Polyominojen pakkaaminen tasossa

Jon Sahlberg

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaajat: Erkki Mäkinen, Timo Poranen
Lokakuu 2014

Tampereen yliopisto
Informaatiotieteiden yksikkö
Jon Sahlberg: Polyominojen pakkaaminen tasossa
Pro gradu -tutkielma, 53 sivua
Lokakuu 2014

Tiivistelmä

Polyominot ovat kaksiulotteisia kappaleita, jotka muodostuvat yhdestä tai useammasta, särmistään toisiinsa liitetyistä, samankokoisista neliöstä. Tason täyttäminen valitulla polyominojoukolla on NP-täydellinen ongelma. Tässä tutkielmassa esitetään uusi tarkistussummiin perustuvan menetelmä, joilla polyominojen pakkaamista tasolle pyritään tehostamaan aiempiin vastaaviin pakkausmenetelmiin verrattuna. Kutsun kehittämääni menetelmää moduloidun bittikentän menetelmäksi. Moduloidun bittikentän menetelmässä täytettävälle alueelle sekä polyominoille etsitään sopiva jakaja. Jakajan perusteella täytettävälle alueelle ja polyominoille lasketaan tarkistussummat. Sopivalla jakajan valinnalla tarkistussummat saadaan kuvaamaan samanaikaisesti sekä polyominon muotoa että sijaintia täytettävällä alueella. Kaikki mahdolliset ratkaisuvaihtoehdot alueen täyttämiseksi voidaan löytää alueen ja kappaleiden yhteenlaskettuja tarkistussummia vertailemalla. Moduloidun bittikentän menetelmällä alkuperäinen vaikea tehtävä jakautuu useammaksi pienemmäksi tehtäväksi, joiden ratkaiseminen voi usein olla huomattavasti alkuperäistä tehtävää helpompaa. Tutkielmassa esitetään myös muutokset, jotka mahdollistavat hajota ja hallitse -menetelmän käytön moduloidun bittikentän menetelmän yhteydessä. Hajota ja hallitse -menetelmään soveltamiseksi tarkistussummat muunnetaan sopivaan binääriseen kantalukuesitykseen. Sopivalla binäärisellä kantaluvulla kappaleille voidaan muodostaa tarkistussummat aiempaa pienemmällä jakajalla säilyttäen tieto kappaleen muodosta ja sijainnista. Huomattavan nopeusedun ansiosta moduloidun bittikentän menetelmä mahdollistaa huomattavasti suurempien alueiden pakkaamisen kuin vastaavat bittikenttiin perustuvat menetelmät. Hajota ja hallitse -menetelmän ansiosta moduloidun bittikentän algoritmi on myös erittäin tehokkaasti rinnakkaistettavissa.

Sisällys

1 Johdanto

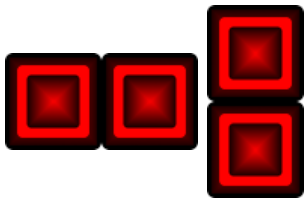
Tässä tutkielmassa tarkastellaan yksinkertaisten geometristen kappaleiden, *polyominojen*, pakkaamista tasolle. Polyominot ovat kaksiulotteisia kappaleita, jotka muodostuvat yhdestä tai useammasta, särmistään toisiinsa liitetystä, samankokoisesta neliöstä. Polyominojen pakkaaminen tasolle on melko yksinkertaista, mutta kappaleiden pakkaaminen tasolle mahdollisimman tiiviisti, jonkin tietyn alueen täyttäminen kokonaisuudessaan tai sen selvittäminen, onko tiettyä aluetta mahdollista kokonaisuudessaan täyttää valitulla polyominojoukolla, on tilanteesta riippuen usein erittäin vaikeaa. Kaikissa kolmessa edellä mainituissa ongelmassa polyominojen erilaisten yhdistelmien sekä niiden erilaisten latomisjärjestysten lukumäärä kasvaa niin nopeasti, että kaikkien mahdollisten ratkaisuvaihtoehtojen läpikäyminen yksitellen ei usein ole mahdollista. Tässä tutkielmassa esitellään menetelmiä polyominon pakkaamiseksi tasolle sekä esitetään uusi menetelmä, joilla alueen pakkaamista pyritään tehostamaan.

Tutkielman toisessa luvussa esitellään erilaisia polyominoja, polyominojen ominaisuuksia sekä polyominojen luokkia. Kolmannessa luvussa esitellään yleisluontoisia menetelmiä polyominojen pakkaamiseksi tasolle. Neljännessä luvussa käsitellään polyominojen pakkaamisen aikakompleksisuuteen vaikuttavia tekijöitä sekä esitellään NP-täydellisten ongelmien luokka, johon myös polyominojen pakkaaminen tasolle kuuluu. Viidennessä luvussa esitellään polyominojen pakkaamiseen liittyviä tarkistussummiin ja pariteettitarkistukseen perustuvia menetelmiä. Kuudennessa luvussa esitellään kirjoittajan oma menetelmä polyominoiden pakkaamiseksi tasolle. Menetelmän nimi on moduloitu bittikenttä. Moduloidun bittikentän menetelmä perustuu tarkistussummien laskemiseen erilaisille täytettävän alueen lohkojaoille sekä tulosten yhdistämiseen koko täytettävän alueen pakkaamisen selvittämiseksi. Seitsemännessä luvussa esitellään parannus moduloidun bittikentän menetelmään. Parannetun menetelmän ansiosta moduloidun bittikentän menetelmän lohkojakoon ja alueen pakkauksen etsintään voidaan soveltaa hajota ja hallitse -periaatetta. Kahdeksannessa luvussa esitellään muita tutkielman tekemisen yhteydessä havainnoituja aiheita polyominojen pakkaamiseen liittyen.

2 Polyominot

Polyominot ovat yhdestä tai useammasta samankokoisesta neliöstä koostuvia kappaleita. Yhteen polyominoon kuuluvat neliöt kytkeytyvät toisiinsa koko särmän pituudelta siten, että neliöt muodostavat keskenään yhden yhtenäisen aluen [Golomb, 1994]. Polyominot voidaan jakaa erillisiin osajoukkoihin kappaleen muodostamiseen käytettyjen neliöiden lukumäärän mukaan. Yksinkertaisimmat polyominotyypit ja samalla niihin kuuluvat kappaleet ovat *monomino* (ks. kuva 2.1), joka koostuu yhdestä yksinäisestä neliöstä, sekä *domino* (ks. kuva 2.2), joka koostuu kahdesta toisiinsa yhdistetyistä neliöistä. Monominoja on vain yksi neliönmuotoinen kappale. Dominoita on laskentatavasta riippuen joko yksi tai kaksi kappaletta. Tunnetuimmat polyominot lienevät aiemmin mainittu domino, joka on tunnettu Domino-pelistä, sekä neljästä neliöstä muodostuvat *tetriminot* (ks. kuva 2.4), jotka ovat useimmille tuttuja Tetris-pelistä.

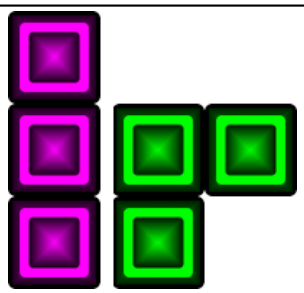
Kuva 2.1 Yksinäinen monomino on ainoa monomino.



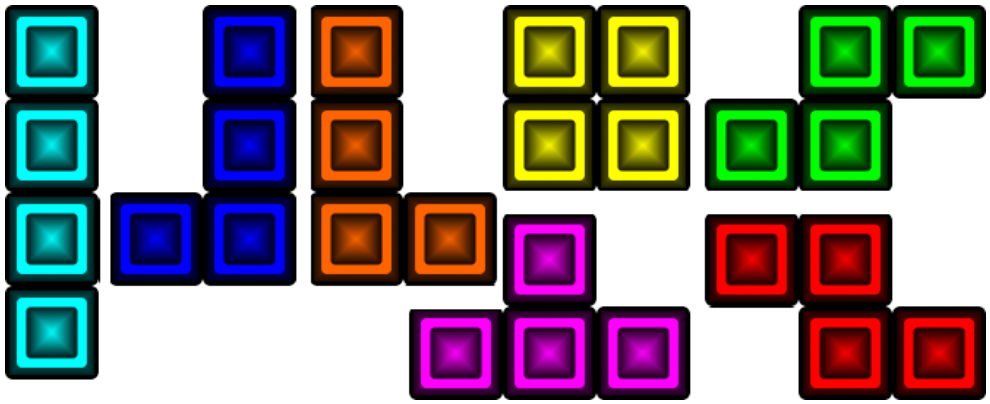
Kuva 2.2 Domino sekä sama domino 90 astetta kierrettynä.

2.1 Polyominojen ominaisuuksia

Erilaisten polyominojen lukumäärä on verrannollinen, polyominojen muodostamiseen käytettyjen neliöiden lukumäärään. Polyominon asteluvun lisäksi erilaisten polyominojen lukumäärää rajoittaa myös polyominojen muodostamiseen käytettävissä olevan alueen pituus sekä leveys. Erilaisten polyominojen lukumäärä riippuu myös siitä, lasketaanko polyominoista kiertämällä tai peilaamalla muodostetut kappaleet erilaiseksi alkuperäiseen polyominoon verrattuna. Mikäli polyominokappaleiden kiertäminen (ks. kuva 2.5) myötä- tai vastapäivään sallitaan,

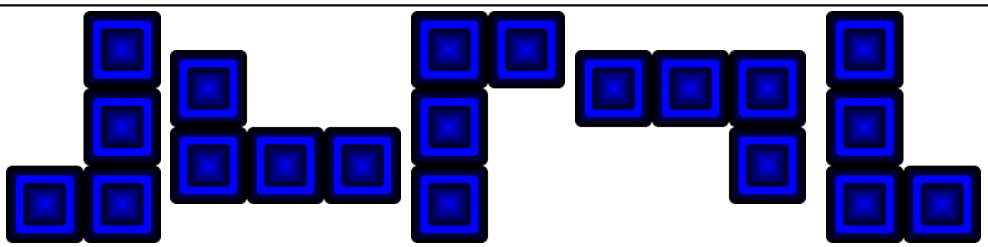


Kuva 2.3 I- ja V-triomino.



Kuva 2.4 Kaikki seitsemän yksipuolista tetriminoa.

toisin sanoen kiertämällä muodostettu kappale lasketaan samaksi polyominoksi alkuperäisen kiertämättömän kappaleen kanssa, polyominoja kutsutaan *yksipuolisiksi (engl. one-sided)*. Mikäli sekä kiertäminen että peilaaminen (ks. kuva 2.5) pysty- tai vaaka-akselin ympäri sallitaan, polyominoja kutsutaan *vapaiksi (engl. free)*. Mikäli kappaleiden kiertämistä tai peilaamista ei sallita, polyominoja kutsutaan *kiinnitetyksi (engl. fixed)*.



Kuva 2.5 J-tetrimino kierretty 4 kertaa 90 astetta myötäpäivään sekä alkuperäinen kappale peilattu y-akselin ympäri.

2.2 Polyominojen osajoukot

Tasokuvioille tai kuvioiden joukoille voidaan asettaa erilaisia määritelmiä, joilla kappaleet voidaan jakaa osajoukkoihin sekä suorittaa vertailuja kappaleiden välillä. Polyominot jaetaan usein osajoukkoihin niihin kuuluvien neliöiden lukumäärän perusteella. Neliöiden lukumäärän, eli kappaleen *asteluvun* perusteella muodostettuja osajoukkoja kutsutaan *n-ominoiksi*. N-ominoa voidaan käyttää yleisnimityksenä kaikille polyominoille. Merkinnässä n-omino käytetty 'n' voidaan korvata neliöiden lukumäärän mukaisella lukuarvolla, jolloin esimerkiksi 4-omino tarkoittaa tetriminoa.

- Monomino (ks. kuva 2.1) koostuu yhdestä neliöstä. Monominoja on vain yksi riippumatta siitä, sallitaanko kappaleen kiertäminen tai peilaaminen.

- Domino (ks. kuva 2.2) koostuu kahdesta neliöstä. Dominoja on yksi kappale. Mikäli kappaleiden kiertämistä ei sallita, dominojen lukumäärä kohoaa kahteen. Peilaamisen sallimisella tai kieltämisellä ei ole vaikutusta dominojen lukumäärään [Redelmeir, 1981; Silva, 2014; Jensen, 2009].

- *Triomino*, toiselta nimeltään *tromino*, koostuu kolmesta neliöstä. Triominoja on kaksi erilaista. Kuvassa 2.3 on vasemmalla I-triomino, jossa kaikki kolme neliötä sijoittuvat peräkkäin samalle riville tai sarakkeelle, ja oikealla kuvassa 2.3 on V-triomino jossa kolme neliötä muodostavat kulman [Golomb, 1994]. Mikäli kappaleiden kiertämistä ei sallita, triominojen lukumäärä kohoaa kuuteen. Peilaamisen salliminen kierron lisäksi ei vaikuta triominojen lukumäärään. [Redelmeir, 1981; Silva, 2014; Jensen, 2009]

- Tetrimino, toiselta nimeltään tetromino, koostuu neljästä neliöstä. Vapaita tetriminoja on viisi erilaista, yksipuolisia on seitsemän ja kiinnitettyjä yhdeksäntoista erilaista [Redelmeir, 1981; Silva, 2014; Jensen, 2009]. Tetrominojen nimitykset kuvan 2.4 mukaisessa järjestyksessä vasemmalta oikealle ja ylhäältä alas ovat I, J, L, O, S, T ja Z [Golomb, 1994].

- Tetriminoa suurempien, 5-10 neliöstä koostuvien, polyominojen nimitykset järjestyksessä pienimmästä suurimpaan ovat pentomino, heksomino, heptomino, oktomino, nonomino sekä dekomino. [Redelmeir, 1981; Silva, 2014; Jensen, 2009]

Jensenin [2003] mukaan Klarner [1967] todistaa, että polyominojen lukumäärä kasvaa eksponentiaalisesti n-omino-osajoukkojen asteluvun suhteen. Polyominojen lukumäärän kasvunopeudelle ei ole löydetty tarkkaa laskentakaavaa [Redelmeir, 1980; Jensen, 2003], joten polyominojen laskeminen tapahtuu enumeroimalla tietokoneiden avulla. Eksponentiaalisesta polyominojen lukumäärän kasvunopeudesta johtuen myös laskemiseen vaadittava aika kasvaa eksponentiaalisesti. Erilaisten polyominojen lukumäärä on laskettu muutamien kymmenien ruutujen kokoisiin polyominoihin saakka. Taulukossa 2.1 esitetään 1-10 kokoisten n-ominojen osajoukkojen alkioiden lukumäärät.

n	nimi	vapaa	yksipuolinen	kiinnitetty
1	monomino	1	1	1
2	domino	1	1	2
3	tromino	2	2	6
4	tetromino	5	7	19
5	pentomino	12	18	63
6	heksomino	35	60	216
7	heptomino	108	196	760
8	oktomino	369	704	2725
9	nonomino	1285	2500	9910
10	dekomino	4655	9189	36446

Taulukko 2.1: N-ominojen eri laskusääntöjen mukaisia lukumääriä polyominojen asteluvun mukaan jaoteltuna [Redelmeir, 1981; Silva, 2014; Jensen, 2009].

3 Tason pakkaaminen polyominoilla

Polyominoja voidaan pakata tasolle useaa eri tavoitetta silmälläpitäen. Yleensä polyominojen pakkaamistehtävissä ongelmalle löytyy joko useita toisistaan hieman poikkeavia ratkaisuja tai vastakohtaisesti tehtävälle ei löydy yhtään täsmällistä ratkaisua. Poikkeuksellisesti suuri osa pulmatehtävistä on tarkoituksella suunniteltu siten, että tehtävälle on vain yksi tai hyvin pieni määrä tavoitteet täyttäviä ratkaisuja. Tehtävän vaikeuden määrittämiseksi pitäisi tietää, millaista tavoitetta tehtävällä pyritään täyttämään ja millainen tulos on hyväksyttävissä ratkaisuksi tehtävän suorittamiselle. Yksittäisen ratkaisun löytäminen polyominojen pakkaustehtävissä on yleensä helppoa, mutta kaikkien mahdollisten ratkaisuvaihtoehtojen esittäminen tai parhaan mahdollisen tuloksen löytäminen on usein huomattavasti työläämpää. Seuraavaksi esitetään muutamia erilaisia pakkaustehtävätyyppejä sekä niihin liittyviä ongelmia.

3.1 Täydellinen ja epätäydellinen pakkaaminen

Täydellisessä pakkauksessa koko täytettävä alue pyritään pakkaamaan täyteen käytettävissä olevilla polyominoilla. Täydellisessä pakkauksessa täytettävälle alueelle ei saa jäädä yhtään täyttämätöntä tilaa, koska silloin kyseessä olisi *epätäydellinen pakkaus*. Epätäydellinen pakkaus voidaan muuntaa täydeksi pakkaukseksi lisäämällä alueelle pakkauksen täydentävät polyominot. Golomb [1994] esittää ongelman shakkilaudan täyttämisestä suorilla triomino-kappaleilla. Koska shakkilaudan pinta-ala, 64 ruutua, ei ole jaollinen triominon pinta-alalla, täysi pakkaus ei ole mahdollinen ilman täydentäviä kappaleita. Golombin esimerkissä täyden pakkauksen mahdollistamiseksi kahdenkymmenenyhden triominon lisäksi tarvitaan yksi yhden ruudun kokoinen monomino. Golombin [1994] esittämässä ongelmassa kysytään "onko shakkilaudan täyttäminen triominoilla sekä yhdellä monominolla mahdollista?". Tehtävän todistuksessa esitetään *kombinatoriseen geometriaan* perustuva algoritmi, jolla monominon ainoat mahdolliset sijainnit laudalla voidaan selvittää ennen triominojen pakkaamisen aloittamista.

Epätäydellisessä pakkauksessa täytettävä alue pyritään pakkaamaan mahdollisimman täyteen polyominoja. Joissain tapauksissa on kuitenkin mahdotonta täyttää aluetta kokonaan, esimerkiksi kun alueen pinta-ala ei ole tasan jaollinen käytettävissä olevien polyominojen ruutujen lukumäärällä tai polyominot sekä täytettävä alue eivät vain muuten ole yhteensopivia keskenään. Esimerkkinä epätäydellisestä pakkauksesta logistiikka-alalla toimivat yritykset pyrkivät saa-

maan kuormalavan mahdollisimman täytteen kuljetettavaa materiaalia. Kuormaa ei kuitenkaan jätetä kuljettamatta, vaikka lavaa ei saataisi ahdettua absoluuttisen täyteen. Logistiikassa kuorman pakkaamiseen vaikuttaa kuorman täyttöasteen lisäksi myös muun muassa yksittäisten pakettien määränpää reitin varrella, pakettien pinoamiskestävyys sekä paino. Kuorma-auton pakkaamisongelma on huomattavasti polyominojen pakkaamista monisyisempi ongelma. Ongelmat ovat yhteneviä erityistapauksessa, jossa kaikkien kuormaan pakattavien laatikoiden korkeus on sama, laatikoita ei voi kääntää ylösalaisin tai kyljelleen, minkä lisäksi laatikoiden sivujen pituudet tulisi voida muodostaa jonkin vakiollisen mitan tulona. Tällöin pakattavien laatikoiden voidaan ajatella olevan vakiokokoisista neliöistä koostuvia polyominoja, joilla on tarkoitus täyttää kuorma-auton lava kerroksittain mahdollisimman täydellisesti.

3.2 Löyhä pakkaus

Löyhässä pakkauksessa on tarkoitus pakata alueelle mahdollisimman monta polyominoa. Täydelliseen pakkaukseen verrattuna täytettävää aluetta ei kuitenkaan tarvitse kokonaisuudessaan täyttää polyominoilla. Löyhässä pakkauksessa erilaiset säännöt määrittelevät, miten kappaleita voi asettaa täytettävälle alueelle. Usein kappaleet pitää asetella joko osittain tai kokonaan määritellylle tai toisinaan vapaavalintaiselle mahdollisimman pienikokoiselle alueelle. Takefuji [1992] mainitsee polyominojen pakkausongelman esiintyvän piirilevytuotannossa. Piirilevylle sijoitettavat komponentit halutaan usein asetella siten, että piirilevyn pinta-alasta tulisi mahdollisimman pieni. Silti piirilevyä ei voi latoa liian täyteen komponentteja muun muassa komponenttien välisen oikosulkuvaaran sekä hukkalämmön haihtumisen takia. Golomb [1994] esittelee löyhään pakkaukseen liittyvän pelin, jossa pelaajat asettavat polyominoja vuorotellen pelialueelle. Pelaaja, joka ei pysty omalla vuorollaan asettamaan uutta kappaletta pelialueelle, häviää pelin. Golomb [1994] esittää myös muutamia tehtäviä, joissa mahdollisimman vähäisellä määrällä monominoja yritetään estää valitun pentominon asettaminen shakkilaudan kokoiselle alueelle.

Golombin esimerkin innoittamana keksimässäni löyhää pakkausta käyttävässä pelissä pelaajan tulee pyrkiä asettamaan valitun asteluvun polyominoja pelilaudalle siten, että mahdollisimman vähäisellä kappaleiden lukumäärällä voidaan estää uusien polyominojen lisääminen alueelle. Keksimäni pelin tavoitteen kanssa yhtenevä tehtävä on, että mahdollisimman vähäisellä polyominojen lukumäärällä täytettävälle alueelle ei saa jäädä valitun asteluvun kokoisia

tai sitä suurempia yhtenäisiä tyhjiä alueita. Samalla täyttämättömien ruutujen lukumäärä halutaan maksimoida parhaan mahdollisen tuloksen aikaansaamiseksi. Kuvatun kaltaista kappaleiden pakkausta voisi sanoa täydellisen pakkauksen vastakohdaksi, sillä siinä halutaan asettaa polyominoja alueelle mahdollisimman vähän ja mahdollisimman harvaan.

3.3 Alueen täyttäminen polyminoilla

Golomb [1994] esittää lukuisia esimerkkejä siitä, onko jonkin alueen täyttäminen valituilla polyominoilla ylipäätään mahdollista. Tässä kohdassa esitellään keinoja, joilla pakkaamista voi yrittää helpottaa, esimerkiksi käyttämällä hyväksi tietoa kyseisestä ongelmasta tai käytettävistä polyominoista. Esimerkiksi kaikki polyominot eivät sovi suorakulmaisille nurkka-alueille, tai tietyt kappaleet joko sopivat tai vastaavasti eivät sovi yhteen keskenään.

3.3.1 Valitaan seuraava täytettävä alue

Valitaan seuraavaksi täytettävä polyominon sijainti sen mukaan, montako erilaista polyominoa kyseiselle alueelle on mahdollista sijoittaa. Valinnan tarkoitus on pienentää hakupuun haarojen lukumäärää täyttämällä sellaisia alueen osia, joihin sopii vain vähän erilaisia polyominoja. Jäljellä olevan alueen pienentyessä eri siirtovaihtoehtojen määrä vähenee entisestään, joten pakkaamisen tarkasteluun käytettävään hakupuuhun tulee mahdollisimman vähän haaroja. Erittäin yksinkertainen valikoiva algoritmi voisi kiertää täytettävää aluetta myötä- tai vastapäivään alueen reunoja myötäillen. Alueen reunat ja erityisesti alueen nurkat rajoittavat kappaleiden sijoittamista alueelle, jolloin erilaisia vaihtoehtoja on vähemmän kuin alueen tyhjässä keskiosassa.

3.3.2 Alueen pakkaaminen osissa

Alueen jakaminen sopivasti valikoituihin osiin voi helpottaa pakkaamistyötä pienentämällä paikallisen alueen pakkausvaihtoehtojen hakupuuta ja mahdollistamalla alueen pakkaamisen useassa samanaikaisessa prosessissa. Mikäli pakattava alue koostuu useammasta kapealla käytävällä toisiinsa yhdistetyistä alueista, on mahdollista jakaa alue osiin kapeaa aluetta hyväksikäyttäen siten, että molemmat puoliskot voidaan pakata erikseen ja yhdistää alueiden pakkaus tulosten valmistuttua. Suorakulmasten alueiden pakkaamisen voisi aloittaa samanaikaisesti kappaleen jokaisesta nurkasta. Täytettävien alueiden symmetrisiä piirteitä voi

myös käyttää hyväkseen hakupuun minimoinnissa. Kahden tai useamman alueen ollessa symmetrisiä keskenään alueiden pakkaaminen voidaan hoitaa yhdellä yhteisellä laskennalla. Mikäli alue on symmetrinen peilauksen tai kierron suhteen, tulee ottaa huomioon, että myös täyttämiseen käytettävät kappaleet on voitava muuntaa samalla operaatiolla alueen kanssa. Menetelmää voi käyttää vaikka peilaus tai kierto-operaatio ei olisikaan sallittu kyseisessä pakkaustehtävässä. Tällöin vastaavat polyominokappaleet tulee löytyä valmiiksi muunnettuina käytössä olevien polyominojen joukosta.

3.3.3 Paikallinen etsintä

Yksi perinteinen menetelmä, jota käytetään muissakin optimointitehtävissä, on *paikallinen etsintä*. Paikallisen etsinnän menetelmässä alueelle asetetaan kappaleita, kunnes kaikki triviaalit kappaleiden lisäykset on käytetty. Mikäli pakkauksen lopputulokseen ei tässä vaiheessa olla tyytyväisiä, täytettävältä alueelta valitaan kohta, jonne asetetaan uusi polyomino aiemmin lisätyistä polyominoista ja täyttösäännöistä välittämättä. Uuden polyominon lisäämisen tai muun satunnaisen muutoksen jälkeen alueen pakkaus korjataan sääntöjen mukaiseksi esimerkiksi poistamalla uuden kappaleen alle jääneet vanhat polyominot. Tämän jälkeen algoritmia jatketaan alusta toiselle kierrokselle mahdollisten triviaalien polyominojen lisäämisellä, lopputuloksen tarkastamisella ja tarvittaessa uudella polyominojen vaihtokierroksella. Algoritmin suoritus päättyy, kun algoritmin tulos on tyydyttävä tai jokin muu algoritmin lopetusehto täyttyy. Algoritmin lopetusehtona voidaan käyttää algoritmin suoritukselle varattua aikarajaa, algoritmin iteraatiokierrosten enimmäislukumäärää, vaatimusta algoritmin tuloksen parantumisesta kierrosten välillä tai jotain näiden yhdistelmiä. Paras lopetusehdon täyttymiseen mennessä löytynyt tulos palautetaan algoritmin lopullisena tuloksena. Menetelmän ongelma on, että sitä käyttämällä ei aina voi tietää, miten lähelle optimaalista tulosta sillä päädytään. [Aho et al., 1983]

3.3.4 Tilastolliseen analyysiin perustuva pakkaaminen

Olisi mielenkiintoista selvittää, mikäli jotkut polyominot esiintyvät useammin suuressa joukossa polyominoilla peitetyistä alueista. Polyominojen esiintymistiheyttä erilaisten alueiden pakkaamisessa voisi käyttää hyväksi pakkaamisessa mukaillen polyominojen esiintymistiheyttä. Alueen täyttäminen voisi olla mielekästä aloittaa useimmin esiintyvillä kappaleilla, sillä suuren esiintymistiheyden johdosta, niiden oikea sijainti osuisi todennäköisemmin oikeaan pelkällä satunnaisel-

la arvauksella. Toisaalta mikäli vähän esiintyvät kappaleet saisi sijoitettua pakkauksen aikaisessa vaiheessa, jäljelle jäisi vain muutamia erilaisia paljon esiintyviä kappaleita, mikä myös helpottaisi jäljelle jääneen alueen pakkaamista. Erityisen hyödyllistä olisi löytää yhteyksiä kappaleiden esiintymien välillä. Esimerkiksi havainto, jossa kaksi tai useampia kappaleita esiintyy usein lähekkäin yhdessä, voisi helpottaa paikallisen alueen täyttämistä merkittävästi. Mikäli polyominoiden esiintymiselle löytyisi jokin selkeä kaava, olisi mahdollista jakaa täytettävä alue osiin ja suorittaa alueen täyttäminen osissa erilaisten polyominojen esiintymistiheyttä mukaillen.

3.3.5 Muita ratkaisutapoja

Täydellisen pakkauksen saavuttamista helpottaa, mikäli pelialueen täyttämätön alue pidetään mahdollisimman yhtenäisenä. Mikäli täytettävä alue jostain syystä jakautuu kahdeksi täysin erilliseksi alueeksi, molemmat alueet tulee pystyä täyttämään, jotta pakkauksesta voisi tulla täysi. Alueen voi toki jakaa osiin, mikäli tietää, että vähintään ainakin toisen lohkon pakkaaminen yksistään on mahdollista. Vaikka tiedettäisiin, että koko alue on jollain tavalla täytettävissä, pelkästään toisen alueenpuolikkaan pakkaaminen ei takaa koko alueen pakkaamisen onnistumista. Yksinkertainen tapa pitää täytettävä alue yhtenäisenä on aiemmin esitetty yksinkertaiseen algoritmi, jossa alueen pakkausjärjestys valittiin kiertämällä alueen reunoja myötä- tai vastapäivään.

Koska polyominoihin kuuluuvat kaikki neliöistä koostuvat alueet, alueen täyttämisen sijaan voisi olla helpompaa leikata täytettävästä alueesta sopivan kokoisia yhtenäisiä lohkoja. Pakkaamisen sijaan suoritettavan alueen lohkomisen ajatus on hyväksikäyttää ennalta tunnettuja alueiden täyttötapoja. Samalla erillisiä lohkottuja alueen osia on mahdollista täyttää rinnakkaisissa prosesseissa.

4 Alueen täyttämisen kompleksisuus

Erilaisilla täytettävillä alueilla on omia ominaispiirteitä, jotka voivat vaikuttaa pakkaamiseen. Erityisen suoraviivaisesti alueen täyttämistä vaikeuttaa täytettävän alueen koon kasvaminen. Mitä suurempi täytettävä alue on, sitä suurempi operaatio sen täyttäminen on. Niin täytettävän alueen kuin myös pakkauksessa käytettävien kappaleiden koko, lukumäärä sekä muoto vaikuttavat alueen täyttämisen vaikeusasteeseen. Kappaleiden tai täytettävän alueen muodon vaikutusta pakkaamisen vaativuuteen voi olla erityisen hankala arvioida etukäteen. Kuten Golombin [1994] esittämässä sekä myöhemmin tässäkin tutkielmassa esimerkkinä käytettävässä dominojen pakkaamisessa shakkilaudalle osoitetaan, joissain tapauksissa on mahdollista yksinkertaisin menetelmin todistaa, että jokin pakkaus on mahdoton. Menetelmä ei kuitenkaan toimi käänteisesti, eli pakkausta ei voida automaattisesti pitää mahdollisena pelkästään sen perusteella, että pakkausta ei voida todistaa mahdottomaksi. Tutkielmassani pyrin välttämään oletuksia täytettävän alueen tai käytettävien polyominojen suhteen siten, että esittämäni menetelmät toimivat mahdollisimman laajasti kaikilla alueilla tai kappaleilla. Pyrkimyksenä on esittää yleisluontoinen ja mahdollinen yksinkertainen menetelmä alueen pakkaamiseen sekä arvio alueen täyttämisen vaikeusasteesta täytettävän alueen sekä kappaleiden koon ja lukumäärän funktiona. Käytän esimerkinomaisesti tetriminoja kappaleiden eri vaihtoehtojen esittelyyn. Täytettävät alueet ovat tutkielman esimerkeissä suorakulmioita, mutta lähes minkä muotoinen alue tahansa voisi yhtä hyvin tulla kyseeseen.

4.1 NP-täydellisyys

NP-ongelmien (engl. nondeterministic polynomial time) luokkaan kuuluvat ongelmat, jotka ovat ratkaistavissa polynomisessa ajassa käyttäen epädeterminististä Turingin konetta. NP-ongelmien luokan tehtäviin esimerkiksi arvaamalla saadut ratkaisuehdotukset voidaan tarkistaa polynomisessa ajassa. *NP-täydelliset (engl. NP-complete)* ongelmat ovat NP-ongelmien osajoukko, jossa minkä tahansa ongelman polynomisen ratkaisualgoritmin keksiminen johtaisi myös muiden NP-täydellisten ongelmien ratkeamiseen vastaavalla aikakompleksisuudella. Samalla yhden NP-täydellisen ongelman polynominen ratkeaminen todistaisi tunnetun $P = NP$ -ongelman, eli voidaanko kaikki epädeterministisellä Turingin koneella polynomisessa ajassa ratkaistavissa olevat ongelmat ratkaista polynomisessa ajassa myös deterministisellä Turingin koneella. Yleisesti arvellaan, että $P \neq NP$.

Korf [2003] on todistanut, että nelikulmioiden pakkaaminen mahdollisimman pieneen tilaan on NP-täydellinen ongelma. Vastaavankaltaisen todistuksen triominoille ovat esittäneet Horiyama ja kumppanit [2012]. Lähteissä esitettyjen todistusten perusteella voisi olettaa, että yleinen polyominojen pakkausongelma yksittäisiä erityistilanteita lukuunottamatta olisi myös NP-täydellinen. Oletus polyominojen pakkaamisen NP-täydellisyydestä riittää toistaiseksi. Myöhemmin tutkielmassa osoitetaan, että *repuntäyttöongelma* (engl. knapsack tai backpack problem) eli *alijoukon summa -ongelma* (engl. subset sum -problem) esiintyvät myöhemmin esiteltävässä moduloidun bittikentän menetelmässä polyominojen pakkaamisongelman osaongelmana. Myös nämä ongelmat ovat NP-täydellisiä [Garey ja Johnson, 1979].

Tunnettu NP-täydellinen ongelma on myös *kauppamatkustajan ongelma* (engl. traveling salesman problem). Kauppamatkustajan ongelmassa henkilön tulee löytää mahdollisimman lyhyt reitti lähtöpaikasta kaikkien ennalta valittujen kaupunkien kautta takaisin lähtöpaikkaansa. Mikäli kaupunkeja on d kappaletta, kaikkien järjestysten lukumäärä on $(d-1)!$. Kauppamatkustajan ongelman tapaan polyominojen pakkaaminen tasolle voidaan myös ajatella toteutettavaksi polyominojen järjestyksen mukaisena ongelmana. Mikäli polyominot sijoitetaan täytettävälle alueelle vierekkäin, kokeillen kappaleiden erilaisia järjestyksiä, kaikkien sijoitteluvaihtoehtojen lukumääräksi tulee polyominojen lukumäärän kertoma $n!$.

En ole kirjallisuudesta löytänyt algoritmia, joka pakkaisi polyominoja tasolle kappaleiden järjestyksien mukaan, joten menetelmä on täysin teoreettinen. Menetelmä on yksinkertainen, mutta myös ainakin teoriassa kohtuullisen tehokas. Menetelmän esityksessä oletetaan, että jokaisen alueelle asetettavan kappaleen sijainti edellisen kappaleen viereen voidaan löytää vakiollisessa ajassa. Tässä tutkielmassa menetelmää käytetään myöhemmin esiteltävän algoritmin vertailukohtana. Vertailussa oletetaan, että kappaleiden tulevat sijainnit voidaan löytää pienessä vakiollisessa ajassa ja menetelmän aikakompleksisuuden oletetaan olevan suuruusluokkaa $d!$.

Seuraavissa kohdissa on esitetty kolme tekijää, jotka vaikuttavat polyominojen pakkaamisen kompleksisuuteen. Vaikka tässä luvussa esitellään yhteensä kolme tekijää, täytyy huomata, että kyseiset tekijät, alueen koko A, polyominojen lukumäärä c sekä polyominojen koko, eli asteluku n, ovat vuorovaikutuksessa keskenään. Käytännössä mitkä tahansa kaksi mainituista tekijöistä määrittelevät kokonaan tai ainakin suurelta osin myös kolmannen tekijän.

4.2 Täytettävän alueen koko

Täydellisessä pakkauksessa täytettävän alueen jokainen ruutu tulee peittyä yhden polyominon tai sen osan alle. Polyominoja ei saa sijoittaa limittäin toisiinsa nähden. Samoin käytettävien polyominojen tulee sijaita kokonaan täytettävän alueen sisäpuolella. Aseteltaessa ensimmäistä polyominoa tyhjälle täytettävälle alueelle voidaan sanoa, että vaihtoehtoisten sijoituspaikkojen lukumäärä on samaa suuruusluokkaa täytettävän alueen pinta-alan kanssa. Seuraavien kappaleiden asettamisessa täytettävän alueen pinta-alan voidaan sanoa pienentyneen aiemmin sijoitettujen kappaleiden verran, jolloin mahdollisia hyväksyttäviä paikkoja uudelle kappaleelle on jäljellä vähemmän. Todellisuudessa alueen koon lisäksi vaihtoehtoisten kappaleen sijoituspaikkojen lukumäärään vaikuttaa polyominon koko sekä muutamat yksittäiset alueen muodon tai aiemmin sijoitettujen kappaleiden aiheuttamat rajoitukset, jotka voivat estää sijoittamasta osaa uusista polyominoista tietyille täytettävän alueen osille. Alueen pinta-ala $A = d * n$.

4.3 Polyominojen lukumäärä

Myös polyominojen lukumäärä vaikuttaa alueen täyttövaihtoehtojen läpikäymisen lukumäärään. Huomionarvoista on, että N kappaletta erilaisia polyominoja voidaan asettaa N! erilaiseen järjestykseen. Mikäli joukossa on keskenään samanlaisia polyominoja, täyttövaihtoehtojen lukumäärä kasvaa hieman maltillisemmin. Erilaisiin kappaleisiin verrattuna samanlaisten kappaleiden sijoittelun järjestyksellä ei ole merkitystä alueen täytössä, jolloin samanlaisten kappaleiden vaihtaessa keskenään paikkaa pakkaus pysyy muuttumattomana. Polyominojen erilaisten järjestysten lukumäärä kuvaa yksinkertaista pakkausmenetelmää, jossa alueen täyttäminen aloitetaan yhdestä paikasta ja pyritään järjestelmällisesti asettamaan kappaleita vierekkäin siten, että jokainen seuraava kappale asetetaan tiiviisti edellisen kappaleen viereen. Kappaleiden lukumäärä $d = \frac{A}{n}$.

4.4 Polyominojen koko

Polyominojen koko vaikuttaa myös omalta osaltaan alueen täyttämiseen. Suurikokoisilla polyominoilla alueen pinta-ala täyttyy nopeammin, jolloin kappaleiden lukumäärä pienenee. Toisaalta taas pieniä kappaleita tarvitaan enemmän saman alueen täyttämiseen, mikä voi vaikeuttaa alueen täyttämistä. Pitää myös ottaa huomioon, että suurenkin alueen täyttäminen monominoilla on triviaalia. Polyominojen asteluku $n = \frac{A}{d}$.

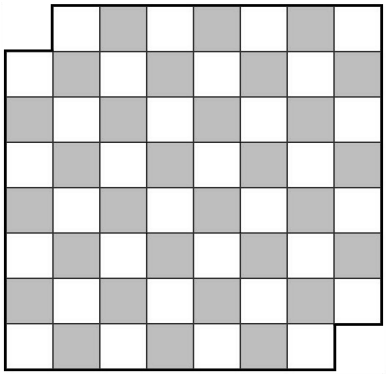
4.5 Polyominojen muoto

Osa polyominoista on symmetrisiä kappaleen kiertämisen tai peilaaminen suhteen. Symmetrisyydellä tarkoitetaan, että operaation suorittaminen kappaleelle ei vaikuta kappaleen muotoon eikä siten kasvata erilaisten pakkausvaihtoehtojen lukumäärää. Osa polyominoista on symmetrisiä puolittain, jolloin kiertäminen 90 astetta tuottaa alkuperäiseen verrattuna eri polyominon, mutta 180 asteen kiertäminen pitää polyominon muodon ennallaan. Tämän tutkielman esimerkeissä kiertoa tai peilausta ei sallita, jolloin kierrolla tai peilauksella muodostetut kappaleet lasketaan eri polyominoiksi alkuperäiseen verrattuna.

5 Polyominot ja tarkistussummat

Golomb [1994] esittää kirjassaan Polyominoes useita menetelmiä, joilla on mahdollista päätellä, onko polyominoja ylipäätään mahdollista sijoittaa täytettävälle alueelle. Tässä luvussa käytän esimerkkinä Golombin [1994] esittelemää tapausta tarkistussummiin perustuvista pakkausmenetelmistä. Golomb ei mainitse kirjassaan termejä *tarkistussumma (engl. checksum)* tai *pariteettitarkistus (engl. parity check)*. Pariteettitarkistuksen sijaan hän käyttää esimerkeissään täytettävän alueen värikoodausta, mikä on käytännössä vastaava asia.

Tiedonvälityksessä pariteettitarkistuksessa verrataan viestin lähettämisen ja vastaanoton yhteydessä laskettujen viestin erilaisten osien ilmentymien lukumääriä keskenään. Mikäli ilmentymien osien lukumäärät täsmäävät, voimme olettaa, että viesti on jollain varmuudella säilynyt muuttumattomana. Yksinkertainen pariteettitarkistus, vastaava jota Golomb [1994] käyttää, vertailee alueen ja kappaleiden värikoodattujen ruutujen, tai vastaavasti tiedonvälityksen tapauksessa viestin bittien ilmentymien, lukumääriä keskenään. Polyominojen pakkauksen tapauksessa kappaleiden ja alueen väritysten täsmätessä on mahdollista, mutta ei täysin varmaa, että pakkaus onnistuu. Mikäli värikoodattujen ruutujen lukumäärät eivät täsmää keskenään, voimme olla varmoja, että pakkaus ei ole mahdollinen.



Kuva 5.1 Esimerkki 8*8 ruudun kokoisesta aluesta, josta on poistettu ristikkäiset kulmat.

5.1 Esimerkki pariteettitarkistuksesta

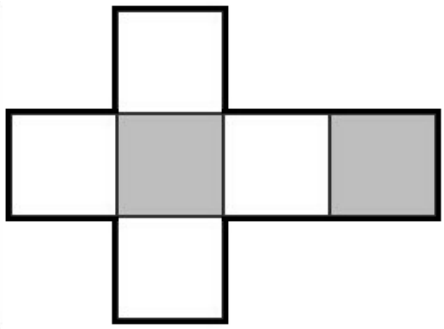
Perinteisen $8 * 8$ ruudun kokoisen shakkilaudan täyttäminen dominoilla on triviaali tehtävä. Muutetaan tehtävää poistamalla shakkilaudalta kaksi ristikkäisistä

kulmaruutua. [Golomb, 1994] Onko kuvan 5.1 esittämää aluetta mahdollista täyttää dominoilla? Vaikka kuvan 5.1 esittämässä alueessa on parillinen määrä ruutuja, kuten myös dominoissa, alueen täyttäminen ei ole mahdollista. Riippumatta siitä, miten yksittäiset dominot asetetaan laudalle, jokainen domino peittää aina alleen yhden tumman ja yhden vaalean ruudun. Koska alueella on eri määrä vaaleita ja tummia ruutuja, aluetta ei ole mahdollista täyttää dominoilla. Golombin [1994] väritysesimerkkiä mukaillen alueen ruutujen värit voidaan tulkita myös binäärisinä lukuarvoina. Valitaan vaalean ruudun arvoksi $2^0 = 1$ ja tumman ruudun arvoksi $2^1 = 2$. Tarkistussumman koko täytettävälle alueelle olisi täten $32 * 1 + 30 * 2 = 92$. Vastaavilla ruutujen väreillä toteutetussa esityksessä yksittäinen domino peittää aina yhden tumman ja yhden vaalean ruudun, mistä seuraa, että jokaisen yksittäisen dominon tarkistussumma on $1 + 2 = 3$. Koska täytettävän alueen summa 92 ei ole jaollinen dominoiden arvolla, voidaan todeta, että kyseisen alueen pakkaaminen dominoilla on mahdotonta.

5.2 Alueen pakkaaminen vapaavalintaisella polyominojoukolla

Valitun alueen täyttäminen ennalta määritellyllä polyominojoukolla on vaikea tehtävä. Helpotetaan alueen täyttämisongelmaa siten, että vapautetaan algoritmi käyttämään mitä tahansa yhden ennalta valitun asteluvun n mukaisia polyominoja. Oletetaan lisäksi, että alueen pinta-ala on jaollinen valitulla asteluvulla. Tällöin alueen koko ruutuina on muotoa $c * n$, jossa c on tarvittavien polyominojen lukumäärä ja n on polyominojen asteluku. Golombin [1994] dominoesimerkistä huolimatta tehtävä saattaa vaikuttaa aluksi triviaalilta. Yksinkertainen menetelmä minkä tahansa alueen pakkaamiseen vapaavalintaisilla vakiollisen asteluvun polyominoilla olisi leikata alue valitun asteluvun kokoisiin lohkoihin. On kuitenkin osoitettavissa, että jokaiselle n-ominojoukolle on olemassa alueita, joita ei ole mahdollista täyttää kyseisellä vapaavalintaisella saman asteluvun polyominojoukolla.

Kuvassa 5.2 esitetään alue, jota ei ole mahdollista täyttää millään vapaavalintaisella 3-ominojoukolla. Alueen pinta-ala on kuitenkin jaollinen triominojen asteluvulla 3. Esimerkin mukainen kuvio on laajennettavissa mille tahansa polyominojen kahta suuremmalle asteluvulle. Kuvio muodostuu, kun $(n+1)$:n ruudun mittainen suora kappale lävistää kohtisuoraan n ruudun mittaisen suoran kappaleen. Kappaleiden yhteisen risteyskohdan tulisi sijaita missä tahansa muualla kuin $(n+1)$:n mittaisen kappaleen kummassakaan päädyssä. Koska $(n+1)$:n ja n:n mittaiset kappaleet jakavat yhden yhteisen ruudun, yhdistetyn alueen pinta-



Kuva 5.2 Esimerkki alueesta, joka ei ole täytettävissä millään triominojoukolla.

alaksi muodostuu $(n + 1) + (n) - 1 = 2n$ ruutua, joka on jaollinen kyseisten n-ominojen asteluvulla n.

6 Moduloitu bittikenttä

Tässä luvussa esitellään kirjoittajan oma tietorakenne sekä algoritmi polyominojen pakkaamiseksi tasolle. Kutsun suunnittelemaani tietorakennetta *moduloiduksi bittikentäksi*. Algoritmin esittelyssä käytetään kuvan 6.1 mukaista 8∗8 ruudun kokoista aluetta, joka on valmiiksi täytetty kuvassa esitetyllä polyominojoukolla. Algoritmin esittelyssä käytetään etukäteen täytettyä aluetta sekä valikoituja polyominoja. Ennalta määritelty ratkaisu ei ole olennaista algoritmin toiminnan kannalta, vaan tarkoitus on selkeyttää algoritmin toiminnan esittelyä. Menetelmässä polyominoja sijoitetaan täytettävälle alueelle määriteltyjen lohkojen perusteella, jolloin polyominojen pakkaamisongelma voidaan muuttaa tarkistussummien mukaisena bittiesityksenä aiempaa tiiviimpään muotoon. Menetelmän nimitys moduloitu bittikenttä tulee siitä, että menetelmässä polyominokappaleet muunnetaan (engl. modulate) binääriseen *bittikenttä (engl. bit field)* -esitysmuotoon. Samalla täytettävän alueen sekä polyominojen esitystä tiivistetään menetelmän tarpeisiin soveltuvalla jakojäännöksellä (engl. modulus).

	A	B	C	D	E	F	G	H
A	0	1	2	3	4	5	6	7
B	8	9	10	11	12	13	14	15
C	16	17	18	19	20	21	22	23
D	24	25	26	27	28	29	30	31
E	32	33	34	35	36	37	38	39
F	40	41	42	43	44	45	46	47
G	48	49	50	51	52	53	54	55
H	56	57	58	59	60	61	62	63

Kuva 6.1 8x8 ruudun kokoinen alue indeksoituna ja täytettynä erilaisilla polyominoilla.

Modulaatiolla tarkoitetaan tässä asiayhteydessä polyominon esitysmuodon muuntamista tai mukauttamista erilaiseen käyttötarkoitusta paremmin palvelevaan, mutta alkuperäistä vastaavaan muotoon. Yleiskielessä modulaatiolla voidaan myös tarkoittaa musiikin sävellajista toiseen siirtymistä tai elektroniikassa signaalin yhdistämistä toiseen kantasignaaliin. Myös nämä määritelmät kuvaavat epäsuorasti moduloidun bittikentän menetelmään, sillä menetelmässä lasketaan yhdelle pakkaukselle useita eri jakojäännöksellä toteutettuja tarkistussummia, joista voidaan muodostaa koko alkuperäisen alueen pakkausvaihtoehdot.

Bittikenttä on käytännössä jono bittejä, joita voidaan käyttää esimerkik-

si tasokuvion täyttämisessä vapaiden tai varattujen ruutujen merkitsemisessä [Busche, 2011]. Bittikenttiä käytetään myös yleisesti datan pariteetin tai muun varmiste- tai tiivistearvon laskemisessa. Bittikenttien käyttö on hyvin toteutettuna erittäin tehokas, sillä bittikentän tilan muutos voidaan usein toteuttaa yhden tietokoneen peruslaskentaoperaation aikana, eli algoritmisessa mielessä lyhyessä vakiollisessa ajassa.

6.1 Algoritmin valmistelut

Algoritmin suorittamisen alussa tiedetään, minkä kokoinen alue halutaan täyttää. Tiedetään myös polyominojen asteluku, jonka perusteella voidaan laskea tarvittavien polyominojen lukumäärän, mikäli sitä ei ole erikseen tehtävänannon yhteydessä annettu. Algoritmin toiminta perustuu pääasiassa lukujen jaollisuuteen ja jakojäännökseen. Se, annetaanko alueen täyttävät kappaleet algoritmille tehtävänannon yhteydessä, ei muuta algoritmin toimintalogiikkaa merkittävästi. Tämän kohdan esimerkissä kuitenkin oletetaan, että alueelle pakattavat polyominot on annettu algoritmille etukäteen. Polyominojen mahdollisia sijainteja alueella ei kuitenkaan tiedetä ennen algoritmin suorittamista. Täysin vapaasti valittavilla polyominoilla $8 * 8$-kokoinen alue olisi triviaalisti täytettävissä esimerkiksi $2 * 2$-kokoisilla neliön muotoisilla tetrominoilla. Yksittäisen vastauksen löytäminen vapaasti valittavilla polyominoilla olisi usein, mutta ei aina, helpompaa kuin ennalta määrättyjen polyominojen pakkaaminen tasolle.

6.1.1 Alueen numeroiminen

Moduloidun bittikentän menetelmässä käytetään hyväksi polyominojen ominaisuutta, jossa polyominojen asteluvusta tiedetään, miten suurelle alueelle kyseisen asteluvun mukainen polyomino enintään voi yltää. Täytettävän alueen koon sekä polyominon asteluvun perusteella algoritmin alustuksessa lasketaan alueen numerointiin sopiva jakaja (ks. liite II). Jakajan avulla täytettävän alueen ruutujen järjestysnumeroista otetaan jakojäännös, jonka perusteella alueen pakkaus suoritetaan. Kuvassa 6.2 on luvun alussa esitetty esimerkkitaulukko uudelleen numeroituna jakajalla 11. Jakajan valintaan vaikuttaa erityisesti pakkauksessa käytettävien polyominoiden asteluku sekä täytettävän alueen leveys. Alueen numeroimiseen käytettävän jakajan tulee olla sellainen, että mikään yksittäisistä alueen täyttämiseen käytettävä polyominoista ei täytettävälle alueelle asetettuna voi peittää kahta tai useampaa samalla jakojäännöksellä numeroitua ruutua. Kuvassa 6.3 on vahvennetulla viivalla rajattu lohko, jonka sisään on mahdollista

sijoittaa mikä tahansa yksittäinen 4-omino ilman, että yksikään alueen numeroimiseen käytetty lukuarvo tulee peitettyä kahdesti. Koska levein n-omino on aina n ruutua leveä ja vastaavasti korkein n-omino n ruutua korkea, alueen jakajaksi soveltuva luku on likimain lohkon ruutujen lukumäärän $a = (n/2) * (n - 1)$ suuruinen. Kuten kuvassa 6.2 alue on numeroitu järjestyksessä luvuilla nollasta kymmeneen, jolloin alueen ruutujen indeksiksi tulee jakojäännös 11:sta. Myös muita numerointivaihtoehtoja on olemassa, sillä jakajana voisi toimia myös muun muassa jokin luvuista 13, 18, 19 tai 20. Kuten kuvan 6.3 esimerkissä, sama luku voi esiintyä rajatun lohkon sisällä useampaan kertaan, kunhan luvun ilmentymien etäisyys toisiinsa on suurempi kuin käytettyjen polyominojen asteluku. Koska koko täytettävä alue on numeroitu järjestyksessä jatkuvalla, kiertävällä numeroinnilla, yksittäiselle lohkolle esitetyt ominaisuudet pätevät jokaisella lohkojaolle koko numeroidulla alueella.

									mod
	A	B	C	D	E	F	G	H	11
A	0	1	2	3	4	5	6	7	
B	8	9	10	0	1	2	3	4	
C	5	6	7	8	9	10	0	1	
D	2	3	4	5	6	7	8	9	
E	10	0	1	2	3	4	5	6	
F	7	8	9	10	0	1	2	3	
G	4	5	6	7	8	9	10	0	
H	1	2	3	4	5	6	7	8	

Kuva 6.2 Alkuperäinen $8 * 8$ ruudun kokoinen alue numeroitu uudelleen modulo 11.

									mod
	A	B	C	D	E	F	G	H	11
A	0	1	2	3	4	5	6	7	
B	8	9	10	0	1	2	3	4	
C	5	6	7	8	9	10	0	1	
D	2	3	4	5	6	7	8	9	
E	10	0	1	2	3	4	5	6	
F	7	8	9	10	0	1	2	3	
G	4	5	6	7	8	9	10	0	
H	1	2	3	4	5	6	7	8	

Kuva 6.3 Esimerkki $8 * 8$ ruudun kokoisen alueen numeroinnista 4-ominoille.

6.1.2 Polyominojen muuntaminen bittiesitysmuotoon

Alueen numeroinnin perusteella on mahdollista muuttaa alueen täyttämisessä käytettävät polyominot bittiesitysmuotoon. Polyominojen bittiesitysmuodossa polyominoille muodostetaan bittijono, joka sisältää aiemmin lasketun alueen numeroimisessa käytettävän lukumäärän bittejä, kuvan 6.3 esimerkkitapauksessa bittejä on 11. Jokaisen polyominon biteistä asetetaan päälle ne jakojäännöslukujen mukaiset bitit, jotka kyseinen polyomino peittää asetettuna täytettävälle alueelle. Päälle asetetut bitit merkitsevät samaan aikaan sekä polyominon muotoa että sen paikkaa täytettävällä alueella. On kuitenkin huomattava, että esitysmuoto ei ole täysin yksiselitteinen. Jokainen alueen numeroinnin yhteydessä muodostettu lohko esiintyy useaan kertaan täytettävällä alueella. Toistuvan numeroinnin seurauksena polyominon sijainti täytettävällä alueella voi olla lähes mikä tahansa yhtenevien lohkojen esiintymistä. Numeroinnista johtuen myös polyominojen muoto ei ole aina täysin yksiselitteinen, sillä bittijonossa ei ole mahdollista määritellä bittien indeksien järjestystä. Lukuarvon bittien $\{0, 8, 9, 10\}$ ollessa asetettuna tulos on sama kuin biteillä $\{8, 9, 10, 0\}$. Selkeyden vuoksi polyominoja kuvaavat luvut esitetään vastedes pääasiassa binäärimuodossa, joka tämän esimerkin tapauksessa on 11100000001_2. Oletetaan toistaiseksi, että sekä polyominojen paikkaan että muotoon liittyvät ongelmat voidaan ratkaista muodostamalla kaksi tai useampia erillistä aluejakoa eri jakojäännöksillä siten, että valittujen jakajien toisistaan eroavien tekijöiden tulo olisi suurempi kuin täytettävän alueen pinta-ala. Polyominojen muodon monitulkinnallisuus voidaan ratkaista myös ylläpitämällä tilatietoa polyominojen muodon ja bittiesityksen välillä. Molempien ongelmien tarkempi tutkiminen sivuutetaan toistaiseksi toteamalla, että ongelma on ratkaistavissa useiden toisistaan poikkeavien tarkistussummien vertailulla. Ongelman ratkaisu käsitellään tarkemmin myöhemmässä alakohdassa 6.2.2.

6.2 Moduloidun bittikentän toimintaperiaate

Kuten aiemmin mainittiin bittikenttien yhteydessä, bittiesitysmuodon parhaita puolia on se, että kappaleita voidaan siirtää täytettävällä alueella käyttäen yksinkertaisia bittioperaatioita. Bittiesityksen 00000001111_2 mukaista polyominoa voidaan siirtää täytettävällä alueella käyttäen *bittisiirto-operaatiota, (engl. bit shift)*, jolloin polyominon bittiesitys muuttuu muotoon 00000011110_2. Verrattaessa bittiesitysten mukaisia polyominoja tasokuviona aiemmin muodostetuilla lohkoilla voidaan huomata, että polyomino on bittisiirrosta huolimatta säilyttänyt

		0	1	2	3	4	5	6	7	8	9	10	X
1	J	1								1	1	1	4
2	I	1	1	1	1								3
3	T	1	1	1							1		3
4	L			1						1	1	1	4
5	O	1	1							1	1		5
6	Z		1				1			1	1		4
7	T	1					1			1	1		3
8	I	1	1	1	1								4
9	J		1				1	1			1		4
10	Z		1				1			1	1		4
11	L	1	1	1						1			3
12	S		1	1						1	1		4
13	J	1								1	1	1	3
14	S		1	1						1	1		3
15	O	1	1							1	1		4
16	I	1	1	1	1								4
Total		**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**5**	**5**	**64**
Count		10	12	8	3	0	4	1	0	11	12	3	**64**
Missing		-4	-6	-2	3	6	2	5	6	-5	-7	2	**0**

Kuva 6.4 Esimerkkitaulukko $8 * 8$-kokoisen alueen 4 ruudun mittaisten polyominojen bittiesityksestä.

alkuperäisen muotonsa ja ainoastaan sen sijainti täytettävällä alueella on vaihtunut. Perinteisessä bittisiirto-operaatiossa bittiesityksen päätyjen yli siirtyvät bitit katoavat. Tässä algoritmissa käytetään perinteisestä bittisiirrosta hieman poikkeavaa toimintalogiikkaa. Algoritmin bittisiirto-operaationa käytetään niin sanottua *kiertävää bittisiirtoa (engl. bit shift with roll-over bits)*, jolloin esityksen ulkopuolelle siirtyvät bitit kierretään omille vastaaville paikoilleen bittijonon toiseen päähän. Esimerkkinä kiertävästä bittisiirrosta aiemmin esitetty J-tetrominon, bittiesitykseltään 11100000001_2, muuttuu muotoon 11000000011_2. Koska alueen ruudut on numeroitu järjestyksessä ja numerointi on toteutettu jakojäännösoperaatiolla, voidaan todeta, että kuvattu bittisiirto-ominaisuus pätee koko numeroidulla alueella. Voidaan myös todeta tunnetuksi, että kiertävälle bittisiirto-operaatiolle on olemassa vakiollisen suoritusajan käänteisoperaatio, jota voidaan tarvittaessa käyttää palauttamaan algoritmi takaisin alkutilaan. Esimerkin tapauksessa moduloidun bittikentän menetelmässä yksittäisten polyominojen sijaintivaihtoehtojen lukumäärä saatiin vähenemään alkuperäisestä 64:stä 11:een. Moduloidun bittikentän menetelmän mukaiset tulokset pitää vielä tarkistaa vähintään toisella jakojäännöslohkolla ennen lopullisia tuloksia, joten mene-

telmien välinen ero ei ole näin merkittävä. Menetelmien välisestä tehoerosta lisää kohdassa 6.3.

6.2.1 Algoritmin tavoite

Kun koko täytettävä alue on numeroitu jakojäännösluvuilla, algoritmin alustuksessa voidaan laskea jokaisen alueen numeroivan arvon esiintymiskerrat. Kuvassa 6.4 annetaan muodostettujen lukuarvojen sekä esimerkkipolyominojen perusteella muodostetut bittijonot algoritmin suorituksen alkutilassa. Kuvan alareunassa nähdään käytettyjen lukujen esiintymiskerrat koko täytettävällä alueella. Samoin kuvassa on esitetty laskurit, jotka esittävät kappaleiden pakkauksen mukaista tilannetta. Esimerkissä luvun 0 tulisi Total-sarakkeen mukaisesti esiintyä taulukossa 6 kertaa. Koska algoritmi on yhä alkutilanteessa, luku 0 esiintyy yhteensä Count-sarakkeen mukaiset 16 kertaa. Missing-sarakkeen osoittamat 10 kappaletta luvun 0 esiintymää pitäisi siirtää täyttämään muita sarakkeita. Algoritmin ensimmäinen tavoite on siirtää polyominojen bittiesityksiä taulukossa siten, että lukuarvojen esiintymiskerrat täsmäisivät alkuperäisen täyden numeroidun taulukon kanssa. Taulukon mukaisten esiintymiskertojen avulla koko täytettävälle alueelle sekä polyominoille voidaan laskea tarkistussummat, joiden perusteella algoritmi voi suorittaa tarkistuksen koko pakkauksen valmistumiselle. Polyominojen bittiesitysten tapaan koko alueen tarkistussumma voidaan tulkita bittiesityksenä, jolloin alueen tarkistussumman arvo voidaan laskea summakaavalla

$$\sum_{n=0}^{A-1} 2^{(n \bmod i)}, \tag{6.1}$$

missä A on täytettävän alueen pinta-ala ja i on alueen numerointiin valittu jakaja. Toinen tapa laskea täytettävän alueen tarkistussumma on laskea alueen ruutujen summa bittiesitysten avulla. Koska tiedetään, että jokaisen alueella sijaitsevan ruudun tulee olla täytetty, voimme laskea saman koko alueen mukaisen tarkistussumman myös muodossa

$$\frac{A - (A \bmod i)}{i} * (2^i - 1) + 2^{(A \bmod i)} - 1. \tag{6.2}$$

Tämän luvun esimerkissä tarkistussummaksi tulee kaavan 6.2 mukaisesti $5_{10} * 11111111111_2 + 00111111111_2 = 10746_{10}$. Selvennyksenä mainittakoon, että kaavassa 6.2 laskutoimitus $2^i - 1$ muodostaa luvun, jonka binäärimuotoisessa esityksessä i ensimmäistä bittiä on asetettu päälle. Kyseinen laskutoimitus on usein

käytössä tietotekniikan binäärioperaatioiden yhteydessä *bittimaskin (engl. bitmask)* muodostamisessa. Bittimaskien käytön yhteydessä on hyvä huomata, että 64-bittisissä ympäristöissä lukuarvo 2^{64} voi olla määrittelemätön, sillä kyseinen lukuarvo voi aiheuttaa ylivuodon tietokoneen muistissa ja tulos voi olla nolla tai jopa määrittelemätön. Tällön myös laskutoimituksen $2^{64} - 1$ tuloksesta ei voida olla täysin varmoja.

Kuvassa 6.5 on esimerkki valmiiksi ratkaistusta taulukosta. Taulukon mukainen paikkatieto polyominoille ei ole yksiselitteinen, joten samalla polyominojoukolla on mahdollista muodostaa lukuisia muita potentiaalisia ratkaisuvaihtoehtoja kuten esimerkissä 6.6. Vaikka kaikki lukuarvot saataisiin täsmäämään, polyominojen pakkaaminen tasolle ei ole vielä täysin valmis. Aiemmin mainituista monitulkinnaisuuksista johtuen on mahdollista, että osa polyominoista on muodoltaan vääränlaisia pakattavaan alueeseen nähden. Osa polyominoista saattaa myös olla asetettu täytettävän alueen reunan päälle, kuten kuvassa 6.6 esitetty viivalla rajattu valkopohjainen alue taulukon vasemmalla ja oikealla reunalla.

		0	1	2	3	4	5	6	7	8	9	10	X
1	J	1								1	1	1	4
2	I		1	1	1	1							3
3	T				1		1	1	1				3
4	L	1						1	1	1			4
5	O		1	1							1	1	5
6	Z	1	1			1				1			4
7	T			1	1		1					1	3
8	I					1	1	1	1				4
9	J			1	1			1			1		4
10	Z	1				1			1	1			4
11	L		1	1	1						1		3
12	S	1	1			1	1						4
13	J								1	1	1	1	3
14	S		1	1			1	1					3
15	O	1							1	1		1	4
16	I				1	1	1	1					4
Total		6	6	6	6	6	6	6	6	6	5	5	64
Count		6	6	6	6	6	6	6	6	6	5	5	64
Missing		0	0	0	0	0	0	0	0	0	0	0	0

Kuva 6.5 Esimerkki valmiiksi täytetystä 8x8-kokoisesta taulukosta aiemmin esitellyssä tilanteessa.

6.2.2 Algoritmin tulosten monitulkinnaisuuden selvittäminen

Algoritmin ensimmäisessä vaiheessa polyominojen bittimuotoiset esitykset sovitetaan numeerisesti taulukkoon siten, että jokaisessa sarakkeessa on oikea lukumäärä bittejä. Lopullinen tehtävä ei kuitenkaan ole vielä ratkaistu, sillä menetelmällä saatu välitulos on tietorakenteen toiminnasta johtuen monitulkinnainen. Kuvassa 6.6 on esitetty J-mallisen tetrominon mahdolliset vaihtoehtoiset sijainnit esimerkkiratkaisussa 6.5. Tämänkaltaisia ongelmia voidaan eliminoida muilla vaihtoehtoisilla jakajilla muodostetulla tarkistussummilla. Toinen laskenta on ensimmäistä laskentaa huomattavasti nopeampi suorittaa, sillä toiseen laskentaan voidaan käyttää hyväksi ensimmäisen laskennan tuloksia. Osa polyominojen koodauksen monitulkinnallisuuden aiheuttamista ongelmista voidaan korjata siirtämällä kaikkia polyominoja sama askelmäärä samaan suuntaan, jolloin polyominojen yhteenlaskettu tarkistussumma säilyy samana. Toinen mahdollisuus korjata monitulkinnaisuudesta aiheutuneita ongelmia on vaihtaa saman bittiesityksen omaavien, mutta keskenään erilaisten kappaleiden paikkoja keskenään.

									mod
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	11
A	0	1	2	3	4	5	6	7	
B	8	9	10	0	1	2	3	4	
C	5	6	7	8	9	10	0	1	
D	2	3	4	5	6	7	8	9	
E	10	0	1	2	3	4	5	6	
F	7	8	9	10	0	1	2	3	
G	4	5	6	7	8	9	10	0	
H	1	2	3	4	5	6	7	8	

Kuva 6.6 Esimerkki polyominon monitulkinnaisuudesta bittitaulukossa.

6.3 Moduloidun bittikentän aikakompleksisuus

Algoritmin tulosten monitulkinnaisuudesta johtuen alueelle joudutaan laskemaan vähintään kaksi erillistä tarkistussummaa. Algoritmin kannalta optimaalisin tilanne syntyy, kun koko alue saadaan jaettua lohkoihin siten, että jakajien tulo on täsmälleen koko alueen pinta-alan suuruinen. Kuudenkymmenenneljän ruudun kokoisen alueen esimerkissä jakajiksi a_n sopivat aiempien esimerkkien mukaisesti pienimmät alueen jakamisen ehdot täyttävät luvut $a_1 = 11$ ja $a_2 = 13$. Valittujen lukujen tulon on oltava suurempi tai yhtä suuri kuin täytettävän alu-

een pinta-ala, jolloin lukujen voidaan sanoa olevan likimain $a = \sqrt{A}$ suuruisia. Moduloidun bittikentän menetelmässä tarkistussummia laskettaessa kappale voi olla siirretty mihin tahansa bittikentän indekseistä. Tällöin kaikkien tarkistussummavaihtoehtojen lukumäärä yhdellä laskentakierroksella on a^c. Toisella tarkistuskierroksella jokainen ensimmäisellä kierroksellä täsmännyt tarkistussumma joudutaan tarkistamaan vastaavalla laskutoimituksella lopullisen tuloksen saamiseksi. Tällä perusteella algoritmin huonoimman vaihtoehdon aikakompleksisuus on $a^{2c} = (\sqrt{A})^{2c} = A^c$.

Algoritmin todellinen suoritusaika riippuu kuitenkin hyvin suuresti aiemmilla tarkistuskierroksilla löytyneistä tarkistussummien täsmäyksistä. Mikäli oletetaan, että kaikki ensimmäisen tarkistuskierroksen vaihtoehdot eivät täsmää alueen tarkistussummaan, voidaan tehdä muutamia havaintoja tarkistussummien täsmäysten enimmäismäärään. Oletetaan löydetyksi yksi kappaleiden tarkistussumma, joka täsmää ensimmäisen lohkon tarkistussummaan. Mikäli mitä tahansa yksittäistä polyominoa siirretään vapaavalintainen askelmäärä silloiselta paikaltaan tarkistussumma voi täsmätä ainoastaan silloin kun siirretty polyomino ei muuta tarkistussummansa arvoa. Polyominon tarkistussumman arvo voi pysyä muuttumattomana ainoastaan silloin, kun polyominon määrittelemät bitit ovat jakautuneet symmetrisesti polyominon bittiesityksen alueelle. Symmetrisesti jakautuneita kahdeksan bitin mittaisia bittiesityksiä ovat muunmuassa 01010101_2, 00110011_2 sekä 11110000_2. Ainoat tapaukset, jolloin polyominon bittiesitys ei voi muuttua minkään bittisiirron seurauksena, ovat sellaisia, joissa kaikki polyominon bittiesityksen bitit ovat joko asetettu päälle tai vastaavasti pois päältä. Mikäli kaikki polyominon bitit olisi asetettu pois päältä polyomino ei bittiesityksen mukaan koostuisi yhdestäkään neliöstä ja polyominon ei olisi kohdan 2 määritelmän mukainen. Vastaavasti mikäli kaikki bittiesityksen bitit olisi asetettu, polyominon siirtäminen alueella olisi hyödytöntä, sillä tarkistussumma pysyisi joka tapauksessa vakiona. Näillä perusteilla ensimmäisen tarkistuskierroksen vaihtoehdoista enintään $\frac{1}{2*c}$ osuus voi täsmätä koko alueen tarkistussummaan. Mikäli yksikään polyomino ei ole aiemmin kuvatulla tavalla symmetrinen bittiesityksen suhteen, mahdollisten tarkistussummien täsmäysten osuus pienenee $\frac{1}{a*c}$ osaan kaikista vaihtoehdoista. Merkitsevin tekijä algoritmin suoritusajan kannalta on kuitenkin funktion eksponentissa oleva kappaleiden lukumäärä c. Esitetyn kaltainen siirtovaihtoehtojen rajaaminen pienentää funktion kantalukua, mutta kasvattaa eksponenttia. Algoritmisessa mielessä kyseinen toimenpide kasvattaa algoritmin kompleksisuutta, mutta käytännön tilanteessa, hyvin toteutettuna algoritmin todellinen ajankulutus saattaa vähentyä. Mikäli eksponentin kantalu-

kua saadaan pienennettyä riittävästi ja ensimmäisten kierrosten tarkistussummat täsmäävät alueen tarkistussummaan erityisen harvoin tai ei ollenkaan, pakkaus voidaan todeta jopa mahdottomaksi, jolloin lisälaskentaa seuraavalle tarkistuskierrokselle siirtyy hyvin vähän tai pakkauksen ollessa mahdoton, ei ollenkaan. Algoritmin todellista ajankulutusta on vaikea arvioida, sillä ensimmäisen kierroksen tarkistussummien täsmääminen alueen tarkistussummaan riippuu suuresti täytettävästä alueesta sekä alueelle asetettavista kappaleista.

6.4 Puutteita ja rajoitteita

Buschen [2011] mukaan bittikenttiin perustuvaa menetelmää on käytetty aiemmin polyominotehtävien ja muiden vastaavan kaltaisten tehtävien yhteydessä paikallisten ongelmallisten alueiden havaitsemiseen. Buschen [2011] esittämässä menetelmässä yksittäisten pelialueen ruutujen tila tallennettiin bittikenttään. Buschen mukaan menetelmä toimi erittäin tehokkaasti, mutta nykytietokoneiden 64-bittisestä arkkitehtuurista johtuen menetelmä rajoittui tarkastelemaan enintään 64 ruudun kokoista aluetta, mikä rajoitti huomattavasti menetelmän käyttöä. Esittämälläni moduloidun bittikentän menetelmällä tehdyt alustavat kokeet osoittavat, että menetelmäni toimivuutta ei juurikaan rajoita tarkasteltavan alueen pinta-ala, vaan alueen täyttöön käytettävien polyominojen asteluku sekä lukumäärä. Myös moduloidun bittikentän menetelmän käytön rajoite liittyy nykyaikaisten tietokoneiden 64-bittiseen arkkitehtuuriin. Mikäli alueen lohkon koko kasvaa liian suureksi, on epätodennäköistä tai jopa mahdotonta numeroida aluetta algoritmin vaatimalla tavalla. Algoritmia käytettäessä polyominojen asteluku n rajoittuu alueen jaollisuudesta riippuen enintään yhdeksästä yhteentoista. Polyominojen asteluvun kasvaessa yksittäisen alueen lohkon pinta-ala a kasvaa, jolloin lohkon numerointiin tarvittavien lukujen kaksoiskappaleiden esiintymisen todennäköisyys lisääntyy. Moduloidun bittikentän algoritmi mahdollistaa bittikentän koon kasvattamisen käyttäen useampia lukuja bittikenttien tallentamiseen. Usean luvun käyttäminen bittikenttiä kohden kuitenkin hidastaa bittisiirtojen ja tarkistussummien laskemista huomattavasti, jolloin algoritmi ei enää toimisi yhtä tehokkaasti.

Moduloidun bittikentän paras puoli on se, että täytettävän alueen koko ei juurikaan rajoita algoritmin toimintaa. Tekemäni testiohjelma (ks. liite I) pystyi löytämään sopivia alueen jakamiseen käytettäviä jakajia jopa 200 ruutua leveille alueille, jolloin alueen koko voisi olla jopa useiden tuhansien ruutujen kokoinen. Jokaiselle alle 50 ruutua leveälle alueelle löytyi useita, jopa kymmeniä sopivia ja-

kajia. Sopivien jakajien löytämistä suurille polyomille vaikeuttaa se, että jakajien tulee olla käytettävän tietokoneen muistiarkkitehtuurin suuruinen, eli nykytietokoneilla enintään 64. Tilanne paranee hieman, mikäli tulevaisuudessa tietokoneiden arkkitehtuuri mahdollistaa tietokoneen perusoperaatiot nykyistä suuremmille lukuarvoille. Rajoitteen aiheuttaja on sama kuin Buschen [2011] esittelemässä bittikenttämenetelmässä, jossa rajoitteen kohteena on pakattavan alueen pinta-ala. Moduloidun bittikentän menetelmässä bittien lukumäärä rajoittaa alueelle muodostettavien lohkojen jakajaa. Jakaja kasvaa polyominojen asteluvun, mutta vain epäsuorasti täytettävän alueen koon suhteen suhteen, jolloin lohkon koon voidaan arvioida olevan

$$(n/2) * (n-1) \approx a \tag{6.3}$$

suhteessa polyominoiden astelukuun n. Kaavan 6.3 mukaan lohkon pinta-alan a ollessa enintän 64 seuraa $n < 12$. Arvio lohkon jakajan suuruusluokalle perustuu kuvassa 6.3 rajatun esimerkkialueen mukaisen pinta-alan kasvufunktioon n-ominoille.

7 Hajota ja hallitse -menetelmä

Tässä luvussa esitellään muunnos moduloidun bittikentän menetelmään, jonka ansiosta moduloidun bittikentän lohkojakoa voidaan pienentää entisestään ja menetelmä saadaan sovitettua toimivaksi hajota ja hallitse -periaatteen mukaiseen binääriseen hakupuuhun. Hajota ja hallitse -menetelmässä pakkaustehtävä jaetaan mahdollisimman pieniin osaongelmiin eli lohkoihin, jotka yhdessä tuottavat ratkaisun alkuperäiselle ongelmalle [Aho et al., 1983]. Alkuperäisessä moduloidun bittikentän menetelmässä toimittiin hieman vastaavasti, sillä pakattavalle alueelle sekä polyominoille muodostettiin jakajan avulla tarkistussummat, joilla alueen pakkaaminen saatiin jaettua kahteen helpompaan osaongelmaan. Hajota ja hallitse -menetelmässä pienten muutosten avulla täytettävä alue pyritään jakamaan mahdollisimman pieneniin osiin, jolloin pakkaaminen tehostuu entisestään, koska lohkojaon avulla epäsopivat pakkausyritykset havaitaan ja jätetään tarkastelun ulkopuolelle mahdollisimman aikaisessa pakkausprosessin vaiheessa.

Menetelmän toimintaa voi kuvata esimerkillä, jossa yritetään arvata satunnaisesti valittua lukua ennalta sovitulta lukuväliltä. Esimerkissä avustajaa pyydetään valitsemaan jokin luku väliltä $[0..A]$. Avustaja ei kuitenkaan kerro lukua kenellekään, joten luku joudutaan selvittämään arvaamalla. Luvun valitsemisen lisäksi avustajan tehtävä on vastata esitettyihin arvauksiin pelkästään kyllä tai ei. Arvaamalla yksittäisiä lukuarvoja arvauksia tarvittaisiin pahimmassa tapauksessa A ja keskimääräisesti $\frac{A}{2}$ kappaletta valitun luvun löytämiseksi. Moduloidun bittikentän menetelmään toteutettujen muutosten avulla pelkkien yksittäisten lukujen arvaamisen sijaan on mahdollista muotoilla jokainen kysymys uudelleen muotoon "Onko i valitun luvun jakojäännös, kun valittu luku jaetaan luvulla a?". Menetelmän mukaisella kysymyksenasettelulla lukualue A voidaan jakaa jokaisella kysymyksellä kahteen yhtäsuureen osaan. Liitteessä III esitetään binääripuu uudistetun kysymysasettelun mukaisen yksittäisen luvun arvaamiselle. Moduloidun bittikentän menetelmässä binääripuu vastaa yksittäisen polyominon sijainnin löytämistä pakattavalta alueelta. Muutoksen seurauksena luvun selvittämiseen tarvittavien arvausten lukumäärän y kasvunopeus, suhteessa kaikkien mahdollisten lukujen määrään, muuttuu lineaarisesta $y = A$ logaritmiseksi $y = log_2(A)$.

Moduloidun bittikentän menetelmällä suoritettua polyominojen pakkausta tasolle ei suinkaan pystytä suorittamaan logaritmisessa ajassa. Alueen jakaminen kuvatun kaltaisesti kahteen yhtäsuureen osaan muuttaa menetelmän lohkojakojen lukumäärän logaritmiseksi suhteessa koko pakattavan alueen pinta-alaan. Samalla tarkastelun yhteydessä jokainen lohko kasvatetaan aina kaksinkertaiseksi edelli-

seen verrattuna. Yksittäisen lohkon kaikkien polyominojen sijaintivaihtoehtojen lukumäärä on 2^c, jolloin koko menetelmän kaikkien pakkausvaihtoehtojen tarkastelun aikakompleksisuus on $2^{c*log_2(A)}$. Koska $2^{log_2(A)} = A$ menetelmän aikavaatimus algoritmisesesa mielessä on täsmälleen sama kuin alkuperäinen A^c. Moduloidun bittikentän menetelmän etu aiempiin pakkausmenetelmiin tulee siitä, että jokaisella lohkojaolla suuri osa pakkausvaihtoehdoista voidaan todeta epäsopiviksi. Seuraavilla tarkistuskierroksilla tarvitsee tarkistaa pelkästään aiemmilla lohkojaoilla sopineet pakkaukset, ja suuri osa myöhempien lohkojakojen pakkausvaihtoehdoista voidaan sivuuttaa epäsopivina.

7.1 Polyominojen tarkistussummien muodostaminen

Tarkastellaan, mitä moduloidun bittikentän mukaisille tarkistussummille tapahtuu, jos lohkojen jakajina käytettäisiin aiempaa pienempiä lukuja välittämättä siitä, että saman bitin indeksi esiintyy polyominon bittiesityksessä useammin kuin kerran. Kuvassa 7.1 esitetään aiemmista luvuista tuttu polyominoilla täytetty alue, joka on indeksoitu uudelleen modulo 4.

									MOD
	A	B	C	D	E	F	G	H	4
A	0	1	2	3	0	1	2	3	
B	0	1	2	3	0	1	2	3	
C	0	1	2	3	0	1	2	3	
D	0	1	2	3	0	1	2	3	
E	0	1	2	3	0	1	2	3	
F	0	1	2	3	0	1	2	3	
G	0	1	2	3	0	1	2	3	
H	0	1	2	3	0	1	2	3	

Kuva 7.1 Esimerkki 8x8 ruudun kokoisen alueen numeroinnista 4-ominoille.

Otetaan lähempään tarkasteluun kuvan 7.1 mukainen J-tetriminon bittiesitys, jossa 0-bitti on asetettu kaksi kertaa sekä 1 ja 2 bitit yhteen kertaan. Tällöin luvun bittiesityksen tulisi olla "0112_2", mikä ei ole binääriselle esitykselle mahdollista, sillä binääriluvut koostuvat pelkästään luvuista 0 ja 1. Huomionarvoista kuvitteellisessa bittiesityksessä "0112_2"on se, että bittisiirto-operaatio kyseisellä kuvitteellisella esityksellä voisi toimia tavanomaiseen tapaan. Miten kuvitteellinen luku "0112_2"pitäisi tulkita? Binääriesitys muodostetaan kahden potensseista, jolloin "0112_2"olisi kymmenkantaisena kokonaislukuna $0*2^3+1*2^2+1*2^1+2*2^0 =$

$0 + 4 + 2 + 2 * 1 = 8_{10}$. Kymmenkantaisesta kokonaisluvusta binääriseksi kokonaislukuesitykseksi muuttamalla saadaan $8_{10} = 1000_2$, jolloin palautus takaisin alkuperäiseen kaksikantaiseen bittiesitykseen ei ole enää yksiselitteinen sillä $8_{10} = 1000_2 =$ "0112_2". Binäärijärjestelmässä kokonaislukua 2_{10} merkitään käyttämällä kahta tai useampaa bittiä jokaista lukuarvoa kohden. Kahdella bitillä kuvattuna 2_{10} voidaan esittää $10_2 = 1 * 2^1 + 0 * 2^0 = 2_{10}$. J-tetriminon bittiesitys voidaan siis muuttaa esitysmuotoon, jossa jokaista esityksen lukuarvoa merkitään aina kahdella bitillä. Kahdella bitillä lukua kohden uudeksi J-tetriminon arvoksi tulee "00, 01, 01, 10"eli tiivistettynä 00010110_2.

Kahden bitin esitysmuodossa kappaleiden siirtäminen ja tarkistussummien laskeminen toimii pääosin samoin kuin alkuperäisessä moduloidun bittikentän menetelmässä. Bittisiirtojen yhteydessä bittejä siirretään kuitenkin aina kaksi bitti-indeksiä kerrallaan aiemman yhden sijaan. Menetelmää varten koko täytettävä alue sekä jokainen alueen täyttämiseen käytettävä polyomino muunnetaan vastaavaan bittiesitysmuotoon. Kahden bitin lukuesityksellä voidaan numeroida enintään 3 saman lukuarvon esiintymää. Mikäli jokin polyomino tai alueen lohko peittää useamman kuin kolme samaa indeksilukua samanaikaisesti, algoritmissa voidaan käyttää useampia bittejä lukuarvoja kohden. Indeksilukujen esiintymien lukumäärä suhteessa tarvittavien bittien lukumäärä voidaan esittää kaavalla $2^b - 1$. Lukuarvoissa käytetty bittien lukumäärä ei vaikuta algoritmin suoritusnopeuteen, sillä muistipaikan sisäinen bittisiirto-operaatio on toteutettavissa vakiollisen ajan operaationa riippumatta siirrettävien bittien lukumäärästä. Yksittäisen lohkon merkitsemiseen voidaan kuitenkin käyttää enintään tietokoneen arkkitehtuurin mukainen bittimäärä. Yksittäisen lohkon numerointiin ja indeksin esiintymille varattu bittien lukumäärän tulo saa täten olla enintään tietokoneen muistipaikan bittisyyden suuruinen. Nykytietokoneiden muistiarkkitehtuuri on yleisesti 64 bittiä. Suurimmat moduloidun bittikentän mukaiset lohkokoot on esitetty liitteessa IV. Lohkokoon rajoitetta voidaan kiertää käyttämällä useampia muistipaikkoja yksittäisten lohkojen ja polyominojen merkitsemiseen mutta silloin lohkojen tarkistussummien laskenta hidastuu merkittävästi.

7.2 Täytettävän alueen numeroiminen

Perinteisessä bittikentässä tarkasteltavaa alueen numeroimista rajoittaa käytettävän tietokoneen arkkitehtuurin yksittäisen muistipaikan koko, joka nykyaikaisilla tietokoneilla on 64 bittiä. Suurempien alueiden tarkastelussa joudutaan turvautumaan useamman lukuarvon käyttämiseen, mikä hidastaa tarkistussummien

laskemista. Alkuperäisen moduloidun bittikentän menetelmän käyttöä rajoittaa se, että menetelmän mukaisia koko alueen osalohkoja ei voi muodostaa polyominoille, joiden asteluku n on suurempi kuin 11. Rajoite alueen lohkokoolle tulee siitä, että jokainen alueen osalohko tulee olla numeroitavissa siten, että lohkon pinta-ala $a = (n-1) * (n/2) \approx 64$. Likiarvoyhtälön mukaan kappaleiden suurin mahdollinen asteluku n on likimain 11. Täytettävän alueen pinta-ala on myös rajoitettu moduloidun bittikentän menetelmässä. Alueen kokoa rajoittaa menetelmän mukaisten eri osalohkojen jakajien yksilöivien tekijöiden keskenäinen tulo. Sopivia moduloidun bittikentän menetelmän mukaisia lohkon jakajia ja täten myös yksilöiviä tekijöitä löytyy kuitenkin pääsääntöisesti niin runsaasti, että alueen pinta-ala voi olla jopa tuhannen ruudun kokoinen.

Alueen jakaminen entistä pienempiin lohkoihin pienentää moduloidun bittikentän menetelmässä polyominojen asteluvulle muodostunutta rajoitetta. Useampaa bittiä käytettäessä yksittäisten polyominojen tai lohkojen tarkistussummat saavat yhä sisältää enintään 64 bittiä. Kuten aiemmin esitettiin bittien jako tarkistussummassa voidaan toteuttaa siten, että yksittäisten indeksilukujen esiintymien laskemiseen käytetään aiempaa useampia bittejä. Tällöin yksittäiset polyominojen numeroimiseen käytettävät indeksit voivat esiintyä lohkojaon alueella useaan kertaan. Kuten moduloidun bittikentän menetelmässä alueen jakamiseen käytettävien lukujen yksilöivien tekijöiden tulon tulee olla suurempi tai yhtä suuri kuin täytettävän alueen pinta-alaan verrattuna kaikkien pakkausvaihtoehtojen läpikäymiseksi. Mikäli yksittäisen alueen jakajaksi valitaan seitsemän ja indeksien esiintymisen laskemiseen käytetään kolme bittiä jokaista indeksilukua kohden, yksittäisen tarkistussumman muodostamiseen tarvitaan $7 * 3 = 21$ bitin mittainen lukuesitys. Kyseisellä tarkistussummalla voidaan numeroida noin $7 * (2^3 - 1) = 7 * 7 = 49$ ruudun suuruinen lohko a_1. Tällöin alueen täyttämiseen käytettävien polyominojen asteluku on kaavan 6.3 mukaisesti noin 9. Vastaavasti aiemmin esitellyssä moduloidun bittikentän menetelmässä 21 bitillä voitiin numeroida noin 21 ruudun kokoinen lohko a_2, jolloin polyominojen asteluku olisi samalla bittimäärällä kaavan 6.3 mukaisesti vain noin 6. Alueelle sopivan jakajan ja samalla suurimman käytettävissä olevan polyominon asteluvun määrittely bittien lukumäärän perusteella riippuu myös käytettävän alueen leveydestä, joten tarkan lohkojaon selvittäminen on tilannekohtaista.

Useamman bitin käyttö jokaista polyominon lukuarvoa kohden rajoittaa lohkon numeroinnissa käytettävän jakajan kokoa. Koska tarkistussummien muodostamiseen voidaan käyttää yhä enintään 64 bittiä, niin käyttämällä 2 bittiä lukua kohden lohkonjakaja voi olla enintään 32. Koska jokainen lukuarvo voi

esiintyä alueella kahden bitin lukuesityksessä enintään kolme kertaa koko numeroitavan alueen enimmäiskoko on $32 * (2^2 - 1) = 32 * 3 = 96$. Taulukko enintään 64 bittisillä luvuilla muodostettavista lohkojakojen kokorajoitteista esitetään liitteessä IV.

Samoin kuin moduloidun bittikentän menetelmässä, lohkon tarkistussummana toimii täytettävään alueeseen kuuluvien ruutujen indeksien summa. Koko lohkon tarkistussumman mukaisessa kaavassa

$$\sum_{n=0}^{A-1} 2^{2*(n \bmod (i))} \tag{7.1}$$

yhteenlaskettavien lukujen eksponentti on kerrottu kahdella vastaamaan kahta bittiä lukua kohden. Kuvassa 7.1 esitetyn alueen tarkistussummaksi tulee $16 * 01010101_2 = 1360_{10}$. Kuvan 7.1 esimerkissä tulee esiin eräs mielenkiintoinen asia polyominojen bittisiirtoihin liittyen. Osa esimerkin polyominoista sattuu peittämään jokaisen alueen numeroinnissa käytetyn luvun kertaalleen, jolloin kyseisten kappaleiden bittimuotoinen esitys kahdella bitillä lukua kohden on 01010101_2. Kiertävä bittisiirto kaksi askelta kerrallaan kyseisillä luvuilla palauttaa täsmälleen alkuperäisen luvun, joten tällaisten polyominojen sijainneista alueella ei saada lisätietoa. Polyomino voidaan jättää tarkistamatta kyseisellä iteraatiokierroksella, mikä vähentää lohkon pakkausvaihtoehtoja ja tarkistamiseen vaadittavaa aikaa. Kyseisen polyominon sijainti joudutaan tarkistamaan seuraavalla iteraatiokierroksella kokonaisuudessaan, kun alueelle on valittu uusi jakaja. Kuten aiemmin esitetyssä kohdassa 6.3 esitettiin, seuraavalla iteraatiokierroksella ja uudella jakajalla voidaan tarkentaa aiemmissa iteraatiossa selvitettyjen polyominojen sijainteja alueella.

7.3 Polyominojen sijainnin monitulkinnaisuus

Alkuperäisen moduloidun bittikentän menetelmän esittelyn yhteydessä, kuvassa 6.6, esitettiin moduloidun bittikentän menetelmään liittyvä ongelma polyominojen sijainnin monitulkinnaisuudesta ja miten moduloidun bittikentän mukaisilla tarkistussummilla ei voida selvittää, meneekö polyominon tarkistussumman mukainen sijainti täytettävän alueen reunojen ulkopuolelle. Ongelma liittyy siihen, että tarkistussumman mukaisilla alueiden kuvauksilla menetetään tieto alueen tarkasta muodosta. Kuudenkymmenen neljän ruudun kokoinen täytettävä alue voisi menetelmän mukaisesti olla esimerkiksi $1 * 64$ tai $4 * 16$ ruudun kokoinen. Koska täytettävän alueen tarkkaa muotoa ei ole erikseen tallennettu tar-

kistussummiin, on mahdoton tarkistaa kappaleiden yksiselitteistä sijaintia pelkkien tarkistussummien perusteella. Alueen reunojen päälle tai ulkopuolelle sijoitetut kappaleet voidaan suodattaa pois oikeiden vastausten joukosta erillisessä tarkistuksessa viimeistään algoritmin suorituksen lopussa. Reunatarkistusta varten algoritmin tulee tietää täytettävän alueen ja polyominojen todellinen muoto, jolloin algoritmin löytämille tarkistussummien mukaisille pakkausvaihtoehdoille voidaan suorittaa alueen ja polyominojen täsmällisiä muotoja käyttäen. Mikäli täytettävä alue on suorakulmion muotoinen polyominojen on myös mahdollista suorittaa kappaleiden sijainnin tarkistus moduloidun bittikentän menetelmän mukaisesti käyttämällä lohkon jakajana täytettävän alueen leveyttä. Lohkon ollessa täytettävän alueen leveyden kokoinen, algoritmi voi keskeyttää tarkistuksen niissä tapauksissa, joissa polyominoa ollaan sijoittamassa lohkon reunojen päälle. Tarkistus vaatii kuitenkin tiedon polyominojen todellisista leveyksistä. Mikäli täytettävä alue ei ole suorakulmion muotoinen, on täytettävä alue aina mahdollista reunustaa suorakulmiolla ja suorittaa osittainen tarkistus vähentämällä suorakulmion tarkistussummasta todelliseen alueeseen kuulumattomien ruutujen indeksit. Alueen reunustamisella päädytään kappaleiden pakkaamiseen suorakulman muotoiselle alueelle, vaikka osa suorakulmioon kuuluvista ruuduista ei olisikaan mukana alueen tarkistussummassa.

Yksittäisen lohkojaon mukainen tulos voidaan joutua tarkistamaan toisilla lohkojaoilla tuloksen varmistamiseksi. Lohkojakojen mukaiset suurimmat mahdolliset pinta-alat on esitetty liitteessä IV. Taulukossa lohkon, joka on numeroitu luvuilla [0..5] eli $N = 6$ ja käyttäen 10 bittiä jokaisen lukuarvon esiintymien laskemiseen, enimmäiskoko voi olla 6138 ruutua. Koska ei voida olla varmoja, onko kyseinen esitys yksiselitteinen tehtävässä käytetylle polyominojoukolle, kyseiset pakkaukset joudutaan jälkikäteen tarkistamaan vaihtoehtoisilla lohkojaoilla. Varmasti alueella toimivat lohkojaot voidaan muodostaa alueen pienemmillä numeroinneilla tässä tapauksessa [0..4] eli $N = 5$, $N = 4$ jne. Tarkistamalla lohkojaon $N = 6$ mukaiset tulokset lohkojaolla $N = 5$ saadaan tarkistetun lohkon koko kasvatettua yhteensä $6 * 5 = 30$ ruudun kokoiseksi. Jatkotarkistus lohkolla 4 kuitenkin kasvattaa lohkon lohkon kokoa vain kaksinkertaiseksi, sillä molemmat lohkojaot neljällä ja kuudella sisältävät lohkojaon 2, jolloin päällekkäinen tarkistus ei edistä tuloksen tarkistusta. Lohkon tuosten tarkistuksessa käytettävät lukujen yksilöivät tekijät ovat siis alkuluvut, esimerkiksi $[2, 3, 5, 7..]$, ja alkulukujen potenssit vastaavasti $[4, 9, 25, 49..]$ sekä $[8, 27, 125, 343..]$ ja niin edelleen. Alkulujujen yhtä suuremmat potenssit kuitenkin edistävät lohkon tarkistusta vain kyseisen alkuluvun verran sillä oletuksella, että tarkistus kyseisellä alkuluvulla

on suoritettu jo aiemmin.

N	Tekijät	Yksilöivä tekijä	Yksilöivien tekijöiden tulo
2	2	2	2
3	3	3	6
4	2	2	12
5	5	5	60
6	2, 3	1	60
7	7	7	420
8	2	2	840
9	3	3	2520
10	2, 5	1	2520
11	11	11	27720

Taulukko 7.1: Lohkojaon tarkistuksessa käytettävät yksiöivät tekijät ja niiden tulot.

Taulukossa 7.1 esitetään lohkojaossa käytettävien yksilöivien tekjöiden tulon perusteella suurimmat varmasti varmennettavissa olevat lohkokoot. Lohkon pakkauksen varmentaminen on tilannekohtaista, joten suurempien alueiden pakkaaminen saattaa myös olla tilanteesta riippuen mahdollista. Liitteenä olevaa taulukkoa IV ja taulukkoa 7.1 vertailemalla voidaan havaita, että suurin varmennettavissa oleva aluekoko on 1143 ruutua. Tulos saavutetaan numeroitaessa täytettävä alue luvuilla [0..8] eli $N = 9$ ja käyttäen enintään 7 bittiä yhden luvun esiintymien laskemiseen. Yhden luvun esiintymiä voi olla lohkolla enintään $2^7 - 1 = 127$ kappaletta, joten pienimmillään numeroitavissa oleva alue $A = 9 * 127 = 1143$ ruudun kokoinen. Yksilöivien tekijöiden tulon mukaan N:n ollessa 9, yhdeksän kokoisilla tai sitä pienemmillä lohkojen numeroinneilla voidaan varmistaa enintään 2520 ruudun kokoinen alue. Riippumatta polyominojen tai alueen muodoista suurin varmennettavissa oleva alue numeroitavissa olevan alueen ja yksilöivien tekijöiden tulon minimi, eli tässä tapauksessa 1143 ruutua.

Vaikka polyominojen monitulkinnaisuutta pystytäänkin varmentamaan tai ainakin rajoittamaan yllä kuvatuilla menetelmillä täytyy muistaa, että tarkistussummiin perustuvat menetelmät eivät välttämättä ole täysin toimintavarmoja. Tarkistussummia käytetään tiedon tiivistämisen yhteydessä ja niiden tarkoitus on tallentaa tieto pienempään tilaan helpottamaan tiedon oikeellisuuden tai muuttumattomuuden tarkistamista. Mikäli tieto tallennetaan käyttäen häviöllisiä me-

netelmiä, kuten moduloidun bittikentän menetelmää, tarpeellista tietoa saattaa kadota ja tällöin kaikki alkuperäinen tieto ei välttämättä ole enää palautettavissa. Samasta syystä moduloidun bittikentän menetelmällä ei aina voida taata täsmälleen oikeaa tulosta. Huomionarvoista kuitenkin on, että moduloidun bittikentän menetelmä on tehokas tapa löytää pakkausvaihtoehtojen joukosta sellaisia tapauksia, jotka eivät voi täyttää valittua aluetta.

7.4 Menetelmän rajoitteet

Sopivien lohkojakojen ja tietokonearkkitehtuurin 64 bittisyyden aiheuttamien rajoitteiden lisäksi myös algoritmin palauttamien pakkausvaihtoehtojen varmentaminen voi rajoittaa algoritmin toimintaa. Pakkausvaihtoehtojen varmentamisen tarpeellisuus riippuu täytettävän alueen sekä käytettävien polyominojen koosta, joten tarkka rajoite ja varmentamisen tarve on usein tilannekohtaista. Huonoimmassa mahdollisessa tilanteessa lohkon pakkaus joudutaan tarkistamaan kaikilla käytössä olevilla lohkojaon yksilöivillä tekijöillä. Tällöin tarkistettavan alueen pinta-ala voi olla enintään minimi lohkojaon suurimman pinta-alan $a = N*(2^b-1)$ ja lohkon esiintymien numerointiin käytettävien lukujen yksilöivien tekijöiden tulon välillä. Lohkojakojen mukaiset suurimmat mahdolliset pinta-alat on esitetty liitteessä IV ja yksilöivät tekijät taulukossa 7.1. Kuten aiemmin osoitettiin, moduloidun bittikentän menetelmällä voidaan varmasti numeroida ja tarkistaa ainakin 1143 ruudun kokoinen alue. Menetelmän suurin rajoite ei liity alueen pinta-alaan, vaan alueen muotoon. Alueen pakkaamisen kannalta alueen muodolla on suuri merkitys pakkaamisen onnistumiselle. Alueen pinta-alan ollessa 100 ruutua alueelle mahtuu 25 kappaletta tetriminoja. Mikäli alueen mitat ovat $1 * 100$ ruutua voidaan todeta, että alueelle sopii pelkästään suoran muotoisia tetriminoja, kun taas $10 * 10$ kokoiselle alueelle on soviteltavissa huomattavasti moninaisempi joukko erilaisia tetriminojen muotoja. Alueiden eroavaisuuksista huolimatta moduloidun bittikentän menetelmän kannalta molempien alueiden tarkistussummat ovat täsmälleen samat, eikä menetelmä ilman lisätietoja osaa erottaa alueita toisistaan. Lisätietojen saamiseksi alue voitaisiin ympyröidä ruuduilla, jotka poistettaisiin lopullisesta tarkistussummasta, samoin kuin voidaan tehdä epämääräisen muotoisen alueen pakkauksen haussa. Voisi sanoa, että moduloidun bittikentän menetelmällä ei varsinaisesti etsitä täytettävälle alueelle sopivia pakkausvaihtoehtoja. Tarkistussummiin perustuen menetelmällä itseasiassa löydetään varmasti vain sellaiset pakkausvaihtoehdot, jotka eivät sovi täytettävälle alueelle. Tarkistussummiin täsmäävien pakkausvaihtoehtojen joukossa voi siis olla myös sellaisia

tuloksia, jotka eivät todellisuudessa sovi täytettävälle aluelle. Menetelmän toimiessa oikein, yhtään täytettävälle alueelle sopivaa pakkausvaihtoehtoa ei kuitenkaan pitäisi löytyä hylättyjen pakkausvaihtoehtojen joukosta.

Kaikille täytettävien alueiden ja polyominojen yhdistelmille ei välttämättä ole mahdollista muodostaa sopivia lohkojakoa. Liitteessä II esitellään algoritmi, jolla voidaan selvittää moduloidun bittikentän menetelmään sopivat lohkojaot polyominon asteluvun sekä täytettävän alueen leveyden perusteella. Liitteen mukaisessa algoritmissa verrataan alueen indeksoivien lukujen esiintymien etäisyyksiä toisiinsa. Vastaava algoritmi on toteutettavissa pienillä muutoksilla myös hajota ja hallitse -menetelmän mukaiselle moduloidun bittikentän toteutukselle. Muutosten avulla funktion tulisi laskea indeksien esiintymien lukumääriä alueen jakajaksi valitun etäisyyden sisällä. Mikäli alueen numeroimiseen käytettävien indeksilukujen esiintymät voidaan numeroida käytettävissä olevalla bittimäärällä, valittu jakaja voidaan hyväksyä kyseiselle alueelle. Yksittäisen tarkistussumman muodostaminen lohkolle ei usein riitä, sillä koko alueen pakkauksen tarkistaminen vaatii lisäksi useita muita lohkojakoja varmentamaan pakkauksen koskemaan koko täytettävää aluetta.

7.5 Hajota ja hallitse -menetelmän aikakompleksisuus

Hajota ja halitse -menetelmässä täytettävälle alueelle lasketaan tarpeen mukaan useita eri lohkojaoilla muodostettuja tarkistussummia. Tarkistussummat yhdessä muodostavat yhä tarkemman kuvan lopullisesta koko alueen peittävästä pakkauksesta. Kaikkien pakkausvaihtoehtojen läpikäyminen mille tahansa yksittäiselle alueelle voidaan esittää muodossa A^c. Kaavan mukainen tulos tulee siitä oletuksesta, että mikä tahansa polyominoista voi sijaita missä tahansa täytettävällä alueella. Kuten aiemmassa kohdassa 6.3 todettiin, menetelmän alueen jakaminen osiin on tehokkainta silloin, kun alue jaetaan kahteen mahdollisimman tasakokoiseen lohkoon, jotka voidaan tarkistaa erikseen huomattavasti yhtä suurta aluetta tehokkaammin. Hajota ja hallitse -menetelmän mukaisella muunnoksilla alueen lohkomisesta voidaan tehdä entistä tehokkaampaa käyttäen useampia entistä pienempiä lohkoja. Esitettyjen muutosten ansiosta alkuperäinen alue voidaan jakaa jakojäännösten avulla kahteen osaan aina uudelleen, kunnes lohkojakojen yksilöivät tekijät yhdessä kattava koko täytettävän alueen tai mahdollisia tuloksia ei enää löydy. Pienin menetelmällä toimiva aluejako on modulo 2, sillä jakojäännös yhdellä on aina nolla. Tällöin kaikkien yksittäisten ruutujen tarkistussummaksi tulisi $1 * 2^0 = 1$, jolloin polyominojen ja niiden sijaintien sekä tarkistus-

summien vertailusta tulisi hyödytöntä. Modulo 2 -jakojäännöksellä täytettävän alueen ruudut jaetaan kahteen osaan ruutujen indeksien perusteella, parillisiin ja parittomiin.

Pienimmän, modulo 2 -jaotellun lohkon kaikkien pakkausvaihtoehtojen läpikäymisen aikakompleksisuus on aiemmin esitetyn perusteella 2^c. Tarkasteltavan lohkon pinta-alan kasvaessa kaksinkertaiseksi algoritmi voi käyttää hyväkseen edellisen iteraatiokierroksen tuloksia liitteessä III esitetyn binääripuun mukaisesti, jollon puolet seuraavan iteraatiokierroksen vaihtoehdoista on jo tarkistettu edellisellä kierroksella. Koska jokaisen seuraavalla tarkistuskierroksella tarkasteltavan alueen koko on kaksinkertainen edelliseen verrattuna tarkistuskierroksia tarvitaan $log_2(A)$ kappaletta, missä A on koko täytettävän alueen pinta-ala. Mikäli lohkojakoa kasvatetaan aina kaksinkertaiseksi seuraavilla iteraatiokierroksella, jokaisen yksittäisen tarkistuskierroksen aikakompleksisuus tulee olemaan sama 2^c. Koska yksittäisen kierroksen aikakompleksisuus on 2^c ja iteraatiokierroksia tarvitaan $log_2(A)$ koko alueen pakkaamiseksi tarvittava aikakompleksisuus on $2^{c*log_2(A)}$. Menetelmän mukainen kaikkien pakkausvaihtoehtojen lukumäärä on $2^{log_2(A)*c} = A^c$, eli täsmälleen sama alkuperäisen menetelmän kanssa. Vaikka kaikkien vaihtoehtojen lukumäärä ei menetelmän myötä vähene alkuperäiseen verrattuna, menetelmän jokaisella iteraatiokierroksella voidaan hylätä kaikki tarkistussummaan täsmäämättömät pakkausvaihtoehdot, jolloin kyseisiä vaihtoehtoja ei tarvitse käydä läpi enää seuraavilla kierroksilla. Kuten kohdassa 6.3 todettiin, yksittäisen täsmäävän tarkistussumman yhteydessä minkä tahansa yksittäisen kappaleen siirtäminen aiheuttaa muutoksen tarkistussummaan, jolloin tarkistussumma ei voi enää täsmätä. Koska polyominojen lukumäärä on c ja polyominoilla on aina 2 sijaintivaihtoehtoa, voidaan todeta, että enintään $\frac{1}{2c}$ osuus tarkistussummista voi täsmätä jokaisella yksittäisellä modulo 2-lohkolla. Tällöin kaikkien mahdollisten tarkistussummien läpikäymiseksi riittää tarkistaa enintään

$$(\frac{2^c}{2c})^{log_2(A)} \tag{7.2}$$

vaihtoehtoa. Yhtälössä jakaja $2c$ tulee siitä, että jokaisella alueella on c polyominoa, joilla oletetaan olevan aina enintään kaksi toisistaan poikkeavaa sijaintivaihtoehtoa. Lohkojakojen lukumäärä on $log_2(A)$, mistä tulee funktion eksponentti kaikkien vaihtoehtojen läpikäynnille. Kaavaa voidaan vielä sieventää muotoon

$$\frac{A^c}{A * c^{log_2(A)}} = \frac{A^{c-1}}{c^{log_2(A)}} \ . \tag{7.3}$$

Kuten kohdassa 7.4 todettiin, pakkauksen kohteena olevan alueen pinta-alan kas-

vaessa riittävän suureksi, pakkauksen täsmällistä tulosta varmentamaan saatetaan tarvita useampia erikokoisia lohkojakoja. Tällöin yhtälö kaikkien pakkausvaihtoehtojen läpikäymiseksi voidaan muuntaa muotoon

$$(\frac{a^c}{ac})^{log_a(A)} = \frac{A^c}{A * c^{log_a(A)}} \ . \tag{7.4}$$

Kaikkein pahimmassa tapauksessa, jossa huomattava määrä polyominojen bittiesityksistä olisi kohdassa 6.3 kuvatun kaltaisesti symmetrisiä, pakkausvaihtoehtoja tulisi tarkistaa enintään

$$(\frac{a^c}{2c})^{log_a(A)} = \frac{A^c}{(2c)^{log_a(A)}} \ . \tag{7.5}$$

Kaavan 7.5 mukainen tulos on käytännössä erityisen harvinainen, sillä jokaisen alueen täyttöön osallistuvan polyominon bittiesityksen pitäisi olla osittain symmetrinen jokaisella pakkauksen tarkistamiseen käytetyllä lohkojaolla. Kaikissa moduloidun bittikentän pakkausvaihtoehtojen kasvunopeutta kuvaavissa kaavoissa merkitsevin tekijä, pakkaamiseen käytettävien polyominojen lukumäärä c, on kuitenkin pysynyt muuttumattomana. Polyominojen lukumäärän kasvaessa tarpeeksi funktion jakajassa oleva lohkojaon pinta-alasta riippuva tekijä $a * c$ on lähes mitätön lohkon pakkausvaihtoehtojen a^c kasvunopeuden suhteen. Useiden lohkojakojen, erilaisten tarkistussummien ansiosta moduloidun bittikentän menetelmällä voidaan kuitenkin sulkea pois tarkastelusta kaikki aiemmilla kierroksilla tarkistussummaan täsmäämättömät pakkausvaihtoehdot liitteen III mukaisesti. Kyseisen kaltainen hakupuu on muodostettavissa myös muille kuin modulo 2 -kokoisille lohkojaoille. Kuten kohdassa 6.3 todettiin, se miten suuri osa kaikista pakkausvaihtoehdoista voidaan sivuuttaa sopimattomina eri kokoisilla lohkojaoilla riippuu täytettävästä alueesta sekä alueelle sijoitettavista polyominoista. Yksittäinen esimerkki moduloidun bittikentän mukaisesta polyominojen pakkausvaihtoehtojen lukumääristä esitetään kohdassa 7.7.

Yksittäisen lohkojaon tarkistussumman täsmääminen modulo 2 -lohkojaolla on muunnettavissa *selkärepun täyttöongelmaksi*. Selkärepun täyttöongelman ja moduloidun bittikentän välinen yhteys kuvataan seuraavassa kohdassa.

7.6 Selkärepun täyttöongelma

Selkärepun täyttöongelmassa henkilöllä on reppu, johon saa pakata vain enintään tietyn painon verran kantamuksia. Henkilö on lähdössä retkelle ja hänellä on suurehko määrä tarpeellisia esineitä mukaanotettavaksi. Valitettavasti kaikkien

tärkeiden esineiden yhteispaino on suurempi kuin repun enimmäiskantokyky. Koska kaikki esineet ovat henkilölle yhtä tärkeitä esineen painoon suhteutettuna, on tärkeää saada pakattua reppu mahdollisimman täyteen esineitä. Kun repun enimmäiskantokyky ja yksittäisten esineiden painot on annettu, mitkä esineet henkilön tulisi valita reppuun, jotta reppu tulisi mahdollisimman täyteen esineitä?

Moduloidun bittikentän menetelmässä jokaisella polyominolla on sen sijaintiin ja samalla myös polyominon muotoon liittyvä tarkistussumma. Koska polyominojen muoto säilyy samana koko pakkaustehtävän ajan, ainoa muutos tarkistussummaan tulee kappaleen sijainnin muutoksen myötä. Kuten aiemmin todettiin moduloidun bittikentän menetelmän binäärisen lohkojaon seurauksena jokaisella polyominolla on lohkojaon tarkistuksen yhteydessä kaksi mahdollista sijaintivaihtoehtoa. Molemmille sijaintivaihtoehdoille on olemassa polyominon muotoon liittyvä tarkistussummat, yksi molemmille sijaintivaihtoehdoille. Moduloidun bittikentän menetelmässä jokaiselle alueen täyttöön osallistuvalle polyominolle valitaan toinen kahdesta tarkistussummasta, jolloin kaikkien polyominojen summan tulisi muodostaa koko alueen peittävä tarkistussumma. Koska jokaiselle polyominolle on kaksi tarkistussummavaihtoehtoa on itsestäänselvää, että luvut ovat vaihtoehtoisesti joko yhtäsuuret tai erisuuret. Mikäli jonkin polyominon vaihtoehtoiset tarkistussummat ovat keskenään täsmälleen yhtäsuuret, kyseisen polyominon siirtäminen tasolla ei muuta polyominojen tarkistussumman yhteistulosta ja kyseisen polyominon siirtäminen sekä tarkistus voidaan sivuuttaa tällä algoritmin iteraatiokierroksella. Koska tiedetään, että jokainen alueen täyttämisen osallistuva polyomino on jossain kohdassa täytettävällä alueella, voidaan olettaa, että polyominojen yhteenlaskettu tarkistussumma sisältää vaihtoehtoisesti aina toisen polyominon sijaintiin liitetyistä tarkistussummista. Koska yksittäisten polyominojen tarkistussummat ovat aina eri suuruisia, jokaisen polyominon pienempi tarkistussumma voidaan vähentää sekä koko alueen tarkistussummasta että saman polyominon suuremmasta tarkistussummasta. Polyominojen tarkistussummien erotus kuvaa selkärepun täyttöongelmassa esineen painoa ja koko täytettävän alueen muunnettu tarkistussumma kuvaa repun painorajaa. Mikäli polyominon sijainti on pienemmän tarkistussumman mukainen selkärepun täyttöongelman tapauksessa esinettä ei lisätä reppuun, sillä polyominon sijainnin mukainen tarkistussumma on jo huomioitu päivitetyssä lohkon tarkistussummassa eli repun painorajassa. Vastaavasti polyominon sijainnin ollessa suuremman tarkistussumman mukainen, valittu esine lisätään reppuun ja myös polyominon sijaintivaihtoehtojen tarkistussummien erotus huomioidaan lohkon tarkistussum-

maan. Ongelman tavoite on vastaava kuin selkärepun täyttöongelmassa: saada esineiden yhteenlaskettu paino tai vastaavasti polyominojen yhteelaskettu tarkistussumma täsmäämään repun painorajan, eli lohkon tarkistussumman kanssa.

7.7 Esimerkki polyominojen pakkausvaihtoehtojen läpikäymisestä

Moduloidun bittikentän menetelmän tehokkuus perustuu suuresti lohkojakojen tarkistussummien täsmäämiseen. Koska menetelmän tehokkuus riippuu täytettävästä alueesta sekä alueen täyttämiseen käytettävistä polyominoista esitetään tässä kohdassa yksittäisenä esimerkkinä kuvan 6.1 mukaisen alueen läpikäymiseksi tarvittavien pakkausvaihtoehtojen lukumääriä muutamilla erilaisilla moduloidun bittikentän mukaisilla lohkojaoilla. Toteutin moduloidun bittikentän C-kielellä, tarvittavat tietorakenteet sekä XML-muotoisen polyominojen ja täytettävän alueen muodoista koostuvan syötedatan. Syötedata koostuu 64 ruudun kokoisesta alueesta, johon pyritään sijoittamaan 16 polyominoa moduloidun bittikentän menetelmän mukaisesti käyttäen kolmea bittiä jokaista indeksiluvun numerointia kohden. Taulukossa 7.2 esitetään muutamien lohkojakojen läpikäymiseen vaadittavien vaihtoehtojen lukumääriä sekä suoritusten tuloksia.

Lohkojako	Kaikki	Lukumäärä	Tarkastetut	Täsmäävät	Aika
modulo 2	2^{16}	65536	64	20	0.005s
modulo 3	3^{16}	43046721	43046721	2018016	9.684s
modulo 2 & 3	$2^{16} * 1.5^{16}$	43046721	11809800	586908	2.755s
modulo 4	4^{16}	4294967296	67108864	703496	5.430s
modulo 2 & 4	$2^{16} * 2^{16}$	4294967296	20971520	703496	3.751s

Taulukko 7.2: Moduloidun bittikentän mukaiset tulokset eräälle lohkojaolle.

Kuudellatoista polyominolla voidaan muodostaa kahdella eri sijaintivaihdolla taulukon 7.2 mukaisesti $2^{16} = 65536$ erilaista sijaintien mukaista yhdistelmää. Koska esimerkissä käytetyt polyominot olivat tetriminoja, modulo 2 -lohkojaolla useat esimerkissä käytettyjen polyominojen bittiesitykset olivat symmetrisiä ja niiden tarkistus voitiin sivuuttaa kyseisellä lohkojaolla. Jäljelle jääneet kuusi bittiesitystä tarkistettiin, jolloin kuuden polyominon kaikkien sijaintivaihtoehtojen lukumäärä yhteensä oli 64. Tarkistetuista vaihtoehdoista 20 täsmäsi kyseisen lohkon tarkistussummaan 108. Koska tetriminot koostuvat neljästä neliöstä, yksikään modulo 3 -lohkojaon mukainen polyominon bittiesitys ei voi olla

symmetrinen. Tämä voidaan todeta myös taulukosta 7.2, sillä kyseisellä lohkojaolla on jouduttu tarkistamaan kaikki $3^{16} = 43046721$ erilaista polyominojen sijaintien mukaista yhdistelmää. Modulo 3 -lohkojaon mukaisista yhdistelmistä yhteensä 2018016 on täsmännyt kyseisen lohkon tarkistussummaan 1534. Taulukon 7.2 kolmannella rivillä esitetään moduloidun bittikentän mukaisen yhdistelmähaun tulos. Haussa on ensin suoritettu etsintä lohkojaolla modulo 2, jonka tulosten perusteella haun kohteena olevaa aluetta on kasvatettu edelleen lohkojaolle modulo 3. Ilman moduloidun bittikentän menetelmää kaikkien polyominojen sijaintivaihtoehtojen yhdistelmien lukumäärä olisi 3^{16}. Moduloidun bittikentän lohkojakojen ansiosta kaikista vaihtoehdoista tarvitsee tarkistaa modulo 2 -tarkistuksen jälkeen taulukon mukaiset 11809800 vaihtoehtoa, joista 586908 täsmää lohkon tarkistussummaan. Modulo 2 -lohkojaosta modulo 3 -lohkojakoon siirryttäessä kaikki modulo 2 -lohkojaolla saadut tulokset tulee tarkistaa erikseen lohkojaolla modulo 3. Kaikkien tarkistettavien vaihtoehtojen lukumäärä on likimain $20 * 1,5^6 * 3^{10} \approx 13452100$, mikä on melko lähellä kaikkien algoritmin tarkastamien vaihtoehtojen lukumäärää. Lukumäärä tulee siitä, että modulo 2 -lohkojaolla löytyi 20 täsmäävää tarkistussummaa, joiden löytämiseksi oltiin suoritettu haku käyttäen kuutta polyominoa. Loput kymmenen polyominoa jouduttiin tarkistamaan kokonaisuudessaan modulo 3 -lohkojaolla. Seuraavassa modulo 4 -lohkojaossa kolmen polyominon bittiesitys oli symmertinen, mikä voidaan huomata kaikkien tarkastettujen vaihtoehtojen tuloksesta $4^{13} = 67108864$. Täsmääviä tarkistussummia löytyi yhteensä 703496 ohjelman suoritusajan ollessa kokonaisuudessaan $5,430$ sekuntia. Vastaavasti modulo 2 -tulosten perusteella suoritettu modulo 4 -lohkojako vaatii enää $20 * 2^6 * 4^7 = 20971520$ tarkistettavan vaihtoehdon läpikäynnin. Koska modulo 4 -lohkojako sisältää modulo 2 lohkojaon kokonaisuudessaan täsmääviä tarkistussummia on yhtä monta kuin aiemmassa modulo 4 -tuloksessa eli 703496. Suoritusaika pienemmillä lohkojaoilla on hieman aiempaa pienempi eli $3,751$ sekuntia.

Algoritmin suoritusaikoihin vaikuttaa huomattavasti ohjelman syöte- ja tulosdatan pituus. Yksittäinen syöte- tai tulostietue vie noin 500 tavua kovalevytilaa, joten algoritmin tulostiedostojen pituus voi olla jopa useita satoja megatavuja. Syöte- ja tulostiedostoissa käytetään XML-formaattia tulosten helpon luettavuuden sekä mahdollisten virhetilanteiden havaitsemisen takia. Kyseisen formaatin käyttö on hidasta, sillä se se edellyttää suurten tiedostojen lukeminen kovalevyltä on usein erityisen hidasta. Kaikki tässä kohdassa esitellyt testiajot suoritettiin tietokoneelle tehdyltä ram-asemalta perinteisen kovalevyn lukunopeuden ja muiden viiveiden minimoimiseksi.

8 Muita ajatuksia ja jatkokehitysideoita

Listaan tähän lukuun jatkokehitysideoita sekä muita huomioita liittyen polyominojen pakkaamiseen tasolle sekä moduloidun bittikentän menetelmään. Menetelmät ovat pääasiassa sellaisia, joita olen miettinyt tutkielman kirjoitusprosessin aikana, mutta joita en ole erinäisistä syistä kehittänyt pidemmälle.

8.1 Rengaspuskurin käyttö bittikentän sijaan

Algoritmini toteutusratkaisuja miettiessäni tutkin myös linkitetyllä listalla toimivan *rangaspuskurin (engl. ring buffer)* käyttöä polyominojen tietorakenteena. Rengaspuskuri osoittautui erittäin käyttökelpoiseksi polyominojen tallentamisessa ja bittisiirtoa vastaavan operaation toteuttamisessa. Rengaspuskuri osoittautui kuitenkin huomattavan tehottomaksi vaihtoehdoksi tarkistussummien laskemisen yhteydessä. Kun bittikentällä kuluu pieni vakiollinen aika yhden alueen pakkauksen tarkistussumman vertailuun, rengaspuskurilta aikavaatimus on suhteessa polyominon esityksen pituuteen. Kun otetaan huomioon, että bittikentissä polyominojen esityksen pituus rajoittuu enintään 64 lukuarvoon, rengaspuskurin mukainen toteutus ei välttämättä ole käytännön tilanteessa merkittävästi bittikenttää hitaampi. Polyominojen esityksen pituus ei ole algoritmin aikavaatimuksen kannalta merkitsevin tekijä, sillä kappaleiden lukumäärä toimii yhä aikavaatimuksen funktion eksponenttina.

Rengaspuskuria voisi ajatella käytettäväksi, mikäli pakattavia kappaleita on vain muutamia tai polyominojen asteluku on erityisen suuri. Käytännössä tällaisia sovelluksia on jo nykyäänkin teollisuudessa. Esimerkiksi metallilevystä irti leikattavia kappaleita voi kuvata bittikarttana, joka on erityisen suuren asteluvun n-omino. Bittikartalla voidaan kuvata aluetta, jossa jokaisen yksittäisen pisteen koko on hyvin pieni, jopa alle millimetrien luokkaa. Kappaleen reunasta voisi ajatella tulevan rosoreunainen, mutta pisteiden ollessa riittävän pieniä kappaleen reunat pyöristyvät. Toinen tapa pyöristää kappaleen kulmia on leikata kappale irti tasosta käyttäen kappaleen reunimmaisia pisteitä pitkin kulkevia janoja. Rengaspuskuri-tieto rakenteen tarkempi tarkastelu ja käyttö jää kuitenkin tämän tutkielman ulkopuolelle.

8.2 Syvyyssuuntainen pakkauksen etsintä

Toteutin algoritmin moduloidun bittikentän menetelmälle leveyssuuntaisena hakuna. Algoritmi käy läpi koko täytettävää aluetta yksi lohko kerrallaan ja tu-

lostaa lohkon tarkistussummaan sopivat tulosvaihtoehdot tiedostoon. Mielessäni kävi myös toteuttaa pakkauksenhaku syvyyssuuntaisesti. Syvyyssuuntainen haku olisi siirtynyt seuraavalle hakutasolle välittömästi yhden tarkistussummaan täsmäävän pakkausvaihtoehdon löydyttyä. Syvyyssuuntainen pakkausvaihtoehtojen etsintä voisi toimia erityisen tehokkaasti, mikäli tarkoitus olisi etsiä mikä tahansa yksittäinen pakkausvaihtoehto alueelle.

8.3 Rinnakkaistettu algoritmi pakkauksen etsimiseksi

Moduloidun bittikentän menetelmä hajota ja hallitse toiminnallisuudella on erityisen tehokkaasti rinnakkaistettavissa. Jokaiselta tasolta löytyy usein monia mahdollisia täsmääviä pakkausvaihtoehtoja. Jokainen yhdeltä tasolta löytynyt pakkausvaihtoehto on mahdollista tarkistaa samanaikaisesti erillisinä rinnakkaisina prosesseina. Rinnakkaistettavuus on erityisen tehokasta myös siksi, että jokaisessa lohkossa on pääsääntöisesti yhtä monta tarkastettavaa pakkausvaihtoehtoa, jolloin lohkovaihtoehtojen tarkistamisessa kuluu lähes yhtä pitkä aika.

8.4 Repuntäyttöongelman tarkempi perehtyminen

Olisi mielenkiintoista perehtyä tarkemmin repuntäyttöongelman erilaisiin ratkaisuvaihtoehtoihin. Huomattuani yhteyden repuntäyttöongelman ja moduloidun bittikentän menetelmän välillä olisin halunnut tutkia repuntäyttöongelmaan liittyviä algoritmeja tarkemmin. Päätin kuitenkin jättää repuntäyttöongelman tämän tutkielman ulkopuolelle, sillä repuntäyttöongelmassa olisi varmasti riittävästi tutkittavaa erilliseen tutkielmaan.

8.5 Alkulukujen käyttö tarkasteltavien lohkojen jakajana

Alkulukujen käyttö moduloidun bittikentän lohkojakojen muodostamisessa vaikutti moduloidun bittikentän menetelmän keksimisen alussa erityisen tehokkaalta tavalta laajentaa tarkasteltavaa aluetta ja löytää mahdollisia pakkausvaihtoehtoja. Myöhemmin osoittautui, että samalla kun alkuluvut jakavat alueen erityisen tehokkaasti sopivan kokooisiin osiin, seuraavien tarkistuskierrosten lohkojen koko kasvaisi myös erityisen nopeasti. Tarkasteltavan lohkon pinta-ala kasvaisi aina alkulukujen tulon mukaan, joten alkulukujen käyttö vaatisi aina seuraavan lohkon tarkistuksen yhteydessä enemmän laskentaa edelliseen lohkoon verrattuna. Lohkon koko kasvaisi erityisen nopeasti siksi, että alkuluvuilla ei ole yhteisiä tekijöitä edellisellä kierroksella tarkastellun alueen pinta-alan kanssa. Tulosteni

valmistuttua päädyin tulokseen, jossa moduloidun bittikentän menetelmä on tehokkaimmillaan mahdollisimman pienellä lohkojaon kasvulla, jolloin pakkaukseen sopimattomat järjestysvaihtoehdot voidaan karsia pois tarkastelusta mahdollisimman aikaisessa vaiheessa. Alkulukuja voidaan joutua käyttämään moduloidun bittikentän menetelmän tulosten varmentamisessa ja erityisen suurilla täytettävillä alueilla.

8.6 Polyominojen järjestyksen mukainen alueen pakkaaminen

Polyominojen pakkaamista tasolle voisi kokeilla suorittaa myös polyominojen järjestysten mukaisesti. Polyominot voisi järjestää listaan, josta polyominoja poimittaisiin järjestyksessä vuorotellen sijoitettavaksi täytettävälle alueelle. Listaan järjestettyjä polyominoja sijoitetaan täytettävälle alueelle järjestyksessä siten, että seuraava polyomino sijoitetaan aina välittömästi edellisen viereen. Mikäli seuraavaa polyominoa ei voi sijoittaa alueelle tai polyominoiden väliin tulee tyhjä alue, jota ei voida enää peittää, polyominolista järjestetään uudelleen seuraavaa pakkausyritystä varten. Koska pakkausvaihtoehtojen lukumäärä on erilaisten listojen järjestysten lukumäärä erilaisia pakkausvaihtoehtoja on polyominojen kertoman verran. Menetelmässä ongelmallista on seuraavien polyominojen sijoittaminen edellisten viereen ja virheellisten pakkausvaihtoehtojen havaitseminen.

8.7 Kappaleiden kiertäminen ja peilaaminen

Tässä teoksessa esitetty moduloidun bittikentän menetelmä ei osa huomioon kappaleiden kiertämistä tai peilaamista. Kiertäminen ja peilaaminen eivät kuitenkaan ole menetelmän kannalta ongelmallisia, mutta ne voisivat aiheuttaa moninkertaisen kompleksisuuden kappaleiden pakkaamiselle sillä erilaisten tarkistussummien ja kappaleiden lukumäärä lisääntyisi entisestään.

9 Yhteenveto

Tässä tutkielmassa tarkasteltiin polyominojen pakkaamista tasolle sekä esiteltiin tekijän kehittämä tarkistussummiin perustuva menetelmä polyominojen pakkaamiseksi. Polyominojen pakkaaminen tasolle on NP-täydellinen ongelma, joten sille ei ole tunnettua polynomisessa ajassa toimivaa ratkaisua, täytettävän alueen ja polyominojen lukumäärän suhteen, erityistapauksia lukuunottamatta.

Tutkielman aluksi esiteltiin polyominoihin liittyviä peruskäsitteitä ja yleisluontoisesti muutamia erilaisia polyominojen pakkaamistehtäviä. Seuraavaksi esiteltiin polyominojen pakkaamisen kompleksisuutta sekä Golombin [1994] esittämä menetelmä polyominojen pakkauksen pariteettitarkistukseen. Tutkielman suurin saavutus on Golombin [1994] esittämän pariteettitarkistuksen pohjalta kehitetty moduloidun bittikentän menetelmä, jossa polyominon pakkausvaihtoehtoja voidaan etsiä useiden erilaisten pariteettitarkastusten perusteella. Moduloidun bittikentän menetelmän esittelyn jälkeen esitettiin muutokset, jolla pakkaustehtävä saadaan jaettua entistä pienempiin osatehtäviin siten, että pakkausten haku voidaan suorittaa hajota ja hallitse menetelmän mukaisesti. Tutkielman lopussa esitettiin lyhyesti useita tutkielman teon aikana esiin tulleita ajatuksia ja jatkokehitystä kaipaavia ideoita polyominojen pakkaamiseen liittyen.

Tämän tutkielman tekemisen ohella on ollut mielenkiintoista seurata, miten moneen suuntaan pienestä havainnosta tai idesta lähtenyt keksintö voi johtaa. Samassa yhteydessä voisi todeta, että uusien ajatusten keksimi on helppoa, mutta idean todistaminen tai kehittäminen toimivaksi kokonaisuudeksi on erityisen työlästä. Miten idean kehittämisen yhteydessä voi tietää, milloin ajatus on valmis kokonaisuus ja siihen ei tule enää mitään merkittävää tietoa lisättäväväksi? Moduloidun bittikentän menetelmää kehittäessä kaikkein hienointa oli seurata menetelmän kehittymistä ja sitä, miten menetelmän toiminta ja käyttötarkoitus muuttui prosessin etenemisen myötä. Tutkielman kirjoittamisen alkussa lähdettiin tutkimaan ongelmaa polyominojen pakkaamiseksi tasolle. Prosessin edetessä päädyttiin kuitenkin menetelmään, joka itseasiassa vastaa lähes päinvastoin aseteltuun kysymykseen "mitkä polyominojen pakkausvaihtoehdot eivät sovi alueelle?". Tätä tutkielmaa olisi voinut jatkaa paljon pidemmälle muunmuassa perehtymällä tarkemmin luvussa 8 esitettyihin ajatuksiin.

Uuden algoritmin kehtitäminen on ollut erityisen palkitsevaa, mutta myös toisinaan hyvin turhauttavaa. Oman algoritmin kehittäminen ei ole ollut mitenkään suoraviivaihen prosessi. Prosessin aikana on kohdattu useita polun haaroja, jotka jopa viikkojen tutkimisen jälkeen ovat osoittautuneet umpikujiksi.

Erityisen raskaaksi prosessista teki se, että etukäteen oli vaikea ennustaa menetelmän toimivuutta tai toimintaympäristön asettamia vaatimuksia sekä rajoitteita. Toisaalta työn tavoitteen on ollut pro gradu -tutkielman kirjoittaminen, joten tehokkaan tai edes toimivan algoritmin kehittäminen ei ole ollut suoranainen vaatimus hyvälle suoriutumiselle tutkielman kirjoittamisen kannalta. Tutkielman kirjoittamisen loppuvaiheessa ymmärsin, että tutkimuksen päätavoite ei voi olla pakottava tarve keksiä jotain uutta. Tutkimuksessa tärkeintä on dokumentoida tutkimuksen aikana todetut havainnot niin hyvin, että seuraavien aiheeseen perehtyvien tutkijoiden ei tarvitse aloittaa uudelleen alusta.

Viiteluettelo

[Aho et al., 1983] Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman. *Data Structures and Algorithms*, Addison-Wesley, 1983.

[Busche, 2011] Matthew T. Busche. *Blog: Solving Polyomino and Polycube Puzzles*, http://www.mattbusche.org/blog/article/polycube/. Checked March 2014.

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

[Golomb, 1994] Solomon W. Golomb. *Polyominoes Puzzles, Patterns, Problems and Packagings Revisited and expanded second edition*, Princeton Science Library, 1994.

[Horiyama et al., 2012] Takashi Horiyama, Takehiro Ito, Keita Nakatsuka, Akira Suzuki and Ryuhei Uehara. Packing trominoes is NP-complete, #P-complete and ASP-complete, In: *Proc. 24th Canadian Conference on Computational Geometry*, August 8-10, 2012.

[Jensen, 2003] Iwan Jensen. Counting polyominoes: a parallel implementation for cluster computing, In: *Proc. International Conference on Computational Science*, LNCS **2659**, 2003, 203-212.

[Jensen, 2009] Iwan Jensen. *Series for lattice animals or polyominoes*, http://www.ms.unimelb.edu.au/∼iwan/animals/Animals_ser.html. Checked April 2014.

[Klarner, 1967] D.A. Klarner. Cell growth problems, *Canadian Journal of Mathematics* **19**, 1967, 851-863.

[Korf, 2003] Richard E. Korf. Optimal rectangle packings: Initial results, In: *Proc. International Conference on Automated Planning and Scheduling*, 2003, 287-295.

[Redelmeir, 1981] D.H. Redelmeir. Counting polyominoes: Yet another attack, *Discrete Mathematics* **36**, 1981, 191-203.

[e Silva, 2014] T.O. e Silva. *Animal enumerations on the* $\{4, 4\}$ *Euclidean tiling*, http://sweet.ua.pt/tos/animals/a44.html, Checked April 2014.

[Takefuji, 1992] Y. Takefuji. *Neural Networking Parallel Computing*, Kluwer Academic Publishers, 1992.

Liitteet

I Tutkielman kaavoissa käytetyt termit sekä niiden selitteet

Tutkielmassa käytetään lukuarvojen yhteydessä alaindeksiä merkitsemään lukuarvon kantalukua. Oletusarvoisesti kaikkien lukuarvojen kantaluku tutkielmassa on 10, mutta tutkielmassa käytetään myös 2-kantalukujärjestelmää. Kaikissa 10-kantalukujärjestelmästä poikkeavissa luvuissa kantaluku on erikseen merkitty.

Lyhenne	Selite termi
A	Koko täytettävän alueen pinta-ala
a	Täytettävän alueen osan eli lohkon pinta-ala
b	Luvun esiintymien laskemiseen käytettävä bittien lukumäärä.
c	Polyominojen lukumäärä
n	Polyominojen asteluku
N	Lohkon numeroimiseen käytettävien lukujen [0..N[lukumäärä
W	Koko täytettävän alueen leveys
H	Koko täytettävän alueen korkeus
w	Täytettävän alueen osan leveus
h	Täytettävän alueen osan korkeus
i	Alueen indeksi, kappaleen sijainti alueella tai lohkolla
x	Kappaleen sijainti alueella vaakasuuntaan vasemmalta oikealla
y	Kappaleen sijainti alueella pystysuuntaan, ylhäältä alas

Taulukko 1: Tutkielman kaavoissa käytettyjen termien lyhenteet.

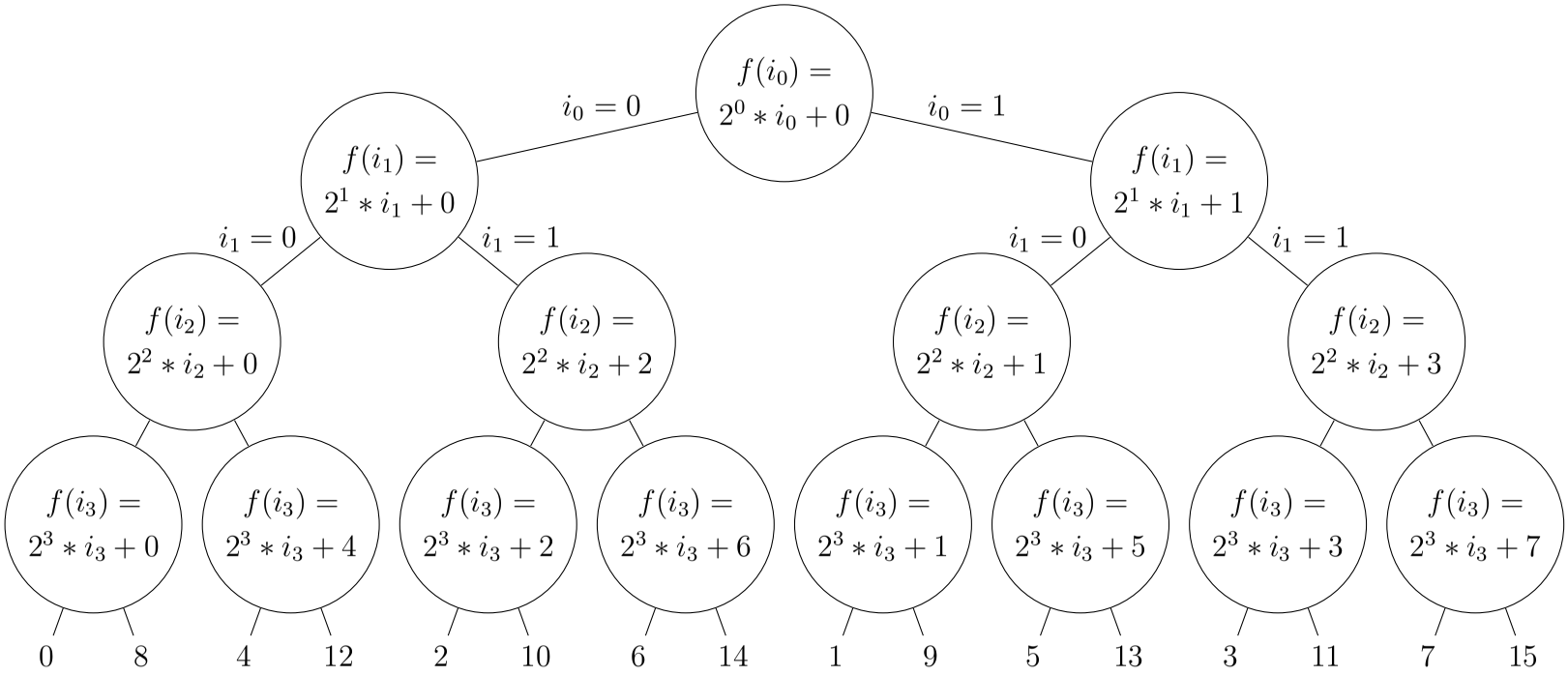
II Funktio täytettävän alueen jakajan laskemiseksi

Funktio tarkistaa, täyttääkö parametrina annettu jakaja a moduloidun bittikenttä algoritmin jakajalle asetetut ehdot. Funktio palauttaa totuusarvon true, mikäli arvo voidaan hyväksyä alueen jakajaksi. Muussa tapauksessa funktio palauttaa arvon false. Algoritmin toiminnan alussa funktion aputaulukko *match* alustetaan tyhjäksi arvolla -1. Tarkastusvaiheessa algoritmi tallentaa jakojäännöslukujen esiintymien koordinaatteja aputaulukkoon. Jakojäännöslukujen esiintymien etäisyyksiä verrataan toisiinsa ja mikäli kahden saman jakojäännöksen etäisyys on pienempi kuin polyominon asteluku *omino*, tarkistuksen kohteena oleva jakaja a hylätään.

```
function testMatch(a, omino, width) {
    var match = new Array();
    for(var count=0; count<width*omino +1; count++) {
        match[count] = -1;
    }
    var index = 1;
    for(var y=0; y<omino; y++ ) {
        for(var x = 0; x<width; x++) {
            if( x+y <= omino ) {
                if( match[index] > 0 ) {
                    if( match[index] <= omino-y ) {
                        return false;
                    }
                }
                match[index] = x;
            }
            index %= a;
            index++;
        }
    }
    return true;
}
```

Kuva 1 Funktio täytettävälle alueelle sopivan jakajan tarkistamiseksi.

Hakupuun solmuissa esitetään yksittäisen polyominon sijainnin jakojäännös $f(i)$ polyominon lohkokohtaisen sijainnin ollessa $i = [0..1]$. Hakupuun juuresta siirrytään lehtiä kohti vain silloin, kun polyominojen yhteenlaskettu tarkistussumma täsmää kyseisen lohkon tarkistussummaan. Siirtymän yhteydessä lohkokoko kaksinkertaistetaan ja polyominon sijainti koko täytettävällä alueella $A = 16$ tarkentuu. Haku alipuussa voidaan keskeyttää, mikäli solmussa ei löydetä lohkon tarkistussummaan täsmäävää tulosta.



Kuva 2 Binääripuu yksittäisen polyominon jakojäännöksen avulla tapahtuvaan sijainnin hakuun täytettävältä alueella.

IV Moduloidun bittikentän mukaiset lohkokoot 64 bitillä esitettynä

Taulukossa on esitetty suurimmat moduloidun bittikentän mukaiset lohkojaot a käyttäen $b = [2..10]$ bittiä yksittäisen luvun esiintymien laskemiseen ja $N = [2..32]$ bittiä polyominon peittämien ruutujen merkitsemiseen lohkolla. Taulukon esittämissä tuloksissa $a = N * (2^b - 1)$.

b = Bittiä / luku

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	...
2	6	14	30	62	126	254	510	1022	2046	...
3	9	21	45	93	189	381	765	1533	3069	...
4	12	28	60	124	252	508	1020	2044	4092	...
5	15	35	75	155	315	635	1275	2555	5115	...
6	18	42	90	186	378	762	1530	3066	6138	-
7	21	49	105	217	441	889	1785	3577	-	
8	24	56	120	248	504	1016	2040	-		
9	27	63	135	279	567	**1143**	-			
10	30	70	150	310	630	-				
11	33	77	165	341	-					
12	36	84	180	372						
13	39	91	195	-						
14	42	98	210							
15	45	105	225							
16	48	112	240							
17	51	119	-							
18	54	126								
19	57	133								
20	60	140								
21	63	147								
22	66	-								
...	...									
31	93									
32	96									

N = Numerointi

Taulukko 2: Lohkon enimmäiskoot 64 bittisellä lukuesityksellä.