todisti seuraavankaltaisen tuloksen: jos käytettävissä on ääretön määrä kopioita käytettävistä paloista, on ratkeamaton ongelma selvittää, voiko näistä paloista koota valmiin äärellisen pelilaudan.

Savelsbergh ja van Embe Boas [1984] todistivat reunatäsmäyspelit NP-täydellisiksi vuonna 1984. Tämän jälkeen ainakin Takenaga ja Walsh [2006] sekä Demaine ja Demaine [2007] ovat todistaneet reunatäsmäyspelit NP-täydellisiksi. Takenaga ja Walsh keskittyivät todistuksessaan Tetravex-peliin [Tetravex, 2008], joka on tietokoneella pelattava reunatäsmäyspeli. Savelsbergh ja van Embe Boas käyttivät todistuksessaan reunatäsmäyspelien palautusta suoraan Turingin koneeseen, kun Takenaga ja Walsh puolestaan tukeutuivat 1-in-3 SAT -ongelmaan. Demainen ja Demainen työkaluna todistuksessa oli 3-ositusongelma (engl. *3-partition problem*).

Kirjallisuudessa on myös käsitelty monia erityyylisiä heuristiikkoja reunatäsmäspeileille. Esimerkkeinä mainittakoon Ansóteguin ja muiden [2008] SAT/CSP-lähestymistapa, jossa pelilauta koodattiin joko Boolean lausekkeiden toteutuvuusongelmaksi (SAT) tai rajoitteiden toteutuvuusongelmaksi (CSP). Benoist ja Bourreau [2008] sekä Schaus ja Deville [2008] puolestaan lähestyivät ongelmaa rajoiteohjelmoinnin puolelta. Muñoz ja muut [2009] käyttivät pelilaudan ratkaisemiseen muun muassa evolutiivisia algoritmeja.

Edellä mainittujen tutkimusten joukosta Ansótegui ja muut sekä Benoist ja Bourreau hakivat tutkittavan reunatäsmäspelin täydellistä ratkaisua, kun taas Muñoz ja muut sekä Schaus ja Deville pyrkivät mahdollisimman hyvään osittaisratkaisuun. Tämä tutkielma keskittyy suuressa määrin Muñozin ja muiden työhön ja heidän kehittämiin evolutiivisiin algoritmeihin.

2. Reunatäsmäyspelit

Reunatäsmäyspelit koostuvat pelilaudasta ja laudalle asetettavista paloista. Pelin tarkoituksena on latoa palat pelilaudalle huomioiden pelin sääntöihin kuuluvat rajoitteet. Yksittäinen rajoite voi esimerkiksi määrätä, että palat tulee latoa tiettyyn yhtenäiseen muotoon pelilaudalle. Lisäksi palojen välillä voi olla erilaisia rajoitteita, jotka hyväksyvät ainoastaan tietyntyypisiä paloja vierekkäin. Reunatäsmäyspelien palat ovat tavallisesti joko yksinkertaisia geometrisia muotoja tai niistä muodostettuja hiukan monimutkaisempia paloja.

Reunatäsmäyspelit muistuttavat suuresti perinteisiä palapelejä, joten oletettavasti reunatäsmäyspelit ovat kehittyneet aikoinaan palapeleistä. Tavallisten palapelien tarkka historia on hieman hämärän peitossa, mutta yleisesti ottaen ollaan yhtä mieltä siitä, että palapelit keksi John Spilsbury vuoden 1760 tienoilla. Spilsbury liimasi erään karttansa ohuelle puulevylle, jonka jälkeen hän sahasi levyn kuviosahalla erimuotoisiin osiin. Aluksi palapelit miellettiin lähinnä opettavaisena huvina, mutta kun niiden suosio kasvoi, käytettiin palapelejä yhä enemmän opetustarkoituksessa. [McAdam, 2004]

Myöskään reunatäsmäyspelien tarkkaa syntyhetkeä ei ole tiedossa, mutta Rob Stegmannin [2007] lähteiden mukaan ensimmäistä patenttia reunatäsmäyspelille haettiin vuonna 1880, joten tuota vuotta voidaan pitää eräänlaisena virstanpylväänä reunatäsmäyspelien historiassa. Vaikka reunatäsmäyspelit eivät ole saavuttaneet palapelien kaltaista suosiota, lienee niilläkin oma vankka kannattajakuntansa.

Reunatäsmäyspelit ovat saaneet tavallista suurempaa huomiota viime vuosina, kun Christopher Monckton julkaisi Eternity- ja Eternity II -lautapelit. Eternity-reunatäsmäyspeli julkaistiin vuonna 1999 ja se koostuu 209 palasta, joista kukin on muodostettu yhdistämällä samankokoisia suorakulmaisia kolmioita, joiden kulmat ovat 30, 60 ja 90 astetta. Palat oli tarkoitus latoa säännöllisen kaksitoistakulmion muotoon. Christopher Monckton lupasi Eternityn ratkaisijalle miljoonan punnan palkkion. Alex Selby ja Oliver Riordan ratkaisivat Eternityn ensimmäisinä 15. toukokuuta 2000. He löysivät pelistä kombinatorisen heikkouden, jota hyväksikäyttämällä ratkaisu löytyi. [Selby, 2001]

Eternity II julkaistiin puolestaan vuonna 2007. Edellisestä tappiostaan viisastuneena Christopher Monckton päätti tehdä jatko-osasta alkuperäistä Eternityä paljon vaativamman ja palkkasi Selbyn ja Oliverin kehittämään kanssaan Eternity II:ta. Monckton mainitsikin kehitystyön aikaan, että Eternity II tulisi olemaan monta kertaluokkaa vaikeampi kuin vuonna 1999 julkaistu Eternity. [The Sunday Times, 2005]

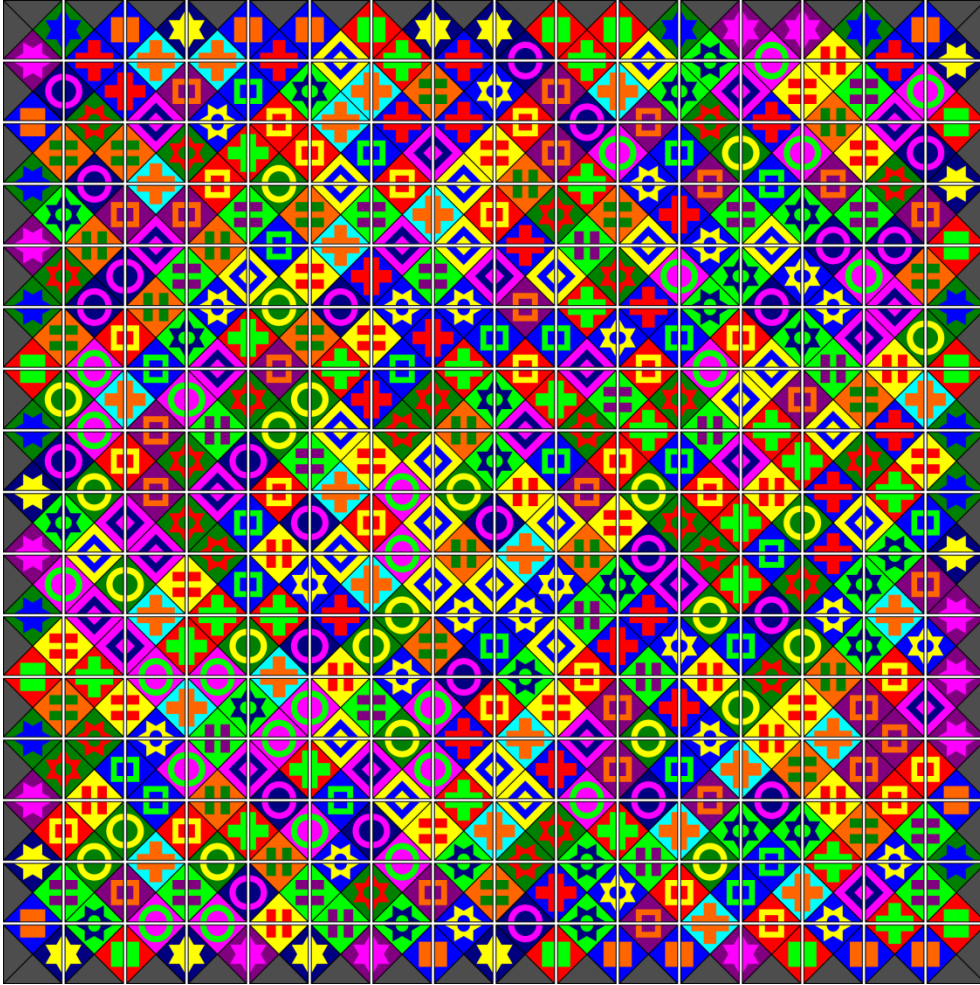
Eternity II sisältää 256 neliön muotoista palaa, jotka tulee sijoittaa 16 x 16 palan kokoiselle pelilaudalle siten, että lukuunottamatta tiettyjen palojen erityisiä ulkoreunoja jokaisen palan jokaisella sivulla on samanlainen vastinsivu. Yksi pelilaudan 256 palasta on pakollinen vihjepala, jonka sijainti ja suunta on ennalta päätetty. Eternity II:n ratkaisemisesta maksettavaksi palkintorahaksi oli määritelty tällä kertaa kaksi miljoonaa dollaria. Kurioositeettina mainittakoon, että kahden miljoonan dollarin palkinto vastasi vuonna 2007 noin

miljoonaa Ison-Britannian puntaa. Ensimmäinen ratkaisujen tarkistuspäivä oli 31.12.2008, mutta tuolloin ei löydetty ainoatakaan täydellistä ratkaisua. Vuotta myöhemmin toisen tarkistuspäivän aikaan siihen mennessä vastaanotetulle parhaalle ratkaisulle maksettiin kymmentuhannen dollarin palkinto. Anna Karlsson Ruotsista oli saanut sijoitettua Eternity II:n 256 palaa pelilaudalle siten, että kaikista pelilaudalle sijoitettujen palojen vierekkäisistä reunapareista 467 kappaletta oli oikein. Kaikkiaan reunapareja on Eternity II:ssa 480 kappaletta. Viimeinen tarkistuspäivä Eternity II:lle oli 31.12.2010, mutta tässäkin tarkistuksessa ei löytynyt yhtään täydellistä ratkaisua, joten kahden miljoonan dollarin palkintosumma jäi jakamatta. Monckton ei ole kuitenkaan julkaissut Eternity II:n oikeaa ratkaisua. [Eternity II, 2007]

2.1. Rakenne

Kuten aiemmin mainittiin, reunatäsmäyspelit koostuvat yleisesti ottaen melko yksinkertaisen muotoisista paloista, joita asetetaan pelilaudalle siten, että kyseisen pelin säännöissä määritellyt tavoitteet täyttyvät. Reunatäsmäyspelit muodostavat kuitenkin laajan peliluokan, joka sisältää monia erilaisia variaatioita. Koska tässä tutkielmassa keskitytään reunatäsmäyspeleistä ainoastaan Eternity II:een (joskin tutkielman sisältö on sovitettavissa myös muille samaan luokkaan kuuluville reunatäsmäyspeleille), ei reunatäsmäyspelien yleisiin ja yhteisiin ominaisuuksiin perehdytä.

Eternity II:ssa käytetään neliön muotoisia paloja, joiden jokaisella neljällä sivulla on jonkinlainen kuvio. Kuten edellä jo mainittiin, palat tulee latoa 16×16 palan kokoiselle neliön muotoiselle pelilaudalle siten, että jokaisen palan jokaisella sivulla on vastassaan samanlainen vastinsivu (poislukien erityisten reunapalojen ulkosyrjät). Koska graafisten kuvioiden esittäminen ei ole kovin tarkoituksenmukaista ratkaisualgoritmien kannalta, voidaan jokaiselle erilaiselle pelin paloissa esiintyvälle kuviolle antaa vastineeksi esimerkiksi jokin kokonaisluku. Pelilaudan laidoilla ja kulmissa sijaitsevilla paloilla ei ole pelilaudan reunojen ulkopuolella enää vastinpaloja, joten kyseisille paloille on määriteltävä erityinen reunatunniste. Tässä tutkielmassa reunatunnistetta merkitään nollalla ja muita tunnisteita positiivisina kokonaislukuina alkaen numerosta yksi. Kuvassa 1 nähdään Eternity II:ta vastaava reunatäsmäyspeli ratkaistuna.



Kuva 1. Esimerkki ratkaistusta Eternity II:n kaltaisesta reunatäsmäyspelistä.

Eternity II:ssa on käytössä myös pakollinen vihjepala, mutta tässä työssä vihjepalaa ei otettu käyttöön. Vihjepalalla ei liene muuta tarkoitusta kuin vähentää pelistä keskenään symmetrisiä ratkaisuja [Schaus and Deville, 2008]. Vihjepalan käyttö pienentää entisestään mahdollisuutta löytää Eternity II:n täydellistä ratkaisua. Vihjepala ei ole oleellinen tämän tutkielman kannalta, vaan tarkoituksena on löytää keskimäärin mahdollisimman hyvän pistemäärän antava algoritmi.

Eternity II:ssa käytössä olevat reunapalat eroavat pelilaudan keskelle tulevista paloista muutenkin kuin vain reunatunnisteiden osalta. Reunapalat toisiinsa liittäville sivuille käytetään vain viittä erilaista tunnistetta, joita voidaan tässä merkitä numeroilla 1–5, kun puolestaan pelilaudan keskellä sijaitsevissa paloissa ja reunapalojen sisemmällä sivulla (pois lukien kulmapalat) käytetään seitsemäätoista erilaista tunnistetta, joita merkitään vastaavasti numeroilla 6–22. Reunapaloissa käytettävä tunnistejoukko on täten täysin erillinen keskuspaloissa käytettävään tunnistejoukkoon nähden.

Erilaisten tunnisteen määrä ja jakauma vaikuttavat pelilaudan koon lisäksi oleellisesti reunatäsmäyspelin vaikeuteen. Reunatäsmäyspeli, joka sisältää ainoastaan yhtä tunnis-

tetta, on algoritmisesti täysin triviaali ratkaista. Käytännössä tilanne on sama silloin, jos tunnisteita olisi yhtä monta erilaista kuin pelilaudalla on vierekkäisiä sivupareja. Tällöin reunatäsmäyspeli muuttuisi käytännössä tavalliseksi palapeliksi, jonka ratkaisemiseksi riittää etsiä pari jokaiselle reunalle. Reunatäsmäyspeli luonnollisesti vaikeutuu verrattuna kahteen edellä mainittuihin ääripäähän, kun tunnisteiden määrä on jotain yhden ja maksimimäärän väliltä.

Reunatäsmäyspelejä voi yleisesti ottaen koota jonkin matkaa ilman suurempia ongelmia vain yhdistelemällä paloja keskenään tavallisen palapelin tapaan, mutta näin saavutettu paikallinen ratkaisu johtaa hyvin harvoin laajempaan ratkaisuun.

Ansótegui ja muut [2008] päätyivät sellaiseen kokeelliseen tulokseen, että jos reunatunnisteita on yhteensä viisi erilaista, niin tällöin 16×16 palan kokoiset reunatäsmäyspelit ovat kaikkein vaikeimpia ratkaista, jos niissä käytetään seitsemäätoista sisätunnistetta. On syytä huomioida, että Ansóteguin ja muiden tulos on kokeellisesti päätelty, eikä sitä ole todistettu analyyttisesti. Eternity II:n kehityksestä ei sen kaupallisen luonteen vuoksi ole saatavilla tarkempaa tietoa, mutta oletettavaa toki on, että Monckton halusi pelistä kokonsa nähden mahdollisimman vaikean ratkaista. Tällöin Monckton ja pelin muut kehittäjät ovat mahdollisesti päätyneet aikoinaan samaan tulokseen Ansóteguin ja muiden kanssa palojen tunnisteiden määrästä ja jakaumista mahdollisimman vaikean lopputuloksen aikaansaamiseksi.

Eternity II:n tunnisteiden määrä on hyvin tasaisesti jakautunut. Keskimäärin kaikkia sisätunnisteita on suunnilleen yhtä paljon. Sama pätee myös ulkotunnisteisiin. Lisäksi pelin jokainen pala on erilainen, joten mahdollisten symmetristen ratkaisujen määrää on pystytty täten karsimaan huimasti.

2.2. Reunatäsmäyspelien aikavaativuus ja approksimoitavuus

On helppoa laskea, että jos Eternity II:n tyyllisessä reunatäsmäyspelissä on n palaa, voidaan nämä palat sijoittaa pelilaudalle $n!$ eri tavalla. Jos lisäksi huomioidaan, että jokainen pala voidaan kääntää neljään eri asentoon, on pelilauta mahdollista täyttää $n! \times 4^n$ eri tavalla. Eternity II:n tapauksessa erilaisia ratkaisumahdollisuuksia on siis noin $1,15 \times 10^{661}$ kappaletta. Tässä laskussa ei ole kuitenkaan huomioitu sitä, että pelilaudan reunoille ja kulmiin voi tulla vain tietynlaisia paloja tiettyyn asentoon sijoitettuna. Jos nämä rajoitteet otetaan huomioon, Eternity II:lla on silti $196! \times 4^{196} \times 56! \times 4! \approx 8,7 \times 10^{559}$ erilaista konfiguraatiota. Selvää on, että näin valtavaa hakuavaruutta on hyödytöntä yrittää käydä läpi kokeilemalla eri vaihtoehtoja.

Seuraavaksi tarkastellaan reunatäsmäyspelien aikavaativuutta ja approksimoitavuutta. Käydään sitä ennen kuitenkin läpi joitain tarvittavia taustatietoja.

2.2.1. Päätösongelmat ja aikavaativuusluokat

Päätösongelmiksi kutsutaan sellaisia ongelmia, joihin voi vastata ainoastaan "kyllä" tai "ei" (vaihtoehtoisesti 1 tai 0). Esimerkkinä päätösongelmasta voisi olla vaikkapa ongelma p_1 : "*Onko reunatäsmäyspelillä R olemassa täydellinen ratkaisu?*". Annettuun kysymykseen ei voi vastata muulla tavoin järkevästi kuin myöntävästi tai kieltävästi. Lisäksi voidaan huomata, että kysymyksessä mainittu annettu reunatäsmäyspelin ilmentymä R ei vaikuta vastausvaihtoehtoihin, vaan millä tahansa annetulla reunatäsmäyspelillä kysymyksen vastausvaihtoehdot pysyvät samana.

Voidaan tarkastella myös ongelmaa p_2 : "*Anna reunatäsmäyspelin R täydellinen ratkaisu*". Kuten huomataan, nyt vastaukseksi kaivataan jokin konkreettinen pelin ilmentymä. Ongelmia, joiden vastaukseksi kaivataan jonkin joukon suotuisinta alkioita, kutsutaan optimointiongelmiiksi. Tämän alakohdan alussa esitetty ongelma p_1 on edellä esitettyä optimointiongelmaa p_2 vastaava päätösongelma. Mistä tahansa optimointiongelma voidaan muodostaa sitä vastaava päätösongelma ja päinvastoin. Huomionarvoista on, että tietyn optimointiongelman ratkaisu antaa ongelmasta enemmän informaatiota kuin vastaavan päätösongelman ratkaisu.

Sellaiset päätösongelmat, jotka voidaan ratkaista polynomisessa ajassa ongelman kokoon nähden, kuuluvat aikavaativuusluokkaan P. Toisin sanoen luokkaan P kuuluvan päätösongelman ratkaisualgoritmin aikavaativuus on $O(n^k)$, kun n on ongelman koko ja k jokin positiivinen vakio. Luokkaan P kuuluvien päätösongelmien katsotaan olevan tehokkaasti ratkaistavissa. Jos päätösongelma kuuluu luokkaan P, niin myös vastaava optimointiongelma voidaan ratkaista polynomisessa ajassa.

Kuten tämän kohdan alussa nähtiin, Eternity II -tyylisten reunatäsmäyspelien erilaisien ratkaisuvaihtoehtojen määrä kasvaa palojen lukumäärän kertoman suhteessa. Tämän kaltainen kasvunopeus on erittäin voimakasta, eikä Eternity II -tyylisille reunatäsmäyspeleille ole löydetty polynomiaikaista ratkaisualgoritmia. Tästä syystä Eternity II -tyyliset reunatäsmäyspelit ovat niin vaikeita ratkaista. Tästä aiheesta pääsemmekin oivasti aikavaativuusluokkaan NP.

Luokka NP (engl. *non-deterministic polynomial time*) sisältää sellaiset päätösongelmat, joiden "kyllä"-vastaukset voidaan ratkaista epädeterministisellä Turingin koneella polynomisessa ajassa. "Tavallinen", deterministinen Turingin kone päättyy tietyn algoritmin suorituksen jälkeen yhteen tiettyyn lopputulokseen, kun epädeterministisen Turingin koneen suorituksen voidaan puolestaan ajatella ikään kuin monistuvan algoritmin jokaisessa haarakohdassa. Näin ollen epädeterministinen Turingin kone tavallaan saavuttaa algoritmin kaikki mahdolliset suoritushaarat yhdellä ajolla. Jos siis jonkin päätösongelman kaikki suoritushaarat voidaan käydä läpi epädeterministisellä Turingin koneella edellä kuvatulla tavalla polynomisessa ajassa läpi, kuuluu kyseinen ongelma luokkaan NP.

Yhtäpitävää edellisen kappaleen kanssa on sanoa, että luokkaan NP kuuluvat sellaiset päätösongelmat, joiden varmenteista (l. ratkaisuehdotuksista) voi deterministisellä Turingin koneella tarkistaa polynomisessa ajassa, onko vastaus kyseiseen päätösongelmaan "kyllä". Yhtäpitävyys perustuu siihen seikkaan, että epädeterministisellä Turingin koneella voidaan ensiksi epädeterministisesti muodostaa jokin varmenne polynomisessa ajassa, jonka jälkeen muodostetun varmenteen mahdollisen "kyllä"-vastauksen voi tarkistaa deterministisesti polynomisessa ajassa. [Cormen et al., 2007]

Luokkien P ja NP määrittelyistä seuraa, että $P \subset NP$, sillä jos luokan P päätösongelmat ratkeavat polynomisessa ajassa deterministisellä Turingin koneella, ratkeavat ne varmasti polynomisessa ajassa myös epädeterministisellä Turingin koneella. Yksi tietojenkäsittelytieteen suurimmista ratkaisemattomista ongelmista on se, ovatko P ja NP samat joukot, vai kuuluuko luokkaan NP joitakin joukkoon P kuulumattomia ongelmia. Yleisesti uskotaan, että $P \neq NP$, mutta tätä ei ole pystytty todistamaan [Gasarch, 2002].

Nykyisen käsityksen perusteella luokkaan NP kuuluu siis muitakin kuin luokan P helpohkosti ratkaistavia ongelmia. Eräs tämän tutkielman kannalta oleellinen ongelmajoukko on NP-täydelliset ongelmat.

Päätösongelma D on NP-täydellinen jos sillä on kaksi seuraavaa ominaisuutta:

1. Ongelma kuuluu luokkaan NP.
2. Jokainen joukon NP päätösongelma on muunnettavissa ongelmaksi D polynomisessa ajassa deterministisellä algoritmilla.

Mainituista ominaisuuksista luokkaan NP kuulumisen selvitettiin jo aiemmin, joten asiaa ei enää käsitellä tässä. Päätösongelman muuntaminen toiseksi ongelmaksi tarkoittaa sitä, että jokainen päätösongelman ilmentymä $k \in K$ voidaan jollain deterministisellä algoritmilla muuntaa toisen päätösongelman ilmentymäksi $d \in D$ siten, että d :n tulos on "kyllä" jos ja vain jos k :n tulos on myös "kyllä". Tämä ominaisuus tarkoittaa yksistään ilman luokkaan NP kuulumista, että päätösongelma D on NP-kova.

NP-täydellisiä ongelmia pidetään luokan NP vaikeimpina ongelmoina. NP-täydellisten ongelmien määrittelyä seuraa, että jos mille tahansa NP-täydelliselle ongelmalle Q keksitään deterministinen polynominen ratkaisualgoritmi, niin tällöin $P = NP$. Tämä johtuu siitä, että jokainen joukon NP ongelma on muunnettavissa polynomisessa ajassa ongelmaksi Q . Nythän voimme valita minkä tahansa luokan NP ongelman ja muuntaa sen polynomisessa ajassa ongelmaksi Q , joka puolestaan ratkeaa myös polynomisessa ajassa. Koko toimenpiteen aikavaativuus on näin ollen polynominen.

Tämän pitkäkhön alustuksen jälkeen huomaamme, että reunatäsmäyspelien mistä tahansa palojen järjestyksestä voimme tarkistaa palojen määrän suhteen lineaarisessa ajassa, onko kyseinen palojen järjestys kyseisen reunatäsmäyspelin täydellinen ratkaisu. Tämä tarkoittaa, että reunatäsmäyspelit kuuluvat luokkaan NP. Tämän lisäksi erilaisia jo tiedos-

sa olevia NP-täydellisiä ongelmia voidaan palauttaa reunatäsmäyspelin ilmentymiksi polynomisessa ajassa. Tästä seuraa se, että reunatäsmäyspelit ovat NP-täydellisiä.

Kuten tutkielman alkupuolella jo mainittiin, reunatäsmäyspelien NP-täydellisyyden on todistanut toisistaan riippumatta ainakin Savelsbergh ja van Embe Boas [1984], Takenaga ja Walsh [2006] sekä Demaine ja Demaine [2007]. Kun pyritään todistamaan jotain ongelmaa NP-täydelliseksi, yleensä vaikeinta on keksiä polynomiaikainen palautus muusta NP-täydellisestä ongelmasta. Toistaalta nykyisin, kun erilaisia NP-täydellisiä ongelmia tunnetaan melko paljon, voi olla helpompaa myös löytää soveltuvampi ongelmakandidaatti palautusta varten. Savelsbergh ja van Embe Boas eivät käyttäneet toista NP-täydellistä ongelmaa todistuksessaan, vaan he rinnastivat suoraan tiettyjä polynomisesti rajoitettuja epädeterministisiä laskentoja reunatäsmäyspelin ilmentymiksi käyttäen Turingin koneen laskumallia. Tällä tavoin toimi myös Cook [1971] todistaessaan lauselogiikan toteutuvuusongelman (SAT) NP-täydelliseksi. Mainittakoon, että lauselogiikan toteutuvuusongelma oli ensimmäinen ongelma, joka todistettiin NP-täydelliseksi. Takenaga ja Walsh tukeutuivat todistuksessaan Cookin työhön, sillä he käyttivät todistuksessaan palautusta 1-in-3 SAT -ongelmaan. Demainen ja Demainen todistus perustui puolestaan 3-ositusongelmaan. Kyseisessä ongelmassa on tarkoitus selvittää, voiko annetun monijoukon alkioista muodostaa sellaisia kolmikoita, joiden kaikkien summa on sama.

2.2.2. Reunatäsmäyspelien approksimointi

Koska reunatäsmäyspeleille (tai mille muullekaan NP-täydelliselle ongelmalle) ei ole löydetty polynomiaikaista ratkaisualgoritmia, kovinkaan suuria reunatäsmäyspelien ilmentymiä ei voi käytännössä ratkaista täydellisesti nykyisin käytössä olevalla laskentateholla. On siis tarpeen keskittyä mahdollisimman hyviin osittaisratkaisuihin. Tämä tarkoittaa käytännössä reunatäsmäyspelien approksimointia.

Edellisessä alakohdassa keskityttiin lähinnä päätösongelmiin, sillä NP-täydellisyys määritellään päätösongelmien avulla. Koska olemme kiinnostuneita pelkän reunatäsmäyspelin ratkeavuuden lisäksi myös siitä, kuinka *hyvin* reunatäsmäyspeli voidaan (osittais)ratkaista, palataan nyt edellisessä alakohdassa esiintyneeseen optimointiongelman käsitteeseen. Optimointiongelman vastaukseksi haetaan siis jonkin joukon suotuisinta tai suotuisimpia alkioita. Käsitellään seuraavaksi hieman optimointiongelmiin liittyvää teoriaa. Tämän alakohdan optimointiongelmiin perustuva tieto on suurelta osin peräisin Cormenin ja muiden [2007] julkaisusta.

Formaalisti optimointiongelma on kolmikko $p = (D, S, c)$, missä D on jonkin tietyn ongelman kaikkien erilaisten tapausten joukko ja S puolestaan edustaa kunkin D :n alkion ratkaisuvaihtoehtoja, joiden hyvyys voidaan evaluoida. Kutakin käsiteltävän ongelman tapausta $d \in D$ vastaa siis tapauksen kaikkien mahdollisten ratkaisuvaihtoehtojen joukko $S(d) \subseteq S$. Ratkaisuvaihtoehtojen evaluointi perustuu kustannusfunktioon $c: D \times S \rightarrow \mathbb{R}$.

Optimointiongelmia tarkasteltaessa kustannusfunktio c on käsiteltävän ongelman suure, jota pyritään minimoimaan tai maksimoimaan. Esimerkinomaisesti voidaan todeta, että reunatäsmäyspelien optimoinnissa D voisi edustaa kaikkia erilaisia Eternity II -tyyppisiä reunatäsmäyspelejä. Tällöin S on sellainen joukko, johon sisältyy kaikkien joukon D alkoioiden d palojen erilaiset järjestykset. Esimerkiksi Eternity II:n $e_2 \in D$ tietty palojen järjestys on alkio $s \in S(e_2)$ ja $|S(e_2)| = 256! \times 4^{256}$. Nyt kustannusfunktio $c(e_2, s)$ tarkoittaa Eternity II:n tietyn palojen järjestyksen pistemäärää.

Optimointiongelma voi olla joko minimointiongelma tai maksimointiongelma. Minimointiongelman ilmentymälle $d \in D$ ratkaisuehdotus $s' \in S(d)$ on optimaalinen, jos $c(d, s') \leq c(d, s)$, kun s on mikä tahansa joukon $S(d)$ ratkaisuehdotus. Vastaavasti maksimointiongelmalle pätee $c(d, s') \geq c(d, s)$, kun s on mikä tahansa joukon $S(d)$ ratkaisuehdotus. Jos optimaalisen ratkaisun kustannuksesta käytetään merkintää $c'(d)$, minimointiongelman ratkaisuehdotuksen $s \in S(d)$ hyvyys $r(d, s)$ voidaan laskea seuraavasti:

$r(d, s) = \frac{c(d, s) - c'(d)}{c'(d)}$, kun $c'(d) \neq 0$. Maksimointiongelman hyvyys määritellään vastaavasti $r(d, s) = \frac{c'(d) - c(d, s)}{c(d, s)}$, kun $c'(d) \neq 0$. Mitä pienempi hyvyys, sitä parempi on silloin siihen

liittyvä ratkaisuehdotus.

Reunatäsmäyspelit ovat pelejä, joiden voidaan ajatella olevan joko ratkaistuja tai keskeneräisiä. Voimme kuitenkin myös ajatella kustannusfunktion merkitsevän reunatäsmäyspeleissä esimerkiksi niiden vierekkäisten sivuparien määrää, joiden vastinsivut ovat yhtenevät. Reunatäsmäyspelistä riippuen toisiaan vastaavien sivuparien määrä m vaihtelee, mutta se on kuitenkin aina nolaa suurempi. Reunatäsmäyspelien ratkaisemiseen liittyvä optimointiongelma voidaan käsittää siis maksimointiongelmana, jossa pelilaudan tuottama pistemäärä yritetään saada mahdollisimman suureksi.

Tiettyä optimointiongelmaa varten kehitetty algoritmi A on approksimointialgoritmi, jos se tuottaa jokaiselle alkioille $d \in D$ jonkin ratkaisuehdotuksen, eli $A(d) \in S(d)$. Lisäksi A on ε -approksimointialgoritmi ($\varepsilon \geq 0$), jos A :n kaikki ratkaisut ovat ε -hyviä, siis $r(d, A(d)) \leq \varepsilon$, kun $d \in D$.

Nyt kun tiedämme ε -approksimointialgoritmien merkityksen, voimme määritellä uusia käsitteitä. PTAS (engl. *Polynomial-Time Approximation Scheme*) tarkoittaa tiettyyn optimointiongelmaan Q liittyvää algoritmijoukkoa \mathcal{A} , jolle on ominaista se, että joukkoon kuuluvat algoritmit ovat polynomisessa ajassa toimivia ε -approksimointialgoritmeja kaikilla $\varepsilon > 0$. Huomioitavaa on, että kullakin yksittäisellä algoritmilla $A \in \mathcal{A}$ arvo ε on vakio.

Aikavaativuusluokka APX sisältää kaikki sellaiset optimointiongelmat Q , joiden päätösongelmaversiot kuuluvat luokkaan NP, ja joille on olemassa polynomiaikainen r -approksimointialgoritmi jollakin $r \geq 0$. Optimointiongelma Q on puolestaan APX-

täydellinen, jos jokainen ongelma, joka kuuluu luokkaan APX, voidaan PTAS-muunnoksella muuttaa polynomisessa ajassa ongelman Q ilmentymäksi.

Antoniadis ja Lingas [2010] todistivat käyttäen Max-3DM-B -ongelmaa apunaan, että reunatäsmäyspelien optimointiversio kuuluu luokkaan APX. Tämän lisäksi he osoittivat, että reunatäsmäyspelien optimointiongelma on APX-täydellinen. Antoniadis ja Lingas todistivat myös, että jos reunatäsmäyspelin tulosta approksimoidaan, approksimoidun tuloksen saaminen kerrointa $\frac{14250}{14249}$ lähemmäs optimaalista tulosta on NP-kova ongelma.

3. Etsintäalgoritmit

Tämä luku esittelee tutkielmassa käytettyjen etsintäalgoritmien taustaa ja toimintaperiaatteita. Tarkasteltaviin algoritmeihin kuuluvat geneettiset algoritmit, monitavoitteiset evoluutiiviset algoritmit, hill climbing -algoritmit sekä simuloituun jäähdytykseen perustuvat algoritmit. Mainitut algoritmityyppit käydään seuraavaksi läpi edellä mainitussa järjestyksessä.

3.1. Geneettiset algoritmit

Geneettiset algoritmit ovat heuristiikkoja, joiden toiminta perustuu evoluutioteorian käsitteisiin. Keskeinen käsite geneettisissä algoritmeissa on populaatio, joka koostuu ongelman ratkaisuehdotuksista. Ratkaisuehdotuksia sisältävää populaatiota pyritään muokkaamaan kohti lopullista tavoitetta evoluutioteorian työkalujen avulla. Geneettiset algoritmit kuuluvat laajempaan evoluutiivisen laskennan piiriin.

Evoluutiivista (l. biologiseen evoluutioon perustuvaa) laskentaa alettiin tutkia 1950- ja 1960-luvuilla usean toisistaan riippumattoman tutkijan voimin. Tutkimuksen perusidea oli jo tuolloin valjastaa evoluution toimintaperiaatteet ratkaisukandidaateista muodostuvan populaation kehittämiseen. Vaikka evoluutiivisen laskennan voidaankin katsoa syntyneen osittain jo 1950-luvulla, ei geneettisiä algoritmeja ollut tuolloin vielä olemassa. Varsinaisesti geneettisten algoritmien perusidea syntyi Michiganin yliopistossa 1960-luvulla, kun John Holland ja hänen tutkimusryhmänsä tutkivat luonnollisen mukautumisen teoreettista taustaa ja sitä, miten tätä ilmiötä voisi hyödyntää ohjelmoinnissa. Vasta Hollandin teos *Adaptation in Natural and Artificial Systems* [1975] esitteli geneettiset algoritmit siinä muodossa, kun niitä pääosin nykyisin käytetään. [Mitchell, 1996]

Geneettiset algoritmit sopivat mitä erilaisimpien ongelmien ratkaisuun, ja niillä on ollut käyttöä tutkimuksen lisäksi paljon myös kaupallisilla ja teollisilla aloilla muun muassa erilaisten aikataulutusten suunnittelussa, tuotekehityksen apuna ja molekyyliarakenteiden tutkimisessa. [Davis, 2003]

3.1.1. Rakenne ja toimintaperiaate

Biologiasta tiedetään, että ihminen koostuu erilaisista soluista. Vaikka erityyppisillä soluilla on erilaisia tehtäviä, sisältävät yhden ihmisen kaikki solut (punasoluja ja verihiutaaleita lukuunottamatta) kuitenkin samat kromosomit. Kromosomit koostuvat yhtäjaksoisesta DNA-rihmasta. DNA sisältää puolestaan ihmisen kaiken geneettisen tiedon. [Guyton and Hall, 2000]

Kun kaksi ihmistä saa uuden jälkeläisen, sisältää jälkeläisen DNA puolet äidin ja puolet isän DNA:sta. Tällä tavoin jälkeläinen perii osan ominaisuuksistaan äidiltä, osan puolestaan isältä. Joskus DNA:n kopiointi kuitenkin epäonnistuu tai DNA:han vaikuttaa jokin ulkopuolinen tekijä. Tällöin DNA:n rakenneosiin voi aiheutua mutaatioita. Mutaatiot

muuttavat mutatoitunutta DNA:ta sisältävän solun ominaisuuksia. Kun mutatoitunut solu jakautuu, periytyy mutaatio myös sen jälkeläisiin. Joskus mutaatio voi parantaa kantajansa jotain tiettyä ominaisuutta, mutta usein kuitenkin mutaatiosta aiheutuu kantajalleen negatiivinen vaikutus. Jos rinnastamme DNA:n sisältämän tiedon jonkin ongelman ratkaisuehdotukseksi, voimme helposti siirtyä geneettisten algoritmien pariin.

Geneettisen algoritmin populaatio koostuu erillisistä yksilöistä, eli käsiteltävän ongelman ratkaisuehdotuksista. Solujen DNA:n tapaan voidaan ajatella, että kukin populaation yksilö sisältää rakennusohjeet, jolla kyseinen yksilö voidaan muodostaa.

Populaation sisältämien yksilöiden määrä on käsiteltävästä ongelmasta riippuvainen, eikä lukumäärälle ole mitään yleispätevää sääntöä. Yleensä käytännöllinen populaation koko selviää testaamalla erilaisia vaihtoehtoja. Ongelman luonteen ja koon perusteella voidaan tosin mahdollisesti arvioida tarvittavan populaation suuruusluokkaa. Monissa ongelmissa jo muutaman kymmenen yksilön populaatiolla voidaan saada tarpeeksi hyviä tuloksia, kun taas esimerkiksi tässä tutkielmassa käytetty populaatio on kymmenen tuhannen yksilön kokoinen.

Geneettinen algoritmi vaatii ennen suoritustaan populaation alustamisen. Alustamisen voi toteuttaa monella eri tyylillä, mutta käytetyin tapa lienee sellainen, jossa populaatio muodostetaan ongelman satunnaisista ratkaisuehdotuksista. On myös mahdollista alustaa populaation ratkaisuehdotukset jollain toisella algoritmilla, jolloin populaation yksilöt voivat olla lähtötasoltaan keskimäärin lähempänä optimaalista ratkaisua kuin satunnaiset yksilöt. Jollain toisella algoritmilla tehty alustus saattaa myös parantaa geneettisen algoritmin lopputulosta.

Alustuksen jälkeen populaatio asetetaan jonkin sopivalla tavalla ratkaisuehdotuksia pisteyttävän sopivuusfunktion mukaiseen järjestykseen. Koska sopivuusfunktio yksin määrittelee populaation yksilöiden välisen järjestyksen, on se samalla myös optimoinnin kohteena oleva funktio. Sopivuusfunktio on oikeastaan sama asia kuin alakohdassa 2.2.2 mainittu kustannusfunktio. Joskus voi olla hankala valita yhtä ainoaa sopivuusfunktiota; tutkittavalla ongelmalla voi olla esimerkiksi monta toisistaan riippuvaa tavoiteominaisuutta, joista jokainen halutaan maksimoida (tai minimoida). Tällaisissa tapauksissa sopivuusfunktio voidaan muodostaa esimerkiksi laskemalla painotettu keskiarvo haluttujen tavoitteiden pistemääristä. Sopivuusfunktion toimintaa voi tällöin hienosäätää muuttamalla eri tavoitteiden painokertoimia. Seuraavan kohdan asia monitavoitteisista evolutiivisista algoritmeista liittyy läheisesti useiden tavoitteiden optimointiin geneettisillä algoritmeilla.

Kun populaatio on järjestetty sopivuusfunktion mukaiseen järjestykseen, on myös löydetty kyseisen sukupolven paras ratkaisuehdotus. Nyt populaatiota tulisi jotenkin muokata sopivammaksi kohti optimaalista ratkaisua. Tavallisimmat toimenpiteet popu-

laation muokkaamiseen ovat elitismi, risteytys ja mutaatio. Nämä operaatiot ja niiden käyttö käsitellään yksityiskohtaisemmin seuraavassa alakohdassa.

Käsiteltävän (l. vanhan) sukupolven populaatio toimii lähtökohtana seuraavan (l. uuden) sukupolven populaatiolle, sillä vanhan sukupolven yksilöt toimivat vanhempina uuden sukupolven yksilöille. Uuden sukupolven populaatioon tehdään lisäksi paikallisia muutoksia hyödyntäen mutaatioita, joiden aiheuttaman satunnaisvaihtelun toivotaan vievän algoritmia kohti optimaalista ratkaisua. Uuden sukupolven muodostamisen jälkeen populaatio järjestetään edeltäjänsä tavoin yksilöiden sopivuuden mukaiseen järjestykseen.

Algoritmi jatkaa tällä tavoin kulkuaan, kunnes populaation jonkin yksilön sopivuusfunktio saavuttaa tavoiteltavan optimiarvon, tai kunnes tietty ennaltamäärätty aika- tai sukupolviraja täyttyy. Listaus 1 esittelee geneettisen algoritmin perustoiminnallisuuden pseudokoodina.

```

algorithm geneettinen_algoritmi(max_sukupolvi : int, max_sopivuus : float) : yksilö
  begin
    alusta populaatio
    sukupolvi := 1
    do
      evaluoi populaation sopivuus
      lajittele populaatio sopivuuden mukaiseen järjestykseen
      paras_yksilö := valitse populaation paras yksilö
      valitse seuraavaan sukupolveen parhaat yksilöt elitismillä
      muodosta loput uuden populaation yksilöt risteytyksellä
      mutatoi satunnaisia uuden sukupolven yksilöitä
      sukupolvi := sukupolvi + 1
    while sukupolvi < max_sukupolvi and paras_yksilö.sopivuus < max_sopivuus
    return paras_yksilö
  end

```

Listaus 1. Esimerkki geneettisen algoritmin toiminnasta pseudokoodina.

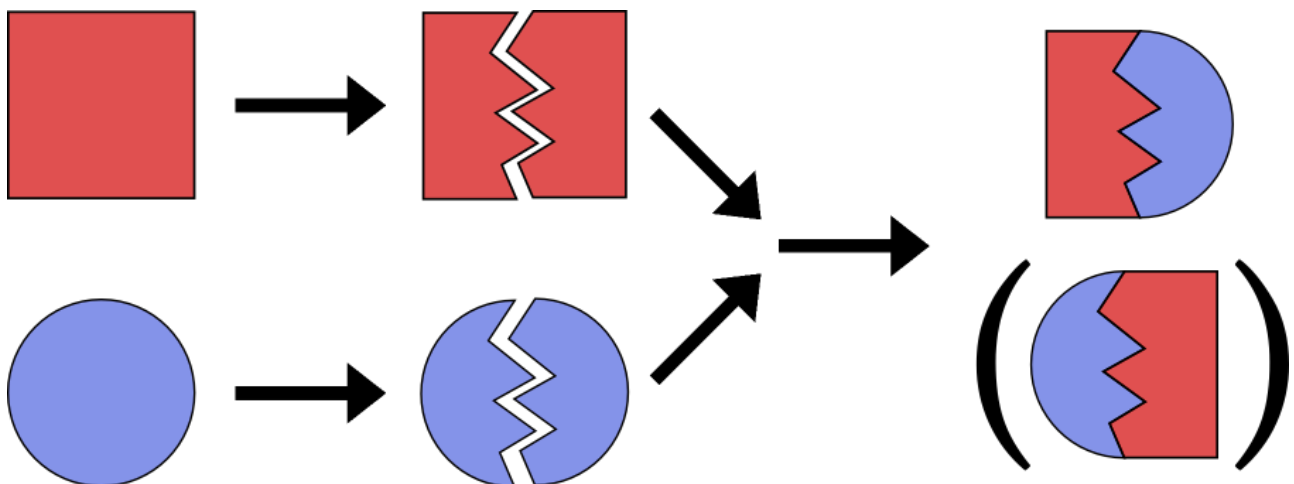
Yllä kuvattu pseudokoodi on melko karkea esimerkki geneettisen algoritmin toiminnasta. Algoritmin toimintaa on kuitenkin yleisellä tasolla hankala avata pseudokoodin avulla paljon tätä tarkemmin, sillä geneettisten algoritmien tarkempi rakenne riippuu paljolti siitä, minkälaisen ongelman ratkaisemiseksi algoritmi on tarkoitettu ja miten ongelman ratkaisuehdotukset on koodattu. Hieman tarkemman kuvan geneettisten algoritmien toiminnasta antanee kuitenkin seuraava alakohta, joka käsittelee algoritmin yksittäisen sukupolven populaation muodostumista ja siihen liittyviä operaatioita.

3.1.2. Populaation muodostaminen

Yleisimmät geneettisissä algoritmeissa käytetyt operaatiot ovat elitismi, risteytys ja mutaatio. Mainittujen operaatioiden hieman yksityiskohtaisempi toiminta kuvataan tässä alakohdassa.

Elitismi on toimenpide, jossa vanhasta sukupolvesta valitaan suoraan tietty määrä parhaita yksilöitä uuteen sukupolveen. Toimenpiteen tarkoituksena on säilyttää populaation parhaat yksilöt myös tulevissa sukupolvissa. Ilman elitismia hyviä yksilöitä voidaan hukata, jolloin algoritmin tulos ei välttämättä kasva niin nopeasti tai nouse niin hyväksi kuin elitismia käyttämällä. Elitismia kannattaa kuitenkin käyttää vain kohtuullisissa määrin, sillä liian suuri eliittiyksilöiden määrä saattaa vähentää populaation diversiteettiä siinä määrin, että algoritmi päättyy loppujen lopuksi johonkin ongelman paikallisista optimiarvoista globaalin optimin sijaan.

Risteytyksen tarkoituksena on muodostaa uusi yksilö käyttäen hyväksi populaation olemassaolevia jäseniä. Populaatiosta valitaan (tavallisesti) kaksi yksilöä, jotka toimivat uuden yksilön vanhempina. Vanhemmat jaetaan ongelmasta riippuen kahteen tai useampaan osaan ja vanhempien osista kootaan uusi jälkeläinen. Risteytystä suoritettaessa tulee olla tarkkana, että muodostettavasta jälkeläisestä rakentuu ongelman sääntöjen mukainen ratkaisuehdotus. Kuva 2 havainnollistaa risteytysoperaation toimintaa. On hyvä huomioda, että kahden vanhemman osista voi muodostaa halutessaan kaksi jälkeläistä yhden sijaan Kuvan 2 osoittamalla tavalla.



Kuva 2. Havainnollistus geneettisen algoritmin risteytysoperaation toiminnasta.

Risteytysoperaation toteutus riippuu siitä, miten algoritmi käsittelee sisäisesti populaation yksilöitä. Usein ratkaisuehdotukset koodataan bittijonoiksi, koska niiden käsittely ohjel-

mallisesti on yksinkertaista. Bittijonot muistuttavat myös suuresti DNA:n koodausta¹, joten tällainen binäärinen esitystapa tuntuu luontevalta. Ongelmien bittikoodaus ei ole aina kuitenkaan välttämättä paras ratkaisu. Monimutkaisempia ongelmia käsiteltäessä bittikoodaus voi olla hankala muodostaa, ja algoritmin tulee joka tapauksessa huolehtia siitä, edustaako luotu bittijono mitään ongelman ratkaisuehdotusta. Merkitään kuitenkin tässä alakohdassa geneettisen algoritmin populaation ratkaisuehdotuksia yhtä pitkillä bittijonoilla niiden selkeyden takia.

Risteytys voidaan käytännössä toteuttaa monella eri tavalla, mutta seuraavana kuvattu menetelmä lienee yksi yleisimmin käytetyistä. Populaatiosta valitaan ensiksi kaksi yksilöä p_1 ja p_2 . Tämän jälkeen arvotaan jokin kokonaisluku $k \in \{1, 2, 3, \dots, n-1\}$, kun $n = |p_1| = |p_2|$. Kokonaisluku k merkitsee, monennenko bitin kohdalta yksilöt p_1 ja p_2 katkaistaan. Uusi yksilö p_3 muodostetaan valitsemalla bittijonosta p_1 k ensimmäistä bittiä, jonka jälkeen uuden bittijonon p_3 loppuun lisätään p_2 :n bitit väliltä $[k+1, n]$. Eli jos $p_1 = 100110101$, $p_2 = 001101011$ ja $k = 6$, niin risteytysoperaation lopputulos on seuraavanlainen:

$$\begin{aligned} p_1 &= 100110 \mid 101 \\ p_2 &= 001101 \mid 011 \\ p_3 &= 100110 \mid 011. \end{aligned}$$

Mikään ei tietenkään estä valitsemasta risteytykseen esimerkiksi kolmea yksilöä, joiden biteistä uusi yksilö muodostetaan. Kahden yksilön tapauksessa bittijonon voisi myös jakaa useampaan kuin kahteen osaan. Geneettiset algoritmit mahdollistavat useita erilaisia variaatioita toteutuksessa, ja muun muassa tämä seikka tekee niistä niin monipuolisia.

Edellä mainittiin, että binäärikoodausta käytettäessä risteytyksessä on huomioitava, että jälkeläisestä muodostuu sääntöjen mukainen ratkaisuehdotus. Sama koskee kuitenkin esimerkiksi tämän tutkielman reunatäsmäyspelejä, jotka on koodattu eräänlaisiksi pelilaudaolioiksi, joiden paloja on helppo käsitellä. Tällöin risteytyksessä uusi yksilö muodostetaan poimimalla vanhemmista sopivalla tavalla paloja. Tällöin täytyy huomioida, että uudessa yksilössä on vain ja ainoastaan yksi kustakin Eternity II:n 256 palasta. Reunatäsmäyspelien risteytyksessä on siis vaarana, että sama pala monistuu jälkeläiseen useammin kuin yhden kerran, jolloin pelilaudasta tulee virheellinen. Käytännössä risteytyksen suunnittelussa täytyy siis ottaa huomioon tietynlaisia rajoitteita ja tehdä erityisiä tarkasteluja oikeellisen lopputuloksen saavuttamiseksi.

Viimeisenä populaation muokkauskeinona käsittelemme mutaatiota. Mutaatio on toimenpide, joka muuntaa hieman populaation satunnaisesti valittuja yksilöitä. Tästä voisi olla esimerkkinä vaikkapa reunatäsmäyspelin yhden satunnaisen palan kääntö tai kahden palan paikan vaihtaminen keskenään. Vaikka mutaation vaikutus ei olekaan aina positiiv-

¹ DNA:n geneettinen koodi muodostuu aakkostosta $\Sigma = \{A, C, G, T\}$, kun $A =$ adenosiini, $C =$ sytosiini, $G =$ guaniini ja $T =$ tymiini.

vinen, on se kuitenkin hyvin tärkeä operaatio algoritmin tehokkaan toiminnan kannalta. Ilman mutaatiota populaation koko potentiaali pysyy samana kaiken aikaa, eikä uusia ominaisuuksia pääse syntymään. Populaation yksilöistä voi muodostaa vain tietyn määrän erilaisia jälkeläisten kombinaatioita, eikä ongelman optimaalinen ratkaisu välttämättä sisälly näihin vaihtoehtoihin.

Mutaatio kohdistetaan yleensä satunnaisesti tietyn ennalta määrätyn todennäköisyyden mukaan populaation yksilöihin. Mutaation tehtävänä ei ole niinkään tehdä yksittäisiä suuria muutoksia populaatioon, jolloin optimaalinen ratkaisu löytyisi saman tien, vaan tarkoituksena on pikemminkin aiheuttaa yksilöissä hiljalleen erilaisia populaation kannalta pienempiä muutoksia, jotka yhdessä muiden geneettisten algoritmien operaatioiden kanssa ajavat sukupolvien kuluessa populaatiota kohti optimaalisempaa ratkaisua.

Sovellettaessa mutaatiota koodattuihin bittijonoihin valitaan yleensä bittijonosta jokin satunnainen indeksi n , jonka sisältämä bitti vaihdetaan. Jos esimerkiksi tarkastelemme jotain tiettyä yksilöä edustavaa bittijonoa $b_1 = 10010110$ ja satunnaisesti valittua indeksia $n = 3$, mutaatio-operaation seurauksena bittijonon b_1 kolmas bitti vaihdetaan nollassa ykköseksi. Tällöin tulokseksi saadaan $b_1 = 10110110$.

Näin pieni mutaatio-operaatio saattaa vaikuttaa merkityksettömältä muutokselta, mutta vaikutus riippuu täysin ongelman koodauksesta. Jos esimerkiksi jonkin tietyn ongelman ratkaisuehdotukset olisivat binääriluvuiksi koodattuja kokonaislukuja, bittijonon $b_2 = 00000010110$ ensimmäisen bitin kääntö muuttaisi ratkaisuehdotuksen arvon kokonaisluvusta 22 lukuun 1046.

Yhden bitin muuttamisen sijaan mutaatio voidaan tehdä tarvittaessa myös jollain muulla tapaa, esimerkiksi n :n ensimmäisen bitin tilaa voidaan vaihtaa yhden bitin sijasta. Kuten risteytysoperaationkin toteutuksessa, mutaatio-operaation variointikeinoja rajoittaa lähinnä käyttäjän mielikuvitus. Myös mutaatiota tehdessä on syytä olla tarkkana siitä, että muodostuva yksilö täyttää ongelman ratkaisuehdotuksen rakennevaatimukset.

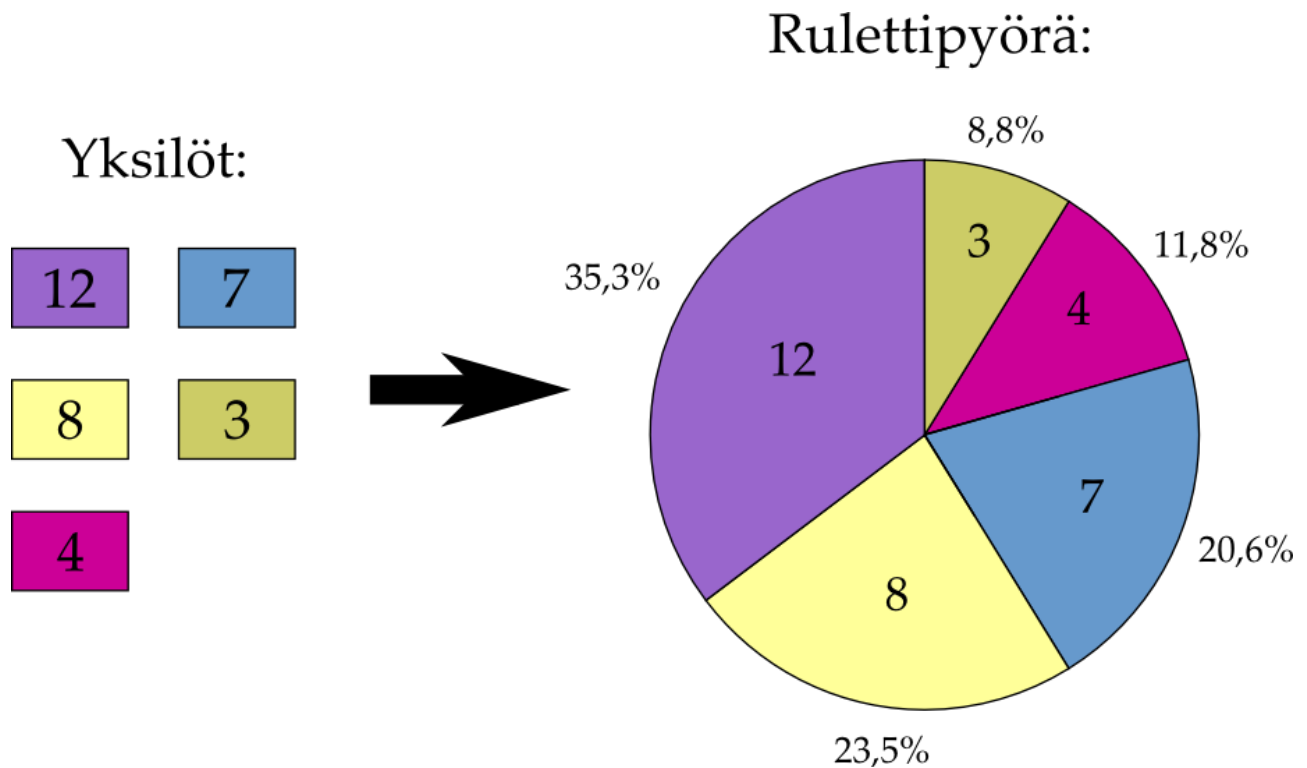
Eräs kokonaisuutena pieni, mutta silti tärkeä osio populaation muodostamisesta on yksilöiden valinta. Valintaa käytetään muun muassa silloin, kun populaation yksilöitä poimitaan risteytystä varten. Toisinaan riippuen algoritmin toteutusyksityiskohdista valinta-operaatiota saatetaan käyttää myös valittaessa yksilöä tai yksilöitä mutaatio-operaatioon. Ei ole lainkaan samantekevää, millä kriteerein yksilöitä populaatiosta valitaan. Jos nimitäin yksilöt valittaisiin populaatiosta satunnaisesti, ei tietoa populaation parhaista yksilöistä hyödynnettäisi tällöin mitenkään. Edellä mainitun kaltaista satunnaisvalintaa käyttämällä geneettisen algoritmin suorituskyky putoaa dramaattisesti, kuten kohdassa 5.2 myöhemmin huomataankin.

Geneettisten algoritmien toteutusyksityiskohtien monipuolisuus koskee myös valinta-operaatiota, sillä myös sille voidaan tehdä lukuisia erilaisia toteutuksia. Kaksi yleisintä valintatapaa lienee kuitenkin rulettivalinta ja turnajaisvalinta.

Rulettivalinnassa lasketaan aluksi jokaisen populaation yksilön sopivuusfunktion arvo, joka jaetaan kaikkien sopivuusfunktion arvojen summalla. Saatu arvo on yksilön normalisoitu sopivuus, ja populaation kaikkien normalisoitujen sopivuuksien summa on tällöin yksi. Tämän jälkeen voidaan kuvitella, että kaikki populaation yksilöt asetetaan rulettipyörän kehälle siten, että kukin yksilö saa kehältä prosentuaalisesti yhtä suuren sektorin kuin yksilön normalisoitu sopivuus on. Tällöin kehä tulee jaettua koko populaatiolle normalisoitujen sopivuuksien suhteessa. Nyt valitaan satunnaisesti jokin kehän piste ja poimitaan se yksilö, jonka sektoriin valittu piste kuuluu. Tällä tavoin populaation paremmat yksilöt tulevat valituiksi huonompia yksilöitä todennäköisemmin. Kuva 3 havainnollistaa rulettivalinnan toimintaa graafisesti.

Turnajaisvalinnassa puolestaan kiinnitetään ensin turnajaisten koko T , jonka jälkeen poimitaan populaatiosta satunnaisesti T yksilöä. Tämän jälkeen lasketaan kunkin poimitun yksilön sopivuus ja "järjestetään turnajaiset", joissa voittaja valitaan sopivuusarvon perusteella. Poimittuja yksilöitä siis verrataan keskenään ja lopuksi valitaan yksilö, jolla on paras sopivuus.

Rulettivalinta ja turnajaisvalinta suosivat siis molemmat vahvoja yksilöitä heikompien sijaan. Vahvempien yksilöiden on siis todennäköisempää saada ominaisuuksiaan jälkipolville risteytyksen ja mutaation avulla. Tämän takia ruletti- ja turnajaisvalinta muistuttavat läheisesti biologista luonnonvalintaa, joka tarkoittaa sitä, että myös luonnossa olosuhteisiin paremmin sopeutuneet yksilöt pärjäävät paremmin ja saavat huonommin sopeutuneita yksilöitä todennäköisemmin jälkikasvua, jolloin paremmin sopeutuneiden yksilöiden geenit dominoivat heikkoja geenejä populaatiossa.



Kuva 3. Rulettivalinnan toiminta. Numerot alueiden sisässä kuvaavat yksilön sopivuutta.

Edellä on kuvattu yleisimmät geneettisen algoritmin populaatioon kohdistettavat toimenpiteet. Näiden operaatioiden avulla voidaan muodostaa populaatiosta uusi sukupolvi, joka edelleen järjestetään yksilöittäin paremmuusjärjestykseen ja josta muodostuu jälleen uusi sukupolvi. Näin jatketaan, kunnes sopivuusfunktio on saavuttanut tavoitellun optimaarvon tai kun algoritmin suoritus on kestänyt tarpeeksi kauan. Uuden sukupolven populaation muodostaminen edellä mainituin keinoin tapahtuu yleensä kolmessa vaiheessa:

1. Valitaan uuteen sukupolven parhaiten pärjänneet yksilöt elitismillä.
2. Muodostetaan osa tai kaikki loput uuden sukupolven yksilöistä risteytyksen avulla.
3. Riippuen edellisen askeleen toteutuksesta aiheutetaan mutaatioita satunnaisille populaation yksilöille, tai muodostetaan loput populaatiosta mutaation avulla vanhasta sukupolvesta.

Käyttäjä voi siis itse päättää, valitaanko mutaatioon osallistuvat yksilöt suoraan vanhasta populaatiosta, vai muodostetaanko uuden populaation yksilöt ensin risteytyksellä, jonka jälkeen mutaatio-operaatio kohdistetaan koko populaatioon. Tavallisesti mutaatio-operaatiolle asetetaan jokin todennäköisyys, jolla populaation tiettyä yksilöä tai bittiä mutatoidaan.

Tämän tutkielman evolutiivisissa algoritmeissa populaation muodostaminen toteutetaan siten, että uuteen sukupolveen valitaan ensiksi haluttu määrä parhaiten pärjänneitä yksilöitä elitismillä. Tämän jälkeen muodostetaan käyttäjän valitsema määrä yksilöitä risteytysoperaation avulla. Loput populaation yksilöistä muodostetaan käyttäen mutaatiota. Mutatoituvat yksilöt valitaan turnajaisvalinnan avulla vanhasta populaatiosta. Tarkempaa tietoa tässä tutkielmassa käytetyistä algoritmeista löytyy luvusta 4.

3.2. Monitavoitteiset evolutiiviset algoritmit

Monitavoitteiset evolutiiviset algoritmit pohjautuvat jo nimensäkin mukaisesti geneettisten algoritmien tavoin evolutiivisiin algoritmeihin. Tämänkaltaisten algoritmien toimintaperiaate ei ainakaan tämän tutkielman osalta eroa oleellisesti geneettisten algoritmien toiminnasta. Tämän kohdan oleellisin uusi lisäys verrattuna geneettisiin algoritmeihin onkin monitavoitteisuus.

Geneettisten algoritmien rakenteen takia populaation yksilöillä voi olla vain yksi tavoite, jonka pistemäärä vaikuttaa yksilön hyvyyteen. Kuten edellisessä kohdassa mainittiin, geneettistä algoritmia käytettäessä useampiakin tavoitteita voidaan kyllä käyttää, mutta lopulta erillisten tavoitteiden pistemäärät tulee jollain keinolla yhdistää yhdeksi suureksi esimerkiksi painotetun keskiarvon avulla. Usein ongelman eri tavoitteet voivat kuitenkin olla toistensa kanssa ristiriitaisia, jolloin yksittäisen suureen muodostaminen voi olla vaikeaa. Monitavoitteiset evolutiiviset algoritmit pyrkivät ratkaisemaan tämän ongelman pisteyttämällä usempaa tavoitetta käyttävät populaation yksilöt monipuolisemmin kuin geneettiset algoritmit.

Geneettisten algoritmien tavoin myös monitavoitteisten evolutiivisten algoritmien luokka on melko löyhästi määritelty, ja erilaisia algoritmien toteutustapoja onkin paljon. Abraham ja muut [2005] luettelevat ensimmäisen sukupolven monitavoitteisiksi evolutiivisiksi algoritmeiksi akronyymit NSGA, NPGA ja MOGA. Uudempaa tuotantoa olevia, toisen sukupolven algoritmeja ovat muun muassa SPEA, SPEA2, PAES, NSGA-II, NPGA 2 ja PESA. Näillä kaikilla algoritmeilla on jokin erityispiirteensä, joka erottaa ne muista monitavoitteisista evolutiivisista algoritmeista. Koska erilaisten algoritmien kirjo on laaja, keskitytään nyt Muñozin ja muiden [2009] kehittämän monitavoitteisen evolutiivisen algoritmin (l. MOEA, *Multi-Objective Evolutionary Algorithm*) taustoihin. MOEA perustuu yllämainittuun NSGA-II -algoritmiin. MOEA:n yksityiskohtaisempi toiminta kuvataan tarkemmin luvussa 4.

Koska MOEA hyödyntää suurelta osin tavanomaisesta geneettisestä algoritmista tuttuja käytäntöjä, keskitytään tässä lähinnä monitavoitteisuuden hallintaan, joka käytännössä tarkoittaa uudenlaista yksilöiden pisteytystä verrattuna geneettisiin algoritmeihin.

Populaation yksilön A sanotaan dominoivan toista yksilöä B , jos A :n kaikkien tavoitteiden pistemäärät ovat vähintään yhtä suuria kuin B :n vastaavien tavoitteiden pistemää-

rät, ja tämän lisäksi ainakin yksi A :n tavoitteista on aidosti suurempi kuin vastaava B :n tavoite. Yksilö on puolestaan dominoimaton, jos mikään muu yksilö ei dominoi sitä.

Geneettisten algoritmien tavoin myös MOEA laskee ongelman kaikille tavoitteille oman pistearvonsa, mutta tavoitteiden pistemääriä ei vain yhdistetä suoraviivaisesti sopivuusfunktioiksi geneettisten algoritmien tavoin. Sen sijaan MOEA laskee populaation jokaiselle yksilölle tavoitteiden pistemäärien perusteella dominanssiarvon, joka tarkoittaa niiden yksilöiden lukumäärää, jotka tutkittavaa yksilöä dominoivat.

Jos yksilön dominanssiarvo on nolla, tarkoittaa se, että kyseinen yksilö on dominoimaton. Populaation dominoimattomien yksilöiden sanotaan kuuluvan niin sanottuun Pareto-rintamaan. Myöhemmin luvussa 4 nähdään, että MOEA käyttää sellaisia tavoitteita, joissa reunatäsmäyspelin päätavoite (eli mahdollisimman monen palan sijoittaminen oikein) on jaettu ikään kuin samantyyliisiin aliongelmiin, joiden optimaaliset ratkaisut eivät ole ristiriidassa keskenään. Täsmällisemmin Pareto-rintamaan kuuluminen tarkoittaa sitä, että yksilö on dominoimaton koko ratkaisuavaruuden suhteen [Sbalzarini et al., 2000]. Tällöin MOEA:n tapauksessa täsmällinen määritelmä tarkoittaisi sitä, että Pareto-rintamaan kuuluisi vain täydellisesti ratkaistut pelilaudat. Tämän takia tässä tutkielmassa määritellään, että yksilö kuuluu Pareto-rintamaan jos ja vain jos se on dominoimaton algoritmin koko populaation suhteen.

MOEA laskee Pareto-rintaman selvittämisen jälkeen rintaman jokaiselle yksilölle etäisyysarvon. Tarkasteltavan yksilön kunkin tavoitteen pistemäärää verrataan muiden Pareto-rintaman yksilöiden vastaaviin pistemääriin, ja aina kun havaitaan tavoitteiden pistemäärien eroavan toisistaan, yksilön etäisyysarvoa kasvatetaan yhdellä yksiköllä.

Lopulta populaation yksilöt asetetaan järjestykseen siten, että Pareto-rintaman yksilöt arvotetaan paremmiksi kuin muut populaation yksilöt. Pareto-rintaman sisällä yksilöt järjestetään etäisyysarvon mukaiseen järjestykseen siten, että se yksilö, jolla on suurin etäisyysarvo, on paras. Muut populaation yksilöt järjestetään puolestaan dominanssiarvon mukaiseen järjestykseen siten, että yksilö on sitä parempi, mitä pienempi sen dominanssiarvo on. Pareto-rintaman muista yksilöistä eniten eroavia yksilöitä painotetaan toisia enemmän, jotta populaation diversiteetti pysyisi suurempana, ja näin ollen algoritmin tulos ei konvergoituisi liian aikaisin [Muñoz et al., 2009]. Muñozin ja muiden MOEA ei siis käytä missään vaiheessa yksilöiden tavoitteiden arvoja suoraan (paitsi parhaan tuloksen tarkistuksessa), vaan ainoastaan sillä on merkitystä, miten tavoitteiden arvot vertautuvat muiden yksilöiden tavoitteiden arvoihin.

Kun populaatio on saatu haluttuun paremmuusjärjestykseen, MOEA:n toiminta jatkuu samanlaisena kuin edellisessä kohdassa kuvailtu geneettisen algoritmin toiminta.

3.3. Hill climbing -algoritmit

Hill climbing -tekniikkaan perustuvat algoritmit ovat paikalliseen etsintään perustuvia algoritmeja. Hill climbing -algoritmit ovat usein helppoja toteuttaa, sillä niiden toiminta on hyvin suoraviivaista.

Hill climbing -algoritmin suoritus alkaa käsiteltävän ongelman jonkin ratkaisuehdotuksen konstruoinnista. Usein ratkaisuehdotus luodaan satunnaisesti ongelman rajoitukset huomioiden, mutta ratkaisuehdotus voi olla aivan hyvin valittu jollain muullakin tavalla.

Tämän jälkeen ryhdytään suorittamaan algoritmin pääsilmuksia. Jokaisella kierroksella ongelman ainoaan ratkaisuehdotukseen tehdään jokin paikallinen muutos. Muutos on tavallisesti sellainen, joka parantaa mahdollisimman paljon ratkaisuehdotuksen laatua yhdellä kertaa. Tällöin algoritmi toimii ahneella periaatteella. Kaikissa ongelmissa ei ole itsestään selvää, mikä muutos kullakin hetkellä on optimaalisin. Tämä voi johtua esimerkiksi siitä, että muutos tulee tehdä ennen kuin sen vaikutus selviää. Tai sitten mahdollisia muutoksia voi olla niin paljon, ettei ole tehokkuuden kannalta järkevää tutkia parasta mahdollista vaihtoehtoa jokaisella kierroksella. Tällöin algoritmi voi tehdä muutoksen satunnaisesti. Satunnaista muutosta tehdessä algoritmi voi tehdä mahdollisesti myös epäoptimaalisen siirron. Tällöin muutosta ei hyväksytä, vaan uusia muutoksia tehdään kunnes parempi ratkaisuehdotus löytyy.

Jos joka kierroksella on selvää, mikä paikallinen muutos algoritmin tulisi tehdä, voidaan algoritmin suoritusta vain jatkaa niin kauan kunnes mikään paikallinen muutos ei enää paranna ratkaisua. Jos taas paikallinen muutos tehdään satunnaisesti, ei voida etukäteen tietää, tuleeko ratkaisu enää parantumaan useankaan muutoksen jälkeen. Tällöin on syytä asettaa algoritmille jokin raja, jota kauemmin algoritmia ei suoriteta jos parempaa ratkaisua ei siihen mennessä löydy.

Hill climbing -algoritmit toimivat usein nopeasti ja niillä voi löytyä käsiteltävään ongelmaan hyviäkin ratkaisuja. Kuitenkin, mitä enemmän paikallisia ääriarvoja ongelmalla on, sitä todennäköisemmin hill climbing -algoritmi jumittuu johonkin tällaiseen ääriarvoon löytämättä lainkaan optimaalista ratkaisua. Tämän takia hill climbing -algoritmeilla löytyy tuskin koskaan optimiratkaisua NP-täydellisille ongelmille.

Koska hill climbing -algoritmit voivat kuitenkin saavuttaa nopeasti kohtuullisen hyvän tuloksen, voidaan niitä käyttää muiden algoritmien alustuksessa. Tällöin toisen algoritmin alustuksessa käytetty hill climbing -algoritmin esiratkaisema ongelma saattaa lyhentää toisen algoritmin suoritusaikaa tai tulos saattaa parantua verrattuna tilanteeseen, jossa käytettäisiin satunnaisesti luotua ratkaisuehdotusta.

3.4. Simuloitu jäähdytys

Simuloituun jäähdytykseen perustuvat algoritmit ovat optimointialgoritmeja, joiden toiminta jäljittelee metalliteollisuuden käyttämää tekniikkaa, jossa metalli ensin kuumennetaan korkeaan lämpötilaan, jonka jälkeen lämpötilaa lasketaan hitaasti siten, että metallin rakenteesta tulisi alkuperäistä kestävämpää materiaalin atomien järjestyessä kestävyys kannalta optimaalisempaan järjestykseen. Simuloitu jäähdytys kehitettiin alun perin vuonna 1953 Metropoliksen ja muiden [1953] toimesta. Myöhemmin myös Kirkpatrick ja muut [1983] sekä Černý [1985] esittelivät vastaavanlaiset tekniikat.

Simuloidun jäähdytyksen toiminta muistuttaa hill climbing -algoritmin toimintaa, mutta simuloitua jäähdytystä käyttävä algoritmi voi tehdä myös epäoptimaalisia siirtoja, joten tällainen algoritmi ei jää niin helposti jumiin paikallisiin ääriarvoihin kuin hill climbing -algoritmit.

Simuloituun jäähdytykseen perustuvan algoritmin toiminnallisuus perustuu fysikaalisen esikuvansa mukaan lämpötilaan T ja sen kontrolloituun laskuun. Kantavana ideana on se, että kun algoritmin suoritus alkaa, lämpötila T saa jonkun käsiteltävästä ongelmasta riippuvan korkeahkon arvon. Tämän jälkeen käsiteltävän ongelman ratkaisuehdotukseen aletaan tehdä paikallisia muutoksia samalla tyylillä kuten hill climbing -algoritmissa. Edelleen hill climbing -algoritmia mukaillen muutos hyväksytään, mikäli se kasvattaa ratkaisuehdotuksen hyvyyttä.

Jos paikallinen muutos ei paranna ratkaisuehdotuksen hyvyyttä, ei tehtyä muutosta hylätä suoraan. Hylkäämisen sijaan muutos saatetaan myös hyväksyä tietyllä todennäköisyydellä. Muutoksen hyväksymistodennäköisyys on funktion $P(\Delta, T) = e^{(-\Delta)/T}$ arvo. Funktiossa Δ tarkoittaa paikallisen muutoksen aiheuttamaa ratkaisuehdotuksen hyvyyden muutosta ja T puolestaan senhetkistä lämpötilaa. Jos algoritmilla yritetään ratkaista maksimointiongelmaa, funktion $P(\Delta, T)$ eksponenttifunktio kirjoitetaan muodossa $e^{\Delta/T}$. Minimointiongelman tapauksessa eksponenttifunktio on vastaavasti $e^{-\Delta/T}$.

Funktiota $P(\Delta, T)$ tarkastelemalla havaitaan, että algoritmin suorituksen alkuvaiheilla lämpötila T on suhteessa paljon suurempi hyvyyden muutosta kuvaavaan Δ :n verrattuna. Tällöin funktion arvo on lähellä yhtä. Tämä siis tarkoittaa, että ratkaisuehdotusta huonontavat muutokset otetaan aluksi käyttöön melko suurella todennäköisyydellä. Lämpötilan T lasiessa algoritmin suorituksen aikana myös ratkaisuehdotusta huonontavien muutosten hyväksymistodennäköisyys laskee, sillä lämpötila lähenee Δ :n arvoa. Algoritmin loppuvaiheilla lämpötilan arvo on jo hyvin pieni suhteessa hyvyyden muutoksen arvoon, jolloin funktio $P(\Delta, T)$ tuottaa myös hyvin pieniä todennäköisyyksiä. Käytännössä lämpötilan jäähtyminen vaikuttaa algoritmin toimintaan siten, että algoritmin suorituksen alussa ratkaisuehdotus poukkoilee hyvin paljon eri tilojen välillä, sillä algoritmi ei tässä vaiheessa pyri vain sokeasti parempaa tulosta kohden hill climbing -algoritmin tavoin. Mitä enemmän lämpötila laskee, sitä harvemmin ratkaisuehdotus enää vaihtaa tilaansa

radikaalisti. Lämpötilan edelleen lasiessa ratkaisuehdotus alkaa asettumaan tietyn tilan läheisyyteen, ja algoritmin toiminta alkaa loppua kohden muistuttaa entistä enemmän hill climbing -algoritmia. Algoritmin toiminta päättyy lopulta kun lämpötila saavuttaa sille asetetun raja-arvon ε . Tavallisesti lämpötilan jäähtymiskertoimena käytetään jotain vakiota α .

Simuloituun jäähtymiseen perustuvan algoritmin toimintaa voidaan ohjailla algoritmin alku- ja loppulämpötilojen T_0 ja ε , sekä jäähtymiskertoimen α arvoilla. Lisäksi lämpötilan jäähtymisfunktio vaikuttaa algoritmin toimintaan. Yksi tavanomainen jäähtymisfunktion toteutustapa on eksponentiaaliseen vähenemiseen perustuva jäähtyminen. Tällaista jäähtymisfunktiota käytettäessä algoritmin lämpötila kerrotaan jokaisen pääsilmuksen kierroksen päätteeksi edellä mainitulla nollan ja yhden väliin sijoittuvalla vakiolla α , joskin yleensä kyseinen vakio on melko lähellä lukua yksi. Eksponentiaalinen jäähtymisfunktio pudottaa lämpötilaa aluksi melko nopeasti, mutta jäähtyminen hidastuu sitä mukaa, mitä kauemmin algoritmin suoritus kestää. Eksponentiaalisen jäähtymisen sijaan voidaan käyttää esimerkiksi lineaarista jäähtymisfunktiota, tai jäähtymisfunktio voidaan suunnitella yksilökohtaisesti ratkaistavan ongelman perusteella.

4. Algoritmien esittely

Tässä luvussa käydään tutkielmassa käytettyjen algoritmien rakenne tarkemmin läpi ja algoritmien toiminta esitellään melko yksityiskohtaisesti. Aluksi käsitellään algoritmien yhteiset ominaisuudet ja yleinen rakenne, jonka jälkeen käydään vuorotellen läpi jokainen neljästä tässä tutkielmassa käytetystä algoritmista.

4.1. Algoritmien yleinen rakenne

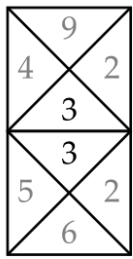
Jokainen tämän tutkielman algoritmeista on rakennettu tutkielmaa varten suunnitellun ohjelmarungon päälle. Tämä ohjelmarunko käynnistyy ennen itse algoritmin suorittamista. Ohjelmarunko päättää aluksi saamistaan parametreista, mitä algoritmia on tarkoitus käyttää ja missä tiedostossa kyseisen algoritmin parametrit määritellään.

Ohjelmarunko käynnistää halutun algoritmin ja toimittaa tälle tiedon algoritmin parametrien sijainnista. Valittu algoritmi lataa käyttäjän syöttämät parametrit tiedostosta ohjelmarungon toiminnallisuuden avulla. Algoritmi alustetaan syötetyin parametrein ja algoritmin toiminta käynnistyy. Algoritmi raportoi suorituksensa aikana pistemäärän kehitymisestä. Suorituksen loputtua ohjelmarunko tallentaa algoritmin tulosteet erilliseen tulostiedostoon.

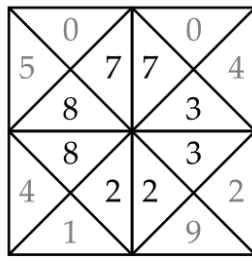
Läheisimmin algoritmien toimintaan liittyvä ohjelmarungon osa on algoritmin käyttämän pelilaudan (tai useiden pelilautojen) pisteiden laskeminen. Ohjelmarunkoon kuuluva pistelaskuri laskee pelilaudalle parametreista riippuen 1–4 erilaista pistemäärää. Jokainen näistä pistemääristä liittyy erilaiseen tavoitteeseen. Erilaisten tavoitteiden käsite reunatäsmäyspeleillä on tässä tutkielmassa peräisin Muñozin ja muiden [2009] julkaisusta, jossa tekijät esittelivät kolme erilaista pisteenlaskutapaa reunatäsmäyspeleille. Muñozin ja muiden kehittelemän kolmen pistelaskun tavoitteen lisäksi kehitin itse myös yhden lisätavoitteen. Seuraavaksi kuvataan tarkemmin edellä minitut neljä tavoitetta.

Ensimmäinen tavoite (Tavoite 1) käsittää tutkittavalta pelilaudalta sellaiset vierekkäiset reunaparit, joilla on sama tunniste. Eternity II:n virallinen pisteidenlasku perustuu Tavoitteen 1 pistemäärään. Toinen tavoite (Tavoite 2) tarkoittaa pelilaudan kaikkia sellaisia 2×2 palan kokonaisuuksia, joissa kyseiseen kokonaisuuteen kuuluvien vierekkäisten palojen vastinsivut ovat samanlaiset. Kolmas tavoite (Tavoite 3) puolestaan tarkoittaa pelilaudan sellaisten palojen määrää, joiden jokaista sivua vastaa samanlainen vastinsivu. Samanlainen vastinsivu ei tarkoita tässä sitä, että jokainen neljästä sivusta olisi samanlainen. Kolme edellä mainittua tavoitetta ovat siis Muñozin ja muiden kehittämiä. Neljäs ja viimeinen tavoite (Tavoite 4) on itse määrittelemäni ja tämä tavoite huomioi sellaiset pelilaudan palojen sivut, jotka on sijoitettu pelilaudan reunaan tai kulmaan ja joiden tunnisteena on käytetty reunatunnistetta. Reunatunnisteena käytetään tässä tutkielmassa numeroa 0. Kuva 4 havainnollistaa pistelaskun tavoitteita.

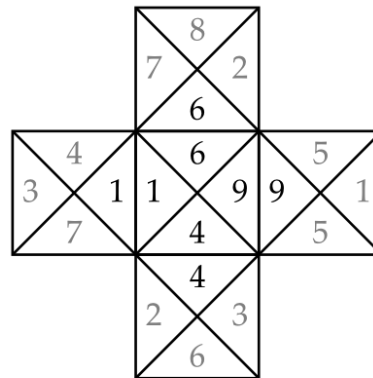
Tavoite 1



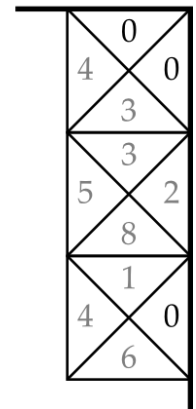
Tavoite 2



Tavoite 3



Tavoite 4



Kuva 4. Reunatäsmäysohjelman pistelaskun tavoitteiden havainnollistusta.

Kuvan 4 Tavoitteiden 1–3 mukaiset konstruktiot tuottaisivat yhden pisteen kullekin tavoitteelle, mutta Tavoitteen 4 mukainen konstruktiio tuottaisi Tavoitteelle 4 kolme pistettä, sillä konstruktiiossa on kolme reunatunnistetta pelilaudan ulkoreunalla.

Tavoitteen 1 maksimipistemäärä on 480, sillä Eternity II -pelilaudalla on $x \cdot (y - 1)$ vaakasuuntaista vierekkäistä reunaparia, ja vastaavasti pystysuuntaisia vierekkäisiä reunapareja on $y \cdot (x - 1)$ kappaletta, kun x on pelilaudan leveys ja y puolestaan pelilaudan korkeus. Tällöin vierekkäisten reunaparien kokonaismäärä on $16 \cdot 15 + 16 \cdot 15 = 480$ kappaletta. Pelilaudan paloista voi puolestaan vaakasuunnassa muodostaa $x - 1$ kappaletta 2×2 -kokoisia palakokonaisuuksia. Pystysuunnassa vastaavia kokonaisuuksia voi muodostaa $y - 1$ kappaletta, joten Tavoitteen 2 maksimipistemäärä on $(16 - 1) \cdot (16 - 1) = 225$ pistettä. Koska pelilaudalla on 256 palaa, on Tavoitteen 3 maksimipistemäärä myös 256 pistettä. Tavoitteen 4 suurin pistemäärä on sama kuin pelilaudan ulkosyrjällä olevien sivujen määrä. Koska Eternity II:n pelilauta on neliön muotoinen ja yhden sivun pituus on 16 palan mittainen, on Tavoitteen 4 maksimipistemäärä $4 \cdot 16 = 64$ pistettä.

Jokaiselle aktiiviselle tavoitteelle lasketaan nollan ja yhden välille sijoittuva normalisoitu pistemäärä, ja lopuksi normalisoiduista pistemääristä lasketaan algoritmista riippuen joko keskiarvo, painotettu keskiarvo tai algoritmi voi käyttää pistemääriä myös sellaisenaan. Saatuja pistemääriä käytetään algoritmien sisäisessä toiminnassa erilaisten pelilautojen keskinäisen paremmuuden määrittämiseen. Eli mitä lähempänä pistemäärä on ykköstä keskiarvotettuja pisteitä käytettäessä, sitä parempi pelilauta on kyseessä.

Ohjelmarunko vastaa myös algoritmien käyttämistä satunnaisluvuista. Monipuoliset satunnaisluvut ovat tärkeitä algoritmien suorituksessa, sillä algoritmien toiminta perustuu suurelta osin satunnaisuuteen ja satunnaislukuja täytyy täten generoida melko paljon. Tästä seuraa, että satunnaislukugeneraattorin jakson tulee olla melko pitkä. Valitsinkin

tämän takia ohjelmarunkoon tunnetun, laadukkaita pseudosatunnaislukuja generoivan algoritmin nimeltään *Mersenne Twister* [Matsumoto and Nishimura, 1998]. Mersenne Twisterin jakson pituus $2^{19937} - 1$ riittää varsin hyvin tämän ohjelman tarpeisiin. Satunnaislukugeneraattori alustetaan siemenluvulla, joka muodostetaan millisekunnin tarkkuudella kulloisestakin ajanhetkestä. Käyttäjä voi halutessaan syöttää algoritmin parametriksi myös vapaavalintaisen siemenluvun, jolla ohjelmarungon satunnaislukugeneraattori alustetaan silloisen ajanhetken sijaan.

Ohjelmarungon tehtäviin kuuluu myös algoritmien käyttämien pelilautojen sekoittaminen. Kun algoritmi käynnistetään, Eternity II:n palat ladataan ohjelmarungon muistiin. Tässä vaiheessa palat ovat aina samassa järjestyksessä, joten ne täytyy luonnollisesti sekoittaa.

Sekoittamisen lisäksi pelilaudat voidaan myös käyttäjän niin halutessa esiratkaisusta. Esiratkaiseminen tarkoittaa sellaista toimenpidettä, jossa pelilaudan reunat ratkaistaan siten, että jokainen pelilaudan reunalle tuleva pala on niin sanotusti oikein sijoitettu. Esiratkaisu suoritetaan raajan voiman tekniikalla hyödyntäen peruutusta, jos ratkaisu päätyy umpikujaan. Ratkaiseminen aloitetaan aina satunnaisesta reunapalasta, jotta eri pelilaudoille tulisi vähemmän todennäköisesti samanlaisia ratkaisuja. Huomattakoon tässä, että Eternity II:n reunat voidaan ratkaista monella eri tapaa. Kun pelilaudan reunat on ratkaistu, ladotaan jäljelle jääneet "keskipalat" reunapalojen sisäpuolelle kiinnittämättä näiden palojen järjestykseen erityistä huomiota.

Jos algoritmin toiminta vaatii usean eri pelilaudan järjestämistä paremmuusjärjestykseen, hoitaa ohjelmarunko tämänkin toimenpiteen. Ennen järjestämistä pelilautojen pistemäärät tulee luonnollisesti laskea, ja tämän jälkeen pelilaudat voidaan toimittaa ohjelmarungolle, joka järjestää pelilaudat paremmuusjärjestykseen käyttäen tavanomaista pikalajittelua.

4.2. Algoritmi 1 (geneettinen algoritmi)

Ensimmäiseksi käydään läpi Muñozin ja muiden [2009] kehittämään geneettiseen algoritmiin perustuva algoritmi, jota kutsutaan tästedes nimellä Algoritmi 1. Algoritmin 1 toiminnallisuus on muuten samanlainen kuin Muñozilla ja muilla, mutta algoritmiin on lisätty edellisessä kohdassa esitelty Tavoite 4, ja pelilaudan esiratkaisu toimii hieman eri tavalla kuin Muñozin ja muiden versiossa.

Algoritmin 1 suoritus alkaa alustuksella, jossa ladataan käyttäjän määrittelemät parametrit, luodaan populaatio ja tarvittaessa esiratkaisusta luodut yksilöt. Populaation koko määräytyy sen mukaan, minkälaisia parametreja käyttäjä on algoritmille antanut syötteenä. Populaatio muodostuu eliittiyksilöistä, risteytysyksilöistä sekä mutaatioyksilöistä. Käyttäjän tulee määritellä jokaisen edellä mainitun kategorian yksilöiden lukumäärät. Alustuksen jälkeen algoritmin varsinainen suoritus alkaa. Algoritmin 1 suoritus vastaa

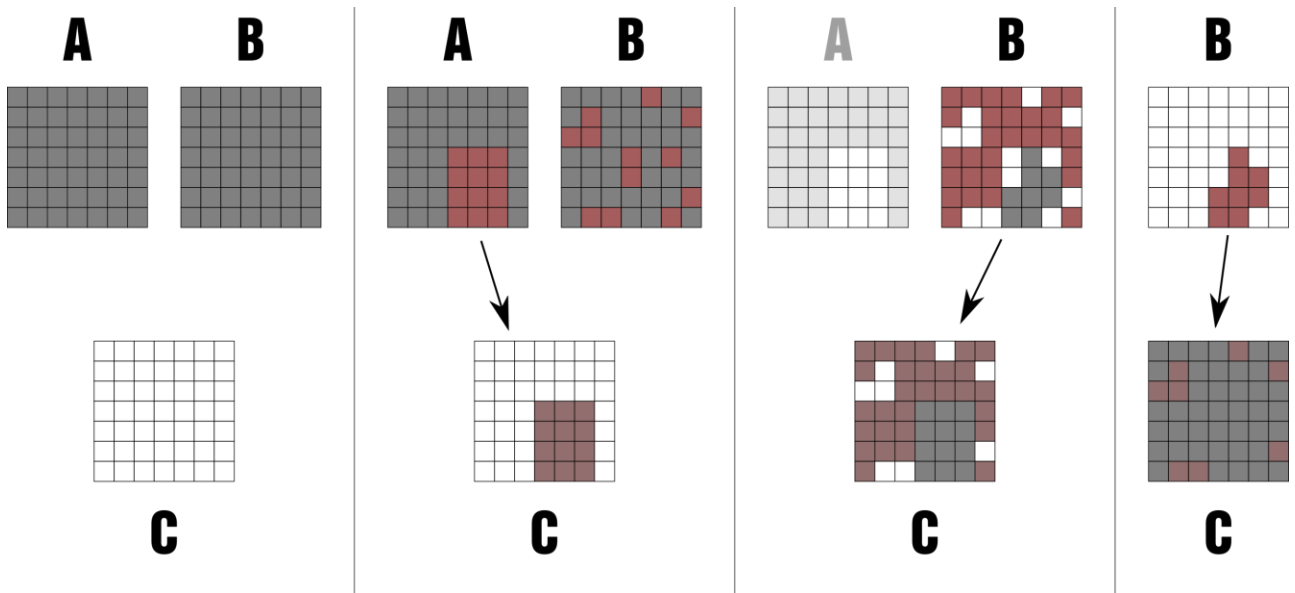
hyvin pitkälti Listauksen 1 kulkua. Käydään tässä vielä kuitenkin algoritmin pääsilmutan yhden kierroksen kulku tarkemmin läpi.

Joka kierroksen alussa lasketaan populaation yksilöiden pistemäärät. Pistemäärien laskeminen tapahtuu edellisessä kohdassa selitetyllä tavalla ohjelmarungossa. Algoritmin suoritus voi olla juuri alkanut, jolloin populaation alkupisteet täytyy selvittää, tai sitten populaatio on epäjärjestyksessä edellisen kierroksen muokkausten jäljiltä, jos algoritmin suoritus on kestänyt jo pidempään. Kun populaation yksilöiden pistemäärät on selvillä, täytyy populaation yksilöt asettaa paremmuusjärjestykseen. Tässä tehtävässä auttaa ohjelmarungon pikalajittelu, joka järjestää populaation jäsenet pistemäärän mukaan laskevaan järjestykseen.

Kun populaatio on järjestetty pisteiden mukaiseen laskevaan järjestykseen, tarkistetaan populaation parhaan yksilön pistemäärä. Jos pistemäärä saavuttaa tavoitellun maksimimäärän, lopetetaan algoritmin suoritus. Muussa tapauksessa algoritmin suoritus jatkuu.

Parhaan yksilön tarkastamisen jälkeen populaatiosta poimitaan käyttäjän valitsema määrä parhaimpia yksilöitä suoraan seuraavaan sukupolveen. Käyttäjä voi myös päättää, että elitismioperaatiota ei käytetä lainkaan asettamalla eliittiyksilöiden määräksi nollan.

Elitismioperaation suorittamisen jälkeen uuden sukupolven alkioita aletaan muodostaa risteytysoperaation avulla. Kuten aikaisemmin mainittiin, myös risteytysyksilöiden määrä on kiinnitetty algoritmin parametreissa. Yksittäisen risteytysyksilön muodostaminen aloitetaan valitsemalla tulevalle yksilölle kaksi vanhempaa. Vanhempien valinta tapahtuu turnajaisvalinnalla, jossa kumpikin vanhempi valitaan omilla turnajaisilla. Käyttäjän tulee määrittellä algoritmin parametreissa turnajaisiin osallistuvien yksilöiden määrä. Tämän tutkielman testeissä on käytetty kolmen yksilön turnajaisia, ellei toisin ole mainittu. Kun vanhemmat on valittu, valitaan pelilaudalta satunnainen suorakulmion muotoinen alue. Valittavan suorakulmion minimi- ja maksimimitat määritellään algoritmin parametreissa. Tämän jälkeen poimitaan ensimmäisestä vanhemmasta äsken muodostetun suorakulmion sisään jäävät palat ja siirretään ne uuden yksilön pelilaudalle vastaaviin kohtiin. Poistetaan nyt siirretyt palat toisen vanhemman laudalta, jotta uudelle pelilaudalle tulisi ainoastaan yksi kappale kutakin palaa. Tämän jälkeen poimitaan toisesta vanhemmasta suorakulmion ulkopuolelle jäävät palat ja asetetaan ne samassa järjestyksessä uudelle pelilaudalle. Uudelta laudalta puuttuvat vielä toisen vanhemman suorakulmion sisään jääneet palat. Nämä palat sekoitetaan ja asetetaan satunnaiseen asentoon uuden pelilaudan vapaisiin kohtiin. Kuva 5 selventää risteytysoperaation toimintaa.



Kuva 5. Algoritmin 1 käyttämä risteytysoperaatio. A ja B ovat vanhempia, C puolestaan uusi luotava yksilö.

Uuden sukupolven populaatio on nyt mutaatioyksilöiden luomista vaille valmis. Mutaatioyksilöt luodaan myös yksi kerrallaan kuten risteytysyksilötkin ja käyttäjä määrittelee myös mutaatioyksilöiden kokonaismäärän. Mutaatio-operaatioita on kaksi erilaista ja jokaiselle luotavalle mutaatioyksilölle arvotaan satunnaisesti sovellettava operaatio. Käydään seuraavaksi molempien mutaatio-operaatioiden toiminta läpi.

Vaihtomutaatiossa valitaan aluksi satunnaisesti yksi populaation yksilö turnajaisvalinnalla. Tämän jälkeen valitaan pelilaudalta kaksi satunnaista samankokoista aluetta. Valittavien alueiden minimi- ja maksimikoko määritellään algoritmin parametreissa. Nyt valittujen alueiden paikkaa vaihdetaan keskenään sekoittamatta palojen järjestystä alueiden sisällä.

Kääntömutaatioon valitaan myös yksittäinen populaation yksilö turnajaisvalinnalla. Tällä kertaa pelilaudalta valitaan satunnaisesti yksi neliön muotoinen alue, jonka koko määräytyy samojen parametrien perusteella kuin vaihtomutaatiossa. Valittu alue käännetään kokonaisuudessaan joko 90° , 180° tai 270° . Käännön suuruus valitaan satunnaisesti. Valittava alue on neliön muotoinen suorakulmion sijaan, jotta alueen rakennetta ei tarvitsisi muuttaa sitä käännettäessä.

Molemmissa mutaatio-operaatioissa uusi yksilö muodostuu siis vanhaan yksilöön tehdyn muutoksen tuloksena. Kun kaikki mutaatio-operaatiot on suoritettu, on uuden sukupolven populaation muodostus valmis. Tämän jälkeen algoritmin pääsilmukkaa suoritetaan uudestaan käytännössä niin kauan kuin käyttäjän määrittelemä maksimikierrös-

määrä täyttyy. Algoritmin suoritus loppuu myös jos pelilaudan maksimipistemäärä saavutetaan.

4.3. Algoritmi 2 (monitavoitteinen evolutiivinen algoritmi)

Toinen tarkasteltava algoritmi on myös Muñozin ja muiden [2009] alunperin kehittämä algoritmi, joka perustuu Debin ja muiden [2002] algoritmiin NSGA-II, joka puolestaan on tehokas monitavoitteinen elitismiä hyödyntävä geneettinen algoritmi. Vaikka Eternity II ei olekaan suoranaisesti monitavoitteinen ongelma, jakavat pistelaskun tavoitteet Eternity II:n useampaan aliongelmaan, joita voidaan yrittää ratkaista samanaikaisesti [Muñoz et al., 2009]. Tässä kohdassa käsiteltävää algoritmia kutsutaan nimellä Algoritmi 2. Koska Algoritmilla 2 on Algoritmin 1 tavoin evolutiivinen tausta, ovat nämä algoritmit toiminnaltaan hyvin samanlaisia. Ainoa oleellinen ero Algoritmien 1 ja 2 välillä on se, että Algoritmin 2 populaation pisteytys ei määräydy niin yksinkertaisesti kuin Algoritmissa 1. Koska Algoritmien 1 ja 2 toiminta on muuten samanlaista, keskitytään tässä kohdassa pääosin Algoritmin 2 käyttämään yksilöiden pisteytykseen.

Algoritmin 2 pisteidenlasku alkaa samalla tavalla kuin Algoritmissa 1, mutta yksilöiden pistelaskun jälkeen tehtävän järjestämisen sijaan jokaiselle populaation yksilölle lasketaan dominanssipisteet. Dominanssin laskeminen tapahtuu siten, että jokaista populaation yksilöä verrataan vuorotellen jokaisen muun yksilön kanssa. Jokaista paria tarkasteltaessa verrataan yksilöiden pistemäärän tavoitteita yksitellen keskenään. Jos vertailussa havaitaan, että yksilön kaikkien tavoitteiden pistemäärät ovat vähintään yhtä hyviä kuin toisen yksilön tavoitteiden pistemäärät ja ainakin yksi tavoite on aidosti parempi, dominoi ensin mainittu yksilö tällöin jälkimmäistä yksilöä. Dominoitavan yksilön pistemäärää kasvatetaan tällöin yhdellä. Jos populaation koko on n , voi yksilön dominanssiksi muodostua täten minimissään nolla ja maksimissaan $n - 1$.

Sellaisten yksilöiden, joita ei dominoi mikään muu yksilö, sanotaan sijaitsevan Pareto-rintamalla. Seuraavaksi lasketaan Pareto-rintaman yksilöiden etäisyydet toisistaan. Etäisyyden laskeminen tapahtuu siten, että Pareto-rintaman yksilöitä verrataan pareittain toisiinsa. Molempien yksilöiden etäisyysarvoa kasvatetaan yhdellä jokaista yksilöiden toisistaan eroavaa pistelaskun tavoitetta kohden. Etäisyyksiä laskiessa pidetään samalla kirjaa suurimmasta kuluvan kierroksen etäisyysarvosta. Lopuksi Pareto-rintaman yksilöiden etäisyydet normalisoidaan välille $[0,1]$ siten, että se yksilö, jolla on suurin etäisyysarvo, saa lopulta etäisyydekseen arvon 1,0. Kaikki muut kuin Pareto-rintaman yksilöt saavat etäisyysarvokseen nollan.

Kun dominanssi ja etäisyysarvot on laskettu, voidaan jokainen populaation yksilö pisteyttää. Yksilön pistearvo lasketaan kaavalla $dom + (1 - dist)$, jossa dom on yksilön dominanssiarvo ja $dist$ vastaavasti yksilön etäisyysarvo. Poiketen muista algoritmeista Algoritmissa 2 pienempi pistemäärä katsotaan suurempaa paremmaksi. Näin ollen dominoi-

mattomat yksilöt (l. Pareto-rintaman yksilöt) pisteytetään aina korkeammalle kuin dominoidut yksilöt, sillä dominoimattomien yksilöiden pistemäärä sijoittuu välille $[0,1]$. Pareto-rintamaan kuuluva yksilö siis katsotaan sitä paremmaksi, mitä suurempi sen etäisyysarvo on. Yksilöt asetetaan siis pistelaskun valmistuttua tavanomaiseen tapaan pistemäärän mukaiseen (nousevaan) järjestykseen. Tämän jälkeen Algoritmin 2 toiminta jatkuu kuten Algoritmilla 1.

4.4. Algoritmi 3 (hill climbing -algoritmi)

Kolmas tämän tutkielman algoritmeista perustuu hill climbing -tekniikkaan. Koska hill climbing -tekniikan idea on melko yksinkertainen, ei itse algoritmikaan ole kovin monimutkainen. Tätä algoritmia kutsutaan nimellä Algoritmi 3.

Algoritmin suoritus alkaa alustuksella, kuten muissakin tapauksissa. Algoritmeista 1 ja 2 poiketen Algoritmilla 3 ei ole käytössä erillistä populaation käsitettä, vaan käytössä on vain yksi pelilauta, jota pyritään parantamaan iteratiivisesti paikallisella etsinnällä. Parametrien lataamisen jälkeen pelilauta alustetaan satunnaiseen järjestykseen ja esiratkaistaan, mikäli käyttäjä on niin määritellyt.

Alustuksen jälkeen aloitetaan algoritmin varsinainen suoritus. Ennen algoritmin pääsilmukan suorittamista pelilaudan pistemäärä lasketaan ohjelmarungon avulla. Tämän jälkeen siirrytään algoritmin pääsilmukkaan. Jokaisen silmukan kierroksen alussa tarkistetaan, onko pelilauta saavuttanut maksimipistemäärän tai onko algoritmia suoritettu käyttäjän määrittelemän maksimikierrosmäärän verran. Jos jompikumpi näistä ehdoista toteutuu, algoritmin suoritus päättyy. Muussa tapauksessa pelilautaa muokataan paikallisesti toisella kahdella vaihtoehtoisesta tavasta, jotka käsitellään jäljempänä. Muokkauksen jälkeen pelilaudan pistemäärä lasketaan uudestaan. Jos muokkaus paransi pelilaudan pistemäärää, algoritmi tallentaa pelilaudan muutoksen ja algoritmin suoritus jatkuu pääsilmukan alusta. Jos pelilaudan muutos ei paranna pistemäärää, ei muutosta oteta käyttöön ja algoritmin suoritus jatkuu tässäkin tapauksessa pääsilmukan uudella kierroksella. Käyttäjä voi tosin eräällä algoritmin parametrilla määritellä, että pistemäärää laskenut pelilauta otetaan pisteiden laskusta huolimatta käyttöön. Tällöin pelilaudan tulevien muutosten aiheuttamaa pistemäärää verrataan kuitenkin suorituksen aikana korkeimpaan saavutettuun pistemäärään, eikä aktiivisena olevan pelilaudan pistemäärään, joka on tällöin matalampi kuin korkein saavutettu pistemäärä.

Algoritmi 3 tarkkailee myös, kuinka monta kierrosta sitten pelilaudan pistemäärä on viimeksi parantunut. Käyttäjä voi määritellä algoritmin parametriksi kierrosrajan, jonka sisään tuloksen pitäisi parantua. Jos algoritmin tulos ei parane siihen mennessä kuin käyttäjän määrittelemä kierrosraja saavutetaan, algoritmin suoritus pysäytetään myös tässä tapauksessa.

Kuten edellä mainittiin, Algoritmilla 3 on kaksi erilaista toimintatapaa, jotka eroavat toisistaan pelilaudaan käytetyn muunnosoperaation osalta. Käyttäjä määrittelee algoritmin parametrien joukossa myös käytettävän muunnosoperaation.

Ensimmäinen muunnosoperaatioista on raakaan voimaan perustuva operaatio, jota kutsutaan Muunnostyyppiä 1. Tämä muunnostyyppi toimii siten, että algoritmin pääsil-
mukan jokaisella kierroksella pelilaudalta valitaan satunnainen pala. Valittua palaa koete-
taan tämän jälkeen vaihtaa vuorotellen jokaisen muun palan kanssa. Keskenään vaihdet-
tavat palat sijoitetaan lisäksi jokaiseen mahdolliseen asentoon. Testatuista asennoista vali-
taan sellainen, joka nostaa pistemäärää kaikista eniten. Pistemäärän kasvaessa palat jätet-
tään suotuisimpaan asentoonsa ja algoritmin suoritus jatkaa pääsil-
mukan uudella kierrok-
sella. Jos mikään käsiteltävien palojen asentokombinaatio ei kasvata pelilaudan pistemää-
rää, hylätään mahdollisesti tehdyt muutokset ja otetaan käyttöön kierroksen aluksi valit-
tua palaa seuraava pala, jota vuorostaan vaihdetaan muiden palojen kanssa. Algoritmin
suoritusta jatketaan, kunnes mikään yllä kuvatuinen muutos ei enää paranna peli-
laudan pistemäärää.

Toinen muunnosoperaatio koostuu Algoritmien 1 ja 2 käyttämisestä vaihtomutaatiosta ja
kääntömutaatiosta. Vaihto- ja kääntömutaatiosta koostuvaa muunnosoperaatiota kutsu-
taan nimellä Muunnostyyppi 2. Muunnostyyppiä 2 käytettäessä käyttäjän tulee määrittellä
Algoritmille 3 erityinen muunnossuhdeparametrin arvo, joka määrittelee, millä todennä-
köisyydellä Algoritmi 3 suorittaa kullakin kierroksella vaihtomuunnoksen ja millä toden-
näköisyydellä puolestaan kääntömuunnoksen. Muunnossuhde annetaan lukuna nollan ja
yhden väliltä. Muunnostyyppillä 2 algoritmi tekee satunnaisen muunnoksen ja tarkistaa,
muuttuiko pelilaudan pistemäärä. Kuten edelläkin, algoritmi hylkää muutoksen, jos pis-
temäärä ei parane ja ottaa toisaalta muutoksen käyttöön, jos pistemäärä kasvaa. Koska
Muunnostyyppin 2 toiminta perustuu satunnaisuuteen järjestelmällisen testaamisen sijaan,
algoritmin suoritus päättyy kun tietty maksimikierrosmäärä on saavutettu, tai kun peli-
laudan pistemäärä ei ole parantunut tietyn kierrosmäärän kuluessa.

4.5. Algoritmi 4 (simuloitu jäähtytys)

Neljäs ja viimeinen käsiteltävä algoritmi on nimeltään Algoritmi 4. Tämä algoritmi muis-
tuttaa toiminnaltaan ja perusidealtaan paljon Algoritmia 3, mutta koska Algoritmi 4 pe-
rustuu simuloituun jäähtytukseen, se pystyy Algoritmin 3 ahneen ja suoraviivaisen toi-
minnan sijaan tekemään myös nykyistä ratkaisua huonontavia valintoja.

Algoritmin 4 alustus tapahtuu samalla tavalla kuin Algoritmessa 3, eli algoritmin pa-
rametrit ladataan, satunnainen pelilauta luodaan ja sen reunat esiratkaistaan, mikäli käyt-
tämä on niin valinnut. Algoritmin 3 parametrien lisäksi käyttäjän tulee määrittellä Algorit-
mille 4 simuloitun jäähtytysten toimintaan liittyen parametrit T_0 , α ja ε . Nämä para-

metrit ovat samat kuin kohdassa 3.4 esitellyt samannimiset parametrit, eli algoritmin alkulämpötila, lämpötilan jäähtymiskerroin sekä loppulämpötila.

Algoritmin 4 pääsilmaan aluksi tarkastetaan aina algoritmin lopetusehdot. Algoritmin suoritus päättyy, mikäli maksimipistemäärä saavutetaan, jos algoritmin lämpötila laskee alle ε :n tai jos algoritmilta mahdollisesti määritelty maksimikierron määrä täyttyy. Lopetusehtojen tarkastuksen jälkeen pelilautaan tehdään jokin paikallinen muutos (tästä lisää myöhemmin), ja jos tehty muutos parantaa pelilaudan pistemäärää, se otetaan käyttöön kuten Algoritmissa 3. Jos tehty muutos huonontaa pelilaudan pistemäärää, se hyväksytään silti todennäköisyydellä $P(\Delta, T) = e^{\Delta/T}$, kun Δ on pistemäärän (negatiivinen) muutos ja T on algoritmin senhetkinen lämpötila. Käyttäjä voi määrittellä algoritmin parametreissa, hyväksytäänkö sellainen pelilautaan tehty muutos, joka ei muuta pelilaudan pistemäärää lainkaan. Jos pelilaudan pistemäärä aleni pelilautaan tehdyn muutoksen seurauksena, muutos siis hylätään todennäköisyydellä $1 - P(\Delta, T) = 1 - e^{\Delta/T}$.

Jokaisen pääsilmaan kierroksen lopuksi algoritmin lämpötilaa tulee laskea. Lämpötilaa voidaan laskea kahdella tavalla, joko lineaarisesti tai eksponentiaalisesti. Näitä tapoja kutsutaan tässä tutkielmassa täten lineaarisiksi ja eksponentiaalisiksi jäähtyysfunktioiksi. Käyttäjän tulee valita haluamansa jäähtyysfunktio algoritmin parametreilla.

Lineaarista jäähtyysfunktioita käytettäessä käyttäjän tulee määrittellä vakion α sijaan maksimikierron määrä, joka määrää, kuinka kauan algoritmia suoritetaan. Tällöin algoritmin lämpötila on aluksi T_0 , ja jokaisella kierroksella lämpötilaa vähennetään sopivalla vakiolla siten, että viimeisellä kierroksella lämpötilan arvo on nolla. Eksponentiaalinen jäähtyysfunktio puolestaan hyödyntää aikaisemmin mainittua parametria $0 < \alpha < 1$, jolla lämpötilan arvo jokaisen kierroksen jälkeen kerrotaan. Tällöin lämpötila-arvot muodostavat vähenevän geometrisen jonon. Lämpötilan pienenemisen jälkeen algoritmin suoritusta jatketaan pääsilmaan alusta.

5. Tulokset

Tässä luvussa esitellään tuloksia, joita algoritmien ajoista on saatu. Tulokset on jaettu kohtiin algoritmeittain. Algoritmien tulokset käsitellään algoritmeittain edellisen luvun esittelyjärjestyksessä siten, että Algoritmien 1 ja 2 tuloksia verrataan myös Muñozin ja muiden [2009] tuloksiin. Muilta osin algoritmeille on pyritty löytämään mahdollisimman korkean pistemäärän tuottamat parametrit.

5.1. Algoritmin 1 vertailua Muñozin ja muiden tuloksiin

Koska tässä kohdassa testattava Algoritmi 1 ei ole itseni suunnittelema, aloitin algoritmin testaamisen samoilla parametreilla, joilla alkuperäisetkin tekijät [Muñoz ja muut, 2009] algoritmia olivat testanneet.

Muñoz ja muut varioivat geneettisessä algoritmissaan risteytyksen, elitismien ja mutaation parametreja, jotka vaikuttavat siihen, millä tavalla populaation jälkeläisiä tuotetaan. Tämän lisäksi he varioivat populaation alustusta sekä pisteidenlaskutapaa. Kuten tiedämme, Muñozin ja muiden algoritmi mahdollistaa populaation yksilöiden pisteytyksen useamman eri tavoitteen avulla. Testiajoissaan he käyttivät populaation yksilöiden arviointiin kahta eri tapaa, joista ensimmäisessä huomioitiin ainoastaan Tavoitteen 1 pistemäärä ja jälkimmäisessä Tavoitteiden 1, 2 ja 3 pistemäärien keskiarvo. Mainittakoon myös tässä, että testeissä käyttämäni pelilautojen esiratkaisu poikkeaa Muñozin ja muiden esiratkaisusta. Siinä missä he pyrkivät ratkaisemaan (vaillinaisesti) koko pelilaudan, oma alustukseni ratkaisee vain pelilaudan reunat. Jos reunoja ei ole saatu ratkaistua kymmenen miljoonan askeleen jälkeen, ratkaisu keskeytetään ja kyseisen pelilaudan palat jätetään aikaisemmin arvottuun sattumanvaraiseen järjestykseen.

Edellä mainittujen seikkojen lisäksi käytän myöhemmissä testiajoissa myös muidenkin parametrien variointia. Näihin kuuluu muun muassa Muñozin ja muiden algoritmista puuttuva, jo aikaisemminkin mainittu Tavoite 4, joka tarkkailee pelilaudan reunapalojen sopivuutta. Tämän lisäksi yritän selvittää tarkemmin populaation optimaalista rakennetta ja muutan myös risteytyksessä ja mutaatioissa käytettäviä parametreja, jotka määrittävät, minkä kokoisia pelilaudan alueita kyseisiin operaatioihin valitaan. Käytännössä tämä tarkoittaa suorakaiteen (mutaatio-operaatioissa neliön) muotoisen pelilaudan alueen minimi- ja maksimimittoja. Risteytys- ja mutaatio-operaatioissa käytettävä turnajaisvalinta on myös eräs muutoksen kohde. Muñoz ja muut käyttivät kaikissa testeissään turnajaisia, joihin osallistuu populaation kolme satunnaista yksilöä, joista valitaan paras.

Taulukossa 1 nähdään Muñozin ja muiden saamat tulokset sekä vastaavilta osin myös omat tulokseni. Jokaista testiajoa on suoritettu 5000 sukupolven ajan, joka vastaa Muñozin ja muiden algoritmin suoritusaikaa. Lisäksi algoritmi on suoritettu kullakin parametriyhdistelmällä kymmenen kertaa, sillä yksittäisen ajon tulos voi heittelehtiä paikoin melko paljonkin. Omissa testituloksissani on siis varioitu samoja parametreja kuin Muñoz ja

muutkin käyttivät. Tämä sillä erotuksella, että populaation alustus eroaa yllä mainituin osin heidän toteutuksestaan. Taulukon lukemat on esitetty siten, että ensimmäinen lukema on Muñozin ja muiden tulos ja toinen lukema on oma tulokseni. Koska yksilön paremmuus määritellään tutkielman eri algoritmeilla eri tavoilla, on tämän tutkielman tulokset raportoitu käyttäen ainoastaan Eternity II:n säännöissä määriteltyä pistemäärää (eli Tavoitetta 1, ks. Kuva 4), joka siis tarkoittaa pelilaudan sellaisten vierekkäisten sivuparien määrää, joilla on sama tunniste. Testeissä käytetyllä 16 x 16 palan kokoisella pelilaudalla tämä tarkoittaa maksimissaan 480 pistettä. Tulosten esitystavaksi on valittu Eternity II:ssä käytetty pistemäärä, koska se on intuitiivinen ja yleisesti käytössä oleva tapa pelilaudan hyvyyden arviointiin. Lisäksi mainittu pisteidenlaskutapa helpottaa eri algoritmien keskinäisten tulosten vertailua, sillä algoritmien sisäisesti käyttämä pistelasku vaihtelee.

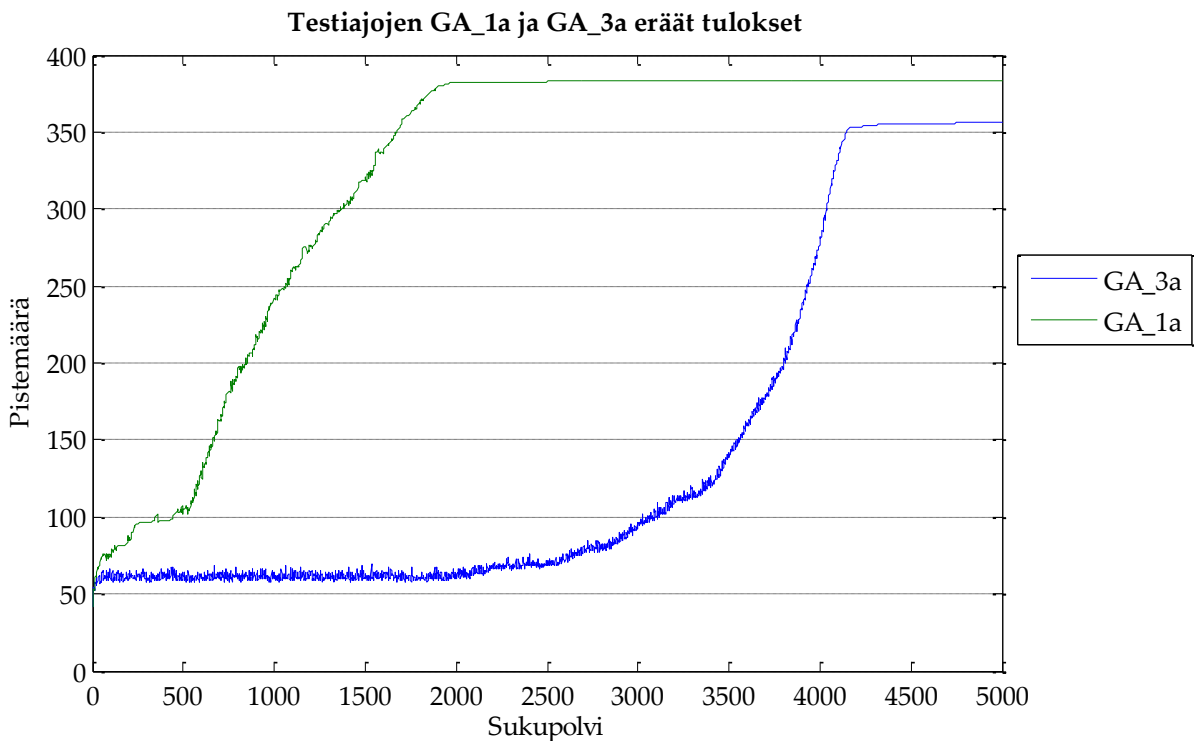
Testiajo	Eliit./Rist./Mut.	Tavoitteet	Esirat.	MAX	MIN	MEAN	STDEV
GA_1a	1/9000/999	1	Ei	303/341	278/322	292,5/333,9	8,56/5,47
GA_1b	1/9000/999	1, 2, 3	Ei	352/387	334/356	345,8/372,6	4,92/9,54
GA_1c	1/9000/999	1	Kyllä	394/396	382/353	385,9/374,6	3,33/15,74
GA_1d	1/9000/999	1, 2, 3	Kyllä	387/393	377/382	382,3/386,0	3,26/3,77
GA_2a	1/5000/4999	1	Ei	169/184	152/169	159,7/175,1	4,78/4,12
GA_2b	1/5000/4999	1, 2, 3	Ei	338/391	144/373	214,7/380,6	78,02/5,38
GA_2c	1/5000/4999	1	Kyllä	358/220	348/192	351,3/209,5	2,76/10,32
GA_2d	1/5000/4999	1, 2, 3	Kyllä	391/393	341/341	355,3/379,2	12,97/14,11
GA_3a	0/7500/2500	1	Ei	105/359	63/59	90,2/183,6	12,79/148,93
GA_3b	0/7500/2500	1, 2, 3	Ei	364/390	344/376	355,9/383,1	6,36/4,68
GA_3c	0/7500/2500	1	Kyllä	343/400	334/380	337,9/392,7	3,08/6,41
GA_3d	0/7500/2500	1, 2, 3	Kyllä	374/394	363/376	367,4/382,3	3,85/5,19

Taulukko 1. Muñozin ja muiden geneettisen algoritmin tulokset rinnakkain omien vastaavien tulosteni kanssa (MAX = maksimipistemäärä, MIN = minimipistemäärä, MEAN = pisteiden keskiarvo, STDEV = pisteiden keskihajonta).

Taulukosta 1 voidaan nähdä, että testiajojen pisteet ovat pääosin samaa suuruusluokkaa niin Muñozilla ja muilla kuin itselläkin. Kuitenkin testiajoissa GA_2b, GA_2c ja GA_3a pisteiden keskimääräinen ero on melko suuri. Tämä on melko hämmäntävää ottaen huomioon, että Algoritmin 1 pitäisi olla lähestulkoon suora kopio toiminnallisuudeltaan Muñozin ja muiden versiosta (algoritmien arkkitehtuuri ja käytetyt tietorakenteet tosin poikkeavat hieman toisistaan). Testiajojen GA_2c ero voi johtua erilaisista algoritmien käyt-

tämistä pelilautojen alustusfunktioista, mutta GA_2b:n ja GA_3a:n erot ovat jo vaikeampia selittää.

Testiajon GA_3a tulokset olivat sikäli erikoisia, että pistemäärien keskihajonta oli sangen suuri. Mukana oli koko joukon huonoin pistemäärä, mutta paras tulos toisaalta ylitti huimasti Muñozin ja muiden vastaavan testiajon parhaan tuloksen. Erikoisen käyttäytymisen voi havaita myös Kuvassa 6, joka esittää testiajojen GA_1a ja GA_3a tiettyjen yksittäisten ajojen pistemäärän kehitystä geneettisen algoritmin sukupolven funktiona. Tavallisesti, jos testiajo tuottaa korkeahkon pistemäärän, lähtee pistemäärä kasvamaan heti alusta alkaen, kuten GA_1a:n pistemäärää kuvaava vihreä käyrä osoittaa. Kuitenkin tapauksissa, joissa testiajo GA_3a tuotti korkeita pisteitä, pistemäärä lähtee kasvuun vasta n. 2000 sukupolven jälkeen. On vaikea sanoa, mistä tämä tarkkaan ottaen johtuu.



Kuva 6. Testiajojen GA_1a ja GA_3a eräiden tulosten pistemäärän kehitys sukupolven funktiona Algoritmilla 1 ajettuna.

Ajoin testiajon GA_3a viisi kertaa ja asetin suorituksen kestämään 20000 sukupolvea. Tämän ajon tulokset (ks. Taulukko 2) avasivatkin hieman tilannetta. Nyt neljä viidestä ajosta päätyi yli 350 pisteen lukemiin. Vain yhden ajon pistemäärä jäi kuudenkymmenen pisteen tietämillä. Mielenkiintoista oli huomata, että kahden yli 350 pistemäärän ajon pistemäärät lähti nousuun vasta n. 10000 ja 15000 sukupolven jälkeen, kun muissa testisarjoissa maksimipistemäärä saavutettiin useasti n. 1000 sukupolven jälkeen. On hankala sanoa, miksi pistemäärä ei nouse tasaisesti alusta lähtien. Voi olla, että käytetyillä parametreilla peli-

lauta tarvitsee tietyn (suhteellisen harvoin esiintyvän) "alkutöytäisyyn", jonka jälkeen pistemäärä lähtee vasta nousuun.

Testiajo	Eliit./Rist./Mut. (kpl)	Pistelaskun tavoitteet	Esiratkaistu	Pistemäärä
GA_3	0/7500/2500	1	Ei	61
GA_3	0/7500/2500	1	Ei	363
GA_3	0/7500/2500	1	Ei	363
GA_3	0/7500/2500	1	Ei	364
GA_3	0/7500/2500	1	Ei	371

Taulukko 2. Testisarjan GA_3 20000 sukupolven mittaisen ajon viisi tulosta.

Palatakseni Taulukon 1 tuloksiin, testiajon GA_2b suhteen tilanne oli melkein vastakkainen testiajona GA_3a:een nähden; tässä tapauksessa oma ajoni on saanut keskimäärin todella hyviä pistemääriä, kun taas Muñozin ja muiden vastaavassa ajossa on ollut melko suuri hajonta.

5.2. Algoritmin 1 tuloksia

Koska Muñozin ja muiden testeissä erilaisia testiajoja oli suhteellisen pieni määrä, emme voi tietää, onko parametrit valittu sattumanvaraisesti, vai ovatko Muñoz ja muut valinneet vain parhaat parametriyhdistelmät esitettäväksi tutkimukseensa. Tästä syystä tein Algoritmilla 1 useita lisäajoja, joilla pyrin säätämään algoritmin parametreja optimaalisemmaksi.

Edellisen kohdan tulosten perusteella näyttää siltä, että ajot, joissa pelilautaa ei ole esiratkaistu ja joissa on käytetty ainoana tavoitteena Tavoitetta 1, eivät pärjää pistemäärässä sellaisille ajoille, joissaistelaskuna on käytetty Tavoitteita 1–3. Toisaalta, jos pelilaudan reunat on ratkaistu alustuksen yhteydessä, pelkkää Tavoitetta 1 käyttävä testiajo GA_3c on saanut edellisessä kohdassa kaikkein korkeimmat pisteet.

Kaikissa tästä eteenpäin käsitellyissä testiajoissa on käytetty kehittämäni lisätavoitetta (Tavoite 4), joka tutkii pelilautojen reunoilla sijaitsevien palojen sopivuutta. Muñozin ja muiden algoritmi ei huomionnut lainkaan pelilaudan reunojen rakenteen oikeellisuutta. Tästä syystä ilman Tavoitetta 4 pelilaudan reunapaloja sijoitetaan surutta myös pelilaudan keskelle. Eternity II:n virallisten sääntöjen mukaan [Monckton, 2007] reunapalat tulisi sijoittaa pelilaudan laidoille, mutta säännöissä ei kuitenkaan mainita tarkemmin, onko reunapaloja nimenomaan kiellettyä sijoittaa muualle kuin pelilaudan laidoille. Koska tavoitteenani on saavuttaa Eternity II:n mahdollisimman korkea pistemäärä, joka on myös pelin sääntöjen mukainen, otin käyttöön Tavoitteen 4, jotta pelilauta täyttäisi mahdollisimman hyvin nämä vaatimukset.

Koska Algoritmilla 1 on varsin monta suoritukseen vaikuttavaa parametria, varioin aluksi populaation rakenteeseen vaikuttavia parametreja, joita ovat elitismillä valittavien yksilöiden määrä, risteytyksellä muodostettavien yksilöiden määrä sekä mutaatiolla muodostettavien yksilöiden määrä. Tämän lisäksi ajoin samat ajot esiratkaistuilla pelilaudoilla ja käytin pistemäärän laskussa sekä tavoiteyhdistelmää 1 ja 4 että yhdistelmää 1, 2, 3 ja 4.

Testiajoissa käytetyt populaation rakenteeseen vaikuttavat parametrit käyvät ilmi tulostaulukosta (Taulukko 4). Populaation lukumäärä on jokaisessa ajossa kokonaisuudessaan 10 000 yksilöä. Jokainen alustava testiajo ajettiin myös käyttäen kahta eri tavoiteyhdistelmää pistelaskussa sekä tämän lisäksi pelilaudan esiratkaisussa oli käytössä kaksi asetusta. Näitä parametrikombinaatioita edustaa testiajon perässä oleva kirjain (A, B, C tai D). Taulukko 3 sisältää jokaista kirjainta vastaavat parametrien arvot.

Parametrit	Pistelaskun tavoitteet	Esiratkaistu pelilauta
Testiajon tarkenne		
A	1 ja 4	Ei
B	1 ja 4	Kyllä
C	1, 2, 3 ja 4	Ei
D	1, 2, 3 ja 4	Kyllä

Taulukko 3. Testiajojen nimen tarkenne ja sen vaikutus lisäparametreihin.

Testiajoja tuli ensimmäisessä vaiheessa edellä mainituilla parametrivalinnoilla 132 kappaletta, joten alla oleviin tuloksiin on kirjattu ainoastaan kunkin testiajoryhmän paras tulos. Paras tulosryhmä valitaan keskimääräisen pistemäärän perusteella.

Testiajo	Eliit./Rist./Mut. (kpl)	Tavoitteet	Esiratkaistu	MAX	MIN	MEAN	STDEV
GA_4D	0/0/10000	1, 2, 3, 4	Kyllä	167	155	160,6	3,31
GA_5D	1/0/9999	1, 2, 3, 4	Kyllä	241	230	235,8	4,66
GA_6D	3/0/9997	1, 2, 3, 4	Kyllä	265	248	254,2	6,66
GA_7D	0/1000/9000	1, 2, 3, 4	Kyllä	195	185	187,7	3,23
GA_8D	1/1000/8999	1, 2, 3, 4	Kyllä	263	246	253,5	5,23
GA_9D	3/1000/8997	1, 2, 3, 4	Kyllä	277	245	260,6	9,98
GA_10D	0/2000/8000	1, 2, 3, 4	Kyllä	210	198	201,9	3,87
GA_11D	1/2000/7999	1, 2, 3, 4	Kyllä	274	257	264,3	5,77
GA_12D	3/2000/7997	1, 2, 3, 4	Kyllä	280	259	269,6	6,93
GA_13D	0/3000/7000	1, 2, 3, 4	Kyllä	228	213	218,2	4,49
GA_14D	1/3000/6999	1, 2, 3, 4	Kyllä	291	271	279,8	6,44
GA_15D	3/3000/6997	1, 2, 3, 4	Kyllä	286	273	279,0	4,35
GA_16D	0/4000/6000	1, 2, 3, 4	Kyllä	259	239	246,8	5,53
GA_17D	1/4000/5999	1, 2, 3, 4	Kyllä	302	286	293,2	5,27
GA_18D	3/4000/5997	1, 2, 3, 4	Kyllä	315	289	303,0	9,83
GA_19C	0/5000/5000	1, 2, 3, 4	Ei	368	338	361,8	8,84
GA_20C	1/5000/4999	1, 2, 3, 4	Ei	370	361	365,7	2,63
GA_21D	3/5000/4997	1, 2, 3, 4	Kyllä	389	333	360,4	21,65
GA_23D	0/6000/4000	1, 2, 3, 4	Kyllä	403	382	389,9	5,67
GA_24D	1/6000/3999	1, 2, 3, 4	Kyllä	400	379	389,3	5,52
GA_25D	3/6000/3997	1, 2, 3, 4	Kyllä	391	374	383,1	4,98
GA_26D	0/7000/3000	1, 2, 3, 4	Kyllä	396	384	388,8	4,29
GA_27D	1/7000/2999	1, 2, 3, 4	Kyllä	390	382	386,8	2,44
GA_28D	3/7000/2997	1, 2, 3, 4	Kyllä	391	370	383,8	6,16
GA_29B	0/8000/2000	1, 4	Kyllä	407	383	394,8	6,48
GA_30D	1/8000/1999	1, 2, 3, 4	Kyllä	397	382	385,7	4,24
GA_31D	3/8000/1997	1, 2, 3, 4	Kyllä	394	375	381,0	5,54
GA_32B	0/9000/1000	1, 4	Kyllä	403	386	393,2	5,63
GA_33D	1/9000/999	1, 2, 3, 4	Kyllä	393	383	387,4	3,47
GA_34D	3/9000/997	1, 2, 3, 4	Kyllä	388	378	383,2	3,36
GA_35D	0/10000/0	1, 2, 3, 4	Kyllä	391	373	382,7	5,44
GA_36D	1/9999/0	1, 2, 3, 4	Kyllä	385	376	381,1	3,35
GA_37D	3/9997/0	1, 2, 3, 4	Kyllä	385	370	378,2	4,57

Taulukko 4. Algoritmin 1 alustavia tuloksia.

Taulukon 4 tuloksista nähdään, että ehdottomasti parhaat keskimääräiset pisteet saatiin, kun risteytysyksilöiden määrä on välillä 6000–9000. Tällä välillä tulosten hajonta oli myös melko pientä. Pelilautojen esiratkaisu tuotti lähes aina parempia tuloksia kuin ratkaisematta jättäminen. Päinvastainen tilanne oli testiajoilla GA_19 ja GA_20, joissa esiratkaisemattomat testiajot tuottivat paremman tuloksen. Toisaalta on hyvä huomioda, että esiratkaisua käyttävän testiajon GA_19D keskimääräinen tulos oli 359,2 pistettä, ja testiajon GA_20D keskimääräinen tulos puolestaan 362,7, joten piste-eroa syntyi molemmissa tapauksissa korkeintaan kolmen pisteen verran.

Jos vertaamme Muñozin ja muiden mukaisten parametrien mukaisia testiajoja GA_1d ja GA_2d (ks. Taulukko 1) testiajoihin GA_33D ja GA_20D (GA_20D ei ole listattu Taulukossa 4), niin huomaamme mikä vaikutus neljännellä pistelaskun tavoitteella on tuloksiin, kun pelilauta on esiratkaistu ja pisteet lasketaan useamman tavoitteen keskiarvona. Sain testiajon GA_1d keskimääräiseksi tulokseksi 386,0 pistettä, kun vastaava neljän tavoitteen testiajo GA_33D:n keskimääräinen pistemäärä oli 387,4. Vastaavasti testiajon GA_2d keskimääräinen tulos oli 379,2, kun testiajo GA_20D tuotti keskimäärin 362,7 pistettä. Ensimmäisessä tapauksessa pistemäärä siis oli suunnilleen samaa luokkaa Muñozin ja muiden tuloksen kanssa, mutta toisessa esimerkissä neljäs tavoite laskee pistemäärää. Jälkimmäisen tapauksen pistemäärän keskihajonta oli tosin keskimääräistä suurempi 21,73, mikä saattaa merkitä sitä, ettei jokaisen testiajon pistemäärä kerennyt kehittyä huippuunsa ennen algoritmin suorituksen päättymistä.

Mielenkiintoisia ovat myös testiajot GA_29 ja GA_32, joissa kaikista muista testiajoista poiketen parempi tulos syntyi pienemmällä määrällä tavoitteita. Testiajo GA_29B saavutti myös tähän mennessä kaikista parhaimman yksittäisen ajon pistemäärän, 407/480 pistettä.

Näiden tulosten perusteella testiajojen parametreja voidaan säätää tarkemmin sellaisiksi, jotka tuottavat mahdollisimman hyvän pistemäärän. Koska näyttäisi siltä, että kolme eliittiyksilöä huonontaa jokaisessa tapauksessa tulosta, tulevia testiajoja on suoritettu ainoastaan yhdellä eliittiyksilöllä sekä kokonaan ilman eliittiyksilöitä. Ei ole myöskään syytä käyttää esiratkaisemattomia pelilautoja, sillä esiratkaistut pelilaudat tuottivat parhaiten pärjänneissä testiajoissa jokaisessa tapauksessa paremman tuloksen kuin esiratkaisemattomat pelilaudat.

Tarkennetut testiajot suoritettiin populaation rakenteen tehdessä sadan yksilön muutoksia risteytys- ja mutaatioyksilöiden suhteen siten, että uudet ajot aloitettiin risteytysyksilöiden ja mutaatioyksilöiden määrän ollessa 6100 ja 3900. Viimeinen ajo oli sellainen, jolla vastaavat yksilömäärät olivat 9400 ja 600. Testiajoja, joissa risteytysyksilöiden määrä oli 6100–7500, ajettiin yhteensä 5000 sukupolven verran, mutta 7600–9400 risteytysyksilön testiajoille algoritmin suoritusta pidennettiin varmuuden vuoksi 6000 sukupolven. Algoritmin suoritusta pidennettiin, sillä 5000 sukupolvea näytti paikoin juuri ja juuri riittävän siihen, että Algoritmin 1 tulos vakiintui kyseisillä parametreilla. Taulukko 5 sisältää

kymmenen tähän mennessä parasta testiajoa, joista ensimmäiseksi sijoittunutta tutkitaan vielä tarkemmin.

Testiajo	Eliit./Rist./Mut. (kpl)	Tavoitteet	Esiratkaistu	MAX	MIN	MEAN	STDEV
GA_29B	0/8000/2000	1, 4	Kyllä	407	383	394,8	6,48
GA_41B	0/7600/2400	1, 4	Kyllä	405	388	394,5	5,06
GA_45B	0/7800/2200	1, 4	Kyllä	403	386	394,5	5,17
GA_32B	0/9000/1000	1, 4	Kyllä	403	386	393,2	5,63
GA_47B	0/7900/2100	1, 4	Kyllä	403	381	393,1	6,74
GA_61B	0/8700/1300	1, 4	Kyllä	401	382	393	6,09
GA_57B	0/8500/1500	1, 4	Kyllä	399	384	392,8	4,24
GA_53B	0/8300/1700	1, 4	Kyllä	404	381	392,4	6,38
GA_51B	0/8200/1800	1, 4	Kyllä	398	383	392,2	4,64
GA_63B	0/8800/1200	1, 4	Kyllä	403	385	391,9	6,72

Taulukko 5. Uuden testiajon kymmenen parasta tulosta.

Huomionarvoista Taulukon 5 tuloksissa on se, että kaikki parhaat pistemäärät saavutettiin pienemmällä tavoitemäärällä. Tosin kaikki tulokset ovat melko lähellä toisiaan, joka toisaalta tarkoittanee sitä, että populaation rakenteen parametrit ovat nyt melko optimaaliset.

Nyt kun populaation tarvittavasta rakenteesta on hyvä kuva, valitaan paras testiajo ja lukitaan populaation rakenteen parametrit, pistelaskun tavoitteet sekä pelilaudan esiratkaaisu. Tämän jälkeen varioidaan jäljelle jääneitä algoritmin sisäisiä parametreja. Näitä ovat aikaisemmin mainitut risteytys- ja mutaatio-operaatioissa käytettävien alueiden minimi- ja maksimirajat sekä yksilöitä valitsevien turnajaisten koko.

Kuten kohdasta 4.2 muistamme, risteytysoperaatioon valitaan ensin kaksi erillistä yksilöä kaksilla turnajaisilla, jonka jälkeen pelilaudalta valitaan satunnaisesti suorakulmion muotoinen alue A , jonka sivun minimimitta on C_{\min} ja maksimimitta C_{\max} . Keskenään kohtisuorat sivut voivat olla erimittaiset. Tämän jälkeen muodostetaan kahdesta valitusta yksilöstä uusi yksilö siten, että ensimmäisestä yksilöstä kopioidaan suoraan alue A sellaisenaan uuteen yksilöön, jonka jälkeen loput puuttuvat palat täytetään toisesta valitusta yksilöstä.

Mutaatio-operaatiota varten on myös valittu hieman samantyylliset muuttujat M_{\min} ja M_{\max} , jotka vastaavasti kuin risteytysoperaatioissa kontrolloivat mutaatio-operaatioihin valittavan alueen kokoa. Mutaatio-operaatiot on selitetty tarkemmin kohdassa 4.2.

Turnajaiset toimivat siten, että populaatiosta valitaan ensiksi satunnaisesti T kappaletta yksilöitä. Valinnan jälkeen valittujen yksilöiden pistemäärät asetetaan suuruusjärjestyk-

seen ja paras yksilö valitaan voittajaksi. Suurilla turnajaisilla on luonnollisesti pieniä turnajaisia suurempi todennäköisyys tuottaa laadukkaampia voittajia. Ei ole kuitenkaan itsestään selvää, että pelkästään mahdollisimman hyvien yksilöiden tuottaminen turnajaisilla tuottaisi geneettisellä algoritmilla mahdollisimman hyvän lopputuloksen, kuten myöhemmin tullaan näkemään.

Koska Eternity II -pelilaudan koko on 16×16 palaa, täytyy parametrien C_{\min} ja C_{\max} olla välillä 1–16 siten, että $C_{\min} \leq C_{\max}$. Vastaavasti parametrin M_{\min} täytyy olla pienempi tai yhtäsuuri kuin parametrin M_{\max} . Tämän lisäksi täytyy olla $M_{\max} \leq 8$, sillä pelilaudan kaksi keskenään vaihdettavaa osaa eivät saa leikata toisiaan. Toisaalta pelilaudalta voisi helposti vaihtaa keskenään vaikkapa kaksi 4×11 -kokoista aluetta, mutta tehokkuussyistä alueiden vaihtoa on rajoitettu siten, että vaihdettavan alueen koko voi olla ainoastaan puolet pelilaudan mitoista.

Muñoz ja muut mainitsivat käyttäneensä testeissään seuraavia arvoja: $C_{\min} = 2$, $C_{\max} = 8$, $M_{\min} = 1$ ja $M_{\max} = 10$. Muñozin ja muiden lähdekoodeissa on kuitenkin käytetty arvoja $C_{\min} = 2$, $C_{\max} = 10$, $M_{\min} = 1$ ja $M_{\max} = 8$. Jälkimmäisiä arvoja on käytetty myös omissa testiajoissani tähän mennessä, sillä kuten edellä mainittiin, ei parametri M_{\max} voi edes olla nykyisellä Muñozin ja muiden algoritmiin perustuvassa toteutuksessa suurempi kuin 8. Näin ollen epäilisin Muñozin ja muiden mainitsemien parametrien sekaantuneen ephuomiossa, tai sitten olen saanut vanhemmat lähdekoodit, joissa edellä mainittu rajoitus vielä esiintyy.

Seuraavat testit ajetaan siten, että $C_{\min} \in \{1, 2, 3\}$ ja $M_{\min} \in \{1, 2, 3\}$. Parametreille C_{\min} ja M_{\min} ei anneta suurempaa arvoa kuin 3, koska risteytys- ja mutaatio-operaatioiden on syytä pystyä tekemään pelilaudalle myös pieniä, hyvin paikallisia muutoksia. Parametrien C_{\max} ja M_{\max} osalta pitäydytään Muñozin ja muiden valitsemien vastaavien arvojen ympärillä. Täten valitaan näille parametreille arvoiksi $C_{\max} \in \{8, 9, 10, 11, 12\}$ ja $M_{\max} \in \{5, 6, 7, 8\}$.

Testeissä ei ajeta kaikkia edellä mainittuja kombinaatioita läpi suuren testiajomäärän vuoksi, vaan ensin suoritetaan testit varioiden parametreja C_{\min} ja C_{\max} , jonka jälkeen varioidaan parametreja M_{\min} ja M_{\max} . Tämän jälkeen valitaan lupaavimmat arvot em. parametreille ja varioidaan vielä yksilön valintaan käytettävän turnauksen kokoa.

Testiajo	C_{\min}	C_{\max}	M_{\min}	M_{\max}	MAX	MIN	MEAN	STDEV
GA_75	1	8	1	8	370	348	364,0	6,93
GA_76	1	9	1	8	387	380,3	380,3	4,37
GA_77	1	10	1	8	391	381	385,8	3,49
GA_78	1	11	1	8	398	381	389,7	5,01
GA_79	1	12	1	8	399	379	388,3	6,25
GA_80	2	8	1	8	399	387	391,2	3,94
GA_81	2	9	1	8	399	391	394,7	2,67
GA_29B	2	10	1	8	407	383	394,8	6,48
GA_83	2	11	1	8	397	207	372,8	58,42
GA_84	2	12	1	8	401	133	274,2	123,75
GA_85	3	8	1	8	405	380	394,0	8,06
GA_86	3	9	1	8	405	225	366,8	63,77
GA_87	3	10	1	8	213	150	186,6	20,88
GA_88	3	11	1	8	140	116	126,5	8,72
GA_89	3	12	1	8	119	114	116,1	1,66

Taulukko 6. Risteytysoperaation parametrien varioinnin vaikutus tuloksiin.

Taulukko 6 sisältää tutulla tavalla raportoidut pistemäärät parametrien C_{\min} ja C_{\max} varioinnin jälkeen. Ehkä hieman yllätyksettömästi Muñozin ja muiden alun perin valitsemit parametrit tuottivat tässä tapauksessa parhaat pisteet testiajolla GA_29B. Melkein täysin samaan keskimääräiseen tulokseen tosin päätyi testiajo GA_81, jolla parametrien arvot olivat $C_{\min} = 2$ ja $C_{\max} = 9$. Kuitenkin testiajon GA_29B paras yksittäinen tulos 407 pistettä on yhä rikkomatta.

Testiajo	C_{\min}	C_{\max}	M_{\min}	M_{\max}	MAX	MIN	MEAN	STDEV
GA_90	2	10	1	5	400	383	392,6	4,40
GA_91	2	10	1	6	397	386	390,7	3,43
GA_92	2	10	1	7	404	389	394,4	4,99
GA_29B	2	10	1	8	407	383	394,8	6,48
GA_93	2	10	2	5	402	383	393,7	6,07
GA_95	2	10	2	6	401	381	390,8	6,61
GA_96	2	10	2	7	398	378	390,4	6,59
GA_97	2	10	2	8	403	387	395,0	4,69
GA_98	2	10	3	5	405	384	393,6	7,57
GA_99	2	10	3	6	401	381	395,3	5,96
GA_100	2	10	3	7	399	382	391,2	4,94
GA_101	2	10	3	8	400	378	392,0	6,78

Taulukko 7. Mutaatio-operaation parametrien varioinnin vaikutus tuloksiin.

Kaiken kaikkiaan Muñozin ja muiden algoritmissa alun perinkin esiintynyt parametriyhdistelmä $C_{\min} = 2$, $C_{\max} = 10$, $M_{\min} = 1$ ja $M_{\max} = 8$ vaikuttaisi ajettujen testien perusteella olevan melko lähellä optimaalista, joskin omilla testeilläni vielä hieman parempaan keskimääräiseen tulokseen ylsi testiajon GA_99:n yhdistelmä $C_{\min} = 2$, $C_{\max} = 10$, $M_{\min} = 3$ ja $M_{\max} = 6$ keskimääräisellä pistemäärällä 395,3 (ks. Taulukko 7). Täten kyseisen testiajon parametrit valitaan vielä viimeiseen Algoritmin 1 testiajoihin, jossa tutkitaan turnaukseen vaikutusta tuloksiin.

Testiajo	C_{\min}	C_{\max}	M_{\min}	M_{\max}	Turnauskoko	MAX	MIN	MEAN	STDEV
GA_102	2	10	3	6	1	46	39	41,2	2,10
GA_103	2	10	3	6	2	115	101	108,2	5,87
GA_104	2	10	3	6	3	398	389	393,1	3,41
GA_105	2	10	3	6	4	394	377	384,4	6,02
GA_106	2	10	3	6	5	372	353	362,4	6,60
GA_107	2	10	3	6	6	360	334	347,2	9,67

Taulukko 8. Turnaukseen vaikutus tuloksiin.

Taulukosta 8 nähdään, että selvästi paras tulos saavutetaan turnaukseen ollessa kolme. Tätä pienemmät turnaukset tuottavat paljon satunnaisempia yksilöitä koko populaatiosta,

joka vaikuttaa negatiivisesti lopulliseen pistemäärään. Erityisen selvästi tämä näkyy yhden yksilön kokoisilla turnajaisilla, joka käytännössä katsoen tarkoittaa satunnaisvalintaa.

Toisaalta tällöin voisi kuvitella, että mahdollisimman suuret turnajaiset tuottaisivat laadukkaampia yksilöitä, ja tätä kautta algoritmin lopputuloskin olisi erityisen hyvä. Kuten voimme huomata, näin ei kuitenkaan tapahdu, vaan tulos kääntyy itseasiassa laskuun, jos turnauksen koko kasvaa kolmea suuremmaksi. Tämä selittyy todennäköisimmin sillä, että suurilla turnajaisilla populaatiosta tullaan valinneeksi pitkällä aikavälillä useasti vain melko pieni osajoukko koko populaation yksilöistä. Tällöin vaikutus on käytännössä sama kuin jos populaation kokoa pienennettäisiin.

Algoritmin 1 testiajoista voidaan todeta, että mitä todennäköisimmin Muñoz ja muut olivat jo omalla tahollaan testanneet parametreja tarkemminkin, vaikka tämä ei käynytkään heidän tutkimuksestaan ilmi. Vaikka Algoritmin 1 toteutuksen pitäisi olla toiminnallisesti yksityiskohtia myöten samanlainen kuin Muñozin ja muiden geneettisen algoritmin toteutus, sain silti samoilla parametreilla parempia tuloksia aikaan. Pitkähköjen testiajojen jälkeen pystyin vielä hiomaan parametrien arvoja siten, että paras keskimääräinen pistetulos nousi 395,3 pisteeseen. Toisaalta on hyvä huomioida, että vaikka kaikissa testiajoissani oli mukana kymmenen erillistä ajoa, voi keskimääräinen pistetulos silti vaihdella siinä määrin, että jokin muu testiajo kuin nyt parhaaksi valittu voisi aivan hyvin nousta kärkeen jollain toisella ajokerralla. Tietenkin tämä saattaa tarkoittaa myös sitä, että algoritmin suorituskyvyn yläraja alkaa tulla tällä toteutuksella vastaan, koska erot eri testisarjojen välillä jäivät paikoin hyvinkin pieniksi.

5.3. Algoritmin 2 vertailua Muñozin ja muiden tuloksiin

Tässä kohdassa verrataan Muñozin ja muiden monitavoitteista evolutiivista algoritmia (tästä eteenpäin *MOEA*, kuten Muñozin ja muiden julkaisussa) Algoritmiin 2, joka on oma toteutukseni Muñozin ja muiden *MOEA*:sta.

Kuten Algoritmin 1 tapauksessa, myös *MOEA*:n ja Algoritmin 2 toimintaa ohjavat pääasiassa populaation rakennetta määrittävät parametrit, eli eliitti-, risteytys- ja mutaatioyksilöiden määrä. Koska *MOEA* on perusluonteeltaan monitavoitteinen, ei tavoitteiden määrää tässä tapauksessa varioida juuri lainkaan, vaan tämän kohdan testiajoissa käytetään Muñozin ja muiden tavoin kolmea tavoitetta mahdollisimman yhdenmukaisten parametrien saavuttamiseksi. Taulukossa 9 on Taulukon 1 tavoin Muñozin ja muiden tulokset sekä jälkimmäisenä vastaavat omat tulokseni.

Testiajo	Eliit./Rist./Mut.	Pistelaskun tavoitteet	Esiratk.	MAX	MIN	MEAN	STDEV
MOEA_1a	10/9000/990	1, 2, 3	Ei	365/379	346/362	356,7/370,3	5,98/5,72
MOEA_1b	10/9000/990	1, 2, 3	Kyllä	394/381	382/369	387,6/377,0	2,94/4,16
MOEA_2a	10/5000/4990	1, 2, 3	Ei	364/383	346/366	358,3/372,9	4,90/6,4
MOEA_2b	10/5000/4990	1, 2, 3	Kyllä	396/381	388/370	392,5/375,2	2,66/4,34

Taulukko 9. Muñozin ja muiden MOEA-tulokset rinnakkain omien vastaavien tulosteni kanssa.

Testiajojen tulokset ovat hieman lähempänä Muñozin ja muiden tuloksia kuin Algoritmin 1 tapauksessa, mutta pieniä eroja on silti havaittavissa. Kaikki omat tulokseni sijoittuivat melko pienelle pistealueelle lähestulkoon parametreista riippumatta, kun Muñozin ja muiden tuloksissa puolestaan näkyi selvemmin esiratkaisun vaikutus tuloksiin. Muñoz ja muut pääsivät parempiin keskimääräisiin tuloksiin esiratkaistuja pelilautoja käytettäessä, kun omat tulokseni puolestaan päihittävät Muñozin ajot testiajoilla, joissa pelilautoja ei esiratkaistu mitenkään. Tällä kertaa tuloksissa ei Algoritmin 1 tavoin ollut erityistä hajontaa, vaan tulokset olivat testiajojen kaikilla kymmenellä ajokerralla melko samansuuntaisia.

5.4. Algoritmin 2 tuloksia

Algoritmin 2 testiajot suoritettiin samansuuntaisesti kuin Algoritmin 1 ajot, sillä molemmat algoritmit hyödyntävät käytännössä samoja parametreja. Pistelaskun tavoitteiden variointia ei tässä tosin hyödynnetty lainkaan, sillä Algoritmin 2 toiminta nimenomaisesti perustuu useaan tavoitteeseen. Kuten aikaisemminkin, niin myös nytkin käytössä on kaikki neljä pistelaskun tavoitetta. Algoritmin 1 tuloksista voitiin huomata, että pelilaudan esiratkaistu oli käytännössä aina kannattavaa. Tästä syystä (ja myös ylimääräisten hitaiden testiajojen karsimiseksi) päätin ajaa testiajot ainoastaan esiratkaistuilla pelilautoilla.

Koska kaikki pistelaskun tavoitteet ovat kaiken aikaa käytössä ja pelilaudat ovat aina esiratkaisuna, on populaation rakenne ainoa asia, jota tässä vaiheessa kannattaa varioida. Muñozin ja muiden MOEA-tulosten raportoinnissa oli listattu vain kahden erilaisen populaation rakenteen mukaiset testiajot. Ensimmäisessä ajossa risteytysyksilöitä oli 9000 ja mutaatioyksilöitä 990, toisessa ajossa vastaavat määrät olivat 5000 ja 4990. Molemmissa ajoissa eliittiyksilöitä oli kymmenen kappaletta.

Päätin toisaalta yllä mainittujen parametrien pohjalta ja osaltaan Algoritmin 1 tulosten ohjaamina ajaa Algoritmin 2 ensimmäiset varsinaiset testiajot siten, että testiajojen risteytysyksilöiden määrä vaihtelee 4500 yksilöstä 9500 yksilöön 500 yksilön välein. Lisäksi jokaista mainittua testiajoa ajetaan siten, että eliittiyksilöiden määrä on aluksi 10 kuten Muñozilla ja muilla, ja tämän jälkeen ajetaan vielä testiajot eliittiyksilöiden määrällä 0, 5 ja

15, kun risteytysyksilöt ovat kuten yllä. Loput populaatiosta täytetään kaikissa tapauksissa mutaatioyksilöillä siten, että populaation kokonaismäärä on 10000 yksilöä.

Algoritmi 2 sisältää Algoritmista 1 poiketen mekanismin, joka tarkkailee algoritmin suorituksen aikana pistemäärän kehittymistä. Tutkin Algoritmin 1 tuloksia ja tein myös alustavia ajoja Algoritmilla 2 ja huomasin, että jos algoritmien tulos lähti kasvamaan jomelko aikaisesta sukupolvesta lähtien, pysähtyi tuloksen kasvu vasta lopulliseen pistemäärään, eikä algoritmien suoritusten aikana pistemäärän kehityksessä esiintynyt erityisiä tasankokohtia. Täten, jos algoritmin tulos ei ole viidensadan sukupolven aikana parantunut, katsotaan tuloksen olevan sama algoritmin suorituksen päättymiseen asti (eli viidentuhannen sukupolven loppuun saakka). Tällainen mekanismi oli tarpeen senkin takia, että Algoritmin 2 testiajot olivat todella hitaita.

Taulukko 10 sisältää Algoritmin 2 ne alustavat testiajot, joissa eliittiyksilöiden määrä on kymmenen.

Testiajo	Eliit./Rist./Mut.	Pistelaskun tavoitteet	Esiratk.	MAX	MIN	MEAN	STDEV
MOEA_3	10/4500/5490	1, 2, 3, 4	Kyllä	325	269	289,8	18,36
MOEA_4	10/5000/4990	1, 2, 3, 4	Kyllä	392	380	384,9	4,04
MOEA_5	10/5500/4490	1, 2, 3, 4	Kyllä	392	378	385,1	4,75
MOEA_6	10/6000/3990	1, 2, 3, 4	Kyllä	392	383	387,4	3,06
MOEA_7	10/6500/3490	1, 2, 3, 4	Kyllä	395	380	386,7	4,47
MOEA_8	10/7000/2990	1, 2, 3, 4	Kyllä	392	377	384,6	4,81
MOEA_9	10/7500/2490	1, 2, 3, 4	Kyllä	393	371	385,5	6,69
MOEA_10	10/8000/1990	1, 2, 3, 4	Kyllä	394	381	386,2	4,37
MOEA_11	10/8500/1490	1, 2, 3, 4	Kyllä	393	372	384,3	6,33
MOEA_12	10/9000/990	1, 2, 3, 4	Kyllä	397	379	386,2	5,96
MOEA_13	10/9500/490	1, 2, 3, 4	Kyllä	390	368	380,8	6,60

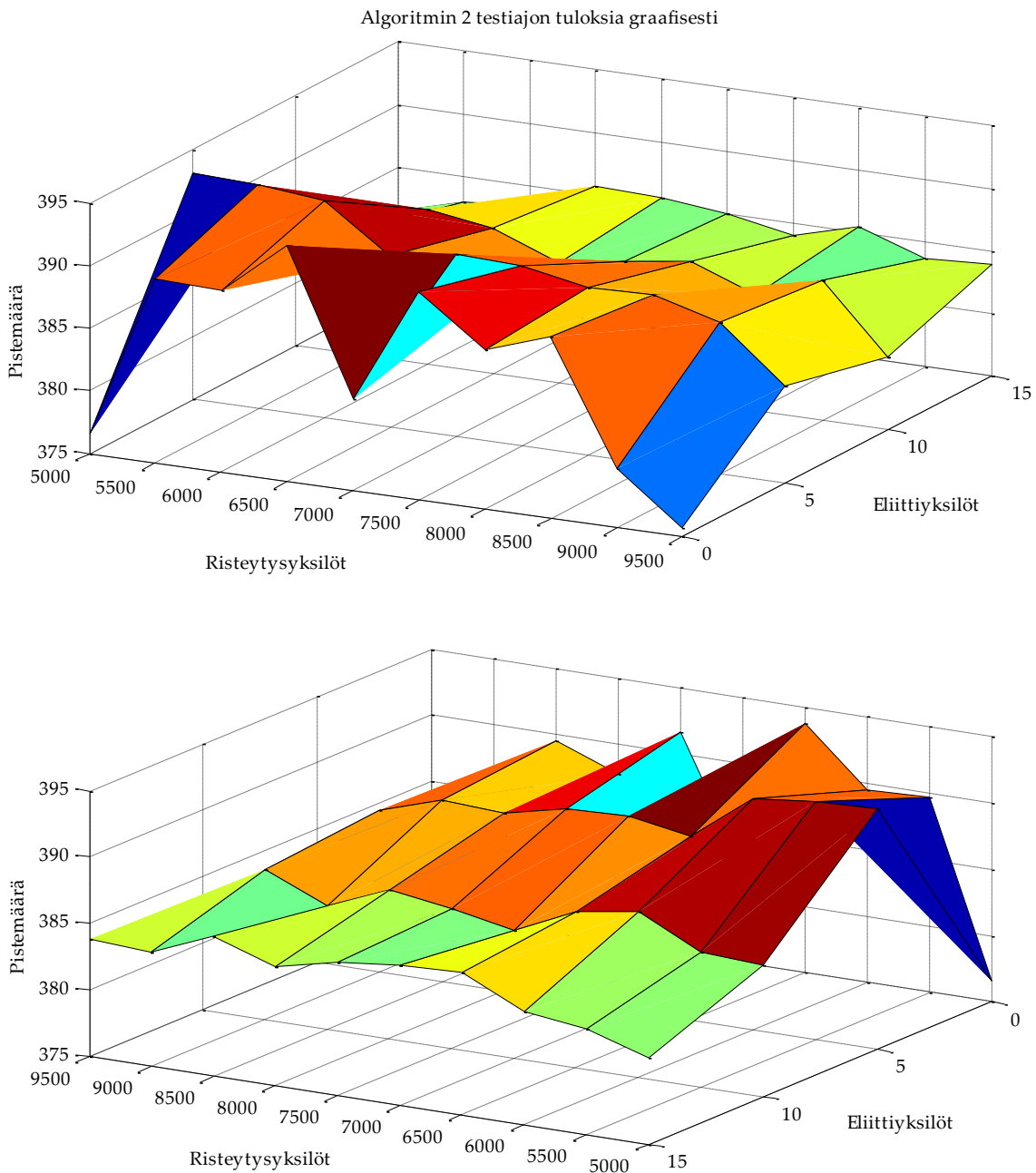
Taulukko 10. Algoritmin 2 testiajoja kymmenellä eliittiyksilöllä.

Tulokset ovat kauttaaltaan melko tasaisia lukuunottamatta testiajoa MOEA_3, jonka pistemäärä oli paljon pienempi kuin muiden testiajojen. Kuten kohdassa 5.2, voimme myös Algoritmin 2 tapauksessa verrata kolmen ja neljän tavoitteen tuloksia parin testiajon osalta.

Tarkastellessa esiratkaistuja pelilautoja testiajo MOEA_1b:n keskimääräinen tulos oli omissa ajoissani 377,0 pistettä. Tätä vastaava neljän tavoitteen testiajo MOEA_12 tuotti keskimäärin 386,2 pistettä. Testiajo MOEA_2b pistemäärällä 375,2 ei myöskään pärjännyt

kilpakumppani MOEA_4:lle, jonka keskimääräinen pistetulos oli 384,9. Näyttäisi siis siltä, että neljäs tavoite parantaa Algoritmin 2 tuottamaa pistemäärää jonkin verran.

Kuva 7 sisältää graafisessa muodossa niiden alustavien testiajojen tulokset, joissa käytetään nollaa, viittä ja viittätoista eliittiyksilöä (poislukien testiajot, joissa käytetään 4500 risteytysyksilöä). Taulukossa 11 on puolestaan ajatun testisarjan kaikki tulokset.



Kuva 7. Algoritmin 2 testiajojen tuloksia graafisessa muodossa eri kulmista

Testiajo	Eliit./Rist./Mut.	Pistelaskun tavoitteet	Esiratk.	MAX	MIN	MEAN	STDEV
MOEA_14	0/4500/5500	1, 2, 3, 4	Kyllä	285	250	266,4	10,86
MOEA_15	5/4500/5495	1, 2, 3, 4	Kyllä	342	260	291,2	28,03
MOEA_16	15/4500/5485	1, 2, 3, 4	Kyllä	360	278	321,7	33,43
MOEA_17	0/5000/5000	1, 2, 3, 4	Kyllä	400	304	376,6	36,92
MOEA_18	5/5000/4995	1, 2, 3, 4	Kyllä	398	388	393,1	3,35
MOEA_19	15/5000/4985	1, 2, 3, 4	Kyllä	388	375	381,5	3,87
MOEA_20	0/5500/4500	1, 2, 3, 4	Kyllä	396	381	389,7	4,45
MOEA_21	5/5500/4495	1, 2, 3, 4	Kyllä	401	386	392,9	4,91
MOEA_22	15/5500/4485	1, 2, 3, 4	Kyllä	392	377	382,9	4,75
MOEA_23	0/6000/4000	1, 2, 3, 4	Kyllä	398	383	389,5	4,20
MOEA_24	5/6000/3995	1, 2, 3, 4	Kyllä	397	385	392,4	3,84
MOEA_25	15/6000/3985	1, 2, 3, 4	Kyllä	388	378	383,5	3,69
MOEA_26	0/6500/3500	1, 2, 3, 4	Kyllä	401	387	393,8	5,73
MOEA_27	5/6500/3495	1, 2, 3, 4	Kyllä	396	380	388,8	4,59
MOEA_28	15/6500/3485	1, 2, 3, 4	Kyllä	391	382	385,7	3,20
MOEA_29	0/7000/3000	1, 2, 3, 4	Kyllä	403	385	382,3	5,36
MOEA_30	5/7000/2995	1, 2, 3, 4	Kyllä	398	380	389,6	5,93
MOEA_31	15/7000/2985	1, 2, 3, 4	Kyllä	392	378	385,5	4,33
MOEA_32	0/7500/2500	1, 2, 3, 4	Kyllä	398	387	391,7	4,00
MOEA_33	5/7500/2495	1, 2, 3, 4	Kyllä	393	384	389,4	2,72
MOEA_34	15/7500/2485	1, 2, 3, 4	Kyllä	392	376	385,0	4,55
MOEA_35	0/8000/2000	1, 2, 3, 4	Kyllä	399	378	387,8	7,79
MOEA_36	5/8000/1995	1, 2, 3, 4	Kyllä	398	379	388,4	6,64
MOEA_37	15/8000/1985	1, 2, 3, 4	Kyllä	397	375	384,0	7,13
MOEA_38	0/8500/1500	1, 2, 3, 4	Kyllä	396	384	389,6	3,84
MOEA_39	5/8500/1495	1, 2, 3, 4	Kyllä	395	377	388,6	6,06
MOEA_40	15/8500/1485	1, 2, 3, 4	Kyllä	394	378	385,5	5,25
MOEA_41	0/9000/1000	1, 2, 3, 4	Kyllä	388	368	379,8	6,21
MOEA_42	5/9000/995	1, 2, 3, 4	Kyllä	399	362	387,1	10,19
MOEA_43	15/9000/985	1, 2, 3, 4	Kyllä	390	374	383,6	5,80
MOEA_44	0/9500/500	1, 2, 3, 4	Kyllä	386	362	375,8	7,94
MOEA_45	5/9500/495	1, 2, 3, 4	Kyllä	399	353	382,8	12,34
MOEA_46	15/9500/485	1, 2, 3, 4	Kyllä	388	379	383,9	2,69

Taulukko 11. Algoritmin 2 tuloksia nollalla, viidellä ja viidellätoista eliittiyksilöllä.

Saadut tulokset noudattavat keskimäärin samaa linjaa kuin kymmenenkin eliittiyksilön ajot Taulukossa 10. Ne testiajot, joissa käytetään 4500 risteytysyksilöä, eivät kuitenkaan tunnu pärjäävän kovinkaan hyvin kummassakaan testiajosarjassa. Taulukossa 11 on muutamia keskimääräistä paremmin pärjänneitä testiajoja, mutta näillä ei näyttäisi olevan selkeää toisiaan yhdistävää tekijää. Kuvan 7 perusteella voidaan havaita, että kymmenen ja viisitoista eliittiyksilöä sisältävät testiajot eivät ole pistemäärältään kärkijoukossa. Korkeampia pistemääriä sen sijaan edustavat melko hajanaiset parametriarvot. Tuloksia aikani tulkittuani päädyin ajamaan tarkentavat testisarjat seuraavaksi kuvatuin parametrein.

Ilman eliittiyksilöitä paras tulos saavutettiin 6500 ja 7500 risteytysyksilön alueella. Tästä syystä lisätetit ajetaan ilman eliittiyksilöitä siten, että risteytysyksilöitä on 5750, 6250, 7250 ja 7750 kappaletta. Myös alue nollan ja viiden eliittiyksilön välillä näyttää kiinnostavalta. Testisarjaan lisätään täten ajot kolmella eliittiyksilöllä siten, että risteytysyksilöt ovat välillä 5000–7750 kahdensadanviidenkymmenen yksilön välein. Viidellä eliittiyksilöllä on saavutettu myös melko korkeita pisteitä risteytysyksilöiden määrän ollessa 5000–6000 kappaleen tienoilla, joten tehdään ajot tällä eliittiyksilömäärällä risteytysyksilöiden määrän ollessa 5250 ja 5750. Varmuuden vuoksi mukana on vielä seitsemällä eliittiyksilöllä testiajot, joissa risteytysyksilöitä on 5000, 5250, 5500, 5750 ja 6000 kappaletta. Tämän jatkotestisarjan kymmenen parasta tulosta on listattu Taulukossa 12.

Testiajo	Eliit./Rist./Mut.	Pistelaskun tavoitteet	Esiratk.	MAX	MIN	MEAN	STDEV
MOEA_51	3/5000/4997	1, 2, 3, 4	Kyllä	407	385	396,7	6,82
MOEA_52	3/5250/4747	1, 2, 3, 4	Kyllä	405	389	396,7	4,74
MOEA_53	3/5500/4497	1, 2, 3, 4	Kyllä	398	389	394,4	3,03
MOEA_47	0/6250/3750	1, 2, 3, 4	Kyllä	398	386	394,3	4,00
MOEA_56	3/6250/3747	1, 2, 3, 4	Kyllä	401	382	392,8	5,43
MOEA_55	3/6000/3997	1, 2, 3, 4	Kyllä	398	387	392,5	3,75
MOEA_64	5/5750/4245	1, 2, 3, 4	Kyllä	399	379	392,5	5,56
MOEA_54	3/5750/4247	1, 2, 3, 4	Kyllä	398	385	392,1	4,82
MOEA_59	3/7000/2997	1, 2, 3, 4	Kyllä	401	384	392,0	5,21
MOEA_63	5/5250/4745	1, 2, 3, 4	Kyllä	398	382	391,5	5,66

Taulukko 12. Algoritmin 2 jatkotestisarjan kymmenen parasta tulosta.

Saman parhaan keskimääräisen tuloksen saavuttivat testiajot MOEA_51 ja MOEA_52 pistemäärällä 396,7. Tämä on samalla korkein tähän mennessä saavutettu keskimääräinen pistemäärä. Myös testiajot MOEA_53 ja MOEA_47 olivat lähellä kärkeä tuloksillaan 394,4 ja 394,3 pistettä. Koska korkeimmat pistemäärät löytyivät tässä tapauksessa selkeästi kol-

men eliittiyksilön ryhmästä risteytysyksilöiden määrän ollessa 5000–5250, ajetaan vielä yksi testisarja, jossa on mukana sellaiset testiajojen kombinaatiot, joissa eliittiyksilöitä on 1, 2, 3 ja 4 kappaletta, sekä risteytysyksilöitä 5000, 5125, 5250 ja 5375 kappaletta (osa mainituista testiajoista ajettiin edellisen testisarjan yhteydessä). Tämän tarkennetun sarjan tulokset nähdään Taulukossa 13.

Testiajo	Eliit./Rist./Mut.	Pistelaskun tavoitteet	Esiratk.	MAX	MIN	MEAN	STDEV
MOEA_70	1/5000/4999	1, 2, 3, 4	Kyllä	404	236	365,7	62,78
MOEA_71	1/5125/4874	1, 2, 3, 4	Kyllä	405	388	395,2	5,79
MOEA_72	1/5250/4749	1, 2, 3, 4	Kyllä	404	173	372,9	70,44
MOEA_73	1/5375/4624	1, 2, 3, 4	Kyllä	402	391	396,3	3,23
MOEA_74	2/5000/4998	1, 2, 3, 4	Kyllä	405	385	396,0	5,50
MOEA_75	2/5125/4873	1, 2, 3, 4	Kyllä	401	383	395,1	4,95
MOEA_76	2/5250/4748	1, 2, 3, 4	Kyllä	404	233	377,6	51,14
MOEA_77	2/5375/4623	1, 2, 3, 4	Kyllä	403	379	393,8	6,80
MOEA_51	3/5000/4997	1, 2, 3, 4	Kyllä	407	385	396,7	6,82
MOEA_78	3/5125/4872	1, 2, 3, 4	Kyllä	407	390	396,5	4,79
MOEA_52	3/5250/4747	1, 2, 3, 4	Kyllä	405	389	396,7	4,74
MOEA_79	3/5375/4622	1, 2, 3, 4	Kyllä	399	383	391,6	4,58
MOEA_80	4/5000/4996	1, 2, 3, 4	Kyllä	396	355	386,7	12,62
MOEA_81	4/5125/4871	1, 2, 3, 4	Kyllä	397	231	377,2	51,47
MOEA_82	4/5250/4746	1, 2, 3, 4	Kyllä	402	383	394,3	6,07
MOEA_83	4/5375/4621	1, 2, 3, 4	Kyllä	402	387	393,6	5,66

Taulukko 13. Algoritmin 2 tarkennetun testisarjan tulokset.

Tarkennetuista tuloksista nähdään, ettei pistemäärä enää parane testiajojen MOEA_51 ja MOEA_52 tuloksista, joten tarkempia populaation rakenteen parametreja ei enää etsitä. Valitaan MOEA_51 parhaaksi testiajoksi sen testiajoa MOEA_52 hivenen korkeamman maksimipistemäärän takia, ja ajetaan Algoritmin 1 testauksen tavoin vielä muutama testiajo, joissa tutkitaan algoritmin sisäisten parametrien vaikutusta tulokseen.

Koska Algoritmien 1 ja 2 populaatioiden muodostus on täysin identtinen ja algoritmit eroavat ainoastaan lähinnä yksilöiden sopivuuden määrittelyssä, ei sisäisiä parametreja lähdetä tässä erityisen laajasti muuntelemaan. Algoritmin 1 sisäisten parametrien varioinnissa saatujen tulosten perusteella ajetaan Algoritmillla 2 sellaiset testiajot, joissa risteytysoperaation käyttämän alueen minimi- ja maksimitat ovat $C_{\min} \in \{1, 2\}$ ja $C_{\max} \in \{9, 10, 11\}$, kun $M_{\min} = 1$ ja $M_{\max} = 8$. Varioidaan vastaavasti mutaatio-operaation

alueen minimi- ja maksimiparametreja siten, että $M_{\min} \in \{1, 2\}$ ja $M_{\max} \in \{6, 7, 8\}$, kun $C_{\min} = 2$ ja $C_{\max} = 10$. Varioimalla saadut tulokset esitellään Taulukossa 14.

Testiajo	C_{\min}	C_{\max}	M_{\min}	M_{\max}	MAX	MIN	MEAN	STDEV
MOEA_86	1	9	1	8	392	383	387,4	3,47
MOEA_87	1	10	1	8	399	380	389,4	5,40
MOEA_88	1	11	1	8	402	386	391,3	4,67
MOEA_89	2	9	1	8	401	316	386,7	25,24
MOEA_51	2	10	1	8	407	385	396,7	6,82
MOEA_90	2	11	1	8	398	293	367,1	39,68
MOEA_91	2	10	1	6	396	384	391,2	4,52
MOEA_92	2	10	1	7	400	333	388,2	19,87
MOEA_51	2	10	1	8	407	385	396,7	6,82
MOEA_93	2	10	2	6	396	246	327,3	45,33
MOEA_94	2	10	2	7	338	199	297,3	37,69
MOEA_95	2	10	2	8	350	210	274,6	39,27

Taulukko 14. Algoritmin 2 sisäisten parametrien vaikutus algoritmin pistemäärään.

Toisin kuin Algoritmin 1 testeissä, tässä tapauksessa algoritmin sisäisten parametrien muuntelu ei tuottanut enää parempaa tulosta, vaan korkeimmaksi pistemääräksi jää edelleen testiajon MOEA_51 keskimääräinen tulos 396,7 pistettä. Erikoista oli se, että vaikka Algoritmien 1 ja 2 populaatioon kohdistuvat muokkaavat operaatiot ovat täysin samanlaisia, oli algoritmien sisäisten parametrien varioinnilla kuitenkin täysin erilaiset tulokset. Tämä toki saattaa johtua siitä, että kun populaation sopivuus arvioidaan muutosten jälkeen, Algoritmi 2 painottaa populaation yksilöiden sopivuutta kovin eri tavoin kuin Algoritmi 1. Tämä saattaa näkyä pitkän suorituksen aikana pistemäärän erilaisena käyttäytymisenä.

Tutkitaan vielä lopuksi, kuinka algoritmin yksilöiden valinnassa käyttämä turnajaisten koko vaikuttaa lopputulokseen. Tässä viimeisessä Algoritmin 2 testisarjassa on käytetty edelleen tähän mennessä parhaat pisteet saanutta testiajoa MOEA_51 tavanomaisilla algoritmin sisäisillä parametreilla $C_{\min} = 2$, $C_{\max} = 10$, $M_{\min} = 1$ ja $M_{\max} = 8$. Turnauskoko T saa arvot kolme, neljä ja viisi. Ajoin myös mielenkiinnon vuoksi sellaisen testiajoryhmän, jossa käytetään Algoritmin 1 parhaaksi havaittuja sisäisiä parametreja $C_{\min} = 2$, $C_{\max} = 10$, $M_{\min} = 3$ ja $M_{\max} = 6$ turnaukseen ollessa $T \in \{3, 4, 5\}$. Taulukko 15 sisältää yllä kuvatun testisarjan tulokset.

Testiajo	C_{\min}	C_{\max}	M_{\min}	M_{\max}	Turnauskoko	MAX	MIN	MEAN	STDEV
MOEA_51	2	10	1	8	3	407	385	396,7	6,82
MOEA_96	2	10	1	8	4	391	374	382,6	6,35
MOEA_97	2	10	1	8	5	389	376	381,9	3,90
MOEA_98	2	10	3	6	3	338	285	310,3	17,87
MOEA_99	2	10	3	6	4	390	376	381,2	4,71
MOEA_100	2	10	3	6	5	385	369	375,2	4,47

Taulukko 15. Turnaukseen vaikutus Algoritmin 2 tuloksiin.

Muuttamalla testiajon MOEA_51 turnauskokoja kolmea yksilöä suuremmaksi aiheuttaa samanlaisen ilmiön kuin Algoritmilla 1, eli tulos huononee hivenen. Erikoista oli toisaalta huomata se, että vaikka testiajo MOEA_98 Algoritmin 1 parhailla sisäisillä parametreilla ei kovin hyvin pärjännytkään 310,3 pisteen tuloksellaan, nosti turnajaiskoon kasvatus neljään pistemäärän kuitenkin melko hyvään 381,2 pisteen tulokseen. Tästä turnauskokoja vielä suurentamalla pistemäärä kuitenkin kääntyi hienoiseen laskuun. Saaduista tuloksista voi siis selvästi huomata sen, ettei turnaukseen optimaalinen arvo ole lainkaan yksiselitteinen, vaan sekin riippuu suuresti muista algoritmin parametreista.

Tältä osalta evolutiivisten algoritmien Algoritmin 1 ja Algoritmin 2 testit päättyvät. Tarkoituksena oli ajaa melko kattavat testisarjat varioiden mahdollisimman monilta osin algoritmien erilaisia parametreja, ja Muñozin ja muiden tuloksiin verrattuna algoritmien suorituskykyyn tulikin joitain parannuksia. On kuitenkin selvää, ettei kaikkia mahdollisesti oleellisiakin parametrikombinaatioita käyty tässä läpi, eikä perinpohjainen testaus ole välttämättä mahdollistakaan testien viemän ajan vuoksi. Perinpohjaisen järjestelmällisen testaamisen sijaan voisikin olla järkevämpää tutkia eri parametrien välisiä riippuvuuksia, jonka jälkeen algoritmeilla voisi ajaa kohdistetumpia ajoja, joilla pistemäärää voitaisiin mahdollisesti entisestään korottaa.

5.5. Algoritmin 3 tuloksia

Tässä kohdassa käsiteltävä Algoritmi 3 perustuu hill climbing -tekniikkaan, joten se eroaa Algoritmien 1 ja 2 evolutiivisesta toiminnasta hyvin paljon. Algoritmi 3 käsittää suuren populaation sijasta yhden ainoan pelilautayksilön, jota pyritään pienillä muutoksilla ajamaan kohti mahdollisimman korkeaa pistemäärää. Algoritmille 3 ja seuraavan kohdan Algoritmille 4 ei ole olemassa lainkaan vertailutuloksia, vaan testejä lähdetään ajamaan ilman aikaisempaa tietoa sopivista parametreista.

Kuten kohdassa 4.4 kerrottiin, toteutin Algoritmiin 3 kaksi erilaista pelilaudan muunnosoperaatiota. Ensimmäinen muunnosoperaatio on melko suoraviivainen ja se perustuu

raakaan voimaan. Tämä raakan voiman tekniikka valitsee pelilaudalta satunnaisen palan, jonka paikkaa yritetään vaihtaa vuorotellen jokaisen seuraavan palan kanssa kokeillen samalla jokaista mahdollista kiertoa näiden palojen välillä. Kuten kohdassa 4.4 todettiin, tällaista raakaan voimaan perustuvaa muunnosoperaatiota kutsutaan nimellä Muunnostyyppi 1.

Toinen muunnosoperaatio sisältää Algoritmeista 1 ja 2 jo tutuksi tulleet operaatiot. Tämä muunnosoperaatio sisältää kaksi erilaista muunnostapaa, jotka ovat kääntömuunnos ja vaihtomuunnos. Vaihtomuunnos valitsee pelilaudalta kaksi samanmuotoista erillistä aluetta, joiden paikkoja vaihdetaan. Kääntömuunnos puolestaan valitsee pelilaudalta satunnaisesta kohdasta neliön muotoisen alueen, joka ikäänkuin nostetaan kokonaan irti pelilaudasta ja käännetään edelleen kokonaisena satunnaiseen suuntaan ja lasketaan takaisin pelilaudalle. Käyttäjä voi määrittellä muunnossuhteen p , joka tarkoittaa sitä todennäköisyyttä, jolla vaihtomuunnos valitaan. Vastaavasti kääntömuunnos valitaan todennäköisyydellä $1 - p$. Tätä muunnostapaa kutsuttiin puolestaan nimellä Muunnostyyppi 2. Muunnostyyppillä 2 käyttäjä voi myös määrittellä, miten toimitaan tilanteissa, joissa pelilaudan tulos ei parantunut satunnaisen muutoksen jälkeen. Käyttäjä voi päättää, palauteaanko pelilaudan tila muutosta edeltäneeseen hetkeen, vai valitaanko muokattu pelilauta suoraan seuraavalle kierrokselle. Kutsutaan tätä parametria nimellä *pMuutos*. Pelilaudan muutos palautetaan aina muutosta edeltävään tilaan, kun parametrin *pMuutos* arvo on tosi.

Valittiin kumpi muunnostyyppi hyvänsä, niin molemmissa tapauksissa pelilaudan pistemäärä lasketaan kunkin muutoksen jälkeen. Jos muutos kasvattaa pistemäärää, aloitetaan kierros alusta uuden muunnellun pelilaudan kanssa. Muussa tapauksessa muutos palautetaan ja algoritmin suoritusta jatketaan kunnes kaikki mahdolliset vaihdot on koekeltu (Muunnostyyppi 1) tai kun tietty määrä muutoksia on tehty (Muunnostyyppi 2).

Aloitetaan testaus Muunnostyyppillä 1. Tässä tapauksessa varioitavia parametreja on melko vähän, sillä hill climb -algoritmi on toiminnaltaan yksinkertainen ja täten suuri osa algoritmin toiminnasta on kiinnitetty itse algoritmin rakenteeseen. Ainoat kontrolloitavat parametrit ovat Muunnostyyppiä 1 käytettäessä pistelaskun tavoitteet sekä pelilaudan reunojen esiratkaistu. Parametrien vähäisestä määrästä johtuen otin Algoritmilla 3 käyttöön pistelaskun tavoitteille painokertoimet. Nyt, sen sijaan että tietty tavoite vain kytkettäisiin päälle tai pois päältä, voidaan tavoitteille määrittellä suhteellinen merkittävyys välillä 0–100 %. Tämä tarkoittaa sitä, että esimerkiksi Tavoitteelle 1 voidaan asettaa suurempi paino kuin muille tavoitteille, jolloin Tavoite 1 määrää suhteessa suuremman osan pelilaudan kokonaispistemäärästä.

Algoritmin 3 kukin testiajo sisältää kymmenen algoritmin suorituskertaa samoilla parametreilla, kuten aikaisemmissakin ajoissa. Koska Muunnostyyppiä 1 käytettäessä algoritmia suoritetaan kaavamaisesti niin kauan kuin pistemäärä paranee, ei käytössä ole eril-

listä parametria, joka rajoittaisi algoritmin suoritusaikaa. Taulukko 16 sisältää erilaisia testiajoja, joissa pistelaskun tavoitteille on asetettu erilaisia painokertoimia. Näissä testiajoissa pelilautaa ei ole esiratkaistu, joten testiajojen perään lisätään tämän takia tunnus 'A'.

Eri testiajojen tavoitteiden painokertoimien muodostamisessa kiinnitettiin huomiota siihen, että yksittäinen tärkein tavoite on Tavoite 1, sillä kyseinen tavoite määrittelee suoraan pelilaudan pistemäärän ja on myös selkein yksittäinen mittari tietyn pelilaudan hyvyyteen. Tavoitteet 2–3 toisaalta tukevat Tavoitetta 1 ja ne myös pyrkivät varmistamaan, että pelilaudan alueita tarkastellaan myös yhtä palaa suurempina kokonaisuuksina. Tavoite 4 on puolestaan siinä mielessä muista tavoitteista eroava, että se ei ota mitään kantaa pelilaudan kokonaispistemäärään. Silti Tavoite 4 on myös tärkeä, sillä se on ainoa tavoite, joka pyrkii sijoittamaan reunapalat pelilaudan laiduille ja säilyttämään täten reunatäsmäyksen säännönmukaisen rakenteen. Näistä syistä Tavoitteita 1 ja 4 on painotettu testiajoissa keskimäärin Tavoitteita 2 ja 3 enemmän.

Testiajo	Tavoite 1	Tavoite 2	Tavoite 3	Tavoite 4	Esiratkaistu	MAX	MIN	MEAN	STDEV
HC_5A	50 %	16,67 %	16,67 %	16,67 %	Ei	378	355	368,0	6,86
HC_10A	40 %	20 %	10 %	30 %	Ei	383	354	365,2	8,22
HC_11A	40 %	10 %	20 %	30 %	Ei	374	355	365,2	6,66
HC_14A	60 %	10 %	10 %	20 %	Ei	374	356	364,9	5,26
HC_16A	40 %	15 %	15 %	30 %	Ei	375	360	364,4	4,60
HC_12A	10 %	30 %	30 %	30 %	Ei	376	352	363,8	7,87
HC_17A	30 %	15 %	15 %	40 %	Ei	371	346	362,3	7,66
HC_7A	40 %	10 %	10 %	40 %	Ei	379	353	362,1	7,85
HC_8A	33,33 %	16,67 %	16,67 %	33,33 %	Ei	370	349	360,7	8,35
HC_9A	20 %	30 %	30 %	20 %	Ei	367	350	358,3	5,72
HC_15A	50 %	10 %	10 %	30 %	Ei	369	351	358,1	4,79
HC_13A	70 %	10 %	10 %	10 %	Ei	367	346	356,2	7,87
HC_6A	50 %	0 %	0 %	50 %	Ei	344	324	355,9	7,02
HC_3A	25 %	25 %	25 %	25 %	Ei	365	348	355,0	5,01
HC_2A	33,33 %	33,33 %	33,33 %	0 %	Ei	362	335	347,7	9,27
HC_4A	75 %	0 %	0 %	25 %	Ei	346	327	339,9	5,70
HC_1A	100 %	0 %	0 %	0 %	Ei	337	317	325,2	6,20

Taulukko 16. Algoritmin 3 alustavia tuloksia ilman reunojen esiratkaistua (Muunnostapa 1).

Taulukko 16 sisältää paremmuusjärjestyksessä ajatut testiajot. Kärkituloksien erot eivät olleet järin suuria, mutta eroa syntyi kuitenkin huonoimpiin tuloksiin verrattuna keskimäärin 30–40 pisteen verran. Kaikkia kärkituloksia yhdistää se, että niissä Tavoittella 1 on aina suurin painoarvo. Tämä puoltaisi yllä mainittua seikkaa siitä, että Tavoite 1 on tämäntyylisten reunatäsmäysojien tärkein pistelaskuri.

Jotta nähtäisiin, onko pelilautojen esiratkaistulla yhteyttä tavoitteiden painoarvoihin, tai parantaako esiratkaistus yleensä tulosten laatua, ajetaan ylläolevat testiajot myös esiratkaistulla pelilautoilla (merkitään tunnuksella 'B' testiajon perässä). Esiratkaistujen pelilautojen testiajot tulokset nähdään Taulukossa 17.

Testiajo	Tavoite 1	Tavoite 2	Tavoite 3	Tavoite 4	Esiratkaistu	MAX	MIN	MEAN	STDEV
HC_5B	50 %	16,67 %	16,67 %	16,67 %	Kyllä	378	364	370,9	4,91
HC_7B	40 %	10 %	10 %	40 %	Kyllä	380	353	368,1	8,38
HC_11B	40 %	10 %	20 %	30 %	Kyllä	375	355	367,8	6,96
HC_14B	60 %	10 %	10 %	20 %	Kyllä	382	355	367,7	7,97
HC_15B	50 %	10 %	10 %	30 %	Kyllä	371	358	364,8	4,02
HC_16B	40 %	15 %	15 %	30 %	Kyllä	374	357	364,5	5,48
HC_17B	30 %	15 %	15 %	40 %	Kyllä	367	359	363,1	3,28
HC_10B	40 %	20 %	10 %	30 %	Kyllä	371	352	361,1	5,53
HC_13B	70 %	10 %	10 %	10 %	Kyllä	366	353	360,9	4,12
HC_9B	20 %	30 %	30 %	20 %	Kyllä	373	348	359,8	8,65
HC_12B	10 %	30 %	30 %	30 %	Kyllä	376	347	359,5	9,01
HC_8B	33,33 %	16,67 %	16,67 %	33,33 %	Kyllä	370	352	358,4	5,32
HC_3B	25 %	25 %	25 %	25 %	Kyllä	362	343	356,0	6,36
HC_2B	33,33 %	33,33 %	33,33 %	0 %	Kyllä	363	342	349,2	8,18
HC_4B	75 %	0 %	0 %	25 %	Kyllä	347	322	338,1	7,22
HC_1B	100 %	0 %	0 %	0 %	Kyllä	346	330	337,9	5,22
HC_6B	50 %	0 %	0 %	50 %	Kyllä	342	329	335,8	4,39

Taulukko 17. Algoritmin 3 alustavia tuloksia esiratkaistulla pelilaudalla (Muunnostapa 1).

Esiratkaistus paransi tuloksia jonkin verran, mutta suurelta osin muutokset olivat melko marginaalisia. Esiratkaistun käyttö ei myöskään muuttanut suuremmin sitä edellä havaittua seikkaa, että parhaiden testiajot joukossa Tavoite 1 oli pääosin dominoiva tavoite.

Yritetään etsiä vielä tarkemmin Algoritmin 3 pistelaskun painokertoimien optimaalisempia arvoja. Vaikka pelilaudan esiratkaistu ei suurta muutosta tuloksiin tehnytkään, otetaan se kuitenkin pienen hyötynsä takia käyttöön. Saadut tulokset ovat paremmuusjärjestyksessä parhaasta huonoimpaan Taulukossa 18.

Testiajo	Tavoite 1	Tavoite 2	Tavoite 3	Tavoite 4	Esiratkaistu	MAX	MIN	MEAN	STDEV
HC_27B	55 %	15 %	15 %	15 %	Kyllä	379	363	372,9	4,75
HC_20B	88 %	4 %	4 %	4 %	Kyllä	380	367	371,3	3,92
HC_21B	85 %	5 %	5 %	5 %	Kyllä	381	361	371,2	7,28
HC_32B	60 %	10 %	20 %	10 %	Kyllä	379	360	370,1	6,71
HC_25B	73 %	9 %	9 %	9 %	Kyllä	378	360	367,6	6,22
HC_31B	60 %	20 %	10 %	10 %	Kyllä	380	361	367,4	5,76
HC_23B	79 %	7 %	7 %	7 %	Kyllä	374	357	366,2	4,52
HC_28B	80 %	5 %	5 %	10 %	Kyllä	373	356	364,1	6,30
HC_22B	82 %	6 %	6 %	6 %	Kyllä	370	357	363,6	5,21
HC_26B	70 %	10 %	10 %	10 %	Kyllä	369	355	362,8	5,51
HC_18B	94 %	2 %	2 %	2 %	Kyllä	369	352	362,3	6,48
HC_19B	91 %	3 %	3 %	3 %	Kyllä	374	347	361,2	8,51
HC_29B	75 %	10 %	10 %	5 %	Kyllä	372	342	361,0	9,08
HC_24B	76 %	8 %	8 %	8 %	Kyllä	378	349	360,8	8,64
HC_30B	45 %	5 %	5 %	45 %	Kyllä	374	352	360,5	7,35

Taulukko 18. Algoritmin 3 tarkennettuja tuloksia (Muunnostapa 1).

Muunnostapaa 1 käyttäen tarkennettujen testiajojen parhaaksi tulokseksi päätyi testiajo HC_27B, jonka keskimääräinen tulos ylsi 372,9 pisteeseen. Tulos ei parantunut järin paljon edellisen testisarjan parhaasta tuloksesta, joka oli 370,9 pistettä. Tuloksista voidaan päätellä, ettei Algoritmin 3 tavoitteiden painokertoimien hienosäädöllä ja pelilaudan esiratkaistulla ole lopputulokseen erityisen suurta vaikutusta.

Otetaan seuraaviin testiajoihin käyttöön Muunnostapa 2. Tällöin algoritmin toimintaperiaate muuttuu hieman, sillä Muunnostavan 2 toiminta ei ole Muunnostavan 1 tapaan järjestelmällistä, vaan pelilautaa muutetaan aina satunnaisesta kohdasta. Tämän takia algoritmin suoritus keskeytetään, jos parannusta pistemäärään ei tule 500000 askeleen jälkeen.

Muunnostavan 1 parametrien lisäksi nyt kontrolloitavia lisäparametreja ovat edellä mainittu muunnossuhde sekä parametri pMuutos, joka määrää, palautetaanko pelilautaan

tehty muutos, jos pistemäärä ei kasva. Lisäksi käytetään muunnosoperaation sisäisiä parametreja M_{\min} ja M_{\max} , joilla on täysin vastaava toiminnallisuus (ja siksi sama nimi) kuin Algoritmien 1 ja 2 mutaatio-operaatioiden parametreilla. Aloitetaan testaus Muunnostavalla 2 ajamalla Taulukon 18 mukaiset testiajot kun muunnossuhde on 0,5, muunnosoperaation sisäiset parametrit ovat $M_{\min} = 1$ ja $M_{\max} = 8$ ja pMuutos on tosi (eli pelilaudan muutos palautetaan jos pistemäärä ei kasva). Taulukko 19 sisältää saadut tulokset.

Testiajo	Tavoite 1	Tavoite 2	Tavoite 3	Tavoite 4	Esiratkaistu	MAX	MIN	MEAN	STDEV
HC_37B	82 %	6 %	6 %	6 %	Kyllä	333	309	318,2	7,24
HC_45B	45 %	5 %	5 %	45 %	Kyllä	326	306	317,9	6,19
HC_40B	73 %	9 %	9 %	9 %	Kyllä	336	309	317,6	7,62
HC_34B	91 %	3 %	3 %	3 %	Kyllä	325	312	317,5	3,57
HC_46B	60 %	20 %	10 %	10 %	Kyllä	332	307	317,2	8,18
HC_41B	70 %	10 %	10 %	10 %	Kyllä	324	303	315,9	6,49
HC_42B	55 %	15 %	15 %	15 %	Kyllä	328	304	315,6	7,90
HC_43B	80 %	5 %	5 %	10 %	Kyllä	327	296	315,5	10,95
HC_35B	88 %	4 %	4 %	4 %	Kyllä	328	296	314,1	12,20
HC_44B	75 %	10 %	10 %	5 %	Kyllä	324	296	314,1	7,78
HC_38B	79 %	7 %	7 %	7 %	Kyllä	322	303	313,6	6,59
HC_36B	85 %	5 %	5 %	5 %	Kyllä	326	301	313,3	8,74
HC_39B	76 %	8 %	8 %	8 %	Kyllä	319	301	312,3	6,04
HC_33B	94 %	2 %	2 %	2 %	Kyllä	322	294	311,2	9,05
HC_47B	60 %	10 %	20 %	10 %	Kyllä	321	272	307,8	14,03

Taulukko 19. Algoritmin 3 tuloksia (Muunnostapa 2).

Muunnostapaa 2 käyttäen alustavista tuloksista nähdään heti, että testiajojen pistemäärät jäävät kauaksi Muunnostapaa 1 käyttävistä testiajoista. Muunnostavalla 1 huipputulokset olivat n. 370 pisteen luokkaa, kun nyt jäädään vajaan 320 pisteeseen. Ellei Muunnostavan 2 tuloksiin saada lisäparametreilla merkittävää nousua, jää Muunnostavan 2 hyöty melko vähäiseksi.

Jos Muunnostavalla 1 näytti siltä, että testiajojen pistemäärien välillä ei ollut suuremmin hajontaa, niin samaa voidaan sanoa myös Muunnostavan 2 tuloksista. Verrattuna Muunnostavan 1 tarkennettuihin tuloksiin, Muunnostavan 2 tulokset saavuttivat huipunsa hieman eri parametreilla. Muunnostavan 2 tulokset olivat tosin pääosin niin lähellä toisiaan, että ne mahtunevat virhemarginaalin sisään. Jatketaan testausta valitsemalla jat-

koon Taulukko 19 paras testiajo, jonka tavoitepainot lukitaan arvoihin 82 %, 6 %, 6 % sekä 6 %. Varioidaan samalla myös muita aikaisemmin mainittuja parametreja.

Muutetaan aluksi muunnosoperaation parametreja siten, että $M_{\min} \in \{1, 2\}$ ja $M_{\max} \in \{1, 2, 3, 4, 5, 6, 7, 8\}$. Huomioitakoon tässä, että sellaista yhdistelmää, jossa $M_{\min} > M_{\max}$ ei luonnollisesti voida ajaa.

Testiajo	M_{\min}	M_{\max}	MAX	MIN	MEAN	STDEV
HC_48	1	1	332	314	323,1	6,17
HC_49	1	2	344	330	336,3	4,47
HC_50	1	3	345	325	336,9	5,15
HC_51	1	4	340	322	330,6	5,32
HC_52	1	5	344	322	330,1	7,78
HC_53	1	6	332	313	325,7	6,20
HC_54	1	7	332	301	318,8	10,17
HC_37B	1	8	333	309	318,2	7,24
HC_55	2	2	272	242	260,4	9,16
HC_56	2	3	279	257	268,2	6,21
HC_57	2	4	284	247	266,3	9,71
HC_58	2	5	276	258	267,8	5,55
HC_59	2	6	281	243	263,2	10,86
HC_60	2	7	272	244	257,3	8,67
HC_61	2	8	267	242	254,9	8,33

Taulukko 20. Algoritmin 3 tuloksia muunnosoperaation parametrien varioimisen jälkeen (Muunnostapa 2).

Taulukon 20 tuloksista voidaan nähdä, että muunnosoperaation parametreilla on tuloksiin paljon suurempi vaikutus kuin tavoitteiden painojen varioinnilla, kun käytössä on Muunnostapa 2. Muunnosoperaation parametreja muokkaamalla tulos nousi testiajon HC_37B pistemäärästä 318,2 testiajon HC_50 pistemäärään 336,9. Otetaan nyt testiajo HC_50 käyttöön ($M_{\min} = 1$ ja $M_{\max} = 3$) ja varioidaan muunnossuhdetta, joka määrittelee, millä todennäköisyydellä vaihtomuunnos valitaan kääntömuunnoksen sijaan. Ajetaan uusia testiajoja siten, että muunnossuhde saa arvot 0–100 % siten, että arvo muuttuu aina yhdellä askeleella kymmenen prosenttiyksikön verran. Tulokset esitetään Taulukossa 21.

Testiajo	Muunnossuhde	MAX	MIN	MEAN	STDEV
HC_62	0 % (Vain kiertomuunnos)	243	222	235,7	7,50
HC_63	10 %	327	307	317,5	5,66
HC_64	20 %	349	319	328,8	9,21
HC_65	30 %	341	322	332,9	5,90
HC_66	40 %	345	327	335,0	5,16
HC_50	50 %	345	325	336,9	5,15
HC_67	60 %	347	324	335,1	6,03
HC_68	70 %	352	332	338,5	6,02
HC_69	80 %	342	332	337,8	3,39
HC_70	90 %	352	329	339,0	6,96
HC_71	100 % (Vain vaihtomuunnos)	340	320	330,6	6,40

Taulukko 21. Algoritmin 3 tuloksia muunnossuhteen muutoksen jälkeen (Muunnostapa 2).

Hieman yllättäen paras tulos syntyi muunnossuhtetta varioimalla silloin, kun 90 % muunnoksista suoritettiin käyttäen vaihtomuunnosta ja loput 10 % kääntömuunnosta. Tämä saattaa selittyä sillä, että yleisesti ottaen palojen vaihto keskenään muuttaa pelilaudaa radikaalimmin kuin paikallisempaan muutokseen perustuva kierto.

Ajetaan vielä lopuksi muunnossuhteilla 30 %, 50 %, 70 % ja 90 % sellaiset testiajot, joissa pelilaudan niin sanotun epäonnistuneen muutoksen takaisin palauttamista ohjaava parametri pMuutos asetetaan epätodeksi. Nämä tulokset löytyvät alta Taulukosta 22.

Testiajo	Muunnossuhde	pMuutos	MAX	MIN	MEAN	STDEV
HC_65	30 %	Tosi	341	322	332,9	5,90
HC_50	50 %	Tosi	345	325	336,9	5,15
HC_68	70 %	Tosi	352	332	338,5	6,02
HC_70	90 %	Tosi	352	329	339,0	6,96
HC_72	30 %	Epätosi	339	320	327,6	6,17
HC_73	50 %	Epätosi	344	321	331,2	7,35
HC_74	70 %	Epätosi	348	323	336,2	6,63
HC_75	90 %	Epätosi	349	330	338,6	6,26

Taulukko 22. Algoritmin 3 tuloksia parametrin pMuutos ollessa epätosi (Muunnostapa 2).

Parametrin pMuutos asettaminen epätodeksi pienensi testiajojen keskimääräisiä pisteitä, mutta vain marginaalisesti. Jos pMuutos on epätosi, tehdään pelilaudalle mahdollisesti montakin sellaista perättäistä muutosta, jotka eivät nosta pistemäärää. Jokainen tällainen muutos vie pelilautaa kauemmas viimeksi löydetystä optimijärjestyksestä. Tällainen toiminta tuntuu ehkä hieman kannattamattomalta, mutta toisaalta algoritmi ei tällöin ehkä jää niin helposti tiettyyn paikalliseen maksimikohtaan, vaan uusia kovinkin erilaisia ratkaisukandidaatteja tutkitaan kaiken aikaa. Mutta koska tutkintatapa on tällöin periaatteessa satunnaistettu etsintä, ei voida olettaa tuloksen kehittyvän poikkeuksellisen hyväksi.

Muunnostapaa 2 käytettäessä parhaaksi tulokseksi tuli testiajon HC_70 tulos keskimääräisellä pistemäärällä 339,0. Testiajo HC_70 käytti parametreinaan Tavoitteiden 1–4 painoja 82 %, 6 %, 6 % ja 6 %. Samaisen testiajon muunnosoperaation parametrit olivat $M_{\min} = 1$ ja $M_{\max} = 3$. Testiajon muunnossuhde oli 90 % ja parametrin pMuutos arvo oli tosi. Muunnostavalla 1 puolestaan korkeimman pistemäärän saavutti testiajo HC_27B tuloksella 372,9. Tällä testiajolla tavoitepainot olivat 55 %, 15 %, 15 % ja 15 %. Testiajon suorituksen aikana parametri pMuutos oli tosi. Muunnostavan 1 paras tulos päihitti siis Muunnostavan 2 parhaan tuloksen kymmenen prosentin erolla.

Algoritmin 3 parhaankaan testiajon pistemäärä ei kuitenkaan noussut yhtä hyväksi kuin Algoritmeilla 1 ja 2. Suurin syy heikompaan pistemäärään lienee Algoritmin 3 ahne toimintaperiaate, joka ajaa algoritmin suorituksen varsinkin reunatäsmäyspelien tapauksessa todella helposti johonkin useista paikallisista huippukohdista, joista tulos ei pääse enää paranemaan. Kuitenkin Algoritmi 3 antoi suhteellisen pienellä kehityspanoksella (tämän tutkimuksen sisällön perusteella) kohtuullisen hyvän tuloksen ilman laajaa testusta monimutkaisilla parametreilla, varsinkin käytettäessä Muunnostapaa 1.

5.6. Algoritmin 4 tuloksia

Tämän luvun viimeinen algoritmi on simuloituun jäähtyäkseen perustuva Algoritmi 4. Kuten luvusta 4 muistamme, Algoritmin 4 toimintaperiaate on hyvin pitkälle samankaltainen kuin Algoritmilla 3. Algoritmista 3 poiketen Algoritmin 4 simuloitu jäähtytys vaikuttaa siten, ettei algoritmi jää niin helposti paikallisiin maksimeihin kuin Algoritmi 3, vaan simuloitu jäähtytys mahdollistaa algoritmille tietyissä tilanteissa myös epäoptimaaliset siirrot, jotka pienentävät pelilaudan pistemäärää.

Algoritmien 3 ja 4 samankaltaisuudesta johtuen Algoritmilla 4 on myös kaksi erilaista muunnostapaa. Koska raakaan voimaan perustuva Muunnostapa 1 ei järjestelmällisen koekilunsa vuoksi sovellu järin hyvin Algoritmiin 4, käytetään tässä hieman vastaavaa uutta muunnostapaa, jossa vaihdetaan joka kierroksella kahden palan paikkaa keskenään. Tässä vaihdettavat kaksi palaa valitaan satunnaisesti. Kutsutaan tätä tapaa Muunnostavaksi 3. Muunnostavan 3 toimintaperiaate on itseasiassa hyvin samanlainen kuin Muunnostavalla 2 silloin, kun muunnosoperaation parametrit ovat $M_{\min} = 1$ ja $M_{\max} = 1$, jolloin muunnosoperaatiot käsittelevät pelilaudan yksittäisiä paloja.

Algoritmin 4 jäähtymisfunktioina käytetään kahta erilaista funktiota, joista toinen on eksponentiaaliseen vähenemiseen perustuva ja toinen perustuu lineaariseen vähenemiseen. Algoritmia 4 ohjaavat myös Algoritmista 3 tutut tavoitteiden painokertoimet ja pelilaudan esiratkaistu sekä Muunnostapaan 2 liittyvät parametrit. Parametri p Muutos jätetään tässä tapauksessa testauksen ulkopuolelle, sillä Algoritmin 3 testituloksista havaittiin, ettei kyseisellä parametrilla ole positiivista käytännön merkitystä tulosten kannalta. Algoritmien samankaltaisuuden takia parametrilla on siis tuskin tässä tapauksessa hyötyä. Algoritmia 4 ohjaavat lisäksi simuloituun jäähtyäkseen liittyvät ohjausparametrit α , ε ja T_0 . Näistä α merkitsee eksponentiaalisen jäähtymisfunktion vähenemiskerrointa, ε tavoiteltua loppulämpötilaa ja T_0 puolestaan algoritmin alkulämpötilaa. Käyttäjä voi myös kontrolloida parametrilla *pisteMuutos* sitä, hyväksytäänkö sellaiset pelilaudalle tehtävät muutokset, jotka eivät muuta lainkaan pelilaudan pistemäärää. Jos parametri on epätosi, pistemäärään vaikuttamatonta muutosta ei oteta käyttöön.

Aloitetaan testaus Muunnostavalla 3, jolloin kontrolloitavia parametreja on hieman vähemmän kuin Muunnostavalla 2. Valitaan Tavoitteiden 1–4 painokertoimiksi 55 %, 15 %, 15 % ja 15 %, jotka ovat samat kertoimet kuin Algoritmin 3 parhaassa testiajossa, jossa käytettiin Muunnostapaa 1. Varioidaan lopuksi painokertoimia hieman tarkemmin muiden parametrien tarkennuttua. Asetetaan lisäksi parametrin *pisteMuutos* arvo epätodeksi. Valitaan jäähtymisfunktioksi eksponentiaalinen jäähtyminen. Käytetään tästedes aina esiratkaistua pelilautaa, sillä se on osoittautunut esiratkaistematonta paremmaksi vaihtoehdoksi. Taulukko 23 sisältää suoritettuja testiajoja parametreineen ja tuloksineen.

Testiajo	α	ε	T_0	Kierroksia	MAX	MIN	MEAN	STDEV
SA_1	0,99	0,01	1,0	458	217	196	204,9	6,37
SA_2	0,99	0,01	1000,0	1145	239	220	229,4	5,36
SA_3	0,99	0,01	1000000,0	1832	251	237	242,8	4,54
SA_4	0,99	0,01	1000000000,0	2520	261	244	253,9	4,91
SA_5	0,999	0,01	1,0	4602	276	263	269,5	4,55
SA_6	0,999	0,01	100,0	9205	296	280	289,0	4,97
SA_7	0,999	0,01	10000,0	13808	310	288	300,8	6,09
SA_8	0,999	0,01	1000000,0	18411	324	306	315,5	5,02
SA_9	0,99	0,01	10000,0	1374	249	229	235,6	6,19
SA_10	0,99	0,002	10000,0	1534	242	226	236,2	4,66
SA_11	0,99	0,00001	10000,0	2061	258	239	247,8	6,11
SA_12	0,9999	0,00001	1,0	115123	353	337	349,1	4,70
SA_13	0,9999	0,00001	100,0	161173	361	350	354,9	4,23
SA_14	0,9999	0,00001	10000,0	207222	361	349	355,3	4,08
SA_15	0,9999	0,00001	1000000,0	253272	363	347	354,1	5,00
SA_16	0,99999	0,00001	1,0	~1151290	369	347	357,5	5,76
SA_17	0,99999	0,00001	0,1	921029	365	350	357,9	5,20
SA_18	0,99999	0,00001	0,01	690772	368	346	356,1	6,51
SA_19	0,99999	0,00001	0,0001	230257	365	350	357,1	4,65
SA_20	0,999999	0,000001	0,00001	~2302580	367	355	360,7	3,53

Taulukko 23. Algoritmin 4 alustavia tuloksia (Ekspontiaalinen jäähtymisfunktio ja Muunnostapa 3)

Ensimmäisen testisarjan tuloksista huomionarvoisinta lienee parametrien osalta se, että algoritmin suorittama kierrosten määrä vertautuu lähes suoraan kyseisen testiajon pistemäärään. Yhteys ei ole lineaarinen, mutta noin 355 pisteen rajaan saakka pisteet kasvavat sitä mukaa, mitä kauemmin algoritmia suoritetaan. Kierrosmäärää ei aseteta manuaalisesti algoritmin parametreihin, vaan kierrosmäärä määräytyy simuloidun jäähtymisen parametrien perusteella. Algoritmin aloituslämpötilalla T_0 ei näyttänyt olevan selkeää arvoaluetta, jolla parametrin arvon tulisi sijaita tietyn pistemäärän saavuttamiseksi, vaan varsinkin korkeamman pistemäärän testiajoilla aloituslämpötilan arvo oli lähestulkoon yhdenkertainen.

Ajetaan siis vielä yksi testisarja ajoja muutellen ε :n ja T_0 :n arvoja siten, että algoritmin kierrosten lukumäärä pysyy samana. Näin voidaan helpommin huomata, jos tietty parametrikombinaatio tuottaa muita parempia pisteitä. Parametreista α määrää voimak-

kaimmin algoritmin kierrosten lukumäärän ja samalla algoritmin lopullisen pistemäärän, joten lukitaan sen arvoksi 0,999999. Testisarjan tulokset nähdään Taulukosta 24.

Testiajo	α	ε	T_0	Kierroksia	MAX	MIN	MEAN	STDEV
SA_21	0,999999	0,1	1,0	~2302580	374	347	359,9	9,18
SA_22	0,999999	0,01	0,1	~2302580	370	353	361,8	5,92
SA_23	0,999999	0,001	0,01	~2302580	371	338	358,1	8,60
SA_24	0,999999	0,0001	0,001	~2302580	369	345	356,2	7,39
SA_25	0,999999	0,00001	0,0001	~2302580	377	349	360,2	8,36
SA_20	0,999999	0,000001	0,00001	~2302580	367	355	360,7	3,53
SA_26	0,999999	0,0000001	0,000001	~2302580	365	354	359,7	3,68

Taulukko 24. Algoritmin 4 jatkotuloksia (Ekspontiaalinen jäähtymisfunktio ja Muunnostapa 3)

Testiajojen SA_20–SA_26 tuloksista nähdään, ettei alku- ja loppulämpötilan valinnalla ole tässä tapauksessa suurta merkitystä, kunhan algoritmin kierrosluku pysyy samana. Valitaan kuitenkin korkeimman keskimääräisen pistemäärän saanut testiajo SA_22, lukitaan sen simuloituun jäähdytykseen liittyvät parametrit ja kokeillaan varioida Tavoitteiden 1–4 painokertoimia. Valitaan painokertoimet Taulukon 18 mukaan, sillä Muunnostavan 1 toimintaperiaate on ajatukseltaan lähellä Muunnostapaa 3, jota Algoritmi 4 tässä tapauksessa käyttää.

Testiajo	Tavoite 1	Tavoite 2	Tavoite 3	Tavoite 4	MAX	MIN	MEAN	STDEV
SA_22	55 %	15 %	15 %	15 %	370	353	361,8	5,92
SA_27	88 %	4 %	4 %	4 %	381	349	364,8	8,50
SA_28	85 %	5 %	5 %	5 %	368	354	358,7	5,03
SA_29	60 %	10 %	20 %	10 %	372	350	360,9	7,36
SA_30	73 %	9 %	9 %	9 %	374	347	362,0	7,77
SA_31	60 %	20 %	10 %	10 %	371	345	356,6	8,64
SA_32	79 %	7 %	7 %	7 %	370	347	360,3	6,63
SA_33	80 %	5 %	5 %	10 %	366	349	358,0	6,04
SA_34	82 %	6 %	6 %	6 %	368	340	358,6	7,95
SA_35	70 %	10 %	10 %	10 %	370	348	359,3	6,75
SA_36	94 %	2 %	2 %	2 %	367	348	354,3	6,11
SA_37	91 %	3 %	3 %	3 %	372	351	358,3	7,15
SA_38	75 %	10 %	10 %	5 %	378	345	358,7	9,68
SA_39	76 %	8 %	8 %	8 %	365	339	356,9	8,01
SA_40	45 %	5 %	5 %	45 %	368	342	356,6	7,09

Taulukko 25. Algoritmin 4 tuloksia tavoitteiden painokertoimien muokkauksen jälkeen (Eksponentiaalinen jäähtymisfunktio ja Muunnostapa 3)

Taulukosta 25 nähdään, että myöskään tavoitteiden painokertoimien muutoksella ei ollut järin suurta vaikutusta tuloksiin. Marginaalisesti paras tulos saavutettiin kuitenkin testiajolla SA_27 tavoitteiden painokertoimien ollessa 88 %, 4 %, 4 % ja 4 % pistemäärällä 364,8. Valitaan vielä Taulukko 25 kolme parasta tulosta ja testataan parametrin pisteMuutos vaikutusta tuloksiin. Yllä olevat testiajot on ajettu pisteMuutoksen ollessa epätosi, joten ajetaan vielä testiajot SA_27, SA_30 ja SA_22, kun pisteMuutoksen arvo on tosi. Tulokset nähdään Taulukossa 26.

Testiajo	Tavoite 1	Tavoite 2	Tavoite 3	Tavoite 4	pisteMuutos	MAX	MIN	MEAN	STDEV
SA_27	88 %	4 %	4 %	4 %	Epätosi	381	349	364,8	8,50
SA_30	73 %	9 %	9 %	9 %	Epätosi	374	347	362,0	7,77
SA_22	55 %	15 %	15 %	15 %	Epätosi	370	353	361,8	5,92
SA_41	88 %	4 %	4 %	4 %	Tosi	378	349	366,2	9,43
SA_42	73 %	9 %	9 %	9 %	Tosi	377	356	364,6	6,15
SA_43	55 %	15 %	15 %	15 %	Tosi	371	347	356,4	6,96

Taulukko 26. Parametrin pisteMuutos vaikutus Algoritmin 4 tuloksiin (Eksponentiaalinen jäähtymisfunktio ja Muunnostapa 3)

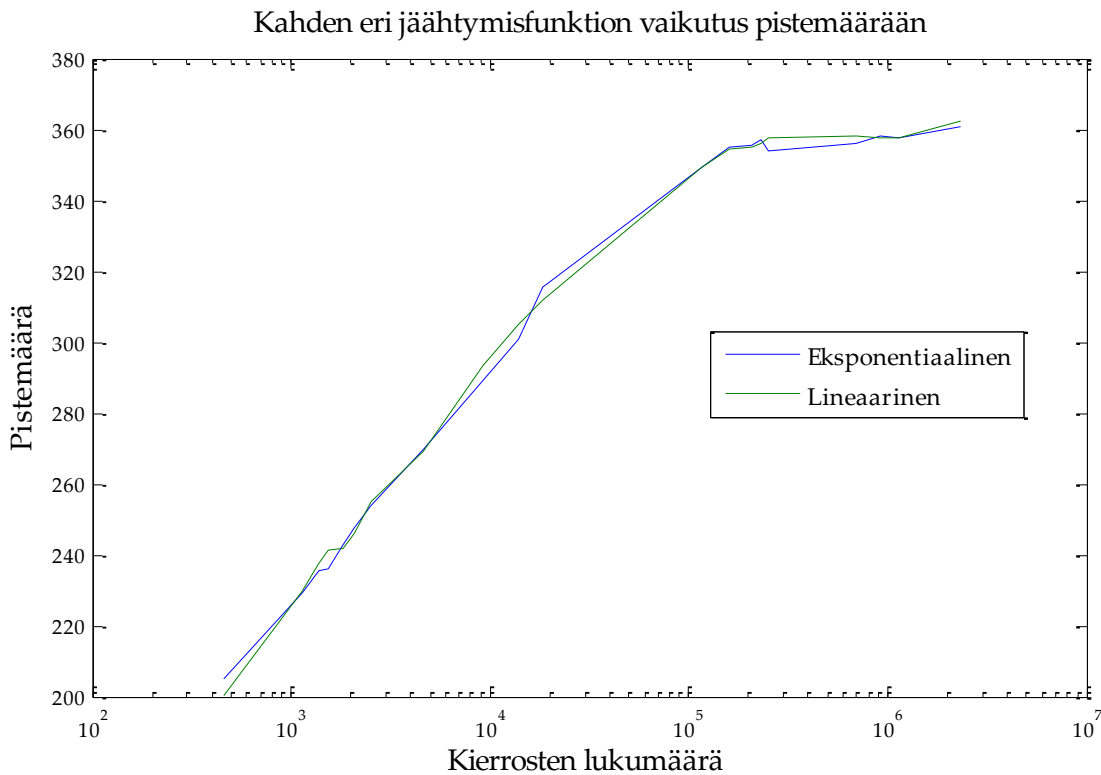
Parametrin pisteMuutos vaikutus on hieman ristiriitainen, eikä nyt ajettujen testiajojen perusteella voida tehdä vielä lopullisia päätelmiä parametrin mahdollisesta vaikutuksesta. Keskimäärin pistemäärä ei ratkaisevasti nouse, vaikka testiajo SA_41 tuottaakin tähän mennessä korkeimman tuloksen Algoritmillä 4. Testiajon SA_41 tulos saattaa kuitenkin olla normaalia vaihtelua, eikä parametrilla pisteMuutos ole välttämättä suurempaa vaikutusta asiaan.

Koska käytetyllä jäähtymisfunktiolla ja muunnostavalla on tuskin keskinäistä yhteyttä toisiinsa, niin ajetaan ennen muunnostavan vaihtoa testejä myös lineaarisella jäähtymisfunktiolla, jotta nähtäisiin, kumpi jäähtymisfunktio lopullisiin testeihin kannattaa valita. Koska lineaarisen jäähtymisfunktion yhteydessä käytetään loppulämpötilan ε ja jäähtymiskertoimen α sijaan algoritmin kokonais kierrosten määrää, tarvitsee algoritmille määrittellä nyt alkulämpötila T_0 ja kierrosten määrä n . Käytetään lineaarisesta jäähtymisfunktiota käyttävien testien ensimmäisissä testiajoissa Tavoitteiden 1–4 painokertoimia 55 %, 15 %, 15 % ja 15 %, sillä nämä kertoimet menestyivät eksponentiaalista jäähtymisfunktiota käytettäessä melko hyvin ja koska samoja kertoimia käytettiin alun alkaenkin Algoritmin 4 testeissä. Parametrit on valittu siten, että ne vastaisivat mahdollisimman hyvin Taulukon 23 parametreja. Parametri pisteMuutos on asetettu epätodeksi. Käytännössä alkulämpötila ja kierrosmäärä on asetettu suoraan Taulukon 23 arvoja vastaaviksi. Taulukossa 27 on Algoritmin 4 tuloksia, kun käytössä on lineaarinen jäähtymisfunktio.

Testiajo	T_0	Kierroksia (n)	MAX	MIN	MEAN	STDEV
SA_44	1	458	210	188	200,3	6,41
SA_45	1000	1145	250	214	229,9	9,64
SA_46	10000	1374	244	230	237,4	5,10
SA_47	10000	1534	245	238	241,3	2,00
SA_48	100000	1832	251	233	241,7	5,03
SA_49	10000	2061	259	236	245,9	7,88
SA_50	1000000000	2520	270	244	255,2	8,34
SA_51	1	4602	276	262	269,1	4,56
SA_52	100	9205	299	286	293,2	4,76
SA_53	10000	13808	318	296	304,8	7,02
SA_54	1000000	18411	320	306	311,7	4,50
SA_55	1	115123	354	343	349,1	4,12
SA_56	100	161173	361	345	354,4	4,97
SA_57	10000	207222	367	347	355,1	5,74
SA_58	0,0001	230257	374	348	356,1	7,98
SA_59	1000000	253272	371	348	357,7	6,27
SA_60	0,01	690772	363	353	358,1	3,63
SA_61	0,1	921029	367	349	357,6	5,15
SA_62	1	1151290	367	349	357,8	5,33
SA_63	0,00001	2302580	372	355	362,3	6,15

Taulukko 27. Algoritmin 4 alustavia tuloksia lineaarisella jäähtytysfunktioilla (Muunnostapa 3)

Lineaarisen jäähtymisfunktion tulokset näyttävät melko samankaltaisilta kuin Taulukon 23 tulokset. Jäähtymisfunktioiden eroja havainnollistaa Kuva 8.



Kuva 8. Jäähdytysfunktioiden vaikutus Algoritmin 4 pistemäärään.

Kuten Kuvasta 8 nähdään, ei kahdella käytetyllä jäähtymisfunktiolla ole tässä tapauksessa käytännössä lainkaan eroja. Jatketaan testejä siis tästä eteenpäin käyttäen eksponentiaalista jäähtymisfunktiota, koska sitä käytetään tavallisesti simuloitussa jäähdytyksessä.

Nyt kun muut testiparametrit on käyty läpi, vaihdetaan algoritmin muunnostapaa. Otetaan käyttöön siis käyttöön edellisestä kohdasta tuttu Muunnostapa 2. Muunnostavan 2 toimintaperiaatteeseenhan kuului Algoritmien 1 ja 2 mutaatio-operaatioiden hyödyntäminen pelilaudan muuntelussa. Asetetaan ensiksi muunnosoperaation parametreille arvot $M_{\min} = 1$ ja $M_{\max} = 3$, sillä kyseiset parametrit tuottivat Algoritmilla 3 parhaan tuloksen. Samasta syystä muunnossuhdeparametri saa arvon 90 %. Ajetaan aluksi Taulukon 23 tyylliset testiajot uusilla parametreilla, sillä muunnostyyppin vaihdos näytti alustavien kokeilujeni perusteella aiheuttavan suuria muutoksia simuloitun jäähdytyksen parametreihin Muunnostyyppiin 3 verrattuna. Seuraavat Taulukon 28 testiajot on ajettu Tavoitteiden 1–4 suhteilla 55 %, 15 %, 15 % ja 15 %.

Testiajo	α	ε	T_0	Kierroksia	MAX	MIN	MEAN	STDEV
SA_64	0,99	0,0001	0,01	458	95	74	85,2	7,04
SA_65	0,99	0,0001	0,001	229	125	111	118,0	4,42
SA_66	0,99	0,00001	0,01	687	120	73	87,3	12,92
SA_67	0,99	0,00001	0,001	458	138	116	129,2	6,20
SA_68	0,99	0,00001	0,0001	229	119	107	113,9	4,77
SA_69	0,999	0,0001	0,001	2301	186	174	178,4	4,27
SA_70	0,999	0,00001	0,001	4602	213	195	203,0	6,36
SA_71	0,999	0,00001	0,0001	2301	191	170	179,8	6,16
SA_72	0,9999	0,00001	0,01	69074	232	220	224,1	4,53
SA_73	0,9999	0,00001	0,001	46049	277	249	262,2	7,55
SA_74	0,9999	0,00001	0,0001	23024	259	241	248,4	5,13
SA_75	0,9999	0,000001	0,0001	46049	283	264	272,2	5,92
SA_76	0,99999	0,00001	0,001	460515	309	289	304,2	5,88
SA_77	0,99999	0,00001	0,0001	230257	319	304	311,7	5,44
SA_78	0,99999	0,000001	0,001	690772	322	301	311,4	6,59
SA_79	0,99999	0,000001	0,0001	460515	330	312	320,1	5,38
SA_80	0,99999	0,000001	0,00001	230257	313	297	304,7	4,90
SA_81	0,999999	0,0000001	0,0001	~6907750	349	329	340,4	7,43
SA_82	0,999999	0,0000001	0,00001	~4605710	350	331	342,7	6,33
SA_83	0,999999	0,0000001	0,000001	~2302580	347	329	338,8	6,89

Taulukko 28. Algoritmin 4 alustavia testiajoja Muunnostapaa 2 käyttäen.

Muunnostavasta 3 poiketen Muunnostapaa 2 käyttäen simuloidun jäähdytyksen alkulämpötilalla T_0 on nyt hyvinkin suuri merkitys. Liian suurella alkulämpötilalla pistemäärä ei käytännössä katsoen lähtenyt lainkaan nousemaan, ja jonkinlaisen kuvan alkulämpötilan vaikutuksesta saa myös tarkastelemalla testiajojen SA_64 ja SA_65 eroja. Testiajo SA_65 on nimittäin saavuttanut testiajoa SA_64 korkeamman pistemäärän pienemmällä kierrosmäärällä.

Vaikka Muunnostapojen 2 ja 3 alustavissa tuloksissa onkin parametrien osalta suuria eroja, näyttäisivät tulokset seuraavan tässäkin tapauksessa suoritettujen kierrosten määrää tiettyyn pisterajaan saakka. Koska testiajolla SA_81 ei ole pistemäärätua testiajoon SA_82 nähden suuremmasta kierrosmäärästä huolimatta, valitaan jatkotesteihin siis testiajo SA_82 pistemäärällä 342,7.

Kuten äsken näimme, muunnostavan muutos voi vaikuttaa parametrien optimaalisiin arvoihin paljonkin, joten seuraavaksi ajetaan testiajoon SA_82 parametreilla uusia testiajoja,

joissa pistemäärän tavoitteille asetetaan erilaisia kombinaatioita. Saadut tulokset nähdään Taulukossa 29.

Testiajo	Tavoite 1	Tavoite 2	Tavoite 3	Tavoite 4	MAX	MIN	MEAN	STDEV
SA_82	55 %	15 %	15 %	15 %	350	331	342,7	6,33
SA_84	88 %	4 %	4 %	4 %	348	327	338,4	7,28
SA_85	85 %	5 %	5 %	5 %	352	336	342,2	5,43
SA_86	60 %	10 %	20 %	10 %	351	327	339,2	7,18
SA_87	73 %	9 %	9 %	9 %	352	321	338,7	9,03
SA_88	60 %	20 %	10 %	10 %	356	330	341,3	7,48
SA_89	79 %	7 %	7 %	7 %	354	333	341,4	6,95
SA_90	80 %	5 %	5 %	10 %	352	328	342,7	7,38
SA_91	82 %	6 %	6 %	6 %	343	329	337,0	4,32
SA_92	70 %	10 %	10 %	10 %	345	320	335,6	8,92
SA_93	94 %	2 %	2 %	2 %	344	320	333,6	7,44
SA_94	91 %	3 %	3 %	3 %	345	327	335,0	6,0
SA_95	75 %	10 %	10 %	5 %	343	328	338,2	4,44
SA_96	76 %	8 %	8 %	8 %	349	327	339,1	6,06
SA_97	45 %	5 %	5 %	45 %	346	328	337,5	6,22

Taulukko 29. Algoritmin 4 tuloksia Muunnostavalla 2 pistemäärän tavoitteiden erilaisilla yhdistelmillä.

Pistelaskun tavoitteiden painokertoimet 55 %, 15 %, 15 % ja 15 % pitivät korkeimman sijoituksen, joskin samaan pistemäärään ylsi myös testiajo SA_90. Toisaalta suurin osa testiajoista sijoittui melko pienen pistemäärän päähän toisistaan, joten uudella ajokerralla tulos voisi olla jokin toinen. Mutta oleellisinta tuloksissa on se, ettei yhtäkään sellaista tulosta löytynyt, joka olisi selkeästi parempi kuin muut.

Jatketaan siis testejä käyttäen edelleen testiajon SA_82 parametreja. Vielä on testaamatta muunnosoperaation parametrien muutos sekä parametrin pisteMuutos vaikutus tuloksiin käyttäen Muunnostapaa 2. Ajetaan seuraavaksi Taulukon 20 parametreja vastaavat ajot, joissa muunnosoperaation parametrit saavat arvot $M_{\min} \in \{1, 2\}$ ja $M_{\max} \in \{1, 2, 3, 4, 5, 6, 7, 8\}$.

Testiajo	M_{\min}	M_{\max}	MAX	MIN	MEAN	STDEV
SA_98	1	1	350	324	333,3	7,57
SA_99	1	2	355	327	340,5	8,55
SA_82	1	3	350	331	342,7	6,33
SA_100	1	4	344	323	337,6	6,40
SA_101	1	5	341	329	338,2	3,77
SA_102	1	6	346	327	334,5	7,93
SA_103	1	7	338	316	329,6	6,11
SA_104	1	8	333	318	327,7	4,88
SA_105	2	2	278	246	268,5	9,88
SA_106	2	3	296	257	270,8	12,48
SA_107	2	4	282	263	273,6	5,60
SA_108	2	5	290	258	272,1	8,65
SA_109	2	6	277	266	271,8	3,19
SA_110	2	7	284	252	266,8	9,20
SA_111	2	8	274	248	262,6	8,86

Taulukko 30. Algoritmin 4 tuloksia muunnosoperaation parametrien varioinnin jälkeen.

Taulukko 30 esittelee muunnosoperaation parametrien muutosten vaikutusta testiajojen tuloksiin. Tulokset noudattelevat Algoritmin 3 vastaavaa testisarjaa, jossa parhaimman tuloksen saavutti sellainen testiajo, jonka parametrit olivat $M_{\min} = 1$ ja $M_{\max} = 3$. Testiajolla SA_82 olivat siis nämä samat parametrit, ja kyseisen testiajon tulos on edelleen korkein saavutettu tulos Algoritmin 4 ja Muunnostavan 2 yhdistelmällä.

Jatketaan tutulla linjalla ja testataan tämän jälkeen, löytyykö muunnossuhdeparametrien muutoksilla tuloksiin jonkinlaista parannusta. Ajetaan siis testisarja, jossa muunnossuhde saa arvot 0–100 % kymmenen prosenttiyksikön hyppäyksin. Tulokset nähdään Taulukossa 31.

Testiajo	Muunnossuhde	MAX	MIN	MEAN	STDEV
SA_112	0 % (Vain kiertomuunnos)	239	216	228,9	7,28
SA_113	10 %	330	317	323,1	3,96
SA_114	20 %	339	320	331,1	5,78
SA_115	30 %	342	327	336,8	4,08
SA_116	40 %	345	323	334,9	6,85
SA_117	50 %	350	330	340,1	6,15
SA_118	60 %	348	331	339,4	5,27
SA_119	70 %	352	326	340,9	8,18
SA_120	80 %	347	329	337,0	5,85
SA_82	90 %	350	331	342,7	6,33
SA_121	100 % (Vain vaihtomuunnos)	338	325	331,4	3,84

Taulukko 31. Muunnossuhteen vaikutus Algoritmin 4 tuloksiin.

Myös muunnossuhteen muutokset noudattelevat Algoritmin 3 vastaavaa linjaa, ja täten testiajon SA_82 tulos pysyy parhaimpana Muunnostapaa 2 käyttävänä testiajona. Parametrin pisteMuutos vaikutus oli hieman ristiriitainen käytettäessä Muunnostapaa 3, joten ajetaan myös tässä vielä yksi testisarja, johon valitaan Taulukon 31 kolme parasta testiajtoa, jotka ajetaan uudestaan parametrin pisteMuutos ollessa tosi. Nämä tulokset löytyvät Taulukosta 32.

Testiajo	Muunnossuhde	pisteMuutos	MAX	MIN	MEAN	STDEV
SA_117	50 %	Epätosi	350	330	340,1	6,15
SA_119	70 %	Epätosi	352	326	340,9	8,18
SA_82	90 %	Epätosi	350	331	342,7	6,33
SA_124	50 %	Tosi	361	345	352,2	4,57
SA_125	70 %	Tosi	363	343	354,8	6,12
SA_126	90 %	Tosi	366	353	357,1	4,58

Taulukko 32. Parametrin pisteMuutos vaikutus Algoritmin 4 tuloksiin Muunnostapaa 2 käytettäessä.

Muunnostavasta 3 poiketen parametri pisteMuutos paransi tällä kertaa pistemääriä keskimäärin yli kymmenen pisteen verran. Parhaaseen tulokseen ylsi testiajo SA_126, jonka tulos parani testiajon SA_82 tuloksesta lähes viidellätoista pisteellä. Tällainen ero Muunnostapaan 3 nähden voi liittyä siihen, että Muunnostavalla 3 suuri osa pelilaudan muutoksista ei muuta pelilaudan pistemäärää lainkaan, mutta Muunnostavalla 2 tällaiset tapaukset ovat harvinaisempia pelilaudan suurempien muutosten takia. Tarkempaa selitystä on kuitenkin vaikea antaa ilman perusteellisempää selvittelyä, jota tässä ei nyt kuitenkaan tehdä.

Käytettäessä Muunnostapaa 2 Algoritmin 4 paras tulos saavutettiin siis testiajolla SA_126, jonka tulos oli 357,1 pistettä. Muunnostavalla 3 parhaaseen tulokseen ylsi puolestaan testiajo SA_41, jonka pistemäärä oli 366,2. Muunnostavan 3 tulokset olivat muutenkin keskimäärin hieman paremmat kuin Muunnostavalla 2. Käytettyjen muunnostapojen eräs suuri ero oli Algoritmin 4 alkulämpötilan vaikutus tuloksiin. Muunnostapaa 3 käytettäessä alkulämpötilalla ei ollut paljoakaan merkitystä tuloksiin. Tämä saattaa johtua siitä, että Muunnostavan 3 pelilautaa vain vähän kerrallaan muokkaava muutosmenetelmä ja pelilaudan pistemäärän laskutapa yhdessä aiheuttavat sen, että erilaisten pelilautojen välillä voi olla välillä vaikea nähdä eroja, sillä niin monet erilaiset pelilaudat voivat tuottaa saman pistemäärän.

Koska Muunnostavalla 3 alkulämpötilan merkitys oli hyvin pieni, voi olla ettei Algoritmi 4 toiminut tällöin täysin sen mukaan, miten simuloidun jäähtytyksen on ajateltu toimivan, vaan Algoritmin 4 toiminta saattoi tällöin muistuttaa enemmän Algoritmin 3 toimintaa.

6. Yhteenveto

Kaikkien testien ajamisen jälkeen kaikista parhaimman keskimääräisen tuloksen saavutti Algoritmi 2 pistemäärällä 396,7. Tästä kovin kauaksi ei jäänyt Algoritmi 1 pistemäärällä 395,3. Hieman huonompaan tulokseen jäivät Algoritmit 3 ja 4, jotka kuitenkin olivat keskenään melko tasaväkisiä. Hill climbing -tekniikkaan perustuva Algoritmi 3 vei kuitenkin voiton Algoritmista 4 keskimääräisellä pistemäärällä 372,9. Algoritmin 4 parhaaksi tulokseksi jäi 366,2. On toisaalta hieman erikoista, että hill climbing -tekniikkaan perustuva algoritmi päihitti simuloituun jäähdytykseen perustuvan algoritmin. Voi tosin olla, että reunatäsmäyspeliin ongelman ratkaisuvaihtoehto on niin valtava ja hajanainen, että yksinkertaisella ahneella algoritmilla saavutetaan parempia tuloksia kuin hieman edistyneemmällä algoritmilla.

Evoluutioon perustuvat algoritmit olivat siis tämän tutkielman perusteella tehokkaampia kuin tutkielmassa käsitellyt paikalliseen etsintään perustuvat algoritmit. Evoluutiiviset algoritmit mahdollistavat laajan etsintävaihtoehtojen tutkimisen suuren populaation avulla, ja ilmeisesti tästä syystä nämä algoritmit myös saavuttivat korkeimmat pistemäärätkin.

Algoritmien parametreja voisi yrittää muokata vielä pitkäänkin, jolloin parasta tulosta voisi ehkä hieman vielä parantaa. Pelkällä parametrien varioinnilla tuskin saavutetaan kuitenkaan enää merkittäviä uudistuksia algoritmien tuloksiin. Näkisin, että parametrien variointiin verrattuna käytettävän algoritmin rakenteellisilla uudistuksilla on paljon merkittävämpi osuus pistemäärän muodostamisessa. Jatkokehityksinä voisi olla esimerkiksi algoritmien muokkaaminen siten, että yksittäisen pelilaudan sisäiseen rakenteeseen kiinnitettäisiin jonkinlaisen heuristiikan avulla enemmän huomiota pelin satunnaisen muuntelun sijasta.

Parhain näkemäni tieteellisessä julkaisussa esitetty Eternity II -pistemäärä on Schausin ja Devillen [2008] julkaisema 458 pisteen tulos. Tämä saavutus on jo melko lähellä Eternity II:n 480 pisteen maksimitulosta, joten jään mielenkiinnolla seuraamaan, saavutetaanko tulevaisuudessa vielä parempia tuloksia. Vai pysyykö Eternity II kenties ikuisesti ratkeamattomana?

Viiteluettelo

[Abraham et al., 2005] Ajith Abraham, Lakhmi Jain and Robert Goldberg, *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*. Springer Verlag, 2005.

[Ansótegui et al., 2008] Carlos Ansótegui, Ramón Bèjar, César Fernández and Carles Mateu, Edge matching puzzles as hard SAT/CSP benchmarks. In: *Proc. of the 14th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* **5202**, 560–565.

[Antoniadis and Lingas, 2010] Antonios Antoniadis and Andrzej Lingas, Approximability of edge matching puzzles. *Lecture Notes in Computer Science* **5901**, 153–164.

[Benoist and Bourreau, 2008] Thierry Benoist and Eric Bourreau, Fast global filtering for Eternity IITM. *Constraint Programming Letters* **3**, 36–49. Available as <http://www.cs.brown.edu/people/pvh/CPL/Papers/v3/BenoistBourreau.pdf>. Checked 17.11.2010.

[Berger, 1966] Robert Berger, The Undecidability of the domino problem. *Memoirs of the American Mathematical Society* **66**.

[Černý, 1985] V. Černý, Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, *Journal of Optimization Theory and Applications* **45**, 1 (Jan. 1985), 41–51.

[Cook, 1971] Stephen A. Cook, The complexity of theorem proving procedures. In: *Proceedings of the 3rd Annual Symposium on Theory of Computing*, 151–158. ACM, 1971.

[Cormen et al., 2007] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms, 2nd ed.*, MIT Press, 2007.

[Davis, 2003] Lawrence Davis, Genetic algorithms and their applications. Available as <http://www.informatics.indiana.edu/fil/CAS/PPT/Davis>. Checked 13.4.2011.

[Deb et al., 2002] Kalyanmoy Deb, Amrit Pratap and Sameer Agarwal, A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* **6**, 2 (Apr. 2002), 182–197.

[Demaine and Demaine, 2007] Erik D. Demaine and Martin L. Demaine, Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity. *Graphs and Combinatorics* **23** (June 2007), 195–208.

[Eternity, 2002] Eternity. MathWorld – A Wolfram Web Resource. Available as <http://mathworld.wolfram.com/Eternity.html>. Checked 25.3.2011.

[Eternity II, 2007] About Eternity II. Available as <http://uk.eternityii.com/about-eternity2>. Checked 5.11.2010.

[Gasarch, 2002] William I. Gasarch, Guest column: The P=?NP poll. *ACM SIGACT News* **33**, 2 (June 2002), 34–47.

[Guyton and Hall, 2000] Arthur C. Guyton and John E. Hall, *Textbook of Medical Physiology*. W. B. Saunders Company, 2000.

[Holland, 1975] John. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[Kirkpatrick et al., 1983] S. Kirkpatrick, C. D. Gelatt Jr. and M. P. Vecchi, Optimization by simulated annealing, *Science* **220**, 4598 (May 1983), 671–680.

[Kujala, 2008] Tuomas Kujala, Reunatäsmäyspeleistä. Teoksessa Erkki Mäkinen (toim.), Pieniä tietojenkäsittelytieteellisiä tutkimuksia (Syksy 2008). Tampereen yliopisto, Tietojenkäsittelytieteen laitos, Raportti **D-2008-12**, Joulukuu 2008, 67–84. Saatavana osoitteesta: <http://www.cs.uta.fi/reports/dsarja/D-2008-12.pdf>. Tarkistettu 9.4.2011.

[Matsumoto and Nishimura, 1998] Makoto Matsumoto and Takuji Nishimura, Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM TOMACS* **8**, 1 (Jan. 1998), 3–30.

[McAdam, 2004] Puzzle history. American Jigsaw Puzzle Society. Available as <http://www.jigsaw-puzzle.org/jigsaw-puzzle-history.html>. Checked 29.10.2010.

[Metropolis et al., 1953] N. Metropolis, A.W. Rosenbluth, A.H. Teller, M.N. Rosenbluth, and E.Teller, Equation of state calculations by fast computing machines. *Journal of Chemical Physics* **21**, 6 (1953), 1087–1092.

[Mitchell, 1996] Melanie Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[Monckton, 2007] Christopher Monckton, US \$2 million EternityII prize competition rules. Available as <http://replay.waybackmachine.org/20090330135402/http://uk.eternityii.com/Download.ashx?id=19934>. Checked 25.3.2011.

[Muñoz et al., 2009] Jorge Muñoz, German Gutierrez and Araceli Sanchis, Evolutionary techniques in a constraint satisfaction problem: Puzzle Eternity II, In: *Proceedings 2009 IEEE Congress on Evolutionary Computation*, 2985–2991. Also available as: http://e-archivo.uc3m.es/bitstream/10016/9915/3/evolutionary_gutierrez_2009.pdf. Checked 20.3.2011.

[Savelsbergh and van Emde Boas, 1984] Martin W. P. Savelsbergh and Peter van Emde Boas, Bounded tiling, an alternative to satisfiability?, In: *Proceedings, 2nd Frege Conference*, vol. 20, 354–363.

[Sbalzarini et al., 2000] Ivo F. Sbalzarini, Sibylle Müller and Petros Koumoutsakos, Multi-objective optimization using evolutionary algorithms, In: *Proceedings of the Summer Program, Center of Turbulence Research*, 63–74.

[Schaus and Deville, 2008] Pierre Schaus and Yves Deville, Hybridization of CP and VLNS for Eternity II, *Actes JFPC 2008*.

[Selby, 2001] Alex Selby, Description of method. Available as <http://www.archduke.org/eternity/method/desc.html>. Checked 25.3.2011.

[Stegmann, 2007] Rob Stegmann, Rob's puzzle page. Available as <http://home.comcast.net/~stegmann/pattern.htm>. Checked 19.11.2010.

[Takenaga and Walsh, 2006] Yasuhiko Takenaga and Toby Walsh, Tetravex is NP-complete. *Information Processing Letters* **99**, 5 (September 2006), 171–174.

[Tetravex, 2008] Tetravex – GNOME Live! Available as <http://live.gnome.org/Tetravex>. Checked 17.11.2010.

[The Sunday Times, 2005] The Sunday Times, £1m says this really is the hardest jigsaw. Available as <http://www.timesonline.co.uk/tol/news/uk/article745506.ece>. Checked 25.3.2011.