Morpion Solitaire

Jouni Laitinen

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Timo Poranen
Toukokuu 2010

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
LAITINEN, JOUNI: Morpion Solitaire
Pro gradu -tutkielma, 71 sivua
Toukokuu 2010

Morpion Solitaire on säännöiltään yksinkertainen peli, jonka ratkaiseminen tietokoneella on kuitenkin NP-vaikea optimointiongelma. Tässä tutkielmassa esitellään aluksi erilaisia optimointimenetelmiä, joilla on aikaisemmin saavutettu hyviä tuloksia Morpion Solitairessa. Tämän jälkeen tutkielmassa perehdytään Morpion Solitairen ongelman ominaisuuksiin useiden testien avulla ja pohditaan tarkemmin, miksi hyvien tulosten löytäminen Morpion Solitairessa on vaikeaa. Lopuksi tutkielmassa esitellään tekijän ehdottama uusi ratkaisumenetelmä, jossa pyritään yhdistelemään aikaisemmin esiteltyjen optimointimenetelmien hyviksi todettuja ominaisuuksia sekä tekijän omia huomioita Morpion Solitairen ongelmasta. Tekijän ehdottamaa menetelmää arvioidaan kokeellisesti.

Avainsanat ja -sanonnat: Morpion Solitaire, optimointimenetelmät, evoluutioon perustuvat algoritmit.

Sisällys

Kuvat

Taulukot

Algoritmit

1 Johdanto

Pelit ovat monille mielenkiintoista ajanvietettä. Useimmissa peleissä tarkoituksena on joko voittaa tai saavuttaa mahdollisimman hyvä tulos. Voittamiseen tai hyvän tuloksen saavuttamiseen tarvitaan usein jonkinlainen strategia. Useimmissa tapauksissa hyvän voittostrategian muodostaminen vaatii pelaajalta hyvää päättely- tai hahmottamiskykyä. Ihmiset kykenevät kuitenkin luonnostaan ratkaisemaan vaikeitakin pelien pelaamiseen liittyviä ongelmia nopeasti ja hyviä tuloksia saavuttaen. Tietokoneelle peleihin liittyvien ongelmien ratkaiseminen ei kuitenkaan ole yhtä yksikertaista.

Tarkastellaan esimerkiksi shakkipeliä. Shakkipelissä ihmispelaajien on helppo hahmottaa erilaisia toistuvia tilanteita ja muodostaa näiden pohjalta mahdollisia voittostrategioita. Tietokoneelle shakkipelin pelaaminen tai edes seuraavasta siirrosta päättäminen ei kuitenkaan ole yhtä helppoa. Tietokone voi tarkastella peliä vain joukkona siirtymiä tilanteesta toiseen ja pyrkiä arvioimaan ennalta määritellyn ohjelmoinnin asettamissa rajoissa, mikä siirtymistä saattaisi olla pelin voittamisen kannalta paras mahdollinen.

Yhtenä motivaationa peleihin liittyvien ongelmien tutkimiseksi tietokoneella on juuri ihmisen ja tietokoneen erilaisuus. Ihmispelaaja tunnistaa ja hahmottaa luonnostaan peleissä esiintyviä kuvioita. Ihmisillä on myös hyvä mielikuvitus ja kyky oppia nopeasti erilaisista toistuvista tilanteista. Tietokone kykenee arvioimaan pelitilannetta vain laskemalla ja pystyy ratkaisemaan pelin vain suorittamalla laillisia siirtymiä tilanteesta toiseen kunnes jokin lopputila saavutetaan. Toisaalta, tietokone on väsymätön suorittaja ja tietokoneiden kasvaneen laskentatehon ansiosta pystytään nopeasti tutkimaan erittäin suuria määriä erilaisia siirtymiä ja näiden vaikutuksia pelin etenemiseen.

Voidaan kuitenkin huomata, että monet peleistä ovat luonnostaan vaikeita tietokoneilla ratkaistaviksi. Monissa peleissä pelaajalle annetaan joukko vaihtoehtoja, esimerkiksi mahdolliset siirrot shakkipelissä, joista yhden valitseminen johtaa täysin uuteen pelitilanteeseen ja uudessa pelitilanteessa pelaaja joutuu valitsemaan täysin uusista vaihtoehdoista. Voidaan helposti huomata, että näiden valintojen määrän kasvaessa erilaisten mahdollisten pelitilanteiden määrä kasvaa eksponentiaalisesti. Useissa peleissä mahdollisia pelitilanteita onkin niin suuri määrä, että koko pelin läpikäyminen ei ole järkevää tai se on jopa mahdotonta. Koska kaikkien mahdollisuuksien läpikäyminen ei ole mahdollista, peleihin liittyvien ongelmien ratkaisemiseksi tietokoneella pyritään usein kehittämään jokin tehokkaampi tapa löytää *tarpeeksi hyvä* ratkaisu lyhyemmässä ajassa.

Tässä tutkielmassa keskitytään tarkastelemaan erityisesti *Morpion Solitaire* -pelin tietokoneella ratkaisemiseen käytettäviä menetelmiä. Monien muiden pelien tapaan, myös Morpion Solitaire on tietokoneelle vaikea ongelma ratkaistavaksi. Morpion Solitairessa hyvän tuloksen saavuttaminen on vaikeaa ja mahdollisten siirtojen suuren määrän ansiosta erilaisia pelitilanteita on erittäin suuri määrä. Myös Pascal Zimmer* huomasi tämän laskiessaan Morpion Solitairen mahdollisten pelitilanteiden lukumäärän 20 ensimmäisen siirron jälkeen. Jättämällä laskuistaan pois keskenään vaihdannaiset pelitilanteet, joissa esiintyivät samat siirrot eri järjestyksissä, Zimmer päätyi yli 22 miljardiin siirroiltaan toisistaan poikkeavaan yhdistelmään. Tämän lisäksi Zimmer arvioi koko pelissä olevan noin 10^{23} siirroiltaan erilaista pelitilannetta.

Vaihtoehtojen suuri määrä vaikeuttaa ongelman ratkaisemista tietokoneella, eikä Morpion Solitairen ongelmalle tällä hetkellä tunneta tehokasta ratkaisumenetelmää. Peli ei kuitenkaan ole mahdoton tavalliselle ihmispelaajalle, koska ihmispelaaja pystyy hahmottamaan Morpion Solitairessa toistuvia kuvioita ja valitsemaan siirtonsa aikaisemman kokemuksen pohjalta. Useista tietokoneella tehdyistä yrityksistä huolimatta, yhdessä suosituimmista Morpion Solitairen pelimuodoista paras tulos on yhä ihmispelaajan käsin pelaamalla saavuttama.

Tämän tutkielman luvussa 2 käsitellään Morpion Solitairea yleisesti pelinä, esitellään pelin säännöt, erilaiset muunnelmat ja kussakin muunnelmassa saavutetut parhaat tulokset sekä tarkastellaan pelin alkuperää.

Tutkielman luvussa 3 perehdytään tarkemmin vaikeiden ongelmien ratkaisemiseen tietokoneella, tarkastellaan optimointimenetelmien toimintaa yleisellä tasolla ja esitellään kokeelliseen algoritmiikkaan kuuluvia käsitteitä.

Luvussa 4 esitellään muutamia toisistaan huomattavasti lähestymistavaltaan poikkeavia optimointimenetelmiä ja esitellään tarkemmin, miten näitä menetelmiä on sovellettu Morpion Solitairen ongelman ratkaisemiseksi.

Tutkielman luvussa 5 perehdytään tarkemmin Morpion Solitaireen tutkimusongelmana ja esitetään tekijän omia huomioita Morpion Solitairen ongelmasta.

Luvussa 6 esitellään tekijän ehdottama uusi menetelmä Morpion Solitairen ongelman ratkaisemiseksi. Uudessa menetelmässä pyritään hyödyntämään luvussa 4 esiteltyjen menetelmien hyviksi todettuja ominaisuuksia sekä luvussa 5 esiteltyjä huomioita Morpion Solitairesta. Luvussa 6 raportoidaan myös uuden menetelmän parametrien testaus sekä menetelmällä saavutetut tulokset.

---

* Pascal Zimmerin saavuttamiin tuloksiin ei liity tieteellistä julkaisua. Zimmerin saavuttamat tulokset on julkaistu Jean-Charles Meyrignacin ylläpitämällä Morpion Solitairen tutkimiseen perehtyvällä sivustolla. [Meyrignac].

2 Morpion Solitairen esittely

Morpion Solitaire on *täydellisen informaation* (perfect information) yksinpeli. Täydellisellä informaatiolla tarkoitetaan, että kaikki aikaisemmin tehdyt siirrot ovat pelaajalle aina näkyvissä. Täydellisen informaation peleissä pelin eteneminen ei riipu sattumanvaraisista tapahtumista, kuten esimerkiksi nopan heitolla saadusta silmäluvusta. Pelaajan päätöksiin vaikuttavat tekijät ovat myös aina näkyvillä, toisin kuin esimerkiksi useissa korttipeleissä, joissa osa korteista on pelaajalta usein piilossa.

Yksinpelejä tarkastellaan usein *päätösteorian* (decision theory) avulla. Päätösteoria voidaan jakaa kahteen osa-alueeseen, *deskriptiiviseen* (descriptive) ja *preskriptiiviseen* (prescriptive) päätösteoriaan. Preskriptiivisessä päätösteoriassa pyritään selvittämään, miten jokin päätös pitäisi tehdä. Deskriptiivisessä päätösteoriassa puolestaan selvitetään, miksi tietynlaisia päätöksiä tehdään. Tässä tutkielmassa Morpion Solitairea tarkastellaan pääasiassa preskriptiiviseltä kannalta. Toisin sanoen, tässä tutkielmassa pyritään selvittämään, miten Morpion Solitairessa eri vaihtoehtojen hyvyyttä voidaan arvioida.

Yksinpelien tarkastelu ei varsinaisesti kuulu *peliteoriaan* (game theory), koska peliteoriassa tarkastellaan useamman kuin yhden toimijan keskinäistä vuorovaikutusta [Ross, 2010]. Toisin sanoen, peliteoriassa tarkastellaan tilanteita, joissa myös muiden pelaajien tekemät päätökset voivat vaikuttaa pelaajan tekemien valintojen hyvyyteen.

Morpion Solitairesta on olemassa joitakin muunnoksia, joissa peliä voidaan tietyin siirroin pelata loputtomiin. Tässä tutkielmassa keskitytään kuitenkin ainoastaan tarkastelemaan Morpion Solitairen sellaisia muunnoksia, joissa peliä ei voida jatkaa loputtomiin.

Seuraavaksi esitellään Morpion Solitairen säännöt, yleisimmät pelin muunnelmat ja tarkastellaan hieman tarkemmin pelin historiaa.

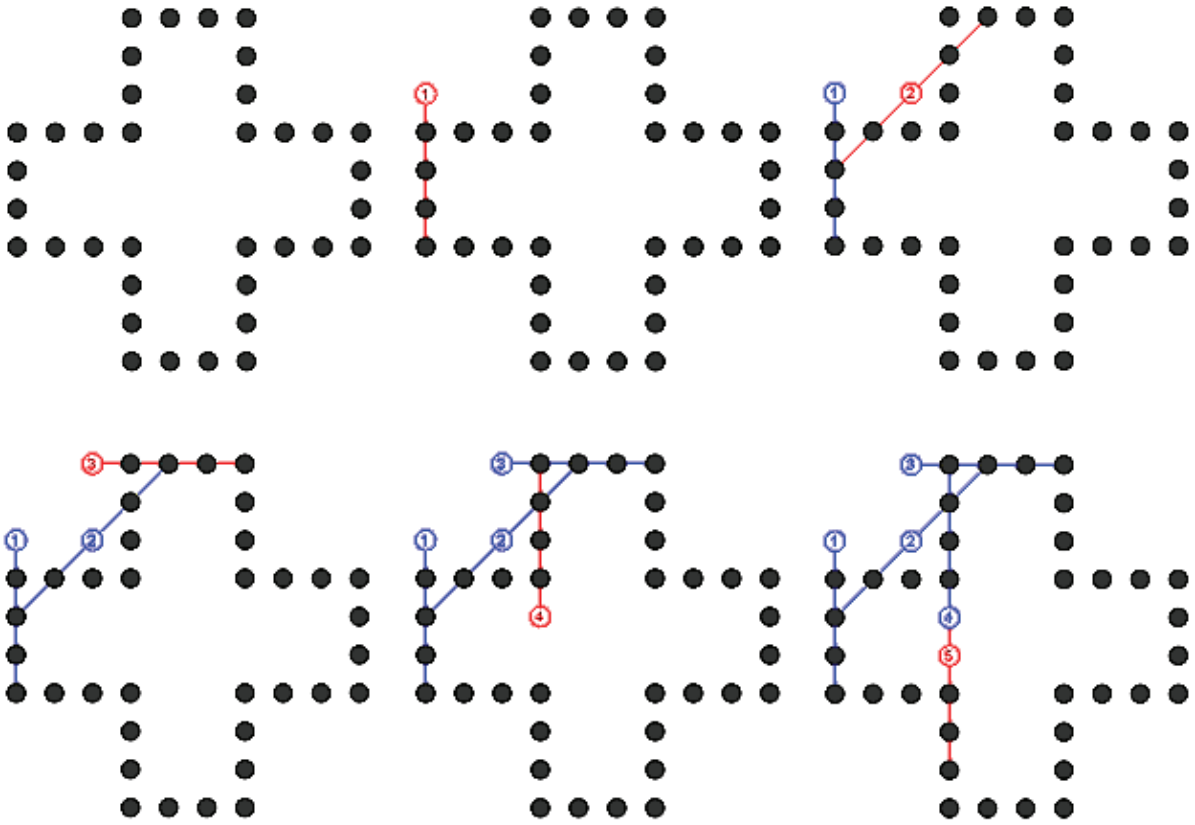
2.1 Morpion Solitairen säännöt ja pelin tarkoitus

Morpion Solitaire on yksinpeli, jossa pelaaja pyrkii tekemään mahdollisimman suuren määrän siirtoja ennen pelin päättymistä. Morpion Solitairen pelialue on kooltaan rajoittamaton ruudukko. Käytännössä Morpion Solitairea voidaan pelata esimerkiksi ruutupaperilla. Ruudukko ei varsinaisesti ole pakollinen, mutta se auttaa huomattavasti pelissä esiintyvien kuvioiden ja mahdollisten siirtojen hahmottamista.

Pelin alussa pelialueelle piirretään pelin aloituskuvio. Suosituimmissa pelin muunnoksissa aloituskuvio on kuvassa 1 esitetty ristin muotoinen kuvio, joka tunnetaan myös *kreikkalaisena ristinä* (greek cross) tai *Solitaire-ristinä* (Solitaire cross) [Boyer].

Jos peliä pelataan ruutupaperilla, aloituskuvion pisteet sijoitetaan perinteisesti ruutupaperin viivojen risteyskohtiin. Tietokoneella pelattavissa pelin toteutuksissa ja kuvissa ruudukko jätetään usein pois kuvan selkeyden parantamiseksi.

Pelin varsinainen peliosuus koostuu pelaajan suorittamista siirroista. Pelaaja lisää pelialueelle jokaisella siirrolla yhden uuden pisteen ja piirtää lisätyn pisteen kautta suoran viivan. Viiva voi kulkea ainoastaan aloituskuvion pisteiden ja aikaisempien siirtojen lisäämien pisteiden kautta ja viivan tulee aina kulkea juuri lisätyn pisteen kautta. Eri suuntiin kulkevat viivat voivat jakaa saman pisteen, samaan suuntaan kulkevat viivat voivat joissain pelin muunnoksissa jakaa viivojen päätepisteen. Viivan pituus vaihtelee pelin eri muunnelmissa, pelin alkuperäisessä muodossa viivan täytyy kulkea viiden pisteen yli. Jos pelimuunnelman viivan pituus on pienempi kuin viisi pistettä, myös aloituskuvio poikkeaa kuvassa 1 esitetystä. Aloituskuvion ristin sakaran leveys on aina viivan pituutta yhtä pienempi. Joissakin pelin muunnoksissa aloituskuvio voi myös olla jokin muu useasta pisteestä muodostuva kuvio. Siirrot voivat kulkea pelialueella joko pystysuoraan tai vaakasuoraan sekä nousevasti tai laskevasti viistoon. Peli loppuu, kun laillisia siirtoja ei ole enää mahdollista tehdä. Pelaajan tarkoituksena on pyrkiä suorittamaan mahdollisimman monta siirtoa ennen pelin loppumista.



Kuva 1. Perinteinen aloituskuvio ja viisi esimerkkisiirtoa, joista viimeinen on mahdollinen vain koskevassa mallissa.

Pelistä on kaksi yleisesti pelattua muunnelmaa, jotka rajoittavat sallittuja siirtoja eri tavoin. *Koskeva malli* (touching model) on pelin alkuperäinen muoto, jossa kaksi samansuuntaista siirtoa voi jakaa yhden yhteisen pisteen, johon toisen siirron lisäämä viiva loppuu ja josta toisen siirron viiva alkaa. *Irrallisessa mallissa* (disjoint model) samansuuntaiset siirrot eivät voi jakaa yhtään pistettä. Yhden pisteen kautta eri suuntiin kulkevien siirtojen määrää ei kuitenkaan ole rajoitettu kummassakaan mallissa, joten jopa neljä tai kahdeksan erillistä siirtoa voi jakaa saman pisteen mallista riippuen. Demaine *et al.* [2006] arvioivat näistä kahdesta muunnelmasta koskevan mallin olevan suositumpi.

Yleisesti eri muunnelmia voidaan merkitä antamalla siirtoja rajoittava malli sekä siirtojen pituus, esimerkiksi 5D irralliselle (disjoint) mallille ja 5T koskevalle (touching) mallille.

2.2 Morpion Solitairen historia ja parhaat tulokset

Morpion Solitairen lyhyeen historiaan mahtuu muutamia mielenkiintoisia ja jopa uskomattomia tapahtumia. Pelin tarkkaa alkuperää ei oletettavasti tiedetä, mutta Morpion Solitairen arvellaan jossain vaiheessa syntyneen kahden Ranskassa suositun pelin yhdistelmänä, jotka tunnetaan nimillä Morpion ja Solitaire [Boyer]. Morpion on itse asiassa Ranskassa käytetty nimi perinteisen ristinollan muunnelmalle, jossa kaksi pelaajaa pelaa rajoittamattomalla ruudukolla ja jossa tarkoituksena on saada viisi omaa merkkiä riviin. Kyseinen muunnelma tunnetaan Suomessa myös ”jätkänshakkina”. Solitaire on Englannissa ja Ranskassa suosittu yksinpeli, jossa esiintyy samankaltainen ristin muotoinen kuvio, kuin Morpion Solitairessa. Todennäköisesti juuri tästä syystä jotkin lähteet kutsuvat aloituskuviota Solitaire-ristiksi. Toisaalta, Demaine *et al.* [2006] esittävät, että Morpion Solitairea kutsutaan myös nimellä *Maltan risti* (Malta cross) ja aloituskuviota kutsutaan joissain lähteissä myös Maltan ristiksi. Oikea Maltan risti poikkeaa kuitenkin huomattavasti muodoltaan Morpion Solitairen ristikuviosta, joten nimitys ei välttämättä ole paras mahdollinen.

Ensimmäinen peliin liittyvä julkaisu oli ranskalaisessa Science & Vie -lehdessä vuonna 1974 Pierre Berloquinin kolumnissa Jeux & Paradoxes [Boyer]. Muun muassa Christian Boyer on yrittänyt selvittää Morpion Solitairen tarkkaa alkuperää ja on onnistunut löytämään henkilöitä, jotka ovat pelanneet alkuperäistä peliä jo vuonna 1962 [Boyer]. Peli on eksynyt Suomeenkin jossain vaiheessa. Timo Poranen [2010] muistaa pelanneensa peliä vuonna 1987. Erityisesti peli on suosittu Ranskassa, jossa peli tuli laajemmin tunnetuksi Berloquinin kolumnin kautta [Boyer].

Alkuperäisen pelin oletetaan olleen viiden mittaisten siirtojen koskeva malli ja muiden muunnelmien oletetaan myöhemmin kehittyneen tästä. Irrallisen malli saattoi hyvinkin

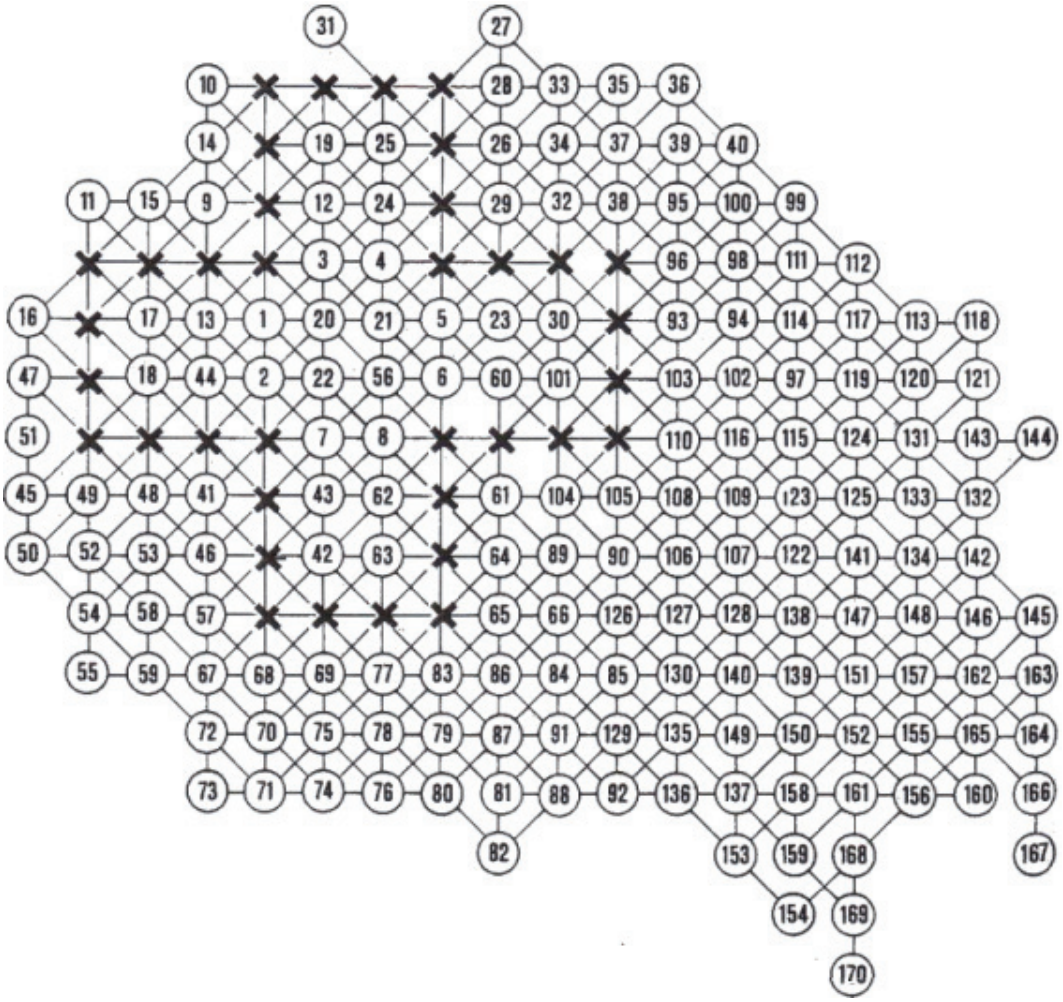
syntyä aivan vahingossa, kun peliä on opetettu eteenpäin. Yhden pisteen jakamisen mahdollisuus kahden eri siirron kesken on saatettu yksinkertaisesti unohtaa mainita ja tämä on johtanut täysin uuteen tapaan pelata peliä [Boyer]. Muunnelmien 4T ja 4D oletetaan syntyneen vasta vuonna 2002, kun Demaine *et al.* [2006][*] keksivät tutkia Morpion Solitairen ongelmaa perinteisestä mallista poikkeavilla siirtojen pituuksilla [Boyer].

Pelistä löytyy myös useita vähemmän tunnettuja muunnelmia. Esimerkiksi Poranen [2010] kertoo käyttäneensä aikaisemmin sääntöä, jossa lisätyn viivan ei välttämättä tarvinnut kulkea kyseisellä siirrolla lisätyn pisteen kautta. Uuden pisteen saattoi lisätä haluamaansa kohtaan pelialueella, jos pelistä löytyi jo viiden pisteen mittainen suora, johon siirron saattoi asettaa. Muita muunnelmia ovat esimerkiksi kahden pelaajan pelattavat muunnelmat Connector ja Morpiae sekä erilaisia aloituskuvioita käyttävät muunnelmat [Boyer].

Koskevan mallin paras julkaistu tulos on käsin pelaamalla saavutettu 170 siirtoa, jonka virallisesti ensimmäisenä saavutti Charles-Henri Bruneau vuonna 1976 [Boyer]. Myös J.B. Bonté saavutti tuloksen 170 ja päätyi jopa hieman erilaiseen ratkaisuun kuin Charles-Henri Bruneau [Boyer]. Bruneaun löytämä 170 siirron ratkaisu on esitetty kuvassa 2.

Bruneaun saavuttamaan huipputulokseen liittyy myös mielenkiintoinen tarina. Alun perin parasta tulosta luultiin Denis Excoffierin saavuttamaksi [Meyrignac]. Todellisuudessa Excoffierin ratkaisu oli Bruneaun Science & Vie -lehdessä [1976] julkaistu ratkaisu, Excoffier oli vain kopioinut Bruneaun ratkaisun ja väitti sitä myöhemmin omakseen. Bruneau, joka ei ollut säästänyt kopiota ratkaisustaan, ei kuitenkaan tiennyt tätä ja luuli itse asiassa, ettei hänen ratkaisuaan koskaan julkaistu lehdessä. Vuonna 2008 Christian Boyer, saadessaan käsiinsä Pierre Berloquinin Science & Vie -lehteen liittyvät arkistot, huomasi kuitenkin ratkaisun kuuluneen alun perin Bruneaulle. Uskomattoman tarinan lopuksi Bruneau sai ensimmäistä kertaa kuulla saavuttaneensa vieläkin voimassaolevan huipputuloksen vasta Boyerin ottaessa häneen yhteyttä 32 vuotta myöhemmin. [Boyer]

[*] Demaine et al. [2006] artikkelin lopullinen versio julkaistiin vuonna 2006. Pelimuunnelmat mainitaan ensimmäisen kerran artikkelista vuonna 2004 julkaistussa versiossa.



Kuva 2. Charles-Henri Bruneaun Science & Vie -lehdessä julkaistu 170 siirron tulos.

Myös tietokoneella on päästy hyviin tuloksiin, joskin tähän asti ne ovat jääneet huomattavasti näitä käsintehtyjä huipputuloksia pienemmiksi koskevassa mallissa. Paras tietokoneella saavutettu tulos koskevassa mallissa on 145, jonka saavutti Haruhiko Akiyama käyttämällä Tristan Cazenaven [2009] kehittämää sisäkkäistä Monte Carlo -hakua. Joissakin lähteissä [Helmstetter and Cazenave, 2004; Cazenave, 2007] Pascal Zimmerin mainitaan saavuttaneen jopa 147 siirron tuloksen, mutta myöhemmin ratkaisun on todettu sisältävän vain 143 siirtoa [Boyer]. Irrallisen mallin paras julkaistu tulos on 80, joka on myös saavutettu Cazenaven [2009] sisäkkäistä Monte Carlo -hakua käyttäen.

Taulukossa 1 on esitetty nämä ja useita muita koskevan ja irrallisen mallin merkittäviä tuloksia. Taulukossa 1 kunkin mallin parhaat tulokset on esitetty tummemmalla taustalla. Esitetyistä tuloksista 4T ja 4D mallien parhaat tulokset ovat todistetusti parhaat mahdolliset. Koska 4T ja 4D malleissa on huomattavasti vähemmän mahdollisia erilaisia pelejä kuin 5T ja 5D malleissa, Michael Quist onnistui käymään läpi kaikki mahdolliset siirtojen yhdistelmät tietokoneella ja näin ollen taulukossa esitetyt 4T ja 4D mallien tulokset ovat todistetusti parhaat mahdolliset saavutettavissa olevat [Boyer].

Malli	Siirtoja	Tekijä	Maa	Aika	Menetelmä
4D	31	Erik D. Demaine, Martin L. Demaine, Arthur Langerman ja Stefan Langerman	Yhdysvallat, Belgia	Toukokuu 2004	-
	35*	Heikki Hyyrö ja Timo Poranen	Suomi	Lokakuu 2007	Simuloitu jäähdytys
4T	56	Erik D. Demaine, Martin L. Demaine, Arthur Langerman ja Stefan Langerman	Yhdysvallat, Belgia	Toukokuu 2004	-
	62*	Heikki Hyyrö ja Timo Poranen	Suomi	Lokakuu 2007	Simuloitu jäähdytys
5D	66	Stefan Schmieta	Yhdysvallat	Lokakuu 1999	-
	68	Arthur Langerman	Belgia	Lokakuu 1999	-
	69	Bernard Helmstetter	Ranska	Syyskuu 2005	-
	74	Heikki Hyyrö ja Timo Poranen	Suomi	Joulukuu 2006	Simuloitu jäähdytys
	78	Tristan Cazenave	Ranska	Toukokuu 2007	Refleksiivinen Monte Carlo -haku
	79	Heikki Hyyrö ja Timo Poranen	Suomi	Kesäkuu 2007	Simuloitu jäähdytys
	80	Tristan Cazenave	Ranska	Helmikuu 2008	Sisäkkäinen Monte Carlo -haku
5T käsin pelaamalla saavutettu	162	Rémy Daubié	Ranska	Marraskuu 1974	-
	163	Charles William Millington	Englanti	Heinäkuu 1975	-
	164	Michel Szeps	Ranska	Marraskuu 1975	-
	164	Joseph Martin	Ranska	Marraskuu 1975	-
	164	Yoland Strehl	Ranska	Marraskuu 1975	-
	170	Charles-Henri Bruneau	Ranska	Huhtikuu 1976	-
5T tietokoneella saavutettu	117 121	Hugues Juillé	Espanja	1995 1999	END SAGE
	143	Pascal Zimmer	Ranska	Tammikuu 2003	END
	143	Heikki Hyyrö ja Timo Poranen	Suomi	Kesäkuu 2007	Simuloitu jäähdytys
	144	Tristan Cazenave	Ranska	Joulukuu 2007	Refleksiivinen Monte Carlo -haku
	145	Haruhiko Akiyama	Japani	Helmikuu 2010	Sisäkkäinen Monte Carlo -haku

Taulukko 1. Parhaat saavutetut tulokset eri Morpion Solitairen muunnoksissa.

* Tulosta ei voi parantaa, koska se on todistetusti kyseisen pelimuunnelman paras mahdollinen tulos.

3 Vaikeat ongelmat ja kokeellinen algoritmiikka

Demaine *et al.* [2006] todistivat tutkimuksessaan, että Morpion Solitairen ongelma on *NP-vaikea* (NP-hard). Tämän lisäksi on myös helppo huomata, että mahdollisten siirtojen eri yhdistelmien suuren määrän ansiosta, kaikkien eri ratkaisuvaihtoehtojen läpikäyminen parhaan ratkaisun löytämiseksi ei ole mahdollista.

Tässä luvussa perehdytään ensimmäiseksi yleisellä tasolla NP-vaikeiden ongelmien ratkaisemiseen käytettäviin menetelmiin. Luvussa käsitellään myös kokeellisen algoritmiikan peruskäsitteet ja tarkastellaan kokeellista algoritmiikkaa algoritmien tutkimusmenetelmänä.

3.1 NP-vaikeat optimointiongelmat

NP-vaikeiden ongelmien määritelmän selittämiseksi määritellään ensin ongelmien luokat *P* (polynomial time), *NP* (nondeterministic polynomial time) ja *NP-täydellinen* (NP-complete) [Garey and Johnson, 1979].

Ongelmien luokkaan P kuuluvat ne *päätösongelmat* (decision problems), jotka ovat ratkaistavissa polynomisessa ajassa deterministisellä *Turingin koneella* (Turing machine). Päätösongelmat ovat ongelmia, joissa pyritään vastaamaan ainoastaan ”kyllä” tai ”ei” annettuun kysymykseen. Päätösongelmassa voidaan esimerkiksi pyrkiä ratkaisemaan, onko verkko A verkon B aliverkko, johon voidaan yksinkertaisesti vastata kyllä tai ei. Turingin kone puolestaan on Alan Turingin [1936] kehittämä abstrakti malli tietokoneen toiminnasta. Deterministinen Turingin kone, kuten kaikki deterministiset algoritmit, toimii samalla syötteellä aina samoin ja tuottaa samalla syötteellä aina saman tuloksen.

Luokkaan NP kuuluvat ne luokan P ongelmat, jotka voidaan ratkaista polynomisessa ajassa epädeterministisellä Turingin koneella. Epädeterministinen Turingin kone sisältää muiden epädeterminististen algoritmien tapaan yhden tai useamman kohdan, jossa toimintavaihtoehto ei ole yksikäsitteinen. Toisin sanoen, kyseisessä kohdassa saatetaan eri suorituskerralla valita jokin muu toimintavaihtoehto kuin aikaisemmin ja näin päätyä eri tulokseen. Esimerkiksi kaikki satunnaisuuteen perustuvat algoritmit ovat epädeterministisiä.

NP-täydelliset ongelmat ovat luokkaan NP kuuluvien erittäin vaikeiden ongelmien muodostama alijoukko. Ongelma on NP-täydellinen, jos se kuuluu luokkaan NP ja jokainen luokkaan NP kuuluva ongelma voidaan polynomisessa ajassa pelkistää samaksi ongelmaksi. NP-täydellisen ongelman ratkaisemisen ei sinänsä tarvitse olla vaikeaa. Ongelman ratkaisemisen hankaluus perustuu ratkaisemisen vaatimaan suoritusaikaan, koska NP-täydellisille ongelmille tunnetaan vain eksponentiaalisen ajan

algoritmeja. Jos mille tahansa NP-täydelliselle ongelmalle onnistuttaisiin löytämään polynomisessa ajassa toimiva ratkaisu, määritelmästä seuraisi, että kaikki luokan NP ongelmat voitaisiin ratkaista deterministisellä Turingin koneella polynomisessa ajassa, jolloin $P = NP$ pitäisi paikkansa. Esimerkiksi kauppamatkustajan ongelma [Garey and Johnson, 1979] on NP-täydellinen ongelma.

NP-vaikeat ongelmat määritellään ongelmiksi, jotka ovat ”vähintään yhtä vaikeita, kuin NP-täydelliset ongelmat”. NP-vaikeat ongelmat eivät kuitenkaan ole NP-täydellisten ongelmien alijoukko. Toisin kuin NP-täydellisten ongelmien kohdalla, NP-vaikean ongelman ei tarvitse välttämättä edes kuulua luokkaan NP. NP-vaikeisiin ongelmiin kuuluu päätösongelmien lisäksi myös *optimointiongelmia* (optimization problems) sekä *hakuongelmia* (search problems).

Hyvän tuloksen etsiminen Morpion Solitairessa on NP-vaikea optimointiongelma. Optimointiongelma on ongelma, jossa pyritään löytämään ongelman kannalta paras ratkaisu kaikista mahdollisista ratkaisuista. Ongelman paras mahdollinen ratkaisu tunnetaan myös *globaalisti optimina ratkaisuna* (global optimum). Kuten yllä esitettiin, NP-vaikeiden ongelmien täydellinen ratkaiseminen kohtuullisessa ajassa on vaikeaa, ellei jopa mahdotonta. Tästä johtuen useimmiten tyydytään etsimään ”riittävän hyvä” tulos *approksimointialgoritmien* (approximation algorithm) tai erilaisten *heuristiikkojen* (heuristic) avulla.

Yleisesti tietojenkäsittelyssä oletetaan, että algoritmien voidaan todistaa toimivan tehokkaasti ja tuottavan hyviä tuloksia. Heuristiikkojen tapauksessa näitä oletuksia ei ole. Heuristiikoilla tarkoitetaan siis menetelmiä, jotka käytännössä johtavat usein nopeasti hyvään tulokseen, mutta hyvää tulosta tai nopeaa toimintaa ei pystytä takaamaan tai todistamaan, joten heuristiikan tuottama tulos voi olla myös erittäin huono. Heuristiikat voivat olla esimerkiksi ongelmakohtaisia ominaisuuksia, joita pystytään käyttämään hyväksi valintoja tehtäessä. Heuristiikkoja käytetään usein approksimointialgoritmien osana.

Approksimointialgoritmeilla pyritään löytämään annettuun ongelmaan riittävän lähellä parasta mahdollista ratkaisua oleva ratkaisu. Toisin sanoen, approksimointialgoritmilla pyritään etsimään likimääräisesti paras mahdollinen ratkaisu. Toisin kuin heuristiikkojen kohdalla, approksimointialgoritmin tuottaman ratkaisun laatua voidaan yleensä arvioida ja samoilla algoritmin parametreilla saavutetaan usein hyvyydeltään samankaltaisia tuloksia.

Esimerkkinä approksimointialgoritmeista voidaan tarkastella paikalliseen etsintään perustuvia algoritmeja. Paikalliseen etsintään perustuvat algoritmit aloittavat useimmiten sattumanvaraisesta ratkaisuehdotuksesta ja yksinkertaisia siirtymiä tehden pyrkivät parantamaan ratkaisua. Yksinkertaisessa ahneessa algoritmissa siirtymä voisi

esimerkiksi olla tarkasteltavan ratkaisuehdotuksen parhaimman naapurin valitseminen uudeksi tarkastelun kohteeksi.

Yksinkertaisessa algoritmissa kyseistä siirtymää voitaisiin toistaa, kunnes saatu tulos ei enää muutu eli tulos on naapurustossaan paras mahdollinen. Toisin sanoen, algoritmi on päätynyt *lokaaliin optimiin* (local optimum). Lokaalisti optimilla ratkaisulla tarkoitetaan ratkaisua, joka on lähinaapurustossaan paras mahdollinen ratkaisu, mutta ei kuitenkaan välttämättä ole koko ongelman paras mahdollinen ratkaisu. Ongelmalla voi olla useita lokaalisti optimaalisia ratkaisuja, jotka voivat kuitenkin olla erittäin kaukana globaalista optimista. Toisaalta, usein globaalin optimin arvoa ei tiedetä etukäteen ja lokaalisti optimi ratkaisu voi hyvinkin olla koko ratkaisun paras mahdollinen ratkaisu tai ainakin ”riittävän hyvä” arvio. Useimmissa tapauksissa yhden lokaalin optimin selvittäminen ei kuitenkaan ole riittävä arvio hyvästä ratkaisusta.

Lokaalista optimista voidaan myös yrittää siirtyä pois, jolloin päästään tutkimaan uusia mielenkiintoisia ratkaisuehdotuksia. Esimerkiksi *simuloituun jäähdytykseen* (simulated annealing) [Kirkpatrick *et al.*, 1983] perustuvissa algoritmeissa voidaan tiettyjen sääntöjen mukaisesti valita myös siirtymä huonompaan suuntaan. Huonompien siirtymien ollessa mahdollisia, ratkaisun etsintä ei ajaudu jumiin lokaalisti optimiin tulokseen yhtä helposti.

Toinen lähestymistapa approksimointiin on sisällyttää ratkaisun etsimiseen satunnaisuutta. Satunnaisuuden johdosta algoritmi on epädeterministisen, koska algoritmi voi saavuttaa eri tuloksen samalla syötteellä. Satunnaisuuden käyttämiseen on monia syitä. Joissakin ongelmissa valintojen hyvyyttä ei voida nopeasti arvioida, jolloin on yksinkertaisempaa valita mahdollisista valinnoista valita yksi sattumanvaraisesti ja pyrkiä arvioimaan tehdyn valinnan hyvyyttä myöhemmin saavutettuihin tuloksiin perustuen. Toisaalta, sattumanvaraisia siirtymiä valitsemalla voidaan myös osittain ehkäistä lokaaliin optimiin ajautumista ja mahdollisesti tutkia lisää mielenkiintoisia hakupolkuja.

Satunnaisuuteen perustuvat algoritmit voidaan jakaa kahteen isoon luokkaan: Monte Carlo -algoritmit ja Las Vegas -algoritmit. Monte Carlo -algoritmit palauttavat aina jonkin tuloksen, mutta tulos ei välttämättä ole paras mahdollinen tai voi pienellä todennäköisyydellä olla väärä vastaus. Las Vegas-algoritmit palauttavat aina oikean vastauksen tai parhaan mahdollisen tuloksen, mutta Las Vegas-algoritmien suoritusaika on täysin sattumanvarainen. Jos Las Vegas-algoritmin suoritusaika rajoitetaan, algoritmi palauttaa joko oikean vastauksen tai ilmoittaa, ettei vastausta voitu löytää [Motwani and Raghavan, 1995].

3.2 Kokeellinen algoritmiikka

Tässä tutkielmassa tullaan myöhemmin tarkastelemaan eri algoritmeja kokeellisesti, joten tässä vaiheessa on aiheellista käsitellä hieman kokeellisen algoritmiikan peruskäsitteitä.

Johnson [2001] esittää, että algoritmiikassa voidaan yleisesti erottaa kolme erilaista lähestymistapaa algoritmien tutkimiseen: pahimman tapauksen tutkiminen, keskimääräisen tapauksen tutkiminen sekä kokeellinen tutkiminen. Pahimman ja keskimääräisen tapauksen tutkiminen ovat analyyttisiä lähestymistapoja tutkia algoritmeja [Anderson, 1997]. Näiden menetelmien vahvuutena on Johnsonin [2001] mukaan vankka matemaattinen pohja, mutta algoritmeja ei yleensä oikeasti toteuteta ja testata. Pahimman ja keskimääräisen tapauksen tutkimisen avulla voidaan tarkastella helposti ja tehokkaasti algoritmien asymptoottista käyttäytymistä ja saadaan tuloksia, jotka pätevät yleisemmällä tasolla kuin kokeellisella tutkimisella saavutetut yksittäiseen toteutukseen perustuvat tulokset. Asymptoottisella käyttäytymisellä tarkoitetaan algoritmin suoritusajan käyttäytymistä ongelman koon kasvaessa.

Toisaalta, näillä teoreettisemmilla tavoilla saavutetut tulokset eivät kerro koko totuutta algoritmien todellisesta tehokkuudesta käytännön sovelluksissa. Esimerkiksi Moret ja Shapiro [2001] antavat tutkimuksessaan esimerkin algoritmista, jolla voidaan tutkia onko jokin verkko toisen verkon *minori* (graph minor). Kyseinen algoritmi on asymptoottisessa mielessä ongelman vaikeuden kannalta tehokas, mutta algoritmin sisältämien suurien vakiotermien takia algoritmin soveltaminen on käytännössä kuitenkin mahdotonta [Moret and Shapiro, 2001]. Asymptoottisten suoritustehokkuusrajojen määrittäminen voi myös olla erittäin vaikeaa joillekin ongelmille. Erityisesti vaikeasti määriteltävinä Moret ja Shapiro [2001] mainitsevat optimointimenetelmät NP-vaikeille ongelmille, joihin myös Morpion Solitaire kuuluu.

Yksinkertaisimmillaan kokeellinen algoritmiikka voidaan käsittää algoritmien toteuttamisena, testaamisena ja testituloksien raportointina. Toisin kuin teoreettisemmissa menetelmissä, kokeellisen algoritmiikan tapauksessa tehokkuutta mitataan havaintojen ja tilastotieteen menetelmien avulla [McGeoch, 1996]. Kokeellisesta tutkimisesta voidaan myös hyötyä yllättävillä tavoilla. Algoritmia toteutettaessa voidaan joutua ratkaisemaan monia algoritmin toteuttamiseen liittyviä ongelmia ja näiden kautta alkuperäisestä ongelmasta saatetaan oppia lisää [Moret and Shapiro, 2001]. Algoritmia toteuttaessa ja testattaessa voidaan myös helposti keksiä täysin uusia suuntia tutkittavaksi. Kokeellinen algoritmiikka onkin vahvasti iteratiivista: testien pohjalta saadaan uusia havaintoja ongelmasta ja hypoteeseja, joiden paikkansapitävyyttä voidaan testata uusilla testeillä [McGeoch, 1996].

Kokeellisessa algoritmiikassa pyritään myös asettamaan jonkinlaiset yleiset säännöt algoritmien testaamiselle. Johnson [2001] määrittelee muun muassa seuraavat asiat, jotka tulisi ottaa testaamisen yhteydessä huomioon.

- Toistettavuus
- Vertailukelpoisuus
- Tulosten esittäminen informatiivisella tavalla

Toistettavuuteen kuuluu yleisesti testitilanteiden tarkka dokumentointi. Esimerkiksi testissä käytetyn algoritmin parametrit sekä mahdolliset syötteet tulisi kirjata ylös. Johnson [2001] ehdottaa myös, että algoritmin tulosteiden tulisi olla itsensä selittäviä eli algoritmin tulisi sisällyttää tulosteeseen testissä käytetyt parametrit. Tämä auttaa myös tutkijaa itseään, koska tällöin tutkijan ei tarvitse pelkästään luottaa muistiinsa, millä parametreilla mikäkin tulos saatiin.

Vertailukelpoisuudella tarkoitetaan tulosten vertailtavuutta keskenään. Esimerkiksi kahden eri algoritmin saavuttamat tulokset eri koneilla suoritettuna eivät selvästikään ole vertailukelpoisia. Tuloksien vertailu on tietysti helpointa, jos kaikkien verrattavien algoritmien toteutukset ovat saatavilla, koska tämän jälkeen verrattavat algoritmit voidaan ajaa samassa testiympäristössä [Johnson, 2001]. Muussa tapauksessa tuloksia voidaan yrittää normalisoida muun muassa *vertailutestien* (benchmark) avulla.

Kokeellisen tutkimuksen tulosten esittämiseen liittyen esimerkiksi Sanders [2002] käsittelee tutkimuksessaan taulukoihin ja graafiseen esittämiseen liittyviä käsitteitä. Myös Johnson [2001] ja McGeoch [1996] käsittelevät tulosten selkeää esittämistä tarkemmin. Erityistä huomiota tutkimuksissa kiinnitetään tulosten selkeään graafiseen esittämiseen sekä tulosten helpon luettavuuden ja ymmärrettävyyden säilyttämiseen.

Näiden seikkojen lisäksi Moret ja Shapiro [2001] nostavat tärkeäksi asiaksi myös testeissä mitattavien muuttujien valinnan. Mielenkiintoisia muuttujia ovat luonnollisesti suoritustehokkuus ja tuloksen laatu approksimointialgoritmien tapauksessa, mutta myös muita mielenkiintoisia testattavia muuttujia voidaan löytää helposti. Erityisesti, tavallisesta poikkeavat tapahtumat tai tuloksissa toistuvasti ilmaantuvat piirteet ovat usein mielenkiintoisia tutkimuksen kohteita [Moret and Shapiro, 2001].

Lopuksi voidaan vielä mainita Zobelin [1997] esittämä huomio kokeellisen tutkimuksen tulosten kirjaamisesta ja tutkimuksen luotettavuudesta. Zobelin mukaan, toisin kuin muilla aloilla, tietojenkäsittelyn alalla ei ole yleisesti hyväksyttyjä normeja tutkimustulosten kirjaamiseksi ja säilyttämiseksi. Zobel käsittelee tutkimuksessaan useita hyviä syitä tulosten oikeaoppiseen kirjaamiseen sekä lähdekoodien eri versioiden säilyttämiseen. Muun muassa säilyttämällä kokeellisesta testauksesta saadut tulokset, tutkijan on helpompaa suojella itseään mahdollisissa ongelmatilanteissa. Testien

tulosten olemassaolo käy esimerkiksi todisteesta, että tuloksia ei ole vain keksitty tyhjästä tai plagioitu [Zobel, 1997]. Noudattamalla myös aikaisemmin mainittua testien toistettavuuden vaatimusta, säilytettyjen testitulosten avulla muiden on myös helppo myöhemmin varmistaa tulosten oikeellisuus.

4 Heuristisia optimointimenetelmiä

Tässä luvussa käsitellään joitakin heuristisia optimointimenetelmiä, joita on jo aikaisemmissa tutkimuksissa käytetty Morpion Solitairen ongelman ratkaisemiseen. Jokaista menetelmää tarkastellaan ensiksi yleisellä tasolla ja tämän jälkeen tarkastellaan, miten menetelmää on sovellettu Morpion Solitaireen.

4.1 Satunnaisotanta

Satunnaisotanta (random sampling) on yksinkertainen tapa yrittää saavuttaa hyviä tuloksia nopeasti. Satunnaisotantaan perustuvat algoritmit ovat aina epädeterministisiä, koska satunnaisotantaan perustuvassa algoritmissa on vähintään yhdessä kohdassa useita erillisiä toimintaohjeita, mutta toimintaa ei ole ennalta määritelty. Tästä johtuen satunnaisotannalle on ominaista, että samalla syötteellä voidaan saada eri tuloksia. Joissakin tapauksissa myös suoritustehokkuus voi vaihdella satunnaisuuden ansiosta [Motwani and Raghavan, 1995]. Satunnaisotantaan perustuvia algoritmeja käytetään usein, jos siirtymien hyvyyttä on vaikea arvioida.

Satunnaisotantaan perustuvien algoritmien perusperiaatteita on todennäköisesti helpompi tarkastella konkreettisen esimerkin avulla. Hyyrö ja Poranen [2007] esittävät tutkimuksessaan kaksi erilaista satunnaisotantaan perustuvaa algoritmia Morpion Solitairen ratkaisemiseksi.

Hyyrön ja Porasen [2007] ensimmäinen satunnaisotantaan perustuva algoritmi (RS1) käy läpi kaikki mahdolliset sallitut siirrot kussakin pelin tilanteessa, arvioi niiden sopivuutta satunnaisotannan avulla ja valitsee siirron, joka on saavutettujen tuloksien perusteella sopivin. Algoritmi arvioi siirron sopivuutta suorittamalla ensiksi arvioitavana olevan siirron ja sitten pelaamalla k kappaletta satunnaisia pelejä tästä tilasta. Jokaista mahdollista siirtoa kohti voidaan esimerkiksi pelata kymmenen satunnaista peliä loppuun asti valitsemalla satunnaisia siirtoja. Siirron sopivuuden arvoksi asetetaan kaikkien tästä siirrosta pelattujen satunnaisten pelien tulosten keskiarvo. Kun kaikki siirrot on arvioitu, valitaan korkeimman sopivuusarvon saanut siirto pysyvästi ja siirrytään valitsemaan seuraavaa siirtoa vastaavalla tavalla. Algoritmin toiminta on esitetty pseudokoodina algoritmissa 1 [Hyyrö and Poranen, 2007].

```
1    while mahdollisten siirtojen määrä > 0 do
2      p := tallenna pelitilanne

3      for each siirto in mahdolliset siirrot do
4        pelitilanne := p + siirto
5        for i := 1 to k do
6          pelaa sattumanvarainen peli uudesta pelitilanteesta
7          laske pelien keskiarvo
8        endfor
9      endfor

10     paras siirto := parhaimman keskiarvon saavuttanut siirto
11     uusi pelitilanne := p + paras siirto
12   endwhile
```

Algoritmi 1. Hyyrön ja Porasen RS1 algoritmi.

Toinen Hyyrön ja Porasen [2007] esittelemä satunnaisotantaan perustuva algoritmi (RS2) ei edellisen algoritmin tapaan erityisesti käy läpi kaikkia mahdollisia siirtoja. Algoritmi on itse asiassa edellistä yksinkertaisempi, koska algoritmi vain pelaa k kappaletta täysin satunnaisia pelejä annetusta tilanteesta loppuun asti. Kun määritelty määrä pelejä on pelattu, algoritmi valitsee parhaimman tuloksen saavuttaneen satunnaispelin seuraavan siirron pysyväksi siirroksi ja siirtyy valitsemaan seuraavaa siirtoa vastaavalla tavalla. RS2 algoritmin toiminta on esitetty pseudokoodina algoritmissa 2 [Hyyrö and Poranen, 2007].

```
1    while mahdollisten siirtojen määrä > 0 do
2      p := tallenna pelitilanne
3      paras tulos := 0

4      for i := 1 to k do
5        pelaa sattumanvarainen peli tilanteesta p
6        if pelin tulos > paras tulos then
7          paras siirto := sattumanvaraisen pelin seuraava siirto
8        endif
9      endfor

10     pelitilanne := p + paras siirto
11   endwhile
```

Algoritmi 2. Hyyrön ja Porasen RS2 algoritmi.

Molemmat algoritmit toistavat valintaprosessia, kunnes mahdollisia siirtoja ei enää ole. Jos k on satunnaisten pelattavien pelien määrä kullakin tasolla ja x on mahdollisten siirtojen määrä annetussa pelitilanteessa, Hyyrön ja Porasen [2007] mukaan ensimmäinen algoritmi tekee noin x kertaa yhtä monta siirtoa kuin jälkimmäinen. Tästä

johtuen ensimmäinen algoritmi on huomattavasti jälkimmäistä hitaampi. Toisaalta, satunnaisten pelien määrän siirtoa kohden ollessa vakio, ensimmäinen algoritmi saavuttaa yleisesti parempia tuloksia kuin jälkimmäinen. Jos jälkimmäisen algoritmin annetaan käydä läpi vastaava määrä satunnaisia pelejä kuin ensimmäisen algoritmin, suoritusajat ja saadut tulokset ovat samankaltaiset [Hyyrö and Poranen, 2007].

Hyyrön ja Porasen [2007] satunnaisotantaan perustuvista algoritmeista saadaan hyvä yleiskuva siitä, miten satunnaisuutta ja satunnaisotantaa voidaan käyttää hyväksi ongelman ratkaisussa valintojen hyvyyden määrittämisen ollessa vaikeaa. Satunnaisotantaa käytetään usein osana monimutkaisempia approksimointialgoritmeja.

Taulukossa 2 on esitetty Hyyrön ja Porasen [2007] saavuttamat tulokset 5D ja 5T malleille. Hyyrö ja Poranen toistivat testit 20 kertaa ja taulukossa annettu luku on testeissä saatujen tuloksien keskiarvo. Taulukossa 2 annettu arvo k on algoritmin pelaamien satunnaisten pelien määrä siirron valintatilanteessa.

	Saavutettujen tulosten keskiarvo arvolla k				
Malli/Menetelmä	**k = 1**	**k = 10**	**k = 100**	**k = 1000**	**k = 10000**
5D / RS1	60.1	61.5	63.0	63.9	64.4
5D / RS2	56.8	60.0	61.8	62.7	64.3
5T / RS1	80.9	85.7	88.9	93.7	96.5
5T / RS2	73.4	80.4	86.2	90.9	94.0

Taulukko 2. Hyyrön ja Porasen saavuttamat tulokset satunnaisotantaan perustuvilla algoritmeilla.

Tuloksista voidaan päätellä, että yksinkertaiset satunnaisotantaan perustuvat algoritmit voivat saavuttaa kohtalaisen hyviä tuloksia, mutta erityisen hyviin tuloksiin niillä ei ylletä. Voidaan myös huomata, että satunnaisten pelien määrän ja näin ollen suoritusajan kasvaessa saavutetaan parempia tuloksia.

4.2 Paikallinen etsintä

Paikalliseen etsintään perustuvissa algoritmeissa aloitetaan useimmiten sattumanvaraisesta ratkaisusta. Tämän jälkeen ratkaisua pyritään parantamaan soveltamalla siihen yksinkertaisia muunnoksia tai valitsemalla yksinkertaisia siirtymiä. Siirtymät ovat luonteeltaan paikallisia, ratkaisua muokataan vähän kerrallaan lähellä olevaksi paremmaksi ratkaisuksi [Mäkinen ja Poranen, 2008].

Yksi tunnetuimmista paikalliseen etsintään perustuvista menetelmistä on *simuloitu jäähdytys* (simulated annealing).

4.2.1 Simuloitu jäähdytys

Simuloitu jäähdytys on menetelmä, jolla voidaan saavuttaa hyviä tuloksia suhteellisen nopeasti. Simuloidun jäähdytyksen idea esiteltiin ensimmäisen kerran jo vuonna 1953 Metropolis-Hastings algoritmina [Metropolis *et al.*, 1953]. Simuloidun jäähdytyksen yleinen muoto, Metropolis-Hastings algoritmin muunnos, esiteltiin kuitenkin vasta vuonna 1983 [Kirkpatrick *et al.*, 1983]. Simuloituun jäähdytykseen perustuvia menetelmiä on hyvällä menestyksellä sovellettu monien NP-vaikeiden ongelmien parhaan ratkaisun arviointiin [Johnson *et al*., 1989; 1991].

Menetelmä on saanut nimensä pyrkimällä jäljittelemään metallien jäähdytykseen käytettyjä menetelmiä hyvän lopputuloksen saavuttamiseksi. Metallien tapauksessa kuumentaminen tiettyyn lämpötilaan ja sen jälkeinen jäähdytys ovat hallittuja tapahtumia, joiden avulla metalli saa halutut ominaisuudet. Kuumentamis- ja jäähdytysnopeudet ja ajat vaikuttavat esimerkiksi atomien siirtymiseen materiaalissa [Kirkpatrick *et al.*, 1983].

Simuloidussa jäähdytyksessä jokaisella algoritmin kierroksella siirrytään sen hetkisestä tilasta tai tuloksesta seuraavaan lähellä sijaitsevaan tilaan yksinkertaisen siirtymän avulla. Simuloidun jäähdytyksen etuna moniin muihin algoritmeihin verrattuna on kyky poistua lokaalista optimista, koska siirtymä myös huonompaan suuntaan voidaan hyväksyä tiettyjen parametrien puitteissa. Lokaalista optimista poistumalla päästään tutkimaan muitakin mielenkiintoisia naapurustoja ja näin voidaan löytää entistä parempia tuloksia.

Simuloitua jäähdytystä ohjataan säätelemällä algoritmin *lämpötilaa* (temperature) [Kirkpatrick *et al.*, 1983]. Lämpötilan ollessa suuri, siirtymät huonompaan suuntaan hyväksytään suuremmalla todennäköisyydellä, jolloin algoritmi ei jää helposti jumiin lokaalisti optimiin tulokseen. Lämpötila asetetaan usein alussa suureksi ja sitä lasketaan hallitusti algoritmin edetessä. Lämpötilaa voidaan laskea esimerkiksi kertomalla nykyinen lämpötila *jäähtymiskertoimella* (cooling ratio), joka on vakio väliltä [0, 1).

Toisin sanoen, koska kerroin on aina ykköstä pienempi, lämpötila laskee jokaisella kierroksella, kunnes ennalta määritelty lopetuslämpötila saavutetaan.

Suuresta alkulämpötilasta seuraa, että algoritmi saa aluksi tutkia ratkaisuavaruutta erittäin vapaasti lämpötilan ollessa suuri. Lämpötilan laskiessa, algoritmi erikoistuu tutkimaan hyväksi havaittua naapurustoa ja toiminta muistuttaa enemmän paikallista etsintää [Mäkinen ja Poranen, 2008].

Huonomman tilan hyväksymisen todennäköisyys $P(\Delta E)$ lasketaan usein kaavalla

$$P(\Delta E) = e^{-\frac{\Delta E}{kT}},$$

missä ΔE on uuden ja vanhan tuloksen välinen erotus, T on algoritmin lämpötila ja k on ennalta määritelty vakio. Toisin sanoen, jos X on satunnaisluku väliltä [0, 1) ja satunnaisluku $X < e^{-\frac{\Delta E}{kT}}$, huonompi tila voidaan hyväksyä [Kirkpatrick *et al.*, 1983]. Lämpötilan laskiessa hyväksymisen todennäköisyys lähestyy nollaa. Hyväksymisen todennäköisyys voidaan luonnollisesti laskea monella muullakin kuin tässä esitetyllä tavalla.

4.2.2 Simuloitu jäähdytys ja Morpion Solitaire

Hyyrö ja Poranen [2007] esittelevät tutkimuksessaan satunnaisotantaan perustuvien algoritmiensa lisäksi myös simuloituun jäähdytykseen perustuvan algoritmin. Suurimpana erona satunnaisotantaan perustuviin algoritmeihin on simuloituun jäähdytykseen perustuvan algoritmin kyky suorittaa siirtoja taaksepäin. Hyyrön ja Porasen esittelemässä simuloituun jäähdytykseen perustuvassa algoritmissa määritellään jokaisessa pelitilanteessa mahdollisten laillisten siirtojen joukko *F* sekä joukko *B,* joka sisältää ne siirrot, jotka ovat sääntöjen puitteissa peruutettavissa.

Siirto on peruutettavissa, jos sen poistaminen ei tee mistään muusta jo pelissä olevasta siirrosta laitonta. Siirron voi siis peruuttaa, jos kaikki aikaisemmat siirrot voidaan jossain järjestyksessä suorittaa ilman poistettavaa siirtoa. Hyyrön ja Porasen algoritmissa siirron peruuttamisen laillisuus tarkistetaan yksinkertaisesti tarkistamalla siirron lisäämän pisteen läpi kulkevien siirtojen lukumäärä. Jos siirron lisäämän pisteen kautta kulkee vain kyseinen siirto itse, siirto voidaan poistaa laillisesti. Siirtojen peruminen eli taaksepäin liikkuminen mahdollistaa sen, ettei algoritmi jää helposti jumiin paikallisesti hyviin tuloksiin, vaan kokeilee muitakin mahdollisuuksia lähinaapuruston ulkopuolelta.

Jokaisessa pelitilanteessa algoritmi valitsee ensin sattumanvaraisesti siirron kaikkien siirtojen joukosta $F \cup B$, johon kuuluvat siis sekä tavalliset että peruuttavat siirrot. Jos valittu siirto m kuuluu tavallisten siirtojen joukkoon *F*, valittu siirto suoritetaan heti. Jos valittu siirto kuuluu peruuttavien siirtojen joukkoon *B,* testataan siirron m

hyväksymisen todennäköisyyttä lämpötilaan T perustuvalla kaavalla $P(m) = e^{-\frac{1}{T}}$. Jos siirron peruuttaminen hyväksytään testin perusteella, suoritetaan valittu peruuttava siirto. Muussa tapauksessa valitaan ja suoritetaan uusi siirto $m \in F$ ja jatketaan algoritmin suorittamista uudesta tilasta [Hyyrö and Poranen, 2007]. Lämpötilaa T lasketaan ennalta määritellyn siirtojen määrän r välein kertomalla se jäähtymiskertoimella α, jolloin todennäköisyys $P(m)$ laskee. Toisin sanoen, peruuttavien siirtojen hyväksymisen todennäköisyys laskee ajan myötä algoritmin lämpötilan laskiessa. Algoritmin pseudokoodi on esitetty algoritmissa 3 [Hyyrö and Poranen, 2007].

```
1    siirtojen lukumäärä := 0
2    T := aloituslämpötila
3    α := jäähtymiskerroin
4    r := suoritettavien siirtojen määrä ennen lämpötilan laskemista

5    while lämpötila T > lopetuslämpötila T_end do
6      for i := 1 to r do
7        valitse sattumanvarainen siirto m joukosta F ∪ B
8        if m kuuluu joukkoon F then
9          suorita siirto m
10         siirtojen lukumäärä := siirtojen lukumäärä + 1
11       else
12         suorita lämpötilaan perustuva todennäköisyystesti
13         if testi onnistui then
14           peruuta siirto m
15           siirtojen lukumäärä := siirtojen lukumäärä - 1
16         else
17           valitse sattumanvarainen siirto m joukosta F
18           suorita siirto m
19           siirtojen lukumäärä := siirtojen lukumäärä + 1
20         endif
21       endif
22       päivitä mahdollisten siirtojen ja
         peruutussiirtojen joukot F ja B
23     endfor
24     T := αT
25   endwhile
```

Algoritmi 3. Hyyrön ja Porasen simuloituun jäähdytykseen perustuva algoritmi.

Hyyrö ja Poranen [2007] toteavat testiensä tuloksissa, että algoritmi saavutti parhaat tuloksensa, kun siirtojen perumisen todennäköisyys oli alussa suhteellisen suuri. Toisin sanoen, parhaat tulokset saavutettiin, kun algoritmin annettiin vapaasti tutkia useita erilaisia vaihtoehtoja ja jäähtyä erittäin hitaasti kohti yksittäistä ratkaisujen naapurustoa.

Hyyrö ja Poranen saavuttivat simuloituun jäähdytykseen perustuvalla algoritmillaan erittäin hyviä tuloksia. Irrallisessa 5D mallissa algoritmilla saavutettiin 79 siirron tulos, joka oli silloinen uusi ennätys. Koskevassa 5T mallissa saavutettiin 143 siirron tulos, joka vastasi koskevan mallin parasta, toisella algoritmilla* löydettyä tulosta.

Hyyrö ja Poranen saavuttivat simuloidulla jäähdytyksellä myös parhaat mahdolliset tulokset 4D ja 4T malleissa. Tulokset 35 siirtoa mallissa 4D ja 62 siirtoa mallissa 4T ovat todistetusti parhaat mahdolliset, koska kaikki mahdolliset siirtojen yhdistelmät kyseisissä malleissa on käyty läpi [Boyer].

4.3 Evoluutioon perustuvat algoritmit

Evoluutioon perustuvat algoritmit (evolutionary algorithms) [Bäck, 1996] ovat eräs vaikeiden optimointiongelmien ratkaisuun soveltuvien algoritmien luokka. Evoluutioon perustuvat algoritmit pyrkivät nimensä mukaisesti jäljittelemään luonnossa tapahtuvaa evoluutiota. Käytännössä tämä tapahtuu usein jäljittelemään evoluutioon kuuluvia käsitteitä: lisääntymistä, mutaatiota, risteytystä ja luonnonvalintaa. Tunnetuin evoluutioon perustuvien algoritmien luokka lienee geneettiset algoritmit [Mitchell, 1998].

Evoluutioon perustuville algoritmeille on ominaista rakentaa hyvä ratkaisuehdotus pala palalta evoluution tai ratkaisuehdotuksien kehittyessä. Paremman ratkaisun etsiminen hyvien ratkaisujen osista tunnetaan myös nimellä *building block hypothesis*. Menetelmien toimintaa on kritisoitu, koska niiden toiminta ei oikeastaan perustu mihinkään todistettavaan ominaisuuteen [Wright *et al*. 2003]. Voidaan oikeastaan vain sanoa, että hyvien ratkaisujen ominaisuuksia yhdistelemällä voidaan joissakin tapauksissa päätyä vieläkin parempaan ratkaisuun.

Monissa muissa menetelmissä keskitytään vain yhteen ratkaisuehdotukseen ja pyritään parantamaan tätä erilaisia siirtymiä tai valintoja tehden. Evoluutioon perustuvissa algoritmeissa tarkastellaan usein suurta joukkoa ratkaisuehdotuksia, joka tunnetaan myös *populaationa* (population). Populaatio on siis määritellyn kokoinen joukko yksittäisiä ratkaisuehdotuksia, jotka muodostetaan usein jollain yksinkertaisella menetelmällä, esimerkiksi satunnaisotannalla tai jollakin yksinkertaisella heuristiikalla [Mäkinen ja Poranen, 2008].

Populaation muodostamisen jälkeen populaatioon sovelletaan algoritmikohtaisia menetelmiä parempien tulosten saavuttamiseksi. Käytännössä tämä tarkoittaa usein hyvien ratkaisuehdotuksien yhdistämistä tai risteyttämistä, huonompien

* Tuloksen saavuttamisen ajanhetkellä paras tietokoneella aikaisemmin saavutettu tulos koskevassa mallissa oli 143 siirtoa, jonka saavutti Pascal Zimmer käyttämällä parantelemaansa toteutusta Hugues Juillén [1995;1999] END-menetelmästä [Boyer].

ratkaisuehdotuksien korvaamista uusilla tai paremmalta näyttävillä ratkaisuehdotuksilla ja ratkaisuehdotuksien mutatoimista eli sattumanvaraisten muutosten tekemistä ratkaisuehdotuksiin. Evoluutiokierroksia voidaan toistaa, kunnes tarpeeksi hyvä tulos on saavutettu tai jokin muu lopetusehto täyttyy [Mäkinen ja Poranen, 2008].

Hugues Juillé esitteli tutkimuksessaan [1995] erään evoluutioon perustuvan algoritmin, *epädeterministisen evoluutiomallin* (Evolving Non-Deterministic, END), jota hän sovelsi myös Morpion Solitairen ongelmaan. Juillén [1995] mukaan END-malli sopii hyvin ongelmiin, joissa ongelmaan liittyvästä ratkaisuavaruudesta ei ole saatavilla paljon tietoa. Kyseinen Juillén mainitsema ominaisuus on itse asiassa yksi evoluutioon perustuvien algoritmien vahvuuksista.

4.3.1 Epädeterministinen evoluutiomalli

Epädeterministinen evoluutiomalli perustuu yritykseen jäljitellä oikeata evoluutiota. Juillén [1995] algoritmi on sarja evoluutiokierroksia, joissa paremmat yksilöt korvaavat huonompia yksilöitä ja pyrkivät samalla kehittymään kohti lopullista ratkaisua.

Algoritmissa ongelman ratkaisuun käytetään populaatiota, joka voidaan kuvata kaksiulotteisena taulukkona. Populaatio koostuu eri *lajeihin* (species) kuuluvista olioista. Olioilla tarkoitetaan tarkasteltavan ongelman yksittäisiä ratkaisuehdotuksia. Olion laji määritellään sen tekemien valintojen mukaan. Jos oliot ovat lajin määrittelevään pisteeseen asti tehneet samat valinnat, ne kuuluvat samaan lajiin.

Juillén [1995] mukaan menetelmän käyttö edellyttää ongelmaa, jonka kaikki ratkaisut voidaan esittää puurakenteena tai suunnallisena syklittömänä verkkona. Ratkaisujen sopivuutta eli paremmuutta ongelman ratkaisuksi tulee myös pystyä arvioimaan jollakin tavalla. Puurakenteessa puun juuri vastaa ongelman alkutilaa. Vastaavasti puun solmut vastaavat yksikäsitteisiä tiloja, joihin on päästy laillisia siirtymiä tehden. Lopulta puun lehtisolmut vastaavat ongelman kaikkia eri ratkaisuja, joihin voidaan päätyä laillisia siirtymiä tehden.

Huomattavaa Juillén evoluutiomallissa on, että geneettisistä algoritmeista poiketen, kaikki menetelmän tuottamat ratkaisut ovat laillisia, koska yksilöt kehittyvät valitsemalla vain laillisia siirtymiä [Juillé, 1995]. Geneettisten algoritmien tapauksessa ratkaisujen mutaatiot ja risteytykset voivat joidenkin ongelmien kohdalla helposti tuottaa ratkaisuja, jotka eivät ole laillisia. Menetelmä eroaa myös paikallisesta etsinnästä, koska se ei varsinaisesti pyri etsimään parempia tuloksia hyvän tuloksen naapurustosta, vaan ratkaisu rakennetaan pala palalta alusta asti kohti parempia tuloksia [Juillé, 1995].

Hyvänä puolena evoluutiomenetelmässä, kuten muissakin populaation omaavissa menetelmissä, on sen hyvä soveltuvuus rinnakkaiseen suorittamiseen. Toisin kuin esimerkiksi simuloidussa jäähdytyksessä, jossa rinnakkaisia toimenpiteitä ei ole juuri

lainkaan, populaation ansiosta menetelmässä on suuri määrä toisistaan riippumattomia tehtäviä, jotka voidaan suorittaa rinnakkain. Rinnakkaisten tehtävien olemassaolo mahdollistaa useamman prosessorin tai tietokoneen käyttämisen ongelman ratkaisemiseen.

Huonona puolena voidaan huomata selvästi korkeammat resurssivaatimukset. Erityisesti muistin käyttö lisääntyy huomattavasti populaation koon kasvaessa, toisin kuin esimerkiksi simuloidussa jäähdytyksessä, jossa muistivaatimukset ovat minimaaliset.

4.3.2 Evoluutiomallin toiminta

Juillé [1995] jakaa algoritminsa toiminnan kahdeksi kokonaisuudeksi: *rakennusvaiheeksi* (construction phase) ja *kilpailuvaiheeksi* (competition phase). Näitä kahta vaihetta toistetaan vuorotellen, kunnes on saavutettu tarpeeksi hyvä tulos tai lopetusehto täyttyy.

Jokaisella evoluutiokierroksella rakennusvaiheessa jokainen populaatiossa oleva olio etenee sen hetkisestä tilanteesta ratkaisuun tehden täysin sattumanvaraisia valintoja. Kun jokainen olio on edennyt ratkaisuun, kunkin olion sopivuutta ongelman ratkaisuksi voidaan arvioida.

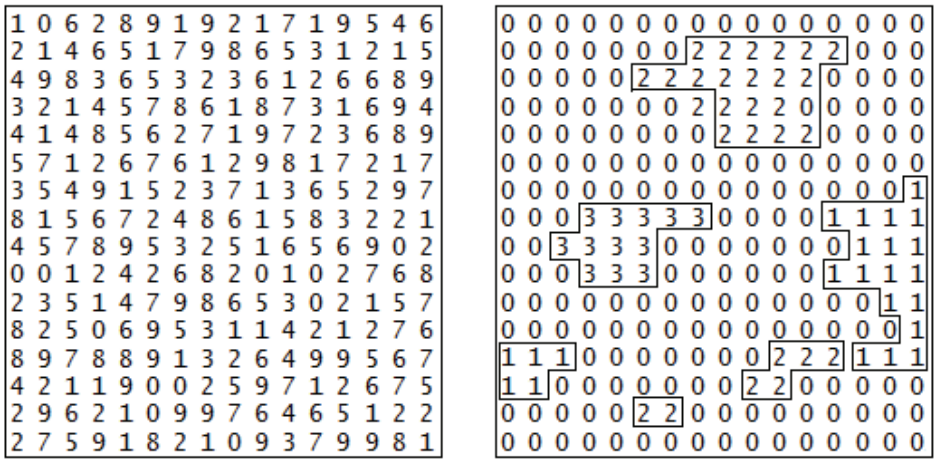
Kilpailuvaiheessa jokaista oliota verrataan lähimpiin naapureihinsa populaatiossa ja sopivin olio naapurustossa korvaa olion, ellei olio itse ole sopivin ratkaisu naapurustossa. Olion naapureihin kuuluvat populaation sisältävässä taulukossa ne oliot, joiden taulukkokoordinaattien *Manhattan-etäisyys* (Manhattan distance) oliosta on alle määritellyn arvon. Tavallisesta etäisyydestä poiketen, Manhattan-etäisyys on koordinaattien erotuksien itseisarvojen summa. Esimerkiksi koordinaatiston pisteiden (0,2) ja (3,1) Manhattan-etäisyys toisistaan on $|3 - 0| + |1 - 2| = 4$.

Lisäksi populaation sisältävän taulukon oletetaan olevan yhdistetty reunoistaan jatkuvuuden säilyttämiseksi. Toisin sanoen, taulukon reunoilla ja kulmissa sijaitsevat oliot voivat laskea naapureikseen taulukon toiselta reunalta löytyviä olioita ja ovat näin vertailussa ja leviämisessään tasaväkisiä muiden olioiden kanssa.

Kilpailuvaiheen ansiosta paremmat oliot korvaavat huonompiin tuloksiin päätyneet oliot. Tästä seuraa, että seuraavalla evoluutiokierroksella paremman lajin edustajia on todennäköisesti enemmän ja evoluutiokierrosten edetessä näistä lajeista muodostuu kilpailevia ala- tai rinnakkaislajeja, joilla saattaa olla pitkästikin saman lajin alkuosa, mutta erilainen loppuosa. Käytännössä huonompien olioiden korvaaminen tarkoittaa käytössä olevien resurssien siirtämistä mielenkiintoiseksi tai paremmiksi todettujen hakupolkujen tarkempaan tutkimiseen.

Tarkastellaan esimerkkiä, jonka avulla myös Juillé [1995] selvensi menetelmänsä toimintaa. Esimerkissä ongelmana on numeroiden 0-9 lajitteleminen nousevaan järjestykseen. Kuvataan yksinkertaisuuden vuoksi olioita ilmoittamalla vain ensimmäinen olion valitsema numero. Kuvassa 3 on vasemmalla esimerkkipopulaatio, joka sisältää sattumanvaraisen alkupopulaation. Esimerkin populaation oliot kuuluvat lajeihin 0-9. Jokainen olio pyrkii järjestämään numerot valitsemalla rakennusvaiheessa sattumanvaraisen numeroiden järjestyksen. Oletetaan nyt, että olion saavuttaman numerojärjestyksen hyvyyttä voidaan jotenkin arvioida. Rakennusvaiheen jälkeisessä kilpailuvaiheessa olioiden saavuttamien numerojärjestyksien hyvyyttä voidaan vertailla ja paremmin kaikkien numeroiden järjestämisessä onnistuneet oliot korvaavat huonommin onnistuneet oliot naapurustoissaan.

Kuvassa 3 on oikealla puolella esitetty yksi mahdollinen tilanne usean evoluutiokierroksen jälkeen. Oikealla puolella olevassa kuvassa osa lajeista on karsiutunut kokonaan pois ja vain parhaiten kaikkien numeroiden lajittelussa onnistuneet lajit ovat säilyneet ja valloittaneet suuren osan koko populaation alasta. Kuvasta voidaan päätellä lajin 0 onnistuneen parhaiten lajittelemaan kaikki numerot nousevaan järjestykseen. Myös numeroista 1-3 aloittaneet oliot ovat onnistuneet järjestämään numerot kohtuullisesti nousevaan järjestykseen ja ovat näin ollen onnistuneet valtaamaan itselleen pieniä osia populaatiosta. Evoluutiokierrosten edetessä lajin 0 voidaan lopulta olettaa valloittavan koko populaation.



Kuva 3. Populaation muuttuminen evoluutiokierrosten edetessä.

Evoluutiokierroksen jälkeen lajien lajitunnuksen pituutta voidaan nostaa, jolloin oliot erikoistuvat lähemmäksi lopullista ratkaisua. Esimerkiksi ensimmäisellä evoluutiotasolla laji voidaan määritellä ensimmäisen valintansa mukaan ja toisella evoluutiotasolla kahden ensimmäisen valinnan mukaan.

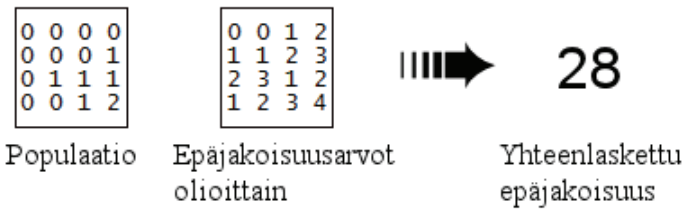
Myös yksittäistä evoluutiotasoa on mahdollista toistaa useamman kerran, jolloin lajin pituus pysyy samana usean evoluutiokierroksen ajan. Sen sijaan, että siirryttäisiin eteenpäin ja muodostettaisiin uusia alalajeja, voidaan nykyisten lajien antaa levitä ja korvata muita lajeja nykyisellä tasolla, kunnes ollaan varmoja jonkin lajin siihen mennessä tehtyjen ratkaisujen sopivuudesta. Toisaalta, jos evoluutiokierroksia jatketaan niin pitkään, että jäljelle jää vain yksi vaihtoehto, saatetaan mielenkiintoisia hakupolkuja menettää kokonaan.

4.3.3 Sitoutumisaste ja populaation epäjakoisuus

Lajien pituutta ohjataan *sitoutumisasteella* (commitment degree), joka määrittelee olioiden lajitunnuksen pituuden. Jos sitoutumisaste on esimerkiksi 3, jokaisen populaatiossa olevan yksilön laji on määritelty kolmen ensimmäisen valinnan perusteella. Toisin sanoen, laji on sitoutunut kolmeen ensimmäiseen valintaansa. Kun sitoutumisastetta nostetaan, jokainen populaation olio valitsee sattumanvaraisen laillisen siirtymän lajitunnuksen jatkokseen, jolloin syntyy taas uusia kilpailevia alalajeja.

Sitoutumisastetta nostetaan algoritmin edetessä, jotta yksilöt ja lajit voivat erikoistua parempiin tuloksiin. Sitoutumisastetta voidaan esimerkiksi yksinkertaisesti nostaa jokaisen evoluutiokierroksen jälkeen. Juillé [1995] määrittelee tutkimuksessaan myös vaihtoehtoisen tavan ohjata sitoutumisasteen kasvua. Juillé ehdottaa, että sitoutumisastetta voidaan nostaa, kun populaation *epäjakoisuus* (disorder measure) on riittävän pieni.

Epäjakoisuuden tarkoituksena on havaita nopeasti, onko jokin laji muita parempi ja tulee lopulta voittamaan muut. Koko populaation epäjakoisuus lasketaan vertaamalla jokaista yksilöä lähimpiin naapureihinsa ja laskemalla yhteen niiden olioiden määrä, jotka ovat erilaisia kuin yksilö itse. Mitä enemmän erilaisia lajeja mallissa on jäljellä, sitä suurempi epäjakoisuuden arvo saadaan. Epäjakoisuus pienenee, kun parempiin tuloksiin päätyneet lajit levittäytyvät ja korvaavat heikompia lajeja. Kuvassa 4 on yksinkertainen esimerkki epäjakoisuuden laskemisesta. Esimerkissä naapuruston koko on määritelty Manhattan-etäisyydellä 1.



Kuva 4. Esimerkki epäjakoisuusarvon laskemisesta populaatiolle, jonka koko on 16.

Evoluutiomallin toimintaa voidaan säädellä muuttamalla populaation kokoa tai naapuruston kokoa sekä muuttamalla sitoutumisasteen nostamiseen käytettävää strategiaa. Suuremmalla populaation koolla varmistetaan, etteivät hyvät tulokset katoa kokonaan muutaman ensimmäisen evoluutiokierroksen aikana sattumanvaraisuuden takia. Naapuruston kokoa säätelemällä voidaan vaikuttaa siihen, miten nopeasti hyvä tulos lisääntyy. Suurempi naapuruston koko nopeuttaa hyvän tuloksen leviämistä, mutta saattaa vaikeuttaa vaihtoehtoisten tulosten löytymistä, koska myös tutkittavat vaihtoehdot vähenevät nopeammin. Pienemmällä naapuruston koolla varmistetaan, että vaihtoehtoiset tulokset eivät karsiudu heti pois, mutta etsimisen voidaan olettaa vievän myös enemmän aikaa, koska hyvät tulokset lisääntyvät hitaammin. Sitoutumisasteen hallintaan käytettävää strategiaa muuttamalla voidaan vaikuttaa kiinnostavien lajien määrään populaatiossa.

4.3.4 Self-Adaptive Greedy Estimate

Myöhemmässä tutkimuksessaan Juillé [1999] esittelee parannellun version epädeterministisestä evoluutiomallistaan. Juillé kutsuu uutta menetelmää nimellä *Self-Adaptive Greedy Estimate* (SAGE). SAGE toimii kuitenkin yhtä lisäystä lukuun ottamatta täysin samoin kuin epädeterministinen evoluutiomalli.

SAGE-mallin ainoa ero END-malliin verrattuna on Juillén ehdotus karsia huonoja olioita jo rakennusvaiheessa ennen kilpailuvaihetta. Ehdotus perustuu käytössä olevien resurssien rajallisuuteen. Esimerkiksi populaation koko on tietokoneen resurssien kannalta ongelmallinen parametri, koska tietokoneen käytössä olevan muistin määrä estää hyvin nopeasti populaation koon kasvattamisen. Juillé [1999] ehdottaa tutkimuksessaan, että yksinkertaisen satunnaisotannan sijaan kukin olio voisi toistaa satunnaisotannan useamman kerran ja vain parhaan tuloksen saavuttanut otanta selviää jatkoon. Tällä menetelmällä suurin osa huonoihin tuloksiin päätyneistä satunnaisista peleistä karsitaan jo ennen kilpailuvaihetta. Toisaalta, toistetun satunnaisotannan ansiosta koko menetelmän toiminta todennäköisesti tehostuu, koska hyviä siirtoja tehnyt olio saavuttaa todennäköisemmin paremman tuloksen usealla satunnaisotannalla kuin huonompia siirtoja tehnyt olio.

4.3.5 Evoluutiomalli ja Morpion Solitaire

Juillén [1995] END-malli on itse asiassa yksi ensimmäisistä Morpion Solitairen ongelmaan sovelletuista ohjelmoiduista ratkaisuista. Morpion Solitairen tapauksessa oliot ja ratkaisut esitetään tiettynä järjestyksenä laillisia siirtoja ja eri lajit määräytyvät sitoutumisasteen mittaisina osaratkaisuina. Jokaisella evoluutiotasolla oliot pelaavat rakennusvaiheessa pelin loppuun sattumanvaraisia siirtoja valiten ja kilpailuvaiheessa parempiin tuloksiin päätyneet pelit korvaavat naapurinsa. Vaikka kyse onkin periaatteessa satunnaisotannasta, suuremmalla todennäköisyydellä parempiin tuloksiin

päätyvät pelit korvaavat lopulta heikompiin tuloksiin päätyvät pelit. Populaation koko ja hyvä sitoutumisasteen hallinta ovat tässä todennäköisesti ratkaisevia, jotta mahdolliset hyvät ratkaisut eivät karsiudu heti pois muutaman niissä sattuneen huonon satunnaisotannan takia.

Algoritmissa 4 on esitetty evoluutiomallin yksinkertainen pseudokoodiratkaisu [Juillé, 1995]. Pseudokoodi on pyritty pitämään tässä tapauksessa selkeänä tehokkaan toteutuksen kustannuksella. Käytännössä esimerkiksi kilpailuvaiheessa ei jokaisen olion kohdalla ole tarpeellista verrata oliota jokaiseen naapuriinsa tarvittavien korvauksien suorittamiseksi.

```
1    alusta populaatio
2    sitoutumisaste := 0
3    paras peli := null

4    while joissakin peleissä on vielä mahdollisia siirtoja do

       -- Rakennusvaihe --
5      for each peli in populaatio do
6        pelaa peli loppuun sattumanvaraisia siirtoja tai jotain
         heuristiikkaa käyttäen
7      endfor

       -- Kilpailuvaihe --
8      for each peli in populaatio do
9        if pelin tulos > paras peli then
10         paras peli := peli
11       endif
12       for each naapuri in pelin naapurit do
13         if pelin tulos < naapurin tulos then
14           peli := naapuri
15         endif
16       endfor
17     endfor

18     tarkista populaation epäjakoisuus tai muu sitoutumisasteen
       hallintaan käytetty strategia
19     if tarkistuksen mukaan sitoutumisastetta tulee nostaa then
20       sitoutumisaste := sitoutumisaste + 1
21     endif
22     for each olio in populaatio do
23       palauta olio takaisin sitoutumisasteen mittaiseksi
         osaratkaisuksi
24     endfor
25   endwhile
```

Algoritmi 4. Juillén epädeterministinen evoluutiomalli Morpion Solitairelle.

Omissa testeissään Juillé [1995] saavutti Morpion Solitairen 5T mallissa 117 siirron tuloksen ja paransi myöhemmin tulostaan 122 siirtoon SAGE-menetelmällä [Juillé,

1999]. Juillé saavutti nämä tulokset käyttämällä 4096 olion populaatiota. Myöhemmin on raportoitu, että Pascal Zimmerin* toteutus evoluutiomallista saavutti tuloksen 143, joka oli pitkään paras tietokoneella saavutettu tulos 5T mallissa [Boyer].

Juillé [1995] toteaa myös tuloksissaan huomanneensa, että oliot, jotka pyrkivät sijoittamaan ensimmäiset siirtonsa samalle alueelle saavuttivat evoluutiokierroksilla sattumanvaraisesti levittäytyviä olioita parempia tuloksia. Juillé ehdottaa myös, että menetelmällä saatettaisiin päästä parempiin tuloksiin käyttämällä esimerkiksi jotain heuristiikkaa karsimaan selvästi huonot valinnat pois hakupuusta ja näin ollen vähentäen sattumanvaraisuuden osuutta olioiden paremmuudessa. Vaihtoehtoisesti jokainen olio voisi pyrkiä nopeasti lähimpään löytämäänsä lokaaliin optimiin esimerkiksi simuloitua jäähdytystä käyttäen, jolloin sattumanvaraisuuden osuus laskisi jälleen, eikä populaation rajallista tilaa tuhlattaisi selvästi huonojen ratkaisujen säilyttämiseen [Juillé, 1995].

4.4 Monte Carlo -menetelmät

Monte Carlo -menetelmät ovat yksi laajasti käytetty vaikeiden ongelmien ratkaisemiseen soveltuva lähestymistapa. Monte Carlo -menetelmien toiminta perustuu satunnaistettuun toimintaan, jonka avulla pyritään saavuttamaan hyviä tuloksia nopeammin kuin deterministisillä algoritmeilla. Käytännössä Monte Carlo -menetelmien satunnaistettu toiminta ilmenee usein toistettuna satunnaisotantana.

Monte Carlo -menetelmät toimivat usein huomattavasti deterministisiä algoritmeja nopeammin, koska tarkoituksena on löytää ratkaisu päätymällä siihen satunnaisotantaa ja mahdollisesti joitakin ongelmakohtaisia oletuksia soveltaen. Toisaalta, Monte Carlo -menetelmät voivat kuitenkin juuri tästä syystä antaa väärän vastauksen pienellä todennäköisyydellä. Väärän vastauksen todennäköisyys päätösongelman tapauksessa riippuu luonnollisesti ongelmasta ja käytetystä menetelmästä. Optimointiongelmien kohdalla ei sinänsä ole vääriä vastauksia, mutta toisaalta, minkä tahansa muun vastauksen kuin globaalin optimin voidaan tietysti katsoa olevan liian huono, väärä vastaus.

4.4.1 Refleksiivinen Monte Carlo -haku

Refleksiivinen Monte Carlo -haku (Reflexive Monte Carlo search) on Tristan Cazenaven tutkimuksessaan [2007] esittelemä Monte Carlo -menetelmä Morpion Solitairen ongelman ratkaisemiseksi. Cazenaven [2007] refleksiivinen Monte Carlo -haku koostuu useasta erillisestä tasosta, joilla pelataan Morpion Solitaire -pelejä. Alimmalla tasolla

* Pascal Zimmerin parantelemaan evoluutiomalliin ei tiedettävästi liity mitään tieteellistä julkaisua. Zimmerin saavuttamat tulokset on julkaistu Jean-Charles Meyrignacin ylläpitämällä Morpion Solitairen tutkimiseen perehtyvällä sivustolla. [Meyrignac].

pelit pelataan sattumanvaraisia siirtoja valiten ja ylemmillä tasoilla olevat pelit muodostetaan alemmalla tasolla sijaitsevien pelien avulla.

Haku alkaa menetelmän ylimmältä tasolta, jossa pyritään selvittämään paras mahdollinen seuraava siirto. Ylemmän tason seuraava siirto selvitetään suorittamalla algoritmi rekursiivisesti yhtä tasoa alemmalla tasolla ja valitsemalla näistä parhaimpaan ratkaisuun yltäneen pelin seuraava siirto. Alemmat tasot arvioivat siirtoja täsmälleen samalla tavalla, kunnes alin taso saavutetaan. Alimmalla tasolla parasta siirtoa arvioidaan pelaamalla määritelty määrä pelejä sattumanvaraisia siirtoja valiten ja valitsemalla ylemmän tason siirroksi sattumanvaraisista peleistä parhaimman tuloksen saavuttaneen pelin seuraavan siirron.

Cazenave [2007] kutsuu ylempiä tasoja *meta-tasoiksi* (meta level). Jos haussa käytetään esimerkiksi kolmea tasoa, ylimmän tason peli on toisen meta-tason peli, tätä alemmalla tasolla olevat pelit ovat ensimmäisen meta-tason pelejä. Viimeinen, alin taso sisältää sattumanvaraisia pelejä.

Käytännössä menetelmässä on yksinkertaisesti kyse toistetusta satunnaisotannasta. Ensimmäisen meta-tason peli muodostetaan pelaamalla jokaista siirtoa varten määritelty määrä sattumanvaraisia pelejä. Parhaan tuloksen saavuttaneen sattumanvaraisen pelin seuraava siirto valitaan ensimmäisen meta-tason pelin seuraavaksi siirroksi. Tämän jälkeen haku toistetaan uudesta ensimmäisen meta-tason pelin tilanteesta. Parhaimman siirron hakuja toistetaan, kunnes ensimmäisen meta-tason peli on saavuttanut jonkin ratkaisun. Toisen meta-tason pelissä pelataan vastaavasti määritelty määrä ensimmäisen meta-tason pelejä ja valitaan vastaavalla tavalla parhaimmalta näyttävä siirto. Myös toisen meta-tason pelissä hakua toistetaan kunnes peli on pelattu loppuun.

Jokaista ensimmäisen meta-tason pelin siirtoa varten pelataan siis k sattumanvaraista peliä. Olkoon m nyt keskimääräinen ensimmäisen meta-tason pelin pituus. Nyt voidaan huomata, että jokaista ensimmäisen meta-tason peliä varten pelataan keskimäärin $m * k$ sattumanvaraista peliä. Määritellään n nyt samalla tavoin toisen meta-tason siirron selvittämiseksi pelattavien ensimmäisen meta-tason pelien lukumääräksi. Lisäksi voidaan huomata, että toisen meta-tason haun syvyys vaikuttaa alimmalla tasolla pelattavien sattumanvaraisten pelien määrään, koska laskettavien kierrosten määrä vähenee toisen meta-tason haun edetessä. Nyt voidaan huomata, että jokaista toisen meta-tason pelin siirtoa varten pelataan keskimäärin noin $\frac{n*m*k}{2}$ sattumanvaraista peliä. Tästä voidaan päätellä, että meta-tasojen määrän kasvaessa odotettu suoritusaika kasvaa huomattavasti. Tästä johtuen Cazenave rajoitti menetelmänsä kolmeen tasoon, joissa ylimmällä tasolla pelattiin vain yksi toisen meta-tason peli.

Algoritmissa 5 on esitetty refleksiivisen Monte Carlo -haun pseudokoodi [Cazenave, 2007]. Pseudokoodi on esitetty kolme tasoa käsittävälle refleksiiviselle Monte Carlo -

haulle, mutta se on helposti laajennettavissa useammalle tasolle. Haku alkaa kutsumalla funktiota, jossa pelataan yksi toisen meta-tason peli

```
1    k := alimman tason pelien määrä
2    n := ensimmäisen meta tason pelien määrä

3    function pelaa_toisen_tason_meta_peli ()
4      peli := uusi peli
5      while pelissä on vielä mahdollisia siirtoja do
6        siirto := etsi_paras_meta_siirto ( peli )
7        peli := peli + siirto
8      endwhile
9    endfunction

10   function etsi_paras_meta_siirto ( pelitilanne P )
11     paras peli := null
12     for i := 1 to n do   // n ensimmäisen meta-tason peliä
13       peli := P
14       while pelissä on vielä mahdollisia siirtoja do
15         siirto := etsi_paras_siirto ( peli )
16         peli := peli + siirto
17       endwhile
18       if pelin tulos > paras peli then
19         paras peli := peli
20       endif
21     endfor
22     return parhaan tuloksen saavuttaneen pelin ensimmäinen siirto
           tilanteesta P
23   endfunction

24   function etsi_paras_siirto ( pelitilanne P )
25     paras peli := null
26     for i := 1 to k do   // k alimman tason peliä
27       peli := pelaa sattumanvarainen peli pelitilanteesta P
28       if pelin tulos > paras peli then
29         paras peli := peli
30       endif
31     endfor

32     return parhaan tuloksen saavuttaneen pelin ensimmäinen siirto
           tilanteesta P
33   endfunction
```

Algoritmi 5. Cazenaven refleksiivinen Monte Carlo -haku.

Cazenaven menetelmä muistuttaa huomattavasti luvussa 4.1 esiteltyjä satunnaisotantaan perustuvia algoritmeja, koska alimmalla tasolla pelataan määritelty määrä täysin sattumanvaraisia pelejä annetusta tilanteesta ja ylemmän tason siirrot valitaan näihin perustuen. Menetelmä muistuttaa myös joiltain osin Juillén [1995] evoluutiomallia. Voidaan huomata, että alemman tason pelit muistuttavat evoluutiomallin kilpailevia olioita, joista vain paras selviää ylemmän tason ratkaisun osaksi. Evoluutiomallin

tapaan lopullinen ratkaisu rakennetaan iteratiivisesti hyväksi todettujen ratkaisujen pohjalta.

Cazenave [2007] saavutti refleksiivisellä Monte Carlo -haulla tuloksen 78 siirtoa irrallisessa 5D mallissa. Tulos saavutettiin pelaamalla yksi toisen meta-tason peli. Haussa pelattiin 100 sattumanvaraista peliä alimmalla tasolla jokaista ensimmäisen meta-tason siirtoa varten ja 1000 ensimmäisen meta-tason peliä jokaista toisen meta-tason siirtoa varten. Cazenaven mukaan koko haku vei lähes 400 000 sekuntia eli yli neljä vuorokautta.

4.4.2 Sisäkkäinen Monte Carlo -haku

Myöhemmin Cazenave [2009] kehitteli refleksiivisen Monte Carlo -hakunsa pohjalta uuden menetelmän, *sisäkkäisen Monte Carlo -haun* (Nested Monte Carlo search). Uudella menetelmällään Cazenave saavutti 80 siirron tuloksen 5D mallissa.

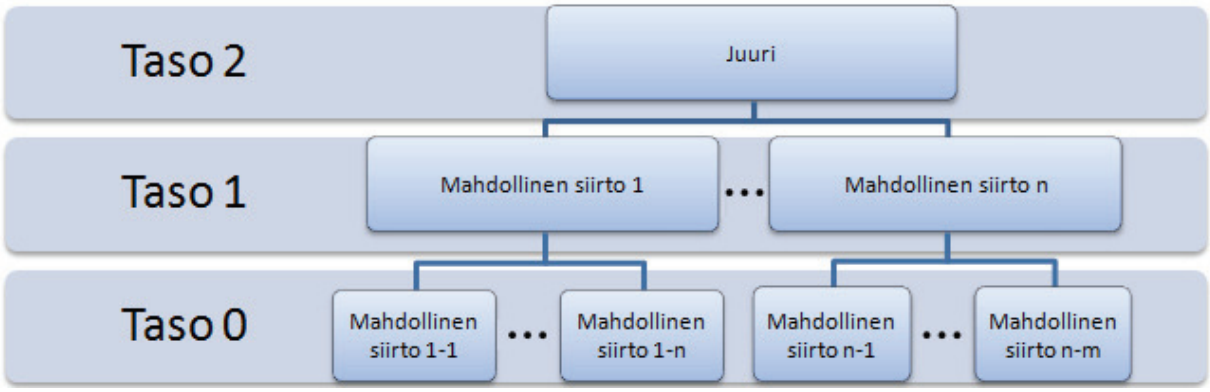
Sisäkkäinen Monte Carlo -haku muistuttaa hieman refleksiivistä Monte Carlo -hakua, koska myös sisäkkäisessä Monte Carlo -haussa hakua suoritetaan usealla tasolla. Refleksiivisessä Monte Carlo -haussa seuraava siirto valittiin pelaamalla annetusta tilanteesta määritelty määrä pelejä ja valitsemalla siirto parhaimman tuloksen saavuttaneesta pelistä. Sisäkkäisessä Monte Carlo -haussa jokaisella tasolla pyritään tarkemmin tutkimaan siirtojen vaikutusta myöhempiin pelitilanteisiin. Alimmalla tasolla paras siirto valitaan yhä samoin kuin refleksiivisessä Monte Carlo -haussa, pelaamalla määritelty määrä sattumanvaraisia pelejä ja valitsemalla näistä parhaan tuloksen saavuttanut. Cazenave [2009] totesi omissa testeissään jokaisen uuden tason vievän noin 200 kertaa alempaa tasoa kauemmin. Tasojen määrän nostaminen vaikuttaa siis suoritusaikaan huomattavasti, kuten myös edellä huomattiin refleksiivisen Monte Carlo -haun tapauksessa. Cazenave saavutti parhaan tuloksensa käyttämällä 4-tasoista hakua, jonka hän arvioi vievän noin 10 päivää yhdellä tietokoneella suoritettuna.

Sisäkkäinen Monte Carlo -haku alkaa ylimmältä tasolta. Sisäkkäisessä Monte Carlo -haussa jokaisella menetelmän tasolla pelataan vuorollaan jokainen mahdollisista siirroista ja kutsutaan sitten alemman tason hakua siirron jälkeisestä tilanteesta. Ylimmällä tasolla pelataan siis jokainen mahdollisista siirroista vuorollaan ja kutsutaan siirron jälkeisestä tilanteesta rekursiivisesti samaa funktiota mutta yhtä tasoa alempana. Näin siirrytään yhdellä siirrolla eteenpäin ja yhdellä tasolla alaspäin mallissa. Alemmalla tasolla kaikkien mahdollisten siirtojen pelaaminen ja rekursiivinen kutsu toistetaan. Näin jatketaan, kunnes alin taso on saavutettu. Alimmalla tasolla paras siirto löydetään pelaamalla jokaista mahdollista siirtoa kohti yksi sattumanvarainen peli ja valitsemalla näistä parhaan tuloksen saavuttanut siirto seuraavaksi siirroksi.

Menetelmä toimii siis lähes samoin kuin refleksiivinen Monte Carlo -haku. Jokaisella tasolla seuraavaksi siirroksi valitaan mahdollisista siirroista alemmalla tasolla parhaimman tuloksen saavuttanut siirto.

Menetelmä voidaan ehkä helpommin ymmärtää puurakenteena, jossa juurena on menetelmän ylin taso. Juuren lapsina ovat kaikki mahdolliset siirrot annetusta pelin tilasta ja näiden lapsina ovat mahdolliset siirrot kunkin solmun siirron jälkeisestä tilasta. Lehtisolmut vastaavat alimman tason sattumanvaraisesti pelattuja pelejä. Juurta lukuun ottamatta jokainen puun solmu on siis yksittäinen suoritettu siirto. Toisin sanoen, siirryttäessä esimerkiksi juuresta alemman tason solmuun, suoritetaan solmun sisältämä siirto.

Rekursiivinen haku palaa lopuksi takaisin juureen. Kun kaikki solmun lasten pelit on pelattu loppuun, tuloksia voidaan verrata keskenään. Esimerkiksi lehtisolmujen vanhemmat valitsevat seuraavaksi siirrokseen parhaimman tuloksen saavuttaneen lehden. Samoin näiden solmujen vanhemmat vertailevat lapsisolmujensa tuloksia keskenään ja vertailua jatketaan. Jokaisen valitun siirron jälkeen haku toistetaan uudesta tilasta, kunnes kyseisen tason peli on saatettu loppuun. Näin menetelmä palaa lopulta takaisin puun juureen ja juuren tasolle selviää vain parhaaksi usealla tasolla todettu siirto. Kyseisen siirto valitaan juuritason pelin seuraavaksi siirroksi. Kuvassa 5 on pyritty hahmottamaan sisäkkäistä Monte Carlo -hakua yksinkertaisen puurakenteen avulla.



Kuva 5. Yksinkertainen esimerkki kaksitasoisesta sisäkkäisestä Monte Carlo -hausta.

Tämän lisäksi Cazenave [2009] ehdottaa, että paras mahdollinen löydetty tulos säilytetään muistissa. Siinä tapauksessa, että alemman tason haut eivät löydä yhtä hyvää tulosta seuraavalla ylemmän tason kutsulla, otetaan seuraava siirto aikaisemmin löydetystä parhaasta tuloksesta sen sijaan, että käytettäisiin nykyisen kierroksen parasta tulosta, joka on kuitenkin aikaisemmin löydettyä huonompi.

Algoritmissa 6 on esitetty sisäkkäisen Monte Carlo -haun pseudokoodi [Cazenave, 2009]. Tasojen määrää on helppo muuttaa määrittelemällä menetelmää kutsuttaessa uusi tasojen määrä. Algoritmin 6 pseudokoodissa kolmannen tason haku käynnistyisi siis kutsulla esimerkiksi nested(alkutilanne, 3).

```
1    function nested ( pelitilanne p, int taso )
2      paras peli := null
3      while  pelissä on vielä mahdollisia siirtoja do
4        foreach siirto in mahdolliset siirrot do
5          pelitilanne := p + siirto
6          if level = 0 then
7            peli := pelaa sattumanvarainen peli pelitilanteesta
8          else
9            peli := nested ( pelitilanne, taso - 1 )
10         endif

11         if peli > paras peli then
12           paras peli := peli
13         endif
14       endfor
15       p := p + parhaan pelin seuraava siirto
16     endwhile

17     return paras peli
18   endfunction
```

Algoritmi 6. Cazenaven sisäkkäinen Monte Carlo -haku.

Kuten pseudokoodin sisäkkäisistä while- ja for-rakenteista, kuten myös menetelmän rekursiivisista kutsuista voidaan päätellä, menetelmässä pelataan erittäin suuria määriä sattumanvaraisia pelejä ylimmän tason siirron selvittämiseksi. Pelien määrää voidaan arvioida kaavalla $\frac{1}{2^t}\prod_{i=0}^{t} X\,Y_i$, jossa t on menetelmän tasojen määrä, X on keskimääräinen mahdollisten siirtojen lukumäärä Morpion Solitairessa ja Y_i on keskimääräinen pelien pituus menetelmän tasolla i, kun $i \in 0 \ldots t$. Näin arvioituna neljännen tason haussa pelataan tekijän arvion mukaan ainakin 10^{12} sattumanvaraista peliä.

5 Morpion Solitaire ongelmana

Tässä luvussa Morpion Solitairea pyritään tarkastelemaan pelin sijaan tutkimusongelmana. Luvussa esitellään ensin aikaisemmissa tutkimuksissa käytetty merkintätapa ja aikaisempien tutkimuksien tuloksia. Tämän jälkeen tarkastellaan ongelman ratkaisuavaruuden muotoa sekä aikaisemmista ratkaisuehdotuksista johdettuja päätelmiä. Lopuksi esitellään kaksi Morpion Solitairen ongelmaan aikaisemmin sovellettua siirtojen valintaheuristiikkaa sekä tarkastellaan kolmatta mahdollista uutta valintaheuristiikkaa.

5.1 Morpion Solitairen tutkimisessa käytetyt merkinnät

Demaine *et al.* [2006] määrittelivät tutkimuksessaan merkintätavan, jota myös Hyyrö ja Poranen [2007] käyttivät tutkimuksessaan. Merkintätavan avulla voidaan helposti ilmaista pelin monet erilaiset muunnelmat. Tässä merkintätavassa arvo *k* määritellään jokaisen siirron olemassa olevien pisteiden lukumääränä. Toisin sanoen, jokaisella siirrolla piirretään viiva, joka yhdistää $k + 1$ pistettä samalle suoralle ja jossa lisätty viiva kulkee siis *k* olemassa olevan pisteen sekä kyseisellä siirrolla lisätyn pisteen kautta. Hyyrö ja Poranen [2007] kutsuivat arvoa *k* tutkimuksessaan kyseisen pelimuunnelman *paksuudeksi* (thickness).

Olkoon $S \subseteq \mathbb{Z}^2$ nyt mikä tahansa aloituspisteiden joukko. Tällöin merkintä $G_k(S)$ tarkoittaa suurinta mahdollista määrää siirtoja, joka voidaan saavuttaa irrallisessa mallissa annetusta aloituspisteiden joukosta *S* pelaamalla siirtoja joiden pituus on $k + 1$. Määritellään vastaavasti merkintä $G'_k(S)$ tarkoittamaan suurinta mahdollista määrää siirtoja, joka voidaan saavuttaa samalla tavoin koskevassa mallissa [Demaine *et al.*, 2006].

Merkitään tavallista ristin muotoista alkuasetelmaa, jonka paksuus on k, merkinnällä A_k. Tällöin esimerkiksi merkintä A_4 tarkoittaa siis alkuperäisen pelin ristin muotoista aloituskuviota, jossa jokaisen ristin sakaran leveys on 4. Nyt voidaan merkitä siirtojen suurinta mahdollista määrää irrallisessa ja koskevassa mallissa vastaavasti merkinnöillä $G_4(A_4)$ ja $G'_4(A_4)$ [Demaine *et al.*, 2006]. Demaine *et al.* [2006] käyttivät merkintöjä myös ilmaisemaan vastaavia pelin muunnelmia. Tällöin esimerkiksi merkintä $G_4(A_4)$ vastaa 5D mallia.

Demaine *et al.* [2006] käsittelivät tutkimuksessaan kaikki mahdolliset *k*:n arvot. Paksuuden arvoilla $k = 1$ ja $k = 2$ todettiin, että pelejä on mahdollista jatkaa loputtomiin tavallisesta aloituskuviosta sekä irrallisessa että koskevassa mallissa, joten

$$G_1(A_1) = G'_1(A_1) = G_2(A_2) = G'_2(A_2) = \infty.$$

Paksuuden arvolla $k = 3$ huomattiin, että tietystä 7 pisteen aloituskuviosta peliä voitaisiin pelata loputtomiin. Toisaalta, Demaine *et al.* [2006] todistivat, että kyseinen kuvio ei voi esiintyä missään pelissä, joka alkaa kuviosta A_3, joten $G_3(A_3)$ ja $G´_3(A_3)$ eivät ole äärettömiä. Mallit $G_3(A_3)$ ja $G´_3(A_3)$ on myös täysin ratkaistu. Hyyrö ja Poranen [2007] saavuttivat simuloituun jäähdytykseen perustuvalla menetelmällään parhaimmat mahdolliset tulokset, joten todistetusti

$G_3(A_3) = 35$ ja $G´_3(A_3) = 62$.

Arvolle $k = 4$ Demaine *et al.* [2006] todistivat Daniel Goffinetin [Science & Vie, 1976] keksimään potentiaalifunktioon perustuen ylärajan siirtojen määrälle. Potentiaalifunktiossa jokaisella pisteellä on tietty potentiaaliarvo ja jokainen lisätty siirto vähentää mallin potentiaalia. Demaine *et al.* [2006] osoittivat, että aloituskuvion A_4 potentiaali riittää parhaimmassa tapauksessa 141 siirtoon irrallisessa mallissa ja 704 siirtoon koskevassa mallissa. Potentiaalifunktiota tullaan tarkastelemaan tarkemmin luvussa 5.4. Arvolla $k = 4$ alarajoiksi voidaan määritellä parhaat löydetyt ratkaisut. Toisin sanoen,

$80 \leq G_4(A_4) \leq 141$ ja $170 \leq G´_4(A_4) \leq 704$.

Achim Flammenkamp [2003] ilmoittaa todistaneensa $G´_4(A_4)$ mallin ylärajaksi 324, mutta todistusta ei ole onnistuttu varmistamaan paikkansapitäväksi tai vääräksi [Demaine *et al.*, 2006; Hyyrö and Poranen, 2007; Boyer].

Arvoilla $k \geq 5$ Demaine *et al.* [2006] huomasivat, että siirron pituuden takia uusia mahdollisia siirtoja ei voida muodostaa aloitussiirtojen jälkeen. Demaine *et al.* [2006] toteavat, että

$G_k(A_k) = G´_k(A_k) = 12,\ k \geq 5$.

Koska muut mallit ovat joko loputtomia, triviaalisti rajoitettuja tai ratkaistu täydellisesti, kiinnostaviksi malleiksi jäävät ainoastaan $G_4(A_4)$ ja $G´_4(A_4)$.

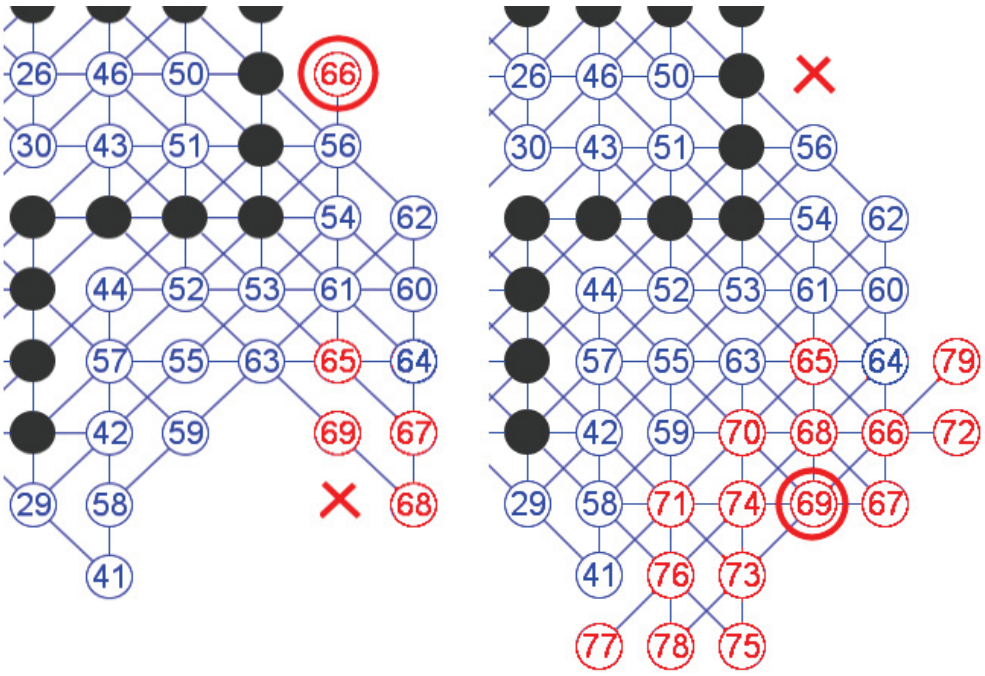
5.2 Morpion Solitairen ratkaisuavaruus

Ongelman *ratkaisuavaruus* (solution space) on sen kaikkien mahdollisten ratkaisujen joukko. Morpion Solitairen ongelman ratkaisuavaruudesta voidaan saada hieman lisää tietoa tarkastelemalla hyvien ratkaisujen naapurustoa. Seuraavassa tarkastellaan Hyyrön ja Porasen [2007] simuloituun jäähdytykseen perustuvalla menetelmällä saavuttamaa 79 siirron ratkaisua. Alla esitetyt tulokset on saatu käyttämällä luvussa 6 tarkemmin esiteltävää Juillén [1995] evoluutiomalliin perustuvaa menetelmää. Testeissä käytettiin

mallille populaation kokoa 128 x 128. Tulosten voidaan olettaa pätevän myös muilla menetelmillä.

Tutkitaan hyvän ratkaisun naapurustoa pelaamalla Hyyrön ja Porasen 79 siirron ratkaisun 64 ensimmäistä siirtoa ja antamalla valitun menetelmän täydentää peli loppuun. Kun 64 ensimmäistä siirtoa on jo lukittu, voidaan löytää ainoastaan kolme arvoltaan toisistaan poikkeavaa lopullista ratkaisua. Kaksi huonompaa ratkaisua päätyy tuloksiin 68 ja 69 ja kolmas saavuttaa parhaan tuloksen 79. Erona kahden huonomman ja kolmannen paremman tuloksen välillä on oikeastaan vain yksi siirto, joka on tarkemmin esitetty kuvassa 6. Kummassakin kuvassa 6 esitetyssä pelissä on pelattu Hyyrön ja Porasen ratkaisun 64 ensimmäistä siirtoa. Vasemmalla puolella kuvassa on peli, joka päättyy 69 siirtoon, oikealla puolella esitetty peli saavuttaa 79 siirtoa. Pelien ratkaisevan eron tekevä siirto on ympyröity ja piste, jota ei siirron takia pystytä lisäämään, on merkitty rastilla.

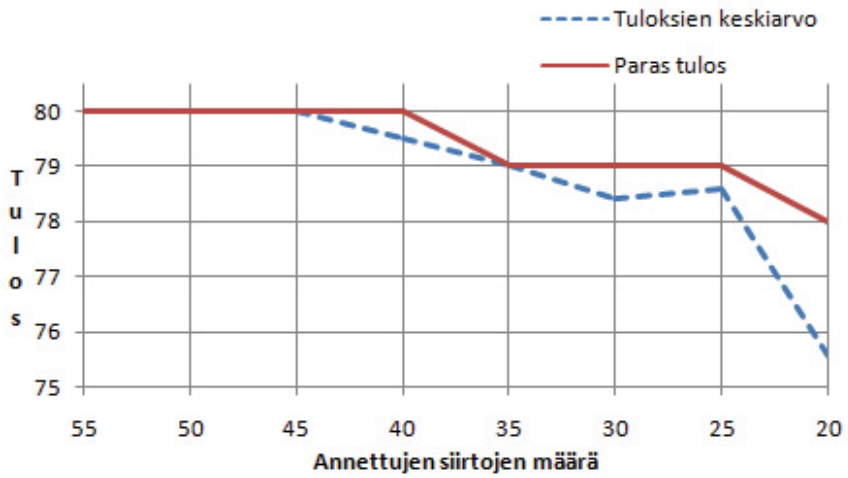
Kyseinen siirto mahdollistaa sarjan uusia siirtoja, joista jokainen avaa uuden siirron, kunnes 79 siirron tulos saavutetaan. Tästä voidaan päätellä, että erittäin hyvät tulokset eivät välttämättä sijaitse muiden lähes yhtä hyvien tulosten naapurustossa. Toisin sanoen, ratkaisuavaruus on muodoltaan piikikäs, koska globaali optimi tai erittäin hyvä lokaali optimi voi sijaita huomattavasti huonompien tuloksien keskellä. Saman ominaisuuden voidaan olettaa pätevän myös koskevassa mallissa.



Kuva 6. Esimerkki yhden huonon siirron vaikutuksesta tulokseen.

Ratkaisuavaruutta voidaan tarkastella myös hieman eri tavalla. Tarkastellaan Cazenaven [2009] 80 siirron tuloksen avulla hyvän tuloksen riippuvuutta ensimmäisistä siirroista. Riippuvuutta voidaan tarkastella lukitsemalla Cazenaven 80 siirron tuloksesta vaihteleva määrä siirtoja. Valmiiksi annettujen siirtojen vaikutusta parhaimman tuloksen löytymiseen testattiin lukitsemalla 80 siirron tuloksesta 20 - 55 ensimmäistä siirtoa viiden siirron askeleen välein.

Kuvassa 7 on esitetty saavutetut tulokset valmiiksi annettujen siirtojen eri määrille. Testi suoritettiin 10 kertaa jokaiselle siirtojen määrälle sattumanvaraisuuden vaikutuksen lieventämiseksi.



Kuva 7. Tuloksien riippuvuus ensimmäisistä siirroista.

Kuvasta voidaan huomata, että menetelmä löysi parhaan mahdollisen tuloksen jokaisella toistolla, kun siirtoja oli annettu valmiiksi 45 – 55 kappaletta. Paras tulos löytyi helposti myös 40 valmiiksi annetulla siirrolla, mutta tulosten keskiarvon käyrästä voidaan nähdä, ettei parasta tulosta saavutettu aivan jokaisella toistolla.

Mielenkiintoinen rajakohta löytyy 35 ja 40 valmiiksi annetun siirron välistä. Testeissä käytetyillä menetelmän parametreilla yksikään 35 valmiiksi annetun siirron testeistä ei saavuttanut 80 siirron tulosta, vaan päätyi hieman poikkeavaan 79 siirron tulokseen. Voidaan todeta, että riippuvuus ei ole enää yhtä vahva 35 annetun siirron kohdalla, mutta paras tulos saatettaisiin yhä löytää suhteellisen helposti käytetyn menetelmän parametreja muuttamalla.

Kuvasta voidaan nähdä selvää riippuvuutta 25 annettuun siirtoon asti, jolloin löydetään yhä erittäin hyvä 79 siirron tulos ja kaikki tulokset päätyvät vähintään 78 siirron tulokseen. Vasta 20 annetun siirron kohdalla voidaan huomata suuri poikkeama tuloksissa. Muutama 20 annetun siirron toistoista päätyi yhä hyvään 78 siirron tulokseen, mutta saavutettujen tulosten keskiarvo oli huomattavasti aikaisempaa huonompi.

Taulukossa 3 on annettu testeihin kuluneet ajat ja kunkin toiston parhaan tuloksen löytämiseen tarvittu keskimääräinen aika. Tuloksista voidaan huomata, että 30 annettuun siirtoon asti paras tulos löytyi nopeasti ja koko testiin kulunut aika kasvoi lähes lineaarisesti annettujen siirtojen määrään nähden. Yllättävänä rajatuloksena voidaan huomata, että menetelmän saadessa vain 20 - 25 siirtoa valmiiksi määriteltynä, menetelmä joutui itse käyttämään huomattavasti enemmän aikaa hyvien siirtojen valitsemiseen. Parhaat tulokset löytyivät myös huomattavasti myöhemmin, koska menetelmä joutui käyttämään huomattavasti pitemmän ajan hyvän naapuruston löytämiseksi.

Annettujen siirtojen määrä	Toistojen määrä	Paras tulos löytyi keskimäärin ajassa	Koko testiin kulunut aika	Paras löydetty tulos
55	10	3,1 sekuntia	95 sekuntia	80
50	10	4,9 sekuntia	128 sekuntia	80
45	10	9,4 sekuntia	176 sekuntia	80
40	10	19 sekuntia	235 sekuntia	80
35	10	35 sekuntia	296 sekuntia	79
30	10	58 sekuntia	384 sekuntia	79
25	10	720 sekuntia	1280 sekuntia	79
20	10	1720 sekuntia	2800 sekuntia	78

Taulukko 3. Riippuvuustestien tulokset.

Vielä 20 annetun siirron kohdalla hyvän tuloksen riippuvuus ensimmäisistä siirroista on huomattava. Joillakin toistoilla saavutettiin 78 siirron tulos alle puolessa tunnissa. Näin hyvän tuloksen löytäminen tässä ajassa ilman ennalta määritettyjä siirtoja on erittäin epätodennäköistä.

Testien perusteella voidaan siis päätellä, että hyvän tuloksen naapurusto määritellään erittäin aikaisessa vaiheessa, todennäköisesti jo ensimmäisillä 20 siirrolla. Huomataan myös, että noin 35 - 40 siirron jälkeen loput siirroista on suhteellisen helppo valita optimaalisesti.

Myös Bernard Helmstetterin ja Tristan Cazenaven [2004] saavuttama tulos tukee oletusta, että hyvän tuloksen naapurusto muodostuu jo aikaisessa vaiheessa. Helmstetter ja Cazenave [2004] todistavat tutkimuksessaan, että Charles-Henri Bruneaun 170 siirron tuloksesta ainakin viimeiset 109 siirtoa ovat optimaalisia. Toisin sanoen, tuloksen naapurusto on vahvasti määritelty koskevassa mallissa jo viimeistään 61. siirron kohdalla.

5.3 Morpion Solitairen vaikeus ja hyvien tulosten löytäminen

Pelin vaikeutta voidaan arvioida kaikkien mahdollisten pelien määrän avulla (game tree complexity). Mahdollisia pelejä ovat kaikki pelipuun lehtisolmut. Monissa tapauksissa pelien määrää on vaikea edes arvioida. Eräs arvio saadaan nostamalla pelitilanteissa mahdollisten valintojen keskimääräinen lukumäärä pelin pituuden keskiarvon potenssiin. Toisin sanoen, jos Morpion Solitairessa olisi irrallisessa mallissa esimerkiksi keskimäärin 12 mahdollista siirtoa ja keskimääräinen pelin pituus olisi 50 siirtoa, pelissä olisi arviolta vähintään 12^{50} eli noin 10^{54} mahdollista siirtojen yhdistelmää. Pascal Zimmer [*] arvioi irrallisessa mallissa olevan noin 10^{22} toisistaan siirroiltaan poikkeavaa ratkaisua. Toisistaan siirroiltaan poikkeavilla ratkaisuilla tarkoitetaan siis pelejä, joissa jokaisessa on vähintään yksi muista poikkeava siirto eikä esimerkiksi samat siirrot toisistaan poikkeavassa järjestyksessä.

Vertailukohteena voidaan antaa myös esimerkkejä muiden pelien mahdollisten pelien määrästä. Esimerkiksi 3x3 ruudukolla pelattavassa ristinollassa on noin 10^5 mahdollista pelivaihtoehtoa, shakissa arviolta 10^{123} ja Go-pelissä arviolta jopa 10^{360} mahdollista siirroiltaan poikkeavaa peliä [Wikipedia].

Hyvien tulosten löytäminen Morpion Solitaire -pelissä on erittäin vaikeaa. Satunnaisia siirtoja valitsemalla pelin alusta alkaen, yli 60 siirron tulokset ovat kohtalaisen harvinaisia. Tämä voidaan huomata myös Cazenaven [2007] saavuttamista tuloksista. Cazenaven pelaamissa 10 miljoonassa sattumanvaraisia siirtoja valitsevassa pelissä paras saavutettu tulos oli 64 siirtoa. Hyvien hakupolkujen löytymisen vaikeus voidaan huomata myös monista menetelmistä, kuten Cazenaven Monte Carlo -haut ja Juillén END-menetelmä, joissa pelataan erittäin suuria määriä sattumanvaraisia pelejä hyvien polkujen löytämiseksi.

Esimerkiksi refleksiivisessä Monte Carlo -haussa parhaan tuloksen löytämiseksi Cazenave päätyi pelaamaan 100 peliä alimmalla tasolla jokaista meta-tason siirtoa varten. Tämä tarkoittaa siis 100 sattumanvaraista peliä jokaista meta-tason pelin noin 65 siirtoa varten. Toisaalta voidaan huomata, että toisen meta-tason haun edetessä sattumanvaraisten pelien määrä laskee laskettavien kierrosten määrän vähentyessä. Tästä johtuen jokainen meta-tason peli koostuu siis keskimäärin noin 3250 sattumanvaraisesta pelistä. Parasta refleksiivisellä Monte Carlo -haulla saavutettua tulosta varten pelattiin 1000 meta-tason peliä jokaista toisen meta-tason siirtoa varten. Koska paras saavutettu tulos oli 78 siirtoa, toisen meta-tason siirtoja varten pelattiin noin 78 tuhatta ensimmäisen meta-tason peliä. Koko haussa pelattiin siis tekijän arvion mukaan yhteensä yli 250 miljoonaa sattumanvaraista peliä lopullisen tuloksen saavuttamiseksi.

[*] Zimmerin esittämään arvioon ei liity tieteellistä julkaisua tai mitään varsinaista todistusta.

Lukuun 5.2 perustuen voidaan todeta, että 20 - 25 ensimmäistä siirtoa määrittelevät vahvasti lopullisten tulosten laadun. Toisaalta, hyviä tuloksia on vaikea saavuttaa, ellei siirtoja lukita ja siirrytä tarkastelemaan jotain ratkaisuavaruuden naapurustoa tarkemmin. Huonot valinnat varhaisessa vaiheessa voivat kuitenkin johtaa ratkaisualgoritmin nopeasti huonoon lokaaliin optimiin. Juuri tämä vastakkainasettelu tekee ongelmasta vaikean. Hyviä tuloksia on vaikeaa saavuttaa sattumanvaraisilla peleillä, ellei hyviksi todettuja siirtoja lukita ja toisaalta siirtojen lukitseminen ajaa haun helposti naapurustoon, joka ei välttämättä sisällä globaalia optimia tai edes hyvää lokaalia optimia.

Luvussa 5.2 pohditusta ratkaisuavaruuden muodosta päätellen erikoistuminen ja eteneminen vain yhtä naapurustoa kohti ei todennäköisesti tuota hyviä tuloksia. Jos ratkaisu esimerkiksi lähestyy naapurustoa, josta on löydetty muutamia yli 75 siirron tuloksia, ratkaisuavaruuden piikikkäästä muodosta johtuen juuri nämä tulokset saattavat olla naapuruston parhaita mahdollisia eikä tähän suuntaan kannattaisi enää erikoistua.

Tästä johtuen voidaan päätellä, että menetelmät, joissa tehtyjä valintoja pystytään perumaan, sopivat ongelman ratkaisuun mainiosti. Esimerkiksi Hyyrön ja Porasen [2007] simuloitu jäähdytys on yksi tällainen menetelmä. Toisaalta, simuloidussa jäähdytyksessä peruuttaminen perustuu lämpötilasta riippuvaan todennäköisyyteen. Jos hyvä naapurusto määritellään jo muutamien ensimmäisten siirtojen avulla, peruuttaminen näiden valintojen muuttamiseksi voi olla erittäin hankalaa.

Luvun 5.2 tuloksien perusteella voidaan myös päätellä, että pelin loppuosa on alkuosaa huomattavasti helpompi pelata hyvin. Tästä voidaan päätellä, että hyvän tuloksen saavuttamiseksi voitaisiin mahdollisesti käyttää algoritmia, joka voidaan asettaa esimerkiksi levittäytymään suuremmalle alueelle tai tutkimaan pelin alkuosaa tarkemmin ennen valintojen kiinnittämistä.

5.4 Siirtojen valintaan käytettävät heuristiikat

Ratkaisuavaruuden muodosta johtuen tuloksien arvioiminen pelkästään saavutettujen siirtojen määrän perusteella ei välttämättä ole paras mahdollinen tapa vertailla siirtojen hyvyyttä. Cazenave [2007] esitteli refleksiivisen Monte Carlo -haun yhteydessä kaksi erilaista heuristiikkaa seuraavan siirron arvioimiseen. Kolmas tässä luvussa esitelty heuristiikka perustuu potentiaalifunktioon.

5.4.1 Liikkuvuus-heuristiikka

Refleksiivisen Monte Carlo -haun yhteydessä Cazenave [2007] esittelee *liikkuvuus-heuristiikan* (mobility heuristic). Liikkuvuus-heuristiikan avulla valitaan tarkasteltaviksi

vain ne siirrot, jotka johtavat pelitilanteisiin, joissa on siirron jälkeen suurin määrä mahdollisia siirtoja.

Liikkuvuus-heuristiikka perustuu yksinkertaisesti mahdollisten siirtojen määrän tarkasteluun. Heuristiikka vertaa pelitilanteita jokaisen mahdollisen siirron jälkeen. Seuraava siirto, valitaan sattumanvaraisesti ainoastaan niiden siirtojen joukosta, jotka johtavat pelitilanteisiin, joissa on eniten mahdollisia siirtoja. Cazenave [2007] saavutti liikkuvuus-heuristiikan avulla keskimäärin yhtä hyviä tuloksia kuin pelaamalla kymmenkertaisen määrän sattumanvaraisia pelejä. Toisaalta tuloksista voidaan kuitenkin huomata, että ajojen suoritusajat heuristiikan kanssa ovat vastaavasti kymmenkertaiset sattumanvaraisiin siirtoihin verrattaessa, joten siirtojen hyvyyden arviointi tällä menetelmällä ei ole kovinkaan tehokasta. Cazenave [2007] toteaakin, että molemmilla tavoilla löydetään yhtä hyviä tuloksia, jos suoritusaika on sama. Tämän lisäksi liikkuvuus-heuristiikka sulkee pois huomattavan määrän ratkaisuvaihtoehtoja, joten on mahdollista, että erittäin hyviinkin tuloksiin johtavia polkuja menetetään.

5.4.2 UCT-menetelmä

Toinen Cazenaven [2007] esittelemä menetelmä siirtojen arvioimiseksi perustuu huonolta näyttävien vaihtoehtojen hylkäämiseen tilastotieteen luottamusrajojen avulla. Cazenaven [2007] esittelemä menetelmä tunnetaan nimellä *UCT* (UCB applied to trees, upper confidence trees). UCT on puurakenteille soveltuva muunnos *UCB*-menetelmästä (upper confidence bounds). Alun perin UCT menetelmän esittelivät Cbasa Kocsis ja Levente Szepesvári [2006]. Menetelmä soveltuu Kocsisin ja Szepesvárin [2006] mukaan hyvin ongelmille, joissa mahdollisten vaihtoehtojen arvot eivät ole suoraan arvioitavissa.

Menetelmä perustuu huonolta näyttävien vaihtoehtojen hylkäämiseen. Käytännössä tämä tarkoittaa jokaisen vaihtoehdon arviointia satunnaisotoksen ja tilastotieteessä käytettyjen luottamusrajojen avulla. Jos tehdyn satunnaisotoksen perusteella voidaan esimerkiksi sanoa 95 % luottamuksella, että jokin vaihtoehto on huono, se voidaan kokonaan ohittaa.

Käytännössä menetelmässä annetaan jokaiselle mahdolliselle siirrolle arvo esimerkiksi satunnaisotantaa käyttäen. Kun satunnaisotannalla on saavutettu jokin tulos, tuloksen arvo voidaan päivittää tuloksen tekemille valinnoille. Jokaisen valinnan arvo määräytyy saavutetun tuloksen perusteella, painotettuna valinnan aikaisemmilla tuloksilla ja sillä, kuinka usein kyseistä valintaa on tutkittu. Toisin sanoen, usean satunnaisotannan avulla jokaiselle siirrolle pyritään laskemaan siirron sisältävän tuloksen odotusarvo ja luottamusraja tälle odotusarvolle.

Myöhempien siirtojen valinnan kohdalla haut odotusarvoltaan ja todennäköisyydeltään huonoilta näyttäviin suuntiin voidaan keskeyttää. Seuraavaksi siirroksi voidaan valita esimerkiksi se vaihtoehto, jolla on paras tuloksen odotusarvo ja suurin luottamus tämän arvon saavuttamiseen.

Cazenave [2007] ei erityisesti esittele käyttämäänsä toteutusta tästä menetelmästä, mutta toteaa testituloksissaan kuitenkin saavuttaneensa hieman satunnaisotantaa ja liikkuvuus-heuristiikkaa parempia tuloksia tämän valintamenetelmän avulla.

Käytännössä valintamenetelmän toiminta muistuttaa hieman Cazenaven [2009] myöhemmin esittelemää sisäkkäistä Monte Carlo -hakua, joka esiteltiin luvussa 4.4.

5.4.3 Potentiaali ja tuhlattu potentiaali

Tässä luvussa pohditaan uutta mahdollista tapaa siirtojen sopivuuden arvioimiseen. Seuraavaksi esitellään menetelmän teoreettinen perusta ja pyritään arvioimaan menetelmän soveltuvuutta siirtojen valintaan.

Menetelmässä siirtojen sopivuutta pyritään arvioimaan *tuhlatun potentiaalin* avulla. Tuhlatun potentiaalin käsitteen selittämiseksi esitellään ensin tarkemmin Morpion Solitairen *potentiaalifunktio.* Potentiaalin käsitteen Morpion Solitairelle esitteli alun perin Daniel Goffinet Science & Vie -lehdessä [Boyer]. Myöhemmin Demaine *et al.* [2006] käyttivät potentiaalin käsitettä tutkimuksessaan todistaessaan ehdottamansa siirtojen ylärajat Morpion Solitairen eri muunnoksille.

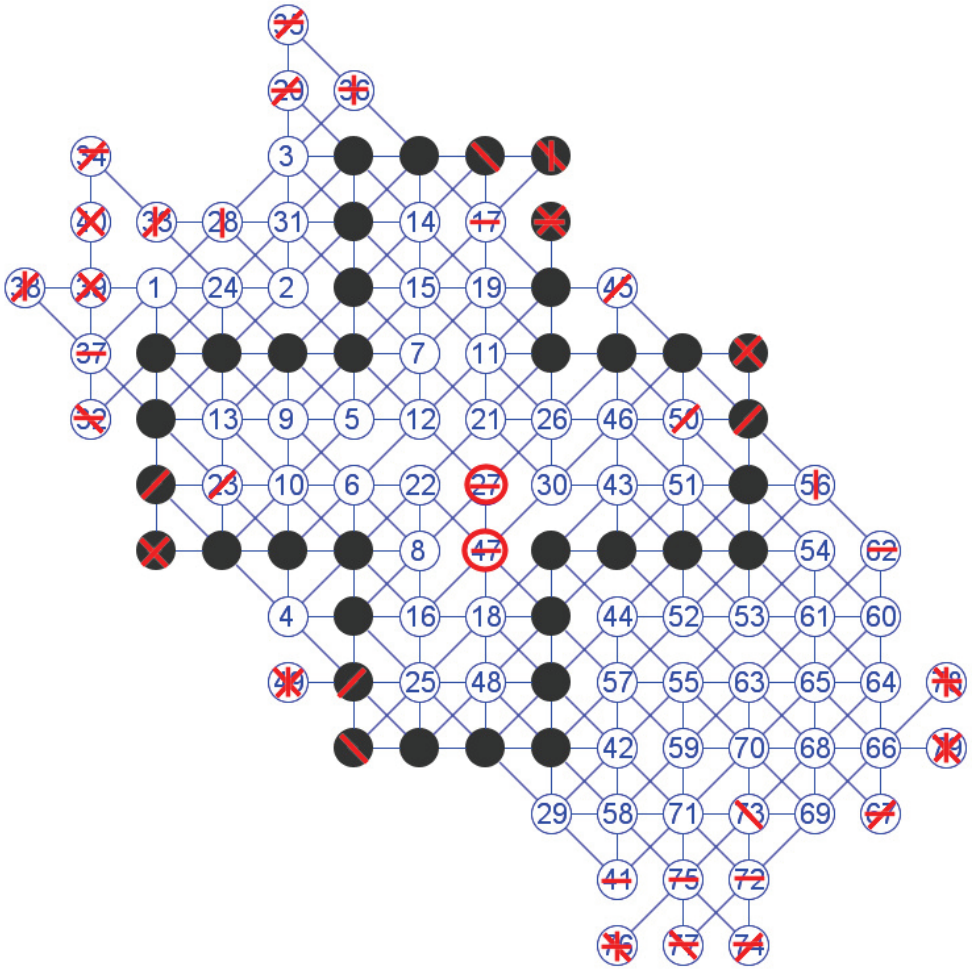
Potentiaalikäsitteen mukaan jokaisella pelialueen pisteellä on jokin potentiaaliarvo. Jokaisen pisteen potentiaaliarvoksi asetetaan niiden siirron suuntien lukumäärä, joihin pistettä voidaan vielä käyttää. Toisin sanoen, jokaista pistettä voidaan koskevassa mallissa käyttää enintään kahdeksaan eri siirtoon, jos jokaisen siirron toinen loppupiste olisi kyseinen piste. Irrallisessa mallissa jokaista pistettä voidaan käyttää enintään neljään siirtoon, koska samansuuntaiset siirrot eivät voi jakaa yhtäkään pistettä. Yleisesti voidaan määritellä, että jokaisen pisteen potentiaaliarvo on 8 ja irrallisessa mallissa jokainen siirto käyttää päätepisteissään kaksi potentiaalia koskevan mallin yhden potentiaalin sijaan. Käytännössä pisteen potentiaali on siis arvo, joka kertoo, kuinka moneen siirtoon pistettä voidaan vielä enintään käyttää.

Demaine *et al.* [2006] esittelemässä potentiaalimallissa jokaisen siirron lisäämä piste kasvattaa koko pelin potentiaalia. Vastaavasti voidaan huomata, että jokainen siirto myös vähentää potentiaalia, koska siirron käyttämien pisteiden potentiaaliarvot vähenevät siirron ansiosta. Esimerkiksi $G_k(A_k)$ -mallissa jokainen uusi piste lisää potentiaalia kahdeksalla, mutta toisaalta viiden pisteen mittainen siirto vähentää potentiaalia kymmenellä, koska jokaisessa viidestä pisteestä menetetään kaksi mahdollista siirron suuntaa.

Määritellään nyt potentiaalin käsitteen perusteella tuhlatuksi potentiaaliksi pisteiden sellaiset suunnat, joiden potentiaalia ei ole käytetty siirron osana, mutta siirto kyseiseen suuntaan ei koskaan tule olemaan mahdollinen. Esimerkiksi yksittäinen piste, joka jää kahden samansuuntaisen siirron väliin tuhlattu, koska pisteen potentiaalia tähän suuntaan ei koskaan voida käyttää kummallakin puolella sijaitsevien siirtojen takia.

Myös pelin reunoilla sijaitsevat pisteet voivat tuhlata potentiaalia. Vaikka nämä pisteet voidaankin teoriassa vielä käyttää, käytännössä peliä voi olla mahdotonta täydentää niin, että pisteiden kautta voitaisiin tehdä siirto tiettyyn suuntaan. Tätä on tietysti huomattavasti vaikeampi arvioida kesken pelin, koska myöhemmässä pelitilanteessa pisteen ympärille voi kerääntyä tarpeeksi pisteitä ja potentiaali pystytäänkin käyttämään.

Tarkastelemalla aikaisemmin saavutettuja hyviä tuloksia, esimerkiksi Hyyrön ja Porasen 79 siirron tulosta, voidaan huomata, että näissä tuloksissa on tuhlattu erittäin vähän potentiaalia. Kuvassa 8 on esitetty Hyyrön ja Porasen 79 siirron tulos, johon on viivoilla merkitty ne pisteet ja suunnat, joiden potentiaali on tuhlattu. Ratkaisun keskellä sijaitsevat tuhlattua potentiaalia sisältävät pisteet on ympyröity.



Kuva 8. Hyyrön ja Porasen 79 siirron tulos, johon on viivoilla merkitty tuhlattu potentiaali viivan osoittamaan suuntaan.

Kuvasta voidaan huomata, että ainoastaan kahdessa ratkaisun keskellä sijaitsevassa pisteessä on tuhlattu potentiaalia ja loput tuhlatusta potentiaalista sijaitsee ratkaisun reunoilla. Voidaan kuitenkin huomata, että reunoillakin potentiaalia on tuhlattu erittäin vähän. Esimerkiksi kahden pisteen tuhlaaminen yhdellä rivillä annettuun suuntaan on erittäin harvinaista ja yleisesti voidaan huomata, että suuri osa riveistä sisältää tasan yhtä monta pistettä kuin rivin siirtoihin tarvitaan. Kuvan 8 yhteenlaskettu tuhlattu potentiaali on 130.

5.4.4 Tuhlattu potentiaali valintaperusteena

Todistetaan seuraavaksi, että koko pelin yhteenlaskettu tuhlatun potentiaalin määrä on vakio jokaiselle pelille, jossa on sama määrä siirtoja.

Huomataan, että A_4 sisältää 36 aloituspistettä, joista jokaisella on 8 potentiaalia alussa. Toisin sanoen, A_4 aloituskuvion potentiaali on alussa $8 * 36 = 288$. Demaine *et al.* [2006] esittävät, että jokainen siirto poistaa $2(k + 1)$ koko mallin potentiaalista ja samalla jokaisen siirron lisäämä piste kasvattaa potentiaalia kahdeksalla. Koska $G_4(A_4)$-mallissa $k = 4$, jokainen siirto poistaa $2(4 + 1) - 8 = 2$ potentiaalia.

Olkoot nyt N siirtojen määrä pelissä. Nyt voidaan nähdä, että $G_4(A_4)$-mallille pitää paikkansa $288 - 2N = P$. Nyt voidaan huomata, että P on yllä määritelty käyttämätön, tuhlattu potentiaali. Toisin sanoen, koko pelin tuhlatun potentiaalin määrä riippuu pelin siirtojen määrästä. Samaa ominaisuutta ei kuitenkaan voida todistaa $G'_4(A_4)$-mallille, koska jokainen siirto poistaa vain 8 potentiaalia, jolloin koko pelin potentiaali ei itse asiassa muutu.

Voidaan kuitenkin nähdä, että koko pelistä laskettu tuhlattu potentiaali on $G_4(A_4)$-mallissa identtinen vertailuperuste siirtojen määrään verrattuna. Toisin sanoen, siirtojen määrän kasvaessa, koko mallin tuhlattu potentiaali vähenee, joten ongelma muuttuu siirtojen määrän maksimoinnista tuhlatun potentiaalin minimoimiseen. Tähän perustuen voidaan todeta, että kaikista pisteistä laskettua tuhlattua potentiaalia ei ole järkevää vertailla. Tämä pätee erityisesti myös $G'_4(A_4)$-mallissa, koska koko mallista laskettu tuhlattu potentiaali ei edes muutu siirtojen ansiosta.

Tuhlatun potentiaalin avulla saatettaisiin kuitenkin saavuttaa hyviä tuloksia, jos tuhlattu potentiaali laskettaisiin vain ongelmallisista kohdista kaikkien pelin pisteiden sijaan. Esimerkiksi Hyyrön ja Porasen [2007] saavuttamassa tuloksessa mallin keskelle on sattunut vain kaksi pistettä, joissa on tuhlattu potentiaalia. Tästä voidaan päätellä, että esimerkiksi tuhlatun potentiaalin minimoiminen pelin keskellä sijaitsevissa pisteissä saattaisi tuottaa hyviä tuloksia.

5.4.5 Tuhlatun potentiaalin testaaminen

Seuraavassa tarkastellaan erilaisia mahdollisia tapoja tuhlatun potentiaalin käyttämiseksi pelin laadun arvioimiseen. Menetelmissä 1 ja 2 lasketaan tuhlattu potentiaali loppuun pelatusta pelistä ja käytetään saatua arvoa pelien vertailuperusteena siirtojen määrän sijaan jossakin optimointimenetelmässä. Menetelmissä 3 ja 4 arvo lasketaan jokaiselle mahdolliselle siirrolle pelitilanteesta ja rajoitutaan siirron valinnassa tiettyyn määrään parhaan pistemäärän saaneita tuloksia.

Menetelmä 1. Lasketaan tuhlattu potentiaali erikseen jokaiselle mahdolliselle siirtojen suunnalle jokaisella pelin rivillä. Kunkin rivin tuhlattu potentiaali korotetaan johonkin potenssiin, jolloin suurempi potentiaalin tuhlaus yhdellä rivillä tekee ratkaisusta selvästi huonomman kuin pienempi potentiaalin tuhlaus jakautuneena useammalle riville.

Menetelmä 2. Keskitytään tarkastelemaan vain pelin keskelle sijoittuvaa tuhlattua potentiaalia.

Menetelmä 3. Jokaista siirtoa tarkastellaan suorittamalla kyseinen siirto ja laskemalla tuhlattu potentiaali menetelmän 1 mukaisesti tästä tilanteesta.

Menetelmä 4. Jokaista siirtoa tarkastellaan suorittamalla kyseinen siirto ja laskemalla tuhlattu potentiaali menetelmän 2 mukaisesti tästä tilanteesta.

Taulukossa 4 on esitetty menetelmillä 1 ja 2 saadut tulokset. Menetelmiä testattiin käyttämällä optimointimenetelmänä luvussa 6 esiteltävää menetelmää populaation koolla 32 x 32. Testit suoritettiin 5D mallissa vertaamalla siirtojen määrän sijaan tuloksista laskettua tuhlattua potentiaalia. Taulukossa on vertailun vuoksi esitetty myös perinteisellä siirtojen määrään perustuvalla vertailulla saavutetut tulokset.

Vertailumenetelmä	Toistojen määrä	Tulosten keskiarvo	Paras tulos	Lisätiedot
Siirtojen määrä	20	69,2	73	-
Menetelmä 1	5	63,8	65	Jokaisen rivin tuhlattu potentiaali korotettu potenssiin 1.5.
Menetelmä 1	5	63,8	66	Jokaisen rivin tuhlattu potentiaali korotettu potenssiin 2.
Menetelmä 2	5	64,4	66	Keskellä olevan pisteen etäisyys reunasta vähintään 1.
Menetelmä 2	5	64,8	67	Keskellä olevan pisteen etäisyys reunasta vähintään 2.

Taulukko 4. Tuhlatun potentiaalin tulokset menetelmillä 1 ja 2.

Taulukon 4 tuloksista voidaan nähdä selvästi, että loppuun pelatuista peleistä lasketun tuhlatun potentiaalin käyttäminen vertailuperusteena johtaa huonompiin tuloksiin kuin siirtojen määrän käyttäminen.

Taulukossa 5 on esitetty menetelmillä 3 ja 4 saadut tulokset. Testeissä pelattiin määritelty määrä pelejä ja peleissä käytettiin seuraavan siirron valintaan annettua menetelmää. Menetelmien avulla mahdollisten siirtojen määrä rajoitettiin puoleen. Toisin sanoen, menetelmillä pisteytettiin kaikki mahdolliset siirrot kussakin pelitilanteessa ja pisteytykseen perustuen mahdollisista siirroista karsittiin puolet pois.

Siirtojen valintamenetelmä	Pelien määrä	Tulosten keskiarvo	Paras tulos	Lisätiedot
Satunnaiset siirrot	1 000 000	42,8	64	-
Menetelmä 3	1 000 000	40.2	62	Jokaisen rivin tuhlattu potentiaali korotettu potenssiin 2.
Menetelmä 4	1 000 000	41,3	63	Keskellä olevan pisteen etäisyys reunasta vähintään 2.

Taulukko 5. Tuhlatun potentiaalin tulokset menetelmillä 3 ja 4.

Taulukon 5 tuloksista voidaan nähdä, että mahdollisia siirtoja rajoittamalla päädyttiin samoihin tai hieman huonompiin tuloksiin kuin täysin sattumanvaraisia siirtoja käyttäen.

Tässä luvussa todettiin hyviin tuloksiin päätyneissä peleissä olevan erittäin vähän tuhlattua potentiaalia. Tämän lisäksi todistettiin, että ainakin irralliselle mallille koko pelistä lasketun tuhlatun potentiaalin minimoiminen on yhtä hyvä vertailuperuste kuin siirtojen määrän vertailu. Näiden pohjalta saatiin hypoteesi, että minimoimalla tuhlatun potentiaalin määrä saatettaisiin saavuttaa pelkkää satunnaista siirtojen valintaa parempia tuloksia.

Taulukon 4 ja 5 tuloksiin perustuen hypoteesi voidaan kuitenkin todeta vääräksi. Tuhlattu potentiaali ei sovellu siirtojen tai pelien hyvyyden vertailuun, ainakaan tässä esitellyillä tavoilla.

6 Monitasoinen evoluutiomalli

Tässä luvussa esitellään tutkielman tekijän oma Morpion Solitairen ratkaisuun käyttämä menetelmä nimeltä monitasoinen evoluutiomalli. Monitasoinen evoluutiomalli perustuu vahvasti Juillén [1995] esittelemään evoluutiomalliin. Menetelmässä pyritään kuitenkin yhdistelemään muidenkin luvussa 4 esiteltyjen menetelmien hyviksi todettuja ominaisuuksia. Joitakin alkuperäisten menetelmien ominaisuuksia on myös pyritty parantelemaan. Tämän lisäksi menetelmässä pyritään käyttämään hyväksi joitakin luvussa 5 esiteltyjä tekijän omia huomioita ratkaisuavaruuden muodosta.

Monitasoisessa evoluutiomallissa pyritään yhdistämään Juillén [1995] esittelemän evoluutiomallin ja Cazenaven [2007, 2009] Monte Carlo -hakujen hyviksi todettuja ominaisuuksia. Juillén evoluutiomallissa hyvänä ominaisuutena on usean eri hakupolun samanaikainen tutkiminen. Monista muista menetelmissä poiketen, Juillén evoluutiomallissa ei suoraan hylätä kaikkia huonompia tuloksia ja edetä vain parhaimmalta näyttävään suuntaan. Kohtalaisen hyvätkin tulokset voivat pitkään säilyä populaatiossa tutkittavina, kunnes parempiin tuloksiin päätyvät ratkaisut lopulta korvaavat ne. Juillén evoluutiomallin hyvinä puolina voidaan siis nähdä laajalle levittäytyvä haku, joka lopulta sijoittaa suuremman osan käytössä olevista resursseista mielenkiintoisien hakupolkujen tutkimiseen.

Toinen hyvä puoli populaatioon perustuvassa mallissa on helppo rinnakkaistettavuus. Koska minkään olion toiminta ei rakennusvaiheessa riipu muista olioista, oliot voidaan helposti jakaa useamman prosessorin tai tietokoneen suoritettavaksi.

Cazenaven [2007, 2009] Monte Carlo -hakujen hyvänä puolena voidaan nähdä monella tasolla tapahtuva haku. Jokaisella ylemmällä tasolla käytetään hyväksi alemmilla tasoilla saavutettuja parhaita tuloksia. Tämän lisäksi jokaisella ylemmällä tasolla oletetaan tästä johtuen saavutettavan parempia tai vähintään yhtä hyviä tuloksia, kuin millä tahansa alemmalla tasolla. Usealla tasolla pelaaminen mahdollistaa myös tehtyjen valintojen vaikutusten tarkemman tutkimisen myöhemmissä pelitilanteissa alemmilla tasoilla vaikuttamatta ylemmän tason pelissä lukittuihin siirtoihin.

Eri tasot voidaan käytännössä myös käsittää eräänlaisena peruuttamisena ja hyvien valintojen varmistamisena ratkaisussa. Cazenaven Monte Carlo -hauissa alemmilla tasoilla pelit pelataan loppuun ja näistä parhaimman tuloksen saavuttaneessa ”peruutetaan” ylemmän tason haettavan siirron kohdalle ja alimman tason haku toistetaan uudesta tilanteesta. Toisin sanoen, valinnan hyvyys pyritään näin tarkistamaan ja samalla pyritään selvittämään, löydetäänkö uudesta tilanteesta parempia siirtoja kuin ennen.

Monitasoisessa evoluutiomallissa pyritään yhdistämään nämä ominaisuudet näistä kahdesta menetelmästä.

6.1 Monitasoisen evoluutiomallin toiminta

Monitasoisessa evoluutiomallissa on yksinkertaistetusti kyse Monte Carlo -haun tapaan monella tasolla pelattavasta Juillén [1995] evoluutiomallista. Ensimmäisellä tasolla toiminta vastaa täsmälleen luvussa 4.3 esitettyä Juillén END-mallin toimintaa. Haun alussa alustetaan populaatio ja jokainen populaation olio pelaa sattumanvaraisen pelin kunkin evoluutiokierroksen rakennusvaiheessa. Olioiden saavuttamia tuloksia vertaillaan kilpailuvaiheessa ja paremmat oliot levittäytyvät laajemmalle alueelle populaatioon. Kilpailuvaiheen jälkeen mallin sitoutumisastetta voidaan nostaa käytettyyn strategiaan perustuen. Sitoutumisasteen nostaminen voidaan toisin sanoen ymmärtää siirtymisellä eteenpäin menetelmän alimmalla tasolla.

Menetelmän ylemmällä tasolla siirrytään eteenpäin, kun alemman tason populaatio on saavuttanut lopulliset tuloksensa tai jokin muu määritelty lopetusehto alimmalle tasolle täyttyy. Käytännössä tämä tarkoittaa jokaisen populaation olion palauttamista ylemmän tason haun syvyyttä vastaavaksi. Kyseinen palauttaminen voidaan myös ymmärtää ylemmän tason sitoutumisasteena. Aivan kuin alemman tason evoluutiokierrosten välissä oliot palautetaan alemman tason sitoutumisasteen tasolle, oliot palautetaan nyt ylemmän tason sitoutumisasteen tasolle ja alemman tason haku toistetaan.

Käytännössä menetelmässä on käytössä vain yksi populaatio, vaikka peliä pelataankin usealla tasolla. Jokaisen alimman tason kierroksen jälkeen hyviksi todetut oliot ovat valloittaneet suuren osan populaatiosta. Toisin sanoen, kun oliot palautetaan ylemmän tason sitoutumisasteen tasolle, haku aloitetaan uudestaan näiden hyviin tuloksiin päätyneiden ja populaatioon levittäytyneiden olioiden ensimmäisistä siirroista. Erona Monte Carlo -hakuihin menetelmässä on kyky säilyttää muitakin kuin aivan paras tulos. Sitoutumisasteen hallinta alimmalla tasolla vaikuttaa ratkaisevasti siihen, kuinka suuren osan populaatiosta parhaimmat tulokset valloittavat. Oikein valitulla strategialla, parhaan tuloksen lisäksi voidaan helposti säilyttää muitakin hyviä tuloksia saavuttaneita olioita.

Jos monitasoista evoluutiomallia haluaa verrata refleksiiviseen Monte Carlo -hakuun, alimman tason olioiden voidaan katsoa vastaavan muuttuvaa määrää ensimmäisen meta-tason pelejä mukautuvalla määrällä sattumanvaraisia pelejä. Toisin sanoen, alimman tason populaation alussa jokaista oliota voidaan verrata ensimmäisen meta-tason peliin, joka pelaa vain yhden sattumanvaraisen pelin. Kilpailuvaiheessa oliot levittäytyvät ja korvaavat muita olioita, jolloin voidaan tietyssä mielessä katsoa ensimmäisen tason meta-pelien määrän vähentyvän, mutta samalla yhden meta-pelin pelaama satunnaisten

pelien määrä kasvaa. Tässä mielessä ylemmän tason populaatio vastaisi siis muuttuvaa määrää toisen meta-tason pelejä mukautuvalla määrällä ensimmäisen meta-tason pelejä.

Monitasoisen evoluutiomallin hyvinä puolina voidaan nähdä Juillén evoluutiomallin tavoin laajemmalle levittäytyvä haku, joka ei suoraan erikoistu parhaiden mahdollisten tulosten suuntaan. Samoin säilytetään menetelmän helposta rinnakkaistettavuudesta saavutetut hyödyt. Menetelmässä pyritään myös saavuttamaan Monte Carlo -hakujen tapaan parempia tuloksia monella tasolla tapahtuvan haun avulla.

Juillén [1995] evoluutiomallissa jokaista evoluutiokierrosta pelataan, kunnes jokin olioista valloittaa suurimman osan populaatiosta eli kunnes jokin vaihtoehdoista on selvästi muita parempi. Monitasoisessa evoluutiomallissa voidaan alimmalla tasolla nostaa sitoutumisastetta vaikka jokaisen evoluutiokierroksen jälkeen. Juillén menetelmässä evoluutiokierroksia toistetaan useaan kertaan, koska sitoutumisasteen nostamisen jälkeen valitut siirrot ovat pysyviä. Toisaalta voidaan huomata, että sitoutumisasteen ollessa alhainen, evoluutiokierroksen toistamisella ei todennäköisesti saavuteta paljonkaan aikaisempaa parempia tuloksia, koska hyvän tuloksen saavuttaminen satunnaisia siirtoja valiten on yhä vaikeaa.

Monitasoisessa evoluutiomallissa on etuna tähän verrattuna kyky erikoistua pelin loppuun nopeammin, koska ylemmällä tasolla siirtoja ei ole vielä valittu pysyvästi. Erikoistumisen ansiosta tehtyjen valintojen arvoja voidaan arvioida paremmin, koska oliot voivat erikoistumisen ansiosta saavuttaa satunnaisia pelejä parempia tuloksia. Peli pelataan loppuun alimmalla tasolla evoluutiomallin tapaan, jonka jälkeen populaatio palautetaan ylemmän tason sitoutumisasteen tasolle. Toisin sanoen, seuraavalla ylemmän tason kierroksella sijoitetaan suurempi osa populaation resursseista mielenkiintoiseksi todettujen olioiden tarkempaan tutkimiseen.

Seuraavassa esitellään muutamia monitasoisen evoluutiomallin ominaisuuksia. Ominaisuuksien vaikutuksia menetelmän toimintaan testataan tarkemmin luvussa 6.3 muiden muuttujien testauksen yhteydessä.

6.1.1 Mukautuva naapuruston koko

Naapuruston koolla tarkoitetaan evoluutiomallissa sitä, kuinka suuren alueen populaatiosta kukin olio laskee naapureikseen. Jokaisen evoluutiokierroksen kilpailuvaiheessa jokainen olio vertaa tulostaan jokaiseen naapuriinsa ja kopioi ratkaisunsa huonompien naapuriolioiden päälle.

Juillé [1995, 1999] käyttää menetelmissään vakioarvoa naapuruston koon määrittämiseksi. Juillé [1995] esittää, että pienemmällä naapuruston koolla voidaan helpommin säilyttää erilaisia ratkaisuvaihtoehtoja ja osittain estää hyvien tuloksien

korvautumista huonon satunnaisotannan ansiosta. Toisaalta suuremmalla naapuruston koolla hyvät tulokset leviävät nopeammin populaatiossa.

Tekijän mielestä evoluutiomallissa voidaan saavuttaa parempia tuloksia käyttämällä mukautuvasti määriteltyä naapuruston kokoa.

Naapuruston koko voidaan määritellä mukautuvasti olion saavuttamaan tulokseen perustuen ja tulosta voidaan verrata kaikkien olioiden saavuttamaan parhaimpaan tulokseen. Toisin sanoen, menetelmässä voidaan määritellä useita tulosten arvojen välejä ja olion saavuttaessa tuloksen tietyltä väliltä, olion naapuruston koko määritellään tämän välin mukaan. Välit voidaan edelleen sitoa parhaimpaan löydettyyn tulokseen, jolloin naapuruston koot mukautuvat erilaisille tuloksille menetelmän edetessä.

Mukautuva naapuruston koko voidaan määritellä esimerkiksi seuraavasti:

- Jos olion saavuttama tulos on yhtä hyvä kuin paras löydetty tulos, käytetään suurinta naapuruston kokoa.
- Jos olion tulos on korkeintaan kolme siirtoa huonompi kuin paras löydetty tulos, käytetään toiseksi suurinta naapuruston kokoa.
- Tätä pienemmillä tuloksilla käytetään naapuruston kokoa 1.

6.1.2 Populaation mutaatiot

Monissa evoluutioon perustuvissa menetelmissä populaatiossa tapahtuu mutaatioita. Mutaatiot voidaan käsittää pieninä, sattumanvaraisina muutoksina populaation olioihin. Koska Morpion Solitairessa ratkaisut koostuvat kokonaisista poluista ongelman alusta johonkin lehteen ja koska yhden valinnan muuttaminen voi tehdä myöhemmistä valinnoista laittomia, mutaatioiden toteuttaminen ei välttämättä ole yksinkertaista.

Yksinkertainen idea mutaatioiden toteuttamiseksi Morpion Solitairessa saadaan tarkastelemalla Hyyrön ja Porasen [2007] simuloituun jäähdytykseen perustuvaa menetelmää. Hyyrö ja Poranen [2007] esittelevät menetelmässään taaksepäin tehtävien siirtojen mahdollisuuden. Taaksepäin suuntautuvan siirron tekeminen ei rajoitu pelkästään edellisen siirron perumiseen, vaan mikä tahansa siirto, jonka lisäämän pisteen kautta kulkee vain siirto itse, voidaan poistaa. Toisin sanoen, jos siirron pistettä käyttää vain siirto itse, siirto voidaan poistaa rikkomatta pelin sääntöjä.

Nyt populaatiossa voidaan suorittaa mutaatioita evoluutiokierrosten välissä. Kun olio on palautettu sitoutumisasteen tasolle, voidaan tietyllä todennäköisyydellä suorittaa siirto taaksepäin ja valita tämän tilalle uusi sattumanvarainen siirto, jolloin olio säilyttää

sitoutumisasteeseen sidotun pituutensa, mutta voi perua muunkin kuin vain edellisen siirtonsa. Koska yhden taaksepäin suoritetun siirron avulla ei välttämättä saada aikaiseksi suurta muutosta, suuremman vaihtelun saavuttamiseksi mutaation tapahtuessa olio voi suorittaa useammankin siirron taaksepäin ja valita jokaisen poistetun siirron tilalle uuden sattumanvaraisen siirron.

Käytännössä mutaatioista voi olla huomattavasti hyötyä evoluutiomallissa, koska hyvät tulokset leviävät usein laajalle alueelle populaatiossa, jolloin samanlaisia olioita on useita. Mutaation ansiosta voidaan tutkia jälleen uusia hakupolkuja, jotka onnistuessaan voivat luoda populaatioon uusia hyviä lajeja.

6.1.3 Sattumanvaraisuuden vaikutuksen lieventäminen

Juillén [1995] esittelemässä evoluutiomallissa jokainen olio pelaa pelin loppuun sattumanvaraisia siirtoja valiten. Tässä lähestymistavassa ongelmana on ratkaisuavaruuden muoto ja kuten luvussa 5.3 todettiin, hyviä tuloksia on erittäin vaikeaa saavuttaa pelkän satunnaisotannan avulla. Erityisesti haun alussa sitoutumisasteen ollessa pieni, sattumanvaraisin siirroin pelattu peli voi päätyä erittäin huonoihin tuloksiin erittäin helposti, vaikka pelin ensimmäiset siirrot olisivatkin parhaat mahdolliset. Juillé [1995] ehdottaakin tutkimuksessaan, että mallin toimintaa voitaisiin ehkä parantaa, jos jokainen olio hakeutuisi johonkin lokaaliin optimiin rakennusvaiheessa täysin sattumanvaraisten pelien sijaan.

Jos sattumanvaraisuuden vaikutusta voitaisiin lieventää, oliot olisivat todennäköisesti huomattavasti helpommin vertailtavissa. Keskimääräistä useammin parempiin tuloksiin johtavat valinnat säilyisivät todennäköisesti varmemmin populaatiossa sen sijaan, että jokin hyvä tulos korvattaisiin yhden huonon sattumanvaraisen tuloksen takia.

Hyyrön ja Porasen [2007] simuloituun jäähdytykseen perustuvalla menetelmällä, kuten monilla muillakin menetelmillä, voidaan tietyin parametrein saavuttaa kohtalaisen hyviä tuloksia erittäin nopeasti. Luvussa 6.3 tullaan tarkemmin tarkastelemaan olioiden kehittämisen vaikutusta menetelmän toimintaan.

6.1.4 Toteutus

Algoritmissa 7 on esitetty mutaation ja mukautuvan naapuruston pseudokoodiesitykset. Mutaatio taaksepäin suuntautuvien siirtojen avulla voidaan toki helposti toteuttaa monella muullakin tavalla kuin tässä on esitetty. Tässä esitetyssä toteutuksessa, mutaation todennäköisyyden ollessa suuri, olioiden on mahdollista suorittaa useita mutaatiosiirtoja. Mukautuvan naapuruston yhteydessä esitetyt naapurustojen kokojen arvot ja koon muutoksen määrittelevä raja ovat luonnollisesti muutettavissa.

```
1    function mutaatio ( pelitilanne peli )
2      mutaatiot := 0
3      while mutaation todennäköisyystesti onnistuu do
4        siirto := valitse sattumanvarainen siirto taaksepäin
5        peli := peli - siirto
6        mutaatiot := mutaatiot + 1
7      endwhile
8      for i := 1 to mutaatiot do
9        siirto := valitse sattumanvarainen siirto
10       peli := peli + siirto
11     endfor
12     return peli
13   endfunction


14   function mukautuva_naapurusto ( int tulos )
15     naapuruston koko := 1
16     if tulos = paras peli then
17       naapuruston koko := 3
18     else if tulos >= (paras peli - 2) then
19       naapuruston koko := 2
20     endif
21     return naapuruston koko
22   endfunction
```

Algoritmi 7. Mutaatio ja mukautuva naapurusto.

Algoritmissa 8 esitetään monitasoisen evoluutiomallin pseudokoodi, jossa kutsutaan myös algoritmissa 7 esiteltyä mutaatiota sekä mukautuvaa naapuruston kokoa. Algoritmissa 8 esitellyssä menetelmässä tasoja on vain kaksi, mutta myös useamman tason lisääminen onnistuu helposti lisäämällä uusia ylemmän tason while-silmukoita.

Sitoutumisasteen hallintaan käytetty strategia voi myös olla erilainen algoritmin eri tasoilla. Alimmalla tasolla sitoutumisastetta voidaan nostaa esimerkiksi kolmen alimman tason evoluutiokierroksen välein ja ylemmän tason sitoutumisastetta voidaan nostaa jokaisen ylemmän tason kierroksen jälkeen.

Eri tasoille voidaan myös määritellä erilaisia lopetusehtoja. Myöhemmin tullaan esittämään, ettei esimerkiksi alemman tason evoluutiomallia tarvitse välttämättä pelata loppuun. Menetelmän toimintaa voidaan siis nopeuttaa lopettamalla alemman tason haku, vaikka oliot eivät olisikaan vielä pelanneet pelejä täysin loppuun asti.

```
1    ylempi sitoutumisaste := 0
2    paras peli := null

     -- Ylemmän tason evoluutiomalli
3    while jollain oliolla on vielä mahdollisia siirtoja do
       -- Alemman tason evoluutiomalli --
4      alempi sitoutumisaste := ylempi sitoutumisaste
5      while alemman tason lopetusehto ei ole voimassa do

         -- Rakennusvaihe --
6        for each peli in populaatio do
7          peli := mutaatio ( peli )
8          pelaa peli loppuun sattumanvaraisia siirtoja käyttäen tai
           käytä jotain menetelmää lokaalin optimin löytämiseksi
9        endfor

         -- Kilpailuvaihe --
10       for each peli in populaatio do
11         if pelin tulos > paras peli then
12           paras peli := peli
13         endif
14         naapuruston koko := mukautuva_naapurusto ( pelin tulos )
15         for each naapuri in pelin naapurit do
16           if pelin tulos < naapurin tulos then
17                peli := naapuri
18           endif
19         endfor
20       endfor

21       tarkista sitoutumisasteen hallintaan käytetty strategia
22       if tarkistuksen mukaan sitoutumisastetta tulee nostaa then
23         alemman tason sitoutumisaste := alemman tason sa. + 1
24       endif

25       for each olio in populaatio do
26         palauta olio takaisin alemman sitoutumisasteen mittaiseksi
           osaratkaisuksi
27       endfor
28     endwhile

29     tarkista sitoutumisasteen hallintaan käytetty strategia
30     if tarkistuksen mukaan sitoutumisastetta tulee nostaa then
31       ylemmän tason sitoutusaste := ylemmän tason sa. + 1
32     endif
33     for each olio in populaatio do
34       palauta olio takaisin ylemmän sitoutumisasteen mittaiseksi
         osaratkaisuksi
35     endfor
36   endwhile
```

Algoritmi 8. Monitasoinen evoluutiomalli.

6.2 Monitasoisen evoluutiomallin muuttujat

Monitasoisessa evoluutiomallissa on useita muuttujia, jotka vaikuttavat haun kestoon sekä tuloksen laatuun. Koska malli perustuu vahvasti Juillén [1995] evoluutiomalliin, malliin vaikuttavat samat muuttujat kuin Juillén evoluutiomallissa. Juillén evoluutiomallista muuttujiksi saadaan populaation koko, naapuruston koko sekä sitoutumisasteen hallinta, joiden toimintaa on jo aikaisemmin esitelty tarkemmin Juillén [1995] evoluutiomallin yhteydessä luvussa 4.3.

Monitasoisessa evoluutiomallissa on näiden lisäksi muutamia uusia muuttujia, joilla voidaan huomattavasti vaikuttaa mallin suoritustehokkuuteen ja löydettyjen tulosten laatuun. Uusia muuttujia ovat populaation kerroin, sitoutumisasteen suurin arvo alimman tason peleissä sekä ylemmän tason evoluutiokierrosten määrä.

Näiden lisäksi malliin voidaan vaikuttaa luvussa 6.1 mainituilla mutaatioilla, mukautuvalla naapuruston koolla sekä kehittämällä olioita rakennusvaiheessa tavallisen satunnaisotannan sijaan.

Populaation kertoimella tarkoitetaan arvoa, jolla pyritään simuloimaan suurempaa populaatiota kuin todellisuudessa käytetään. Tämä tapahtuu korvaamalla jokainen olio k x k - kokoisella minipopulaatiolla, jonka paras olio valitaan edustamaan tätä minipopulaatiota. Käytännössä tämä tapahtuu pelaamalla $X = k^2$ kappaletta sattumanvaraisista siirroista koostuvaa peliä loppuun asti annetun olion sen hetkisestä tilanteesta. Tämän jälkeen valitaan näistä peleistä parhaan tuloksen saavuttanut korvaamaan varsinaisessa populaatiossa sijaitseva olio.

Voidaan helposti huomata, että laskettaessa minipopulaatioiden sattumanvaraiset pelit mukaan varsinaisen populaation kokoon, populaation koko olisi $(n * k)$ x $(n * k)$, jossa n on varsinaisen populaatiotaulukon sivun pituus. Toisaalta voidaan todeta, että sattumanvaraisten pelien pelaaminen ja näistä parhaan valitseminen ei oikeasti ole verrattavissa evoluutiomallin populaation toimintaan, koska hylättyjen sattumanvaraisten pelien keräämä tieto menetetään heti. Menetelmää voidaan kuitenkin käyttää nopeampana ja muistivaatimuksia lieventävänä tapana yrittää simuloida suurempaa populaatiota, kuin esimerkiksi muistin rajallisuuden takia olisi muuten mahdollista.

Myös Juillé [1999] päätyi samankaltaiseen ratkaisuun SAGE-menetelmässään. Juillé ehdotti tutkielmassaan [1999] populaation olioille toistettua satunnaisotantaa, joka on käytännössä sama asia kuin tässä esitelty populaation kerroin.

Sitoutumisasteen suurimmalla arvolla tarkoitetaan erillisen lopetusehdon määrittelemistä alemman tason evoluutiomalleissa asettamalla sitoutumisasteen arvolle yläraja. Hieman populaation koosta riippuen, evoluutiomallin hakua ei välttämättä

tarvitse suorittaa loppuun asti. Esimerkiksi rajoittamalla sitoutumisasteen suurin arvo vaikkapa arvoksi 55, voidaan saada huomattavia eroja suoritustehokkuudessa pienellä kustannuksella tuloksien laadussa. Tämä perustuu todennäköisesti siihen, että populaation ja naapuruston kokojen ollessa riittävän suuria, hyviksi todetut oliot valtaavat suuren osan populaatiosta ja ne ehditään tutkia tarpeeksi tarkasti läpi ilman evoluutiomallin loppuun suorittamista. Evoluutiomallin loppuun suorittamisella tarkoitetaan tässä siis sitoutumisasteen kasvattamista, kunnes kaikki oliot ovat saavuttaneet tilan, jossa mahdollisia uusia siirtoja ei enää ole.

Ylemmän tason evoluutiokierrosten määrä voidaan ymmärtää täsmälleen samalla tavalla kuin yllä esitelty sitoutumisasteen suurin arvo. Tässä tapauksessa tarkoitetaan kuitenkin *ylemmän tason* sitoutumisasteen suurinta arvoa. Myöhemmin tullaan esittämään, ettei ylemmän tason mallia tarvitse suorittaa loppuun asti, koska suuremmilla populaation arvoilla hakuavaruus voidaan olettaa hyvin tutkituksi jo pienemmälläkin kierrosten määrällä ja sitoutumisasteen nostamisesta ei näin ollen enää hyödytä.

Seuraavissa testeissä pyritään selventämään muuttujien vaikutuksia mallin toimintaan ja näin selvittää, millä asetuksilla laajempaa etsintää kannattaisi mahdollisesti yrittää.

6.3 Monitasoisen evoluutiomallin muuttujien testaus

Tässä luvussa pyritään testaamaan monitasoiseen evoluutiomalliin vaikuttavia muuttujia. Testit on suoritettu $G_4(A_4)$-mallissa eli tavallisessa irrallisessa 5D mallissa, koska mallin ratkaisuavaruus on koskevan mallin ratkaisuavaruutta pienempi ja näin ollen nopeampi testattava. Testit suoritettiin Intel Quad Core Q6600 koneella, jossa on käytössä 3 GB muistia ja käyttöjärjestelmänä Windows Vista™ Ultimate SP2. Testeissä käytetty ohjelma toteutettiin Javalla ja käännettiin Javac versiolla 6.0.13. Testit ajettiin Java Runtime Environment versiolla 6.0.20. Ohjelmassa pyrittiin käyttämään hyväksi evoluutiomallin rinnakkaistettavuutta jakamalla olioiden suorittaminen koneen neljälle käytössä olevalle prosessorille.

6.3.1 Populaation koko

Tutkitaan ensimmäiseksi populaation koon vaikutusta mallin suoritusaikoihin ja tuloksen laatuun. Taulukosta 6 voidaan nähdä eri populaatioiden kokojen vaikutukset tuloksiin ja suoritusaikoihin. Taulukon 6 testeissä alimman tason evoluutiomallit pelattiin loppuun asti ja ylemmällä tasolla pelattiin 50 kierrosta nostamalla sitoutumisastetta yhdellä jokaista kierrosta kohti, joten suoritusajat ovat lähes suurimmat mahdolliset annetuille populaation koon arvoille. Testit toistettiin 20 kertaa jokaisella populaation koon arvolla sattumanvaraisuuden vaikutuksen minimoimiseksi.

Populaation koko n tarkoittaa tässä siis $n \times n$ - oliomatriisia. Esimerkiksi, jos populaation koko on 8, niin populaatiossa on $8^2 = 64$ oliota.

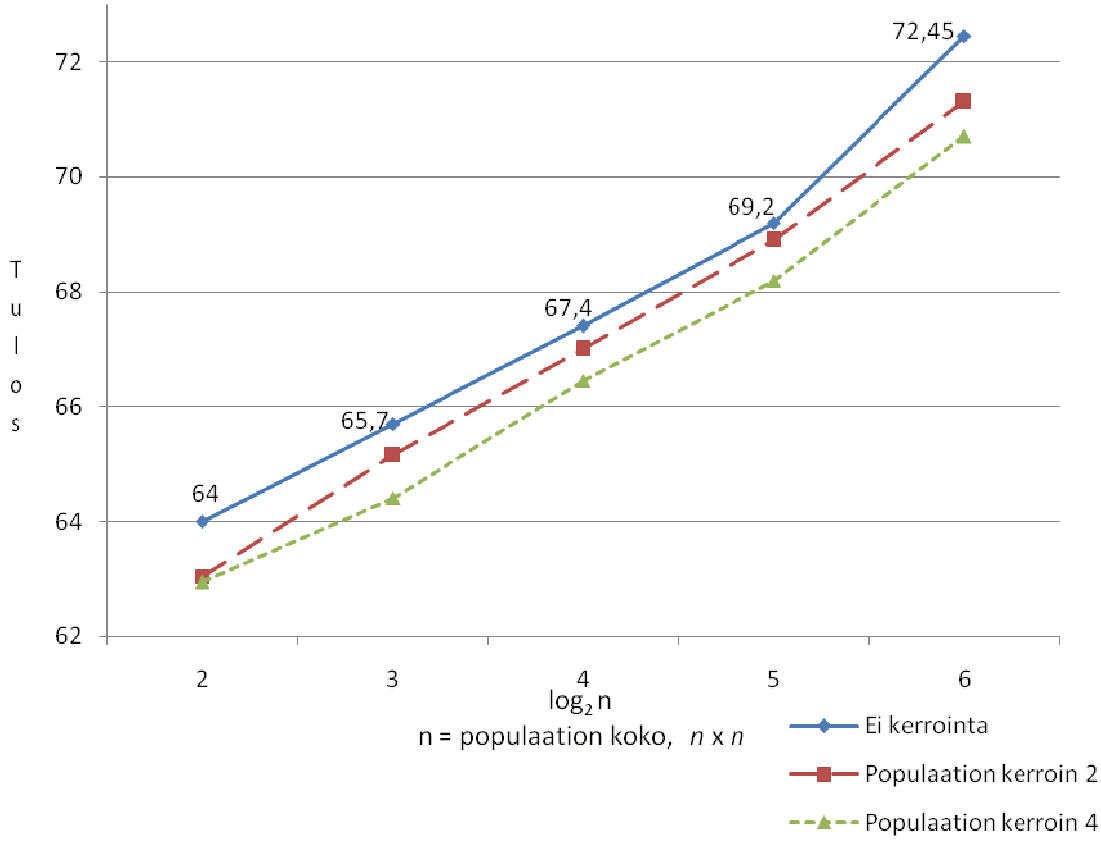
Populaation koko n	Toistojen määrä	Tulosten keskiarvo	Suoritusaika sekunteina	Paras tulos	Otoksesta laskettu keskihajonta
4	20	64	9,8	65	0,86
8	20	65,7	29,5	67	1,25
16	20	67,4	118,4	72	1,67
32	20	69,2	424	73	1,77
64	20	72,5	1325	74	1,64

Taulukko 6. Suoritusajat ja tulokset populaation koon eri arvoilla.

Tuloksista voidaan nähdä selvä riippuvuus tuloksen hyvyyden ja populaation koon välillä. Kuten kuvasta 9 voidaan nähdä, tulosten laatu näyttäisi kasvavan lähes lineaarisesti populaation koon kaksikantaiseen logaritmiin verrattuna. Yllättävänä huomiona voidaan todeta, että populaation koon vaikutus tulosten hyvyyteen näyttäisi itse asiassa kasvavan entistä enemmän populaation koon kasvaessa. Populaation koon arvon kaksinkertaistaminen nelinkertaistaa olioiden määrän, mikä voidaan nähdä myös taulukossa 6 esitetyistä suoritusajoista.

Otoksesta lasketusta keskihajonnasta voidaan nähdä, että saavutetut tulokset poikkesivat jonkin verran keskiarvosta, mutta populaation koon kasvaessa tämä poikkeama pysyy jotakuinkin samansuuruisena. Suurempi hajonta voi olla sekä toivottu että huono ominaisuus. Toisaalta suurempi hajonta voi tarkoittaa, että sattumalta saatetaan eksyä myös huomattavasti keskimääräistä parempiin tuloksiin, mutta toisaalta siitä voidaan myös nähdä, että menetelmällä ei välttämättä aina saavuteta hyviä tuloksia.

Kokeillaan seuraavaksi populaation kokojen simulointia eri populaation kertoimien avulla. Simuloitujen populaation kokojen testit suoritettiin samoilla asetuksilla kuin tavallisten populaation kokojen testit, mutta tällä kertaa populaation kokoa pyrittiin simuloimaan nostamalla populaation kerrointa luvussa 6.2 esitellyllä tavalla. Taulukossa 7 esitetyissä tuloksissa populaation koon merkintä $n \times k$ tarkoittaa $n \times n$-kokoista populaatiota, jossa jokainen populaation olio on korvattu $k \times k$ -kokoisella minipopulaatiolla, jonka paras peli korvaa olion varsinaisessa populaatiossa. Kertoimilla on pyritty simuloimaan samankokoisia populaatioita kuin edellisessä testissä.



Kuva 9. Monitasoisen evoluutiomallin tulokset populaation koon ja kertoimien eri arvoilla.

Populaation koko n x k	Toistojen määrä	Tulosten keskiarvo	Suoritusaika sekunteina	Paras tulos	Otoksesta laskettu keskihajonta
2x2 = 4	20	63,1	10,2	65	1
4x2 = 8	20	65,2	28,8	67	0,88
8x2 = 16	20	67	87,3	70	1,3
16x2 = 32	20	68,9	372	72	1,33
32x2 = 64	20	71,3	1173	74	1,34
1x4 = 4	20	62,9	9,3	65	0,89
2x4 = 8	20	64,4	24,6	67	1,1
4x4 = 16	20	66,5	70,7	70	1,7
8x4 = 32	20	68,2	263	72	1,67
16x4 = 64	20	70,7	864	74	1,72

Taulukko 7. Suoritusajat ja tulokset käyttäen simuloituja populaation kokoja.

Tässä vaiheessa kannattaa huomata, että molemmissa simuloiduissa ja tavallisen populaation koon testeissä pelataan siis täsmälleen sama määrä satunnaisia pelejä. Erona menetelmissä on muistiin tallennetun tiedon määrä. Simuloitujen populaation kokojen kohdalla tietoa tallennetaan vähemmän, koska varsinainen populaatio on

pienempi. Kuten myös kuvasta 9 voidaan helposti huomata, ainakin näin pienillä populaation koon arvoilla simuloitujen populaation kokojen testien tulokset vastaavat yllättävän hyvin samankokoisen tavallisen populaation tuloksia. Myös suoritusaikojen voidaan huomata parantuneen lievästi simulointia käytettäessä kertoimella 2 ja huomattavasti kertoimella 4. Toisaalta voidaan helposti nähdä, että samankokoinen varsinainen populaatio päätyy kuitenkin lopulta selvästi parempiin tuloksiin.

Tuloksista voidaan siis päätellä, että hyväksi käytetyn tiedon määrä vaikuttaa lopullisiin tuloksiin. Populaation koon simuloiminen minipopulaation parhaan olion valitsemalla vastaa käytännössä toistettua satunnaisotantaa. Esimerkiksi Cazenaven [2007, 2009] Monte Carlo -hauissa käytetään vastaavaa valintamenetelmää, joissa ainoastaan paras tulos selviää jatkoon. Tässä esitellyistä tuloksista voidaan kuitenkin päätellä, että näin päädytään itse asiassa huonompiin tuloksiin kuin siinä tapauksessa, että kaikki kerätty tieto säilytettäisiin myöhempää käyttöä varten.

Toisaalta tästä voidaan myös alustavasti huomata, että kehittämällä olioita ennen kilpailuvaihetta toistetun satunnaisotannan avulla voidaan saavuttaa huomattavasti parempia tuloksia kuin samankokoisella populaatiolla normaalisti. Toisin sanoen, minipopulaatioissa suoritettu toistettu satunnaisotanta on yksinkertainen tapa kehittää yksittäistä oliota. Tuloksista päätellen, näin kehitetyt oliot saavuttavat lähes yhtä hyviä tuloksia kuin saavutettiin moninkertaistamalla populaation olioiden määrä. Tästä voidaan siis päätellä, että olioiden kehittämisestä yksinkertaisen satunnaisotannan sijaan saatettaisiin hyötyä huomattavasti.

Otosten keskihajonta pysyi suurin piirtein samana kuin tavallisen populaation tapauksessa. Jostain syystä kertoimella 2 saavutettujen tulosten hajonta oli kuitenkin toistuvasti alhaisempi kuin ilman kerrointa tai kertoimella 4 saavutettujen tulosten hajonta.

Populaation kertoimella saaduista testien tuloksista voidaan päätellä, että suuremmilla populaation arvoilla jonkin asteisesta populaation kertoimesta voi olla huomattavasti hyötyä, koska tällöin huonoihin tuloksiin päätyneet sattumanvaraiset pelit korvataan jo minipopulaatioissa. Toisaalta liian suuresta kertoimesta voi olla myös haittaa. Kertoimen kasvattaminen kasvattaa suoritusaikaa lähes samassa suhteessa kuin populaation koon nostaminen. Sattumanvaraisista peleistä kerättyä tietoa menetetään kuitenkin minipopulaation koon kasvaessa enemmän, koska näistä peleistä vain yksi selviää edustamaan oliota ja tästä johtuen saatu hyöty näyttäisi tuloksiin perustuen laskevan kertoimen kasvaessa.

6.3.2 Sitoutumisasteen ja kierrosten määrän rajoittaminen

Tarkastellaan seuraavaksi sitoutumisasteen rajoittamisen vaikutusta mallin toimintaan. Sitoutumisasteen rajoittamisella tarkoitetaan siis alimman tason evoluutiokierrosten määrän rajoittamista. Taulukossa 8 on esitetty eri sitoutumisasteen rajoituksien vaikutukset mallin toimintaan. Taulukossa 8 esitetyt tulokset on saavutettu käyttämällä 32 x 32 olion populaatiota ja pelaamalla 50 ylemmän tason evoluutiokierrosta. Taulukossa on myös vertailun vuoksi ilman rajoitusta pelatuista peleistä saadut tulokset. Ilman rajoitusta evoluutiokierroksia pelataan, kunnes jokainen populaation olio on saavuttanut tilan, jossa siirtoja ei ole enää mahdollista tehdä.

Sitoutumisasteen rajoitus	Toistojen määrä	Tulosten keskiarvo	Suoritusaika sekunteina	Paras tulos
45	20	67,1	198	70
55	20	69,7	314	72
65	20	69,1	398	72
Rajoittamaton	20	69,2	424	73

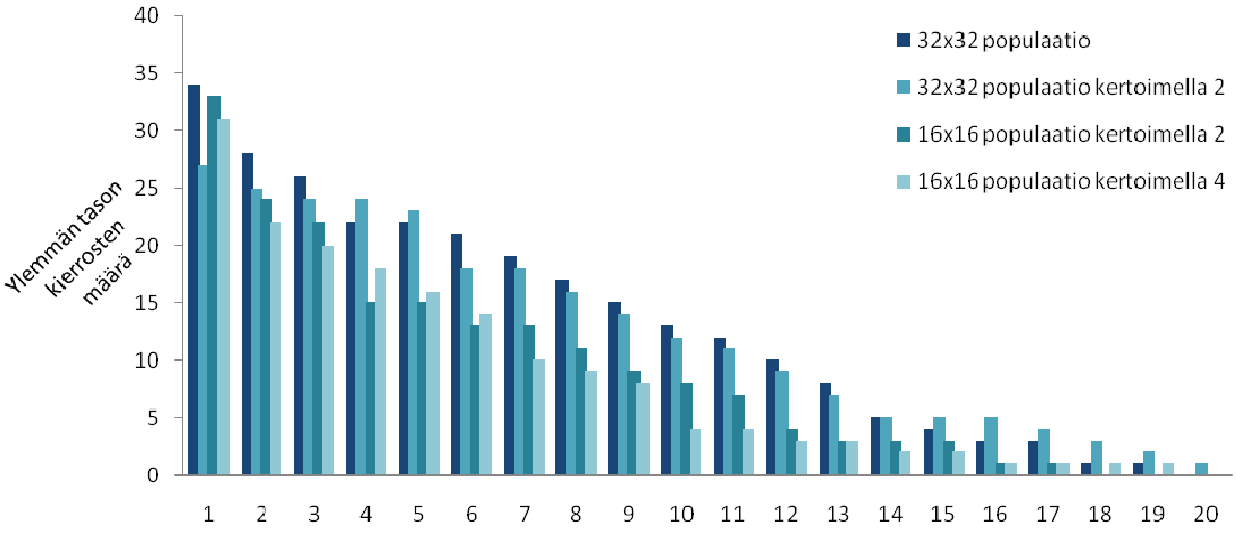
Taulukko 8. Alemman tason evoluutiokierrosten määrän vaikutus monitasoisen evoluutiomallin toimintaan.

Taulukossa 8 esitettyihin tuloksiin perustuen rajoituksen arvo 45 voidaan todeta liian alhaiseksi saatujen tulosten laadusta päätellen. Voidaan kuitenkin huomata, että rajoittamalla alimman tason peleissä sitoutumisasteen suurimmaksi arvoksi 55 tai 65, tuloksissa ei huomata juurikaan eroa ilman rajoitusta saatuihin tuloksiin. Erityisesti huomataan, että sitoutumisasteen rajoittamisen arvolla 65 saadut tulokset ovat lähes identtisiä ilman rajoittamista saatuihin tuloksiin. Ilman rajoitusta saavutettu paras tulos, 73 siirtoa, voidaan myös todennäköisesti laskea onnekkaaksi sattumaksi, koska rajoituksen arvolla 55 saatujen tulosten keskiarvo on huomattavasti parempi. Rajoituksen arvolla 55 voidaan myös huomata selvä ero suoritukseen tarvittavassa ajassa.

Tuloksista voidaan todeta, että sopiva sitoutumisasteen rajoituksen arvo on todennäköisesti välillä 55 – 65. Koska evoluutiomallin alimmalla tasolla loppuun pelaamisesta saatu hyöty näyttäisi olevan lähes mitätön, voidaan näin ollen sitoutumisastetta rajoittamalla lisätä mallin suoritustehokkuutta.

Tarkastellaan seuraavaksi ylemmän tason evoluutiomallin kierrosten rajoittamista. Samoin kuin alemman tason evoluutiomallissa, ylemmän tason evoluutiomallia ei välttämättä tarvitse pelata loppuun asti. Jos ylemmän tason mallin kierrosten lukumäärää voidaan rajoittaa, koko menetelmän suoritustehokkuutta voidaan lisätä huomattavasti.

Kuvassa 10 tarkastellaan tarkemmin neljän jo edellä esitellyn menetelmän saavuttamia tuloksia 20 toistolla. Kuvassa on esitetty laskevassa järjestyksessä, millä kierroksella menetelmä saavutti kyseisellä toistokerralla löytyneen parhaan tuloksen. Taulukossa 9 on puolestaan esitetty kuvasta 10 laskettuja tunnuslukuja, kuten keskiarvo ja keskihajonta.



Kuva 10. Parhaan tuloksen löytämiseksi tarvitut ylemmän tason kierrosten lukumäärät laskevassa järjestyksessä.

Menetelmä	Ylemmän tason kierrosten määrän keskiarvo	Keskihajonta otoksesta	Suurin kierrosten lukumäärä
32 x 32 populaatio	13,2	10,14	34
32 x 32 populaatio kertoimella 2	12,7	8,69	27
16 x 16 populaatio kertoimella 2	9,25	9,14	33
16 x 16 populaatio kertoimella 4	8,5	8,83	31

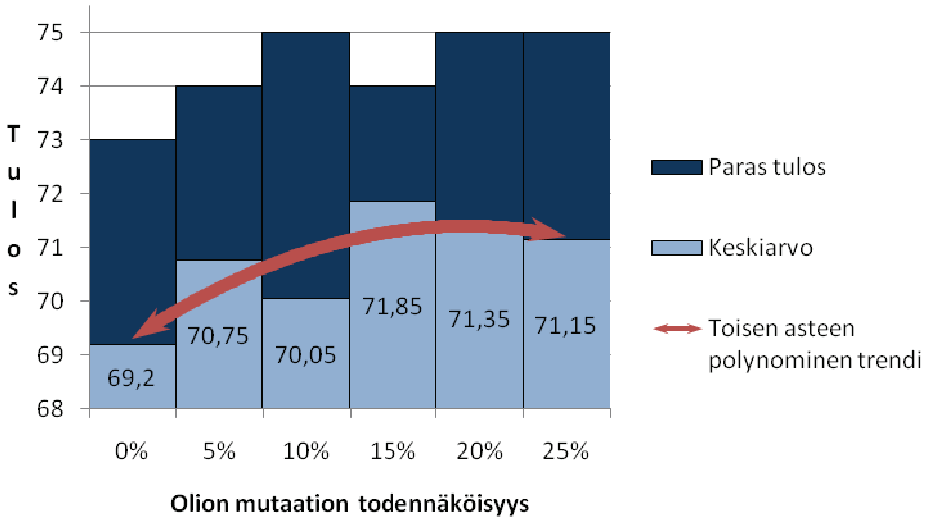
Taulukko 9. Ylemmän tason kierrosten lukumääristä laskettuja tunnuslukuja.

Kuten kierrosten määrän keskiarvosta ja suurimmasta tarvitusta kierrosten lukumäärästä voidaan päätellä, ylemmän tason evoluutiomallia ei tarvitse pelata kokonaan loppuun parhaan tuloksen löytämiseksi. Normaalijakauman tapauksessa tiedämme, että noin 95,45 % todennäköisyysmassasta on kahden hajonnan päässä ja noin 99,73 % on kolmen hajonnan päässä keskiarvosta. Tarvittujen kierrosten lukumäärien ei tietenkään voida olettaa noudattavan tarkalleen normaalijakaumaa, mutta kuvasta 10 voidaan päätellä yllä olevien arvojen pitävän suurin piirtein paikkansa myös tässä tapauksessa. Koska keskihajonta on noin 10 jokaisessa edellä olevista neljästä tapauksesta, voidaan sopiva ylemmän tason kierrosten lukumäärä valita noin 1 - 3 hajonnan päästä keskiarvosta. Pienemmällä arvolla voidaan lisätä suoritustehokkuutta huomattavasti,

mutta suuremmalla arvolla voidaan varmistaa, ettei hyviä tuloksia vahingossa menetetä lopettamalla hakua liian aikaisin. Sopivan ylemmän tason kierrosten lukumäärän voidaan näin ollen olettaa olevan välillä 25 - 40. Tämä tulos tukee myös luvussa 5.2 esitettyä tulosta, että loppupeli on huomattavasti pelin alkua helpompi.

6.3.3 Mutaatiot ja mukautuva naapurusto

Tarkastellaan ensimmäiseksi luvussa 6.1 esitellyn mutaation vaikutuksia menetelmän toimintaan. Kuvassa 11 on esitetty mutaatiolla saavutetut tulokset mutaation eri todennäköisyyksillä. Testi toistettiin jokaisella mutaation todennäköisyydellä 20 kertaa. Testit suoritettiin 32 x 32 olion populaatiossa naapuruston koon ollessa 1. Kuvan 11 vasemmassa reunassa oleva 0 % todennäköisyys vastaa siis tilannetta, jossa mutaatiota ei käytetty ollenkaan.



Kuva 11. Mutaation vaikutukset monitasoiseen evoluutiomalliin.

Kuvasta voidaan nähdä, että jo pienellä mutaation todennäköisyydellä menetelmällä saavutetaan huomattavasti parempia tuloksia. Keskimääräinen paras saavutettu tulos kasvoi joissain tapauksissa jopa kahdella ilman mutaatiota suoritettuun testiin verrattuna. Trendiviivasta voidaan nähdä, että ainakin testituloksien perusteella paras mutaation todennäköisyys on todennäköisesti välillä 15 - 20 %. Tämän jälkeen saatujen tulosten keskimääräinen hyvyys näyttäisi taas kääntyvän laskuun. Tulosten laadun lasku suuremmilla todennäköisyyksillä johtuu luultavasti siitä, että liian suurella mutaation todennäköisyydellä hyvät tulokset alkavat muuttumaan jo ennen kuin ne ovat ehtineet kunnolla levitä populaatioon.

Tuloksien perusteella voidaan todeta, että Hyyrön ja Porasen [2007] esittelemiin taaksepäin suuntautuviin siirtoihin perustuvan mutaation avulla voidaan saavuttaa hyviä tuloksia. Koska mutaation käyttämisen vaikutus suoritustehokkuuteen on minimaalinen,

voidaan mutaatiota käyttämällä lähes samassa ajassa saada huomattavasti parempia tuloksia kuin ilman mutaatiota. Toisaalta, tässä esiteltyihin tuloksiin perustuen on vaikeaa sanoa, miten mutaatio käyttäytyy eri populaation kokojen arvoilla ja vaikuttaako esimerkiksi mukautuvan naapuruston koon käyttäminen siihen, kuinka suuri mutaation todennäköisyys kannattaa valita.

Mukautuvaa naapuruston kokoa on hieman vaikeaa testata luotettavasti. Suuremmat naapuruston koot vaativat huomattavasti suuremman populaation toimiakseen hyvin, mutta suuremmalla populaation koolla testien toistaminen useaan kertaan on ongelmallista. Mukautuvaa naapuruston kokoa testattiin 64 x 64 olion populaatiossa. Testi toistettiin 20 kertaa. Testien tulosten keskiarvoksi saatiin 72,8, jota verrattiin naapuruston koon vakioarvolla 1 saatuihin tuloksiin, joiden keskiarvoksi saatiin 72,45. Tämän perusteella voidaan sanoa, että mukautuvasta naapuruston koosta ei ainakaan ole haittaa. Toisaalta, näin pienellä populaation koolla ja otoskoon ollessa näin pieni, saaduista tuloksista ei voida varmuudella havaita myöskään selvää hyötyä menetelmän käyttämisestä. Intuitiivisesti voidaan kuitenkin olettaa, että suuremmilla populaation koon arvoilla mukautuvasta naapuruston koosta voitaisiin hyötyä, koska tällöin hyvät tulokset leviävät nopeammin suuremmalle alueelle populaatioon. Erityisesti voidaan todeta, että hyvän tuloksen saavuttamisen ollessa vaikeaa, kuten Morpion Solitairessa, hyvän tuloksen löytyessä tätä todennäköisesti kannattaa pyrkiä levittämään nopeasti laajemmalle alueelle.

6.3.4 Olioiden kehittäminen ennen kilpailutusta

Alkuperäisessä evoluutiomallissa jokainen olio pelaa pelin loppuun käyttäen sattumanvaraisia siirtoja. Tarkastellaan, voidaanko olioita helposti kehittää ennen kilpailuvaihetta.

Suuri osa Morpion Solitaireen sovelletuista menetelmistä perustuu satunnaisotantaan. Toisin sanoen, olion kehittäminen vaatisi useimmissa tapauksissa usean satunnaisen pelin pelaamisen. Jokainen olio pelaa jokaisella evoluutiokierroksella jo yhden sattumanvaraisen pelin, joten useiden satunnaisten pelien pelaaminen moninkertaistaa menetelmän aikavaatimuksen. Voidaan siis helposti huomata, että kovinkaan monimutkaista kehittämistä olioille ei voida suorittaa.

Hyyrön ja Porasen [2007] esittelemän simuloidun jäähdytyksen toiminta ei kuitenkaan perustu toistettuun satunnaisotantaan. Tästä johtuen menetelmää voidaankin helposti soveltaa olioiden kehittämiseksi ilman liian suurta vaikutusta suoritustehokkuuteen.

Taulukossa 10 on esitetty täysin sattumanvaraisten pelien ja simuloidulla jäähdytyksellä pelattujen pelien tuloksia. Simuloitua jäähdytystä kokeiltiin useilla eri lämpötiloilla aloitus ja lopetus lämpötilojen ollessa aina määriteltynä samaksi arvoksi.

Menetelmä	Toistojen määrä	Pelien / Siirtojen määrä	Pelien parhaiden tulosten keskiarvo	Paras tulos
Toistettu satunnaisotanta	20	4	53,7	57
	20	8	56,4	59
	20	16	56,9	60
	20	32	58,1	61
Simuloitu jäähdytys	20	200	54,3	59
	20	400	56,4	59
	20	800	57,4	60
	20	1600	58,7	62

Taulukko 10. Sattumanvaraiset pelit ja simuloitu jäähdytys.

Koska yhdessä pelissä näyttäisi näin pienillä arvoilla olevan noin 50 - 60 siirtoa, taulukossa 10 esitetyt satunnaisten pelien määrät vastaavat tehtyjen siirtojen määrältään simuloidun jäähdytyksen testaamisessa käytettyjä siirtojen määriä. Näin pienillä parametrien arvoilla satunnaisotannan ja simuloidun jäähdytyksen välillä ei näyttäisi olevan suurta eroa, mutta simuloidulla jäähdytyksellä saavutetut tulokset näyttäisivät kuitenkin keskimäärin olevan hieman pelkkää toistettua satunnaisotantaa parempia.

Voidaan myös todeta, että tässä esitettyjä suurempia arvoja ei ehkä kannata käyttää. Populaation kertoimen yhteydessä esitettyihin tuloksiin perustuen voidaan todeta, että 16 satunnaisen pelin tai 800 siirron tekeminen lähes kahdeksankertaistaa menetelmän suoritusajan. Arvoja voidaan toki nostaa, mutta tällöin joudutaan todennäköisesti kompensoimaan pienentämällä esimerkiksi populaation kokoa, jolloin oikeastaan siirrytään poispäin monitasoisen evoluutiomallin perusideasta.

Koska tehtyjen siirtojen tai pelien määrää ei juurikaan voida nostaa taulukossa 10 esitetyistä arvoista, voidaan populaation kertoimen olettaa toimivan riittävänä menetelmänä olioiden kehittämiseksi. Voidaan kuitenkin nähdä, että simuloitua jäähdytystä käyttämällä saatettaisiin mahdollisesti päästä hieman parempiin tuloksiin.

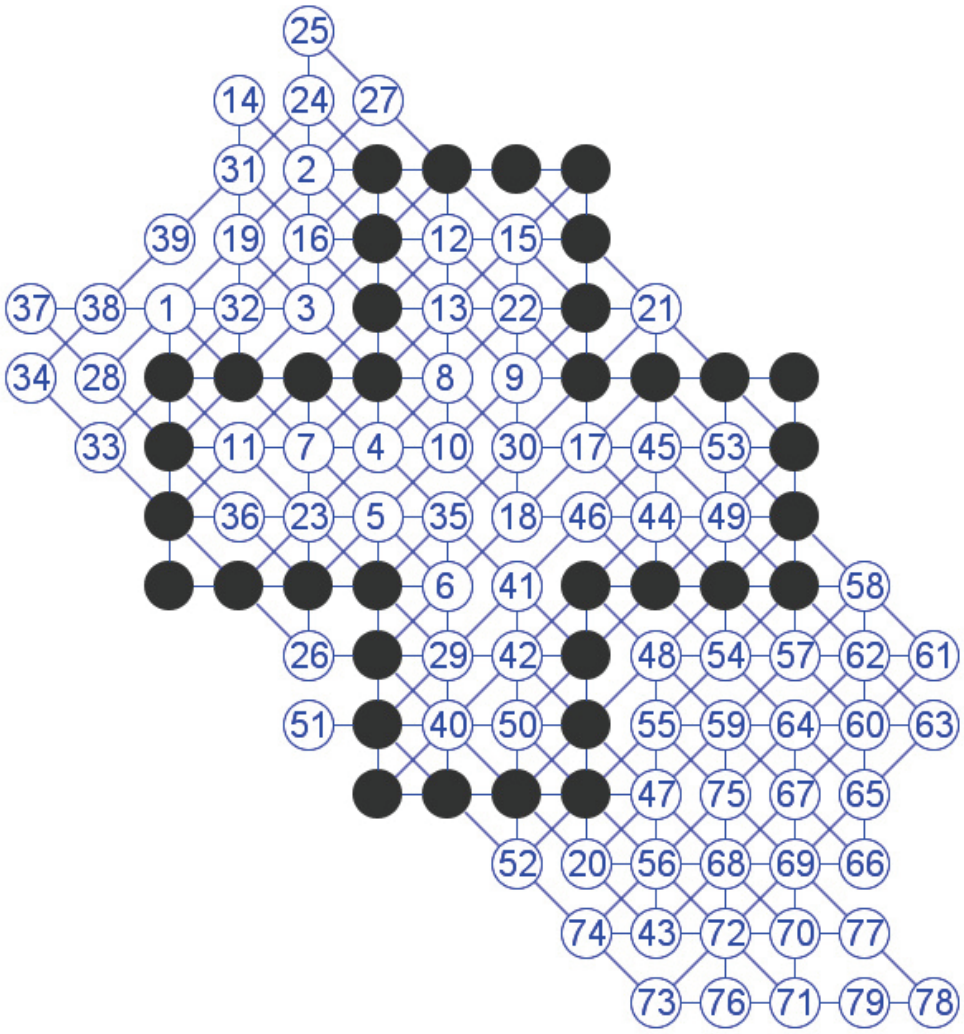
6.4 Monitasoisen evoluutiomallin tulokset

Monitasoisella evoluutiomallilla saavutettiin 79 siirron tulos irrallisessa 5D mallissa ja 141 siirron tulos koskevassa 5T mallissa.

Irrallisessa mallissa löydettiin myös useita 76, 77 ja 78 siirron tuloksia useilla menetelmän eri parametreilla. Erityisesti löydettiin useita toisistaan poikkeavia 78 siirron tuloksia, joista jokainen löytyi noin 2 - 6 tunnin suorituksen jälkeen kyseisellä menetelmän suorituskerralla. Kuvassa 12 on esitetty menetelmän löytämä paras tulos, 79 siirtoa. Kuvassa esitetty tulos löydettiin kokonaisuudessaan noin 36 tuntia

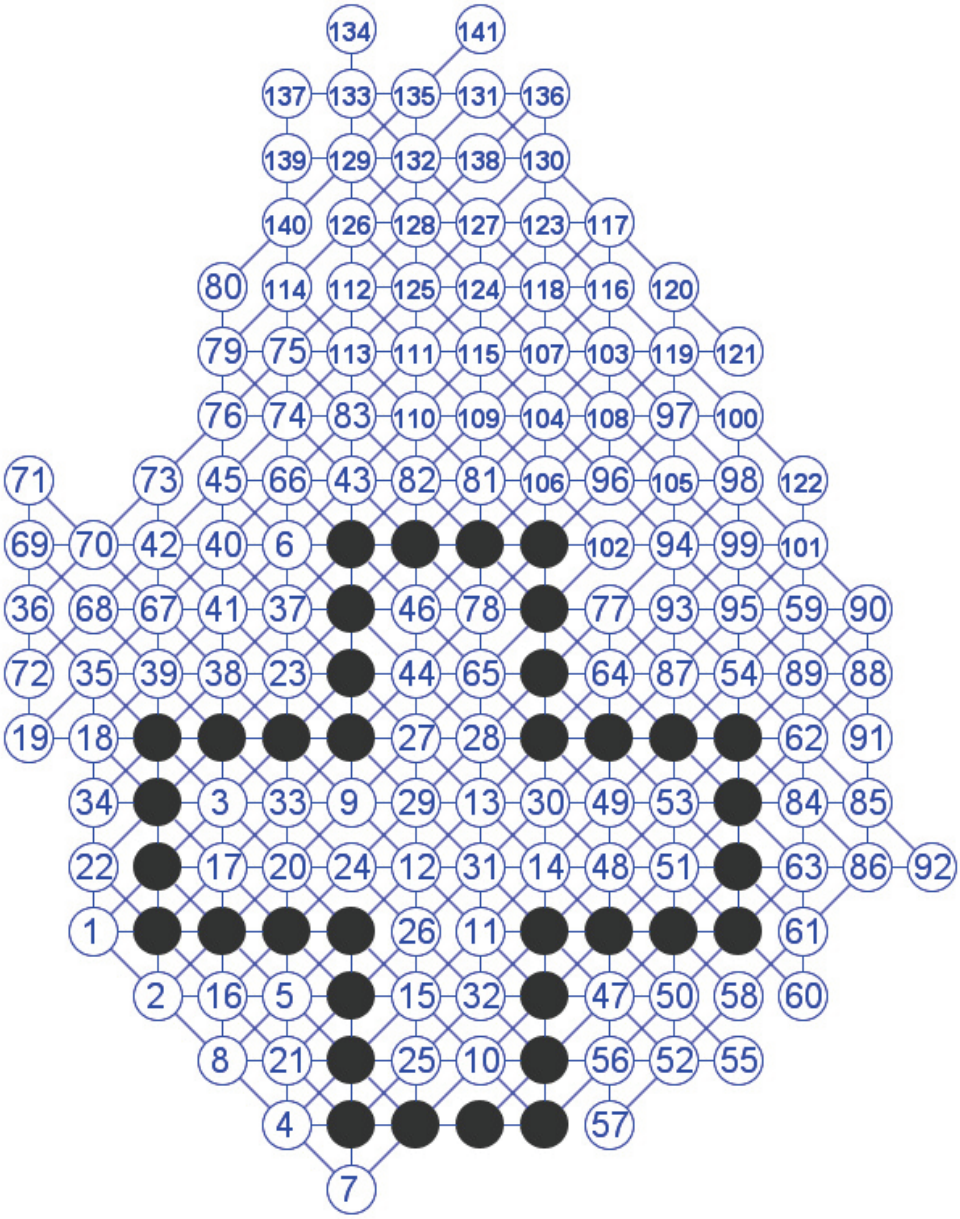
kestäneellä haulla, jossa paras tulos löytyi 23 tunnin ja 12 minuutin kohdalla. Kuvan tuloksen löytäneessä haussa käytetty populaation koko oli 160 x 160 populaation kertoimella 3. Haussa käytettiin mukautuvaa naapuruston kokoa, jossa parasta tulosta vastaavat tulokset käyttivät naapuruston kokoa 4, korkeintaan yhtä siirtoa huonommat tulokset käyttivät kokoa 3, korkeintaan kolmea siirtoa huonommat käyttivät kokoa 2 ja loput arvoa 1.

Ylemmällä tasolla evoluutiokierroksia pelattiin 40 ja jokaisella alemmalla tasolla kierroksia pelattiin kunnes sitoutumisaste saavutti arvon 65. Jokainen alemman tason evoluutiokierros toistettiin kolme kertaa ennen sitoutumisasteen nostamista. Olioiden mutaation todennäköisyys oli 17,5 %.



Kuva 12. Paras monitasoisella evoluutiomallilla löydetty irrallisen mallin tulos.

Koskevan mallin tulos saavutettiin noin 13 tuntia kestäneellä haulla, jossa paras tulos löydettiin 7 tunnin ja 16 minuutin kohdalla. Koskevan mallin haussa käytettiin 128 x 128 populaatiota kertoimella 2 ja mukautuvaa naapuruston kokoa yllä esitetyllä tavalla. Ylemmän tason kierroksia pelattiin 60 ja alemman tason kierroksia aina sitoutumisasteen arvoon 100 asti. Mutaation todennäköisyys tässä haussa oli myös 17,5 %. Koskevan mallin testaamiseen käytetyt parametrit vastaavat jotakuinkin irrallisen mallin 78 siirron tuloksiin päätyneiden testien parametreja suurempia kierrosten lukumääriä lukuun ottamatta. Paras löydetty koskevan mallin tulos on esitetty kuvassa 13.



Kuva 13. Paras monitasoisella evoluutiomallilla löydetty koskevan mallin tulos.

Menetelmällä saavutettiin hyvä 79 siirron tulos ja useita 78 siirron tuloksia irrallisessa mallissa sekä kohtalaisen hyvä 141 siirron tulos koskevassa mallissa. Parhaista aikaisemmin löydetyistä tuloksista jäätiin kummassakin mallissa jonkin verran.

Menetelmästä voidaan kuitenkin sanoa, että se kykenee löytämään hyviä tuloksia muihin aikaisemmin esiteltyihin menetelmiin verrattavassa ajassa. Esimerkiksi Cazenaven [2007] löytämän 78 siirron tuloksen etsintä vei lähes 400 000 sekuntia eli yli neljä vuorokautta, joten monitasoisella evoluutiomallilla 2 - 6 tunnissa löydetyt 78 siirron tulokset ja alle 24 tunnissa löydetty 79 siirron tulos ovat erittäin rohkaisevia tuloksia. Toisaalta voidaan todeta, että tässä tutkielmassa testeihin käytetty kone on suoritusteholtaan huomattavasti Cazenaven [2007] testeissään käyttämää konetta tehokkaampi, joten suoritusajat eivät selvästikään ole suoraan toisiinsa verrattavissa.

Laajemman haun määrittelevillä parametreilla menetelmän voidaan olettaa kykenevän löytämään tässä esitettyjä parempia tuloksia, erityisesti koskevassa mallissa, jossa erittäin laajaa hakua ei ehditty suorittaa. Tämän lisäksi menetelmän toiminta on helposti rinnakkaistettavissa, jonka ansiosta vaativammilla parametreilla suoritettava ajo voidaan helposti jakaa useamman prosessorin tai tietokoneen suoritettavaksi.

Huonona puolena monitasoisessa evoluutiomallissa verrattuna muihin menetelmiin voidaan huomata erilaisten muuttujien suuri määrä. Muuttujien eri arvojen vaikutuksia toisiinsa on vaikeaa arvioida, joten hyvien parametrien valinta laajempaa testiä varten on vaikeaa. Lisäksi voidaan huomata, että menetelmä käyttää huomattavasti enemmän tietokoneen muistia kuin muut vastaavat menetelmät. Populaation koon kasvaessa muistiin tallennettavien olioiden määrä kasvaa nopeasti, joten laitteiston asettamat rajat tulevat vastaan huomattavasti muita menetelmiä nopeammin.

Kolmantena mahdollisena huonona puolena voidaan huomata menetelmän riippuvuus alimmalla evoluutiotasolla saavutetuista tuloksista. Ylemmän tason evoluutio etenee alimman tason tulosten suuntaan ja ylimmän tason siirrot lukitaan satunnaista mutaatiota lukuun ottamatta pysyvästi sitoutumisasteen kasvaessa. Tästä johtuen, hyvien tulosten saavuttamiseksi, alimmalla tasolla täytyy kerätä riittävästi tietoa ennen etenemistä. Sama ongelma esiintyy kuitenkin monissa muissakin menetelmissä.

7 Yhteenveto

Tässä tutkielmassa pyrittiin aluksi esittämään Morpion Solitairen ratkaisemiseen aikaisemmin käytetyt menetelmät yhtenäisessä ja joissakin tapauksissa alkuperäistä selkeämmässä muodossa. Erityisesti Juillén [1995] evoluutiomallin ja Cazenaven [2007, 2009] Monte Carlo -hakujen toimintaa onnistuttiin tekijän mielestä selventämään huomattavasti.

Tämän jälkeen Morpion Solitairen ratkaisuavaruutta tarkasteltiin useasta eri näkökulmasta. Erityisesti huomattiin, että hyvien ratkaisujen naapurusto määritellään lähes kokonaan noin 35 - 40 ensimmäisen siirron avulla. Lisäksi huomattiin selvä riippuvuus hyvän ratkaisun naapuruston ja noin 20 – 25 ensimmäisen siirron välillä. Tästä johtuen Morpion Solitairen loppupelin todettiin olevan huomattavasti alkuosaa helpommin ratkaistavissa. Toisin sanoen, mistä tahansa annetusta noin 40 aloitussiirron joukosta voidaan tutkielmassa esitellyillä optimointimenetelmillä suhteellisen helposti löytää näihin siirtoihin perustuva lokaalisti optimi ratkaisu.

Tämän lisäksi ongelman ratkaisuavaruuden todettiin olevan muodoltaan piikikäs. Toisin sanoen, erittäin hyvien tulosten todettiin voivan sijaita huomattavasti huonompien tuloksien naapurustossa, jonka ansiosta ongelman ratkaisemisen todettiin olevan useimmille optimointimenetelmille erittäin vaikeaa.

Lopuksi tutkielmassa esiteltiin uusi menetelmä Morpion Solitairen ratkaisemiseksi. Menetelmässä pyrittiin käyttämään hyväksi aikaisempien menetelmien hyviä puolia sekä tekijän omia huomioita Morpion Solitairesta. Esitellyn menetelmän parametreja ja toimintaa testattiin useiden testien avulla, joiden tarkoituksena oli pyrkiä selvittämään hyviä menetelmän parametrien arvoja laajempaa testausta varten. Laajemmassa testauksessa saavutettiin kohtalaisen hyviä tuloksia, joskin parhaimmista tähän asti saavutetuista tuloksista jäätiin jonkin verran. Menetelmän todettiin kuitenkin kykenevän löytämään erittäin hyviä tuloksia aikaisempiin menetelmiin verrattavassa ajassa, ellei jopa näitä nopeammin. Tutkielmassa esitellyn menetelmän voidaan olettaa kykenevän löytämään parempia tuloksia vielä laajemman haun avulla.

Uuden menetelmän kehittämisen todettiin myös olevan erittäin iteratiivista. Tässä tutkielmassa esitettyjen testien edetessä, menetelmään keksittiin kokoajan uusia mahdollisia ominaisuuksia aikaisempien menetelmän versioiden käyttäytymisen pohjalta. Vaikka uuden menetelmän kuvauksessa pyrittiinkin kuvailemaan, miten erilaisiin menetelmässä sovellettuihin ominaisuuksiin päädyttiin, menetelmän rakentumisen iteratiivista luonnetta ei ehkä onnistuttu kuvaamaan kovinkaan hyvin.

8 Kiitokset

Lopuksi haluan kiittää Timo Porasta mielenkiintoisesta aiheesta, osasta tässä tutkielmassa käytetyistä lähteistä, kokeelliseen algoritmiikkaan liittyvästä aineistosta sekä hyvästä ohjauksesta. Lisäksi haluan kiittää Timo Porasta ja Heikki Hyyröä heidän simuloituun jäähdytykseen perustuvan menetelmänsä lähdekoodista sekä tulosten visualisointiin käytettävästä Java-ohjelmasta, joiden ansiosta oman testausohjelman kirjoittaminen ja tuloksien testaaminen oli huomattavasti helpompaa.

Viiteluettelo

[Anderson, 1997] Richard J. Anderson, The role of experiment in the theory of algorithms. In: *Proceedings of the 5th DIMACS Challenge Workshop* **59**, 1997.

[Boyer] Morpion Solitaire website maintained by Christian Boyer, http://www.morpionsolitaire.com.

[Bäck, 1996] Thomas Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[Cazenave, 2007] Tristan Cazenave, Reflexive Monte-Carlo search. In: *Proceedings of Computer Games Workshop*, 2007, 165-173.

[Cazenave, 2009] Tristan Cazenave, Nested Monte-Carlo search. *International Joint Conference on Artificial Intelligence*, July 2009. Available as http://www.aaai.org/ocs/index.php/IJCAI/IJCAI-09/paper/view/343.

[Cazenave and Jouandeau, 2009] Tristan Cazenave and Nicolas Jouandeau, Parallel Nested Monte-Carlo search. In: *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, May 2009, 1-6.

[Demaine *et al.*, 2006] Erik D. Demaine, Martin L. Demaine, Arthur Langerman and Stefan Langerman, Morpion solitaire. *Theory of Computing Systems* **39**, 3, June 2006, 439–453.

[Flammenkamp, 2003] Achim Flammenkamp, Le morpion solitaire. Draft, March 2003. Available as http://www.morpionsolitaire.com/Flammenkamp.pdf.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to Theory of NP-Completeness*. W.H. Freeman & Company, 1979.

[Helmstetter and Cazenave, 2004] Bernard Helmstetter and Tristan Cazenave, Incremental transpositions. In: *Proceedings of the 4th International Conference on Computers and Games 2004,* Lecture Notes in Computer Science **3846**, Springer-Verlag, 2006, 220-231.

[Hyyrö and Poranen, 2007] Heikki Hyyrö and Timo Poranen, New heuristics for morpion solitaire. Unpublished manuscript, University of Tampere, 2007. Available as http://www.cs.uta.fi/~tp/pub/morpion-article.pdf.

[Johnson, 2001] David S. Johnson, A theoretician's guide to the experimental analysis of algorithms. In: *Proceedings of 5th and 6th DIMACS Implementation Challenges*, 2001, 215-250.

[Johnson et al. 1989] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch and Catherine Schevon, Optimization by simulated annealing: an experimental evaluation. Part I, graph partitioning. In: *Operations Research* **37**, 7, 1989, 865-892.

[Johnson et al. 1991] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch and Catherine Schevon, Optimization by simulated annealing: an experimental evaluation. Part II, graph coloring and number partitioning. In: *Operations Research* **39**, 3, 1991, 378-406.

[Juillé, 1995] Hugues Juillé, Incremental co-evolution of organisms: a new approach for optimization and discovery of strategies. In: *Proceedings of the 3rd European Conference on Advances in Artificial Life,* Lecture Notes in Computer Science **929**, June 1995, 246–260.

[Juillé, 1999] Hugues Juillé, Methods for statistical inference: extending the evolutionary computation paradigm. PhD thesis, Brandeis University, Department of Computer Science, May 1999, 45-90.

[Kirkpatrick *et al.*, 1983] Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi, Optimization by simulated annealing. *Science* **220**, 4598, May 1983, 671-680.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári, Bandit based Monte-Carlo planning. In: *15th European Conference on Machine Learning*. Lecture Notes in Artificial Intelligence **4212**, Springer, 2006, 282-293.

[Langerman] Morpion Solitaire website maintained by Stefan Langerman, http://slef.org/jeu.

[McGeoch, 1996] Catherine C. McGeoch, Toward an experimental method for algorithm simulation. In: INFORMS Journal of Computing **8**, 1996, 1-15.

[Metropolis *et al.*, 1953] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller and Edward Teller, Equation of state calculation by fast computing machines. *Journal of Chemical Physics* **21**, 1953, 1087-1091.

[Meyrignac] Morpion Solitaire website maintained by Jean-Charles Meyrignac, http://euler.free.fr/morpion.htm.

[Mitchell, 1998] Melanie Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[Moret and Shapiro, 2001] Bernard M.E. Moret and Henry D. Shapiro, Algorithms and experiments: the new (and old) methodology. *Journal of Universal Computer Science* **7**, 2001, 434-446.

[Motwani and Raghavan, 1995] Rajeev Motwani and Prabhakar Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.

[Mäkinen ja Poranen, 2008] Erkki Mäkinen ja Timo Poranen, Algoritmit. Tampereen Yliopisto, Tietojenkäsittelytieteiden laitos, raportti **D-2008-6**, 2008. Saatavilla osoitteesta http://www.cs.uta.fi/reports/dsarja/D-2008-6.pdf.

[Poranen, 2010] Timo Poranen, keskustelut ja kirjeenvaihto.

[Ross, 2010] Don Ross, Game Theory. *Stanford Encyclopedia of Philosophy*, 2010. Available as http://plato.stanford.edu/entries/game-theory.

[Sanders, 2002] Peter Sanders, Presenting data from experiments in algorithmics. *Experimental Algorithmics: From Algorithm Design to Robust and Efficient Software*, Springer-Verlag, 2002, 181-196.

[Science & Vie, 1976] Daniel Goffinet and Pierre Berloquin, Jeux & paradoxes. *Science & Vie* **703**, 1976, 130-131.

[Turing, 1936] Alan M. Turing, On computable numbers, with an application to the Entscheidungsproblem. In: *Proceedings of the London Mathematical Society* **42**, 1936, 230-265.

[Wikipedia] Game tree complexity values of several well-known games on Wikipedia. http://en.wikipedia.org/wiki/Game-tree_complexity.

[Wright *et al.*, 2003] Alden H. Wright, Jonathan E. Rowe and Michael D. Vose, Implicit Parallelism. *Genetic and Evolutionary Computation*, Springer, 2003, 1505-1517. Also available as http://www.cs.bham.ac.uk/~jer/papers/implicit.pdf.

[Zobel, 1997] Justin Zobel, Reliable research: towards experimental standards for computer science. In: *Proceedings of the 21st Australasian Computer Science Conference,* Springer-Verlag, 1997, 217-229.