Jesse Leskelä

Nonogram-ristikoiden ratkaisu syvyyshaulla

Tietotekniikan kandidaatintutkielma

30. huhtikuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Jesse Leskelä

Yhteystiedot: jesse.j.leskela@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: Nonogram-ristikoiden ratkaisu syvyyshaulla

Title in English: Solving nonogram puzzles using depth-first search

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 22+0

Tiivistelmä: Tutkielman aiheena on tutkia japanilaisten ristikoiden eli nonogrammien ratkaisua syvyyshakualgoritmia käyttäen. Nonogram on Japanissa 1990-luvulla kehitetty pulmapeli, jossa on tarkoituksena värittää ristikon ruutuja rivi- ja sarakevihjeiden avulla erilaisia ratkaisumenetelmiä hyödyntäen. Nonogram-ristikoiden ratkaisu on NP-täydellinen ongelma, joten tehokasta polynomisessa ajassa ja kaikille ristikoille toimivaa ratkaisualgoritmia ei ole löydetty. Tutkielmassa esitetään syvyyshakualgoritmin lisäksi vihjeisiin perustuva ratkontaalgoritmi sekä simuloitua jäähdytystä hyödyntävä algoritmi. Tutkielman lopussa vertaillaan eri julkaisuissa saatuja ratkontatuloksia edellä mainittujen algoritmien lisäksi myös näiden yhdistelmien osalta.

Tutkimusten perusteella voidaan havaita, että syvyyshakumenetelmä löytää aina ratkaisun mille tahansa ristikolle, mutta kyseinen menetelmä on erittäin hidas ristikon koon kasvaessa. Simuloitu jäähdytys toimii vastaavasti hyvin pienillä ristikoilla, mutta ristikon koon kasvaessa suoritusaika kasvaa eksponentiaalisesti. Vihjeiden avulla ratkominen onnistuu polynomisessa ajassa, mutta arvaamista vaativissa ristikoissa kyseinen menetelmä ei pysty löytämään kokonaista ratkaisua. Tällaisissa tapauksissa vihjeratkojien ja esimerkiksi syvyyshakualgoritmien yhdistelyn todettiin tuottavan hyviä tuloksia jopa suurten ristikoiden ratkaisussa.

Avainsanat: nonogram, japanilainen ristikko, pulmapeli, ratkaiseminen, algoritmi, syvyyshaku

Abstract: The aim of this thesis is to examine the solving process of japanese puzzles known as nonograms using depth-first search. Nonogram is a puzzle game invented in Japan in the 1990's, and the aim of the game is to color squares in a grid based on the given row and column hints. Solving nonogram puzzles is an NP-complete problem, which means that no efficient algorithm exists that can solve all puzzles in polynomial time. In addition to the depth-first search algorithm, this thesis presents an algorithm, often called a line solver, that uses row and column hints to solve the puzzle as well as an algorithm that uses simulated annealing to find a solution. The final part focuses on the results gathered from different publications regarding the solving time and efficiency of the aforementioned algorithms as well as their combined uses.

The results indicate that depth-first search will always find a solution for any given puzzle but the execution time will increase rapidly in conjunction with puzzle size. Simulated annealing works well with smaller puzzles, but the execution time increases exponentially as puzzle size increases. Line solvers can solve puzzles in polynomial time but cannot find a complete solution should guessing be required to advance the solving process. In such cases combining line solvers and other algorithms, depth-first search for instance, produced commendable results even on larger puzzles.

Keywords: nonogram, paint-by-picture, griddler, puzzle, solving, algorithm, DFS, depth-first search

Kuviot

Taulukot

Sisällys

1 Johdanto

Tutkielman tarkoituksena on tutkia japanilaisten ristikoiden eli nonogrammien toimintaa ja niiden ratkaisemista tietokoneen avulla. Nonogram on japanilainen pulmapeli, jossa on tarkoituksena värittää $m \times n$-kokoisen ruudukon ruutuja aina yksittäiseen ruutuun liittyvien rivi- ja sarakevihjeiden avulla. Ristikoista on olemassa sekä mustavalkoisia että värillisiä versioita, mutta tässä tutkielmassa rajoitutaan vain mustavalkoisiin ristikoihin. Tutkielman motivaationa toimi kirjoittajan kiinnostus ristikoihin ja niiden ratkaisemiseen tietokoneen avulla, mikä taas johti oman ratkaisijan ohjelmoimiseen.

Nonogram-ristikot keksittiin 1980-luvun loppupuolella, ja alkoivat esiintyä lehdissä ja muussa mediassa 1990- ja 2000-luvuilla. Nimi ”Nonogram” on peräisin ristikoiden toisen keksijän Non Ishidan etunimen ja sanan ”diagram” loppuosan yhdistelmästä. Englanninkielisessä kirjallisuudessa usein käytettyjä nimiä ovat esimerkiksi Paint By Picture, Griddler ja Picross. (“Griddler Puzzles and Nonogram Puzzles -Picture Logic Puzzles” 2017)

Tutkielmassa perehdytään nonogram-ristikoiden ratkaisuun syvyyshakualgoritmilla. Nonogram-ristikoiden ratkominen on NP-täydellinen ongelma (Ueda ja Nagao 1996; Hoogeboom ym. 2014), joten mikään algoritmi ei voi ratkaista kaikkia ristikoita polynomisessa ajassa. Polynomisessa ajassa toimivia ratkaisualgoritmeja on kehitetty, mutta kyseiset algoritmit eivät pysty ratkaisemaan ristikoita, joissa ratkaisuprosessin eteneminen vaatisi arvaamista (Batenburg ja Kosters 2009).

Luvussa 2 esitetään ristikoiden toiminta ja muutamia menetelmiä niiden ratkaisuun. Luvussa 3 esitetään kaksi erilaista syvyyshakualgoritmia ristikoiden ratkaisuun, ja luvussa 4 esitetään muita ratkaisualgoritmeja. Lopuksi luvussa 5 käydään läpi esitettyjen algoritmien suorituskykytuloksia ja luvussa 6 pohditaan tutkimusten tuloksia.

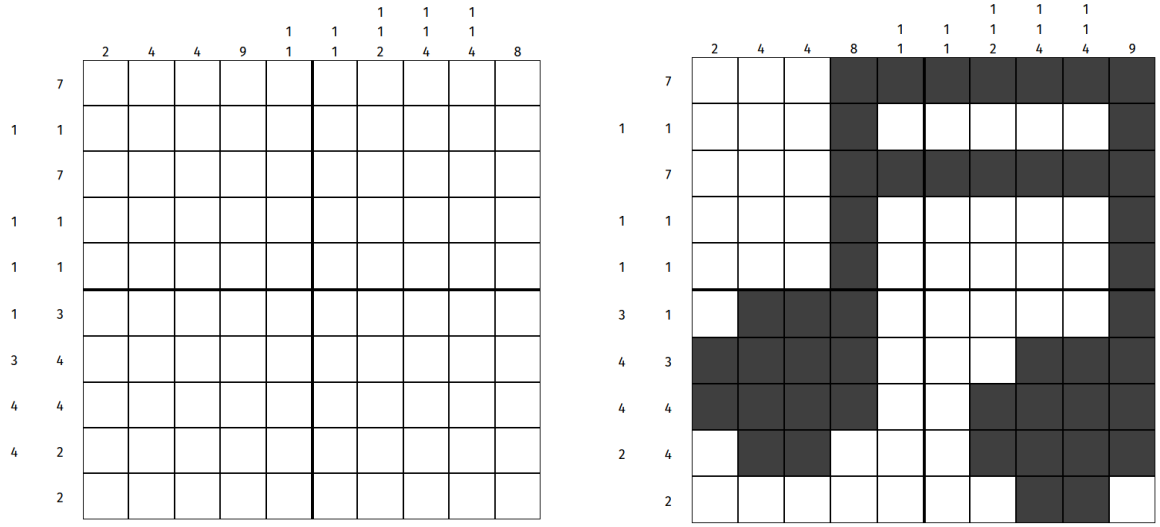
2 Nonogram-ristikot ja niiden ratkaiseminen

Tässä luvussa esitellään ristikoiden toimintaperiaate sekä muutama yksinkertainen menetelmä ristikoiden ratkaisemiseen.

2.1 Ristikoiden toiminta

Nonogram-ristikot ovat $m \times n$-kokoisia ruudukoita, joissa on yleensä ruudukon vasemmalla puolella ja yläpuolella jokaiseen riviin ja sarakkeeseen liittyvät vihjenumerot. Tavoitteena on värittää ruudukko vihjeiden perusteella siten, että vihjeiden numerot vastaavat ruudukkoon väritettyjä ruutuja. Kuviossa 1 on esimerkki tyhjästä ja ratkaistusta ristikosta.

Jos jokin rivi tai sarake sisältää esimerkiksi vihjenumerot ”4 1 2”, niin kyseisellä rivillä tai sarakkeella täytyy olla jossakin kohtaa ensin neljä peräkkäistä mustaa ruutua, yksi musta ruutu, ja kaksi peräkkäistä mustaa ruutua tässä järjestyksessä vasemmalta lukien. Eri vihjenumeroita vastaavien ruutujen välissä täytyy olla vähintään yksi tyhjä valkoinen ruutu, jotta ruudut voidaan yksikäsitteisesti tulkita vihjeitä vastaaviksi.



Kuvio 1. Esimerkki tyhjästä ja ratkaistusta ristikosta

2.2 Menetelmiä ristikoiden ratkaisuun

Nonogram-ristikoiden ratkaisuun on monia eri menetelmiä, ja suurin osa erilaisissa lehdissä ja kokoelmissa esiintyvistä ristikoista voidaan ratkaista vihjenumeroiden ja näiden tekniikoiden avulla (Batenburg ja Kosters 2012). On olemassa kuitenkin ristikoita, joita ei voi vihjeiden perusteella ratkaista, jolloin ruutuja joudutaan värittämään mustiksi esimerkiksi arvaamalla. Tällöin voidaan tehdä vääriä värityksiä, joten on tärkeää tarkistaa ettei uusi väritetty ruutu ole ristiriidassa vihjenumeroiden kanssa.

Seuraavissa alaluvuissa on esitetty muutama yksinkertainen ratkaisumenetelmä ristikoiden ratkaisemiseen (“How to solve a Griddler or Nonogram” 2002; Salcedo-Sanz ym. 2007). Yksinkertaisuuden vuoksi esitettyinä ovat ainoastaan rivit, mutta menetelmät toimivat aivan vastaavalla tavalla myös ristikon sarakkeisiin. Ruudut, jotka eivät voi olla mustaksi väritettyjä, on merkitty kuvioihin symbolilla ”X”.

2.2.1 Täysi rivi

Jos riviä vastaava vihjenumero on sama kuin sarakkeiden lukumäärä, niin kyseinen rivi voidaan ratkaista kokonaan (kuvio 2).

Kuvio 2. Täysi rivi

2.2.2 Täysi rivi vihjevälit huomioiden

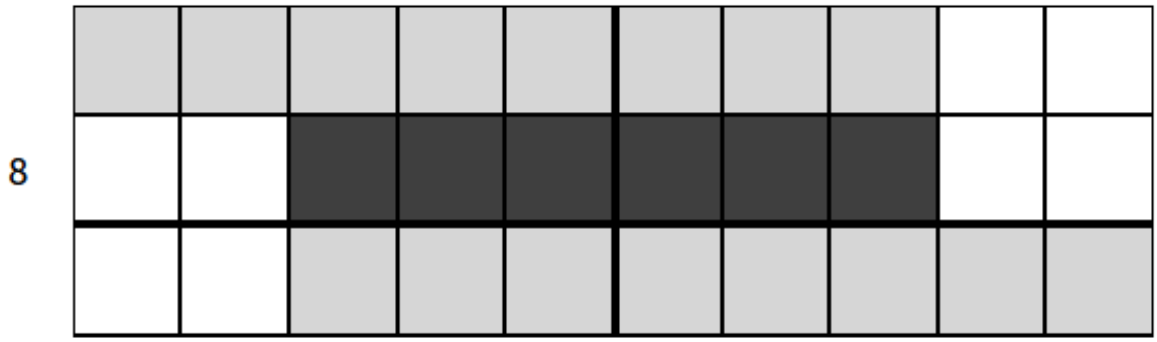
Jos rivin vihjenumeroiden ja niiden välissä olevien yksittäisten tyhjien ruutujen summa on sama kuin sarakkeiden lukumäärä, niin kyseinen rivi voidaan ratkaista kokonaan (kuvio 3).

Kuvio 3. Täysi rivi vihjevälit huomioiden

2.2.3 Pakolliset väritetyt ruudut

Jos rivillä oleva vihjenumero on suurempi kuin sarakkeiden lukumäärä jaettuna kahdella pyöristettynä alaspäin, niin rivillä täytyy olla keskellä mustia ruutuja. Toisin sanoen rivi täytetään vasemmalta ja oikealta vihjenumeron verran ja katsotaan, mitkä ruudut ovat kummassakin tapauksessa väritettyjä. Tätä havainnollistetaan kuviossa 4, missä harmaat ruudut esittävät vasemmalta ja oikealta täytettyä kahdeksan ruudun ketjua ja mustat ruudut ovat kummankin ketjun yhteiset ruudut.

Menetelmä toimii myös yleisessä tapauksessa, jolloin rivillä voi olla useampi kuin yksi numero. Tällöin ratkaistavan vihjenumeron oikealla ja vasemmalla puolella olevat numerot täytyy täyttää riville erotettuna yhdellä tyhjällä ruudulla. Pakolliset mustat ruudut tulkitaan tällöin samalla tavoin kuin yhden numeron tapauksessakin, kunhan vertailu tehdään aina tiettyä numeroa vastaavia värityksiä vertailemalla.



Kuvio 4. Pakolliset väritetyt ruudut keskimmäisellä rivillä

2.2.4 Pakolliset poissuljetut ruudut

Jos riviltä on saatu ratkaistua mustia ruutuja, niin yhden vihjeen tapauksessa voidaan poissulkea ruutuja vasemmalta ja oikealta puolelta sen verran että kummallakin puolella on tyhjiä ruutuja vain puuttuva määrä (ks. kuvio 5).

Kuvio 5. Pakolliset poissuljetut ruudut merkitty symbolilla ”X”

3 Ristikoiden ratkaiseminen syvyyshakualgoritmilla

Tässä luvussa esitellään ristikoiden ratkaisu kahta eri syvyyshakualgoritmia käyttäen.

3.1 Syvyyshaku riviratkaisuja käyttäen

Tässä tarkastellaan algoritmia, jossa ristikkoa ratkaistaan rivi kerrallaan siten, että jokaiselle riville muodostetaan kaikki mahdolliset ratkaisut. Saadut riviratkaisut yhdistetään toistensa kanssa, josta saadaan koko ristikolle ratkaisu. Kun kaikki ratkaisut on muodostettu, niin jokainen ratkaisu käydään läpi sarakkeittain ja katsotaan, mitkä saaduista ratkaisuista vastaavat sarakevihjeiden numeroita. (Yu, Lee ja Chen 2011; Jing ym. 2009) Algoritmin toimintaa on havainnollistettu kuviossa 6. Kyseiselle ristikolle on luotu kaikki neljä eri ratkaisua rivivihjeiden perusteella, mutta selvästi vain oikeassa alanurkassa oleva ratkaisu on oikea.

Algoritmi löytää aina kaikki ratkaisut mille tahansa ristikolle, mutta suorituskyvyltään algoritmi on erittäin hidas, sillä tarkistettavien ratkaisujen lukumäärä voi olla erittäin suuri (Wiggers ja Bergen 2004). Samasta syystä ei ole järkevää luoda ensin kaikkia mahdollisia ratkaisuja ja tarkistaa niitä jälkikäteen, sillä ratkaisujen tallentaminen veisi erittäin suuren määrän tallennuskapasiteettia. Parempi tapa on tarkistaa yksittäinen ratkaisu heti luonnin jälkeen, jolloin turhia ratkaisuja ei tarvitse tallentaa, ja algoritmi voidaan lopettaa heti jos ratkaisu on löytynyt.

Jing ym. (2009) esittävät algoritmista parannellun, haarautumismenetelmää (engl. branch-and-bound) käyttävän toteutuksen, jossa tarkistetaan jo ratkaisua muodostettaessa, onko tähän asti saatu osittaisratkaisu mahdoton. Tällöin ratkaisuja ei tarvitse muodostaa kokonaan, vaan mahdottomat yhdistelmät voidaan hylätä aikaisessa vaiheessa ja siirtyä seuraavan ratkaisun muodostamiseen. Prosessia voi ajatella puuna, jonka lehtisolmut sisältävät aina yhden ratkaisun ja lehtisolmuja edeltävät solmut ovat osittaisratkaisuja. Jos jonkin lehtisolmua edeltävän solmun sisältämä osittaisratkaisu todetaan virheelliseksi, niin kyseinen solmu sekä sen lapsisolmut voidaan hylätä ja siirtyä seuraavaan tutkittavaan solmuun.

3.2 Syvyyshaku peruutusmenetelmän avulla

Toinen tapa toteuttaa syvyyshaku on peruutusmenetelmää (engl. backtracking) hyödyntävä algoritmi, jossa ratkaistaan ristikkoa ruutu kerrallaan, kunnes päädytään vihjeiden kanssa ristiriitaan tai kunnes ristikko on ratkaistu. Jos päädytään ristiriitaan, niin viimeisin muutos kumotaan ja valitaan toinen vaihtoehto.

Algoritmi voidaan toteuttaa Więckowskin ja Shekhovtsovin esittämällä tavalla (Więckowski ja Shekhovtsov 2021), jossa valitaan jokin aloitusruutu, esimerkiksi vasemmassa yläkulmassa oleva ruutu, ja väritetään kyseinen ruutu mustaksi. Värityksen jälkeen tarkistetaan jollakin tarkistusfunktiolla, onko tehty väritys sallittu rivi- ja sarakevihjeet huomioiden. Jos väritys on sallittu, niin algoritmia voidaan jatkaa seuraavaan ruutuun. Jos väritys taas ei ole sallittu, niin tehty väritys kumotaan. Nyt tarkistetaan ruutu jälleen tarkistusfunktiolla ja edetään seuraavaan ruutuun, jos väritys on sallittu. Muuten palataan edelliseen ruutuun ja kumotaan kyseisen ruudun viimeisin väritys, ja tätä toistetaan kunnes ristikko on ratkaistu.

Więckowski ja Shekhovtsov (2021) kuvaavat myös pseudokooditoteutuksen algoritmista, joka on esitettynä alla. Koodin rakenne on säilytetty samanlaisena, mutta muuttujia on uudelleennimetty kuvaavammiksi ja koodiin on lisätty toimintaa selkeyttäviä kommentteja. Huomattavaa on, että viimeiselle riville päästään vain siinä tapauksessa että ristikolle ei löydy ollenkaan ratkaisua, mikä taas tarkoittaa sitä että rivi- ja sarakevihjeet ovat toistensa kanssa ristiriidassa. Tarkistusfunktio palauttaa kummassakin kutsussa tällöin arvon epätosi ja algoritmin suoritus loppuu.

```
def findSolution(row, col):

    // Jos ollaan käyty läpi kaikki rivit,
    // niin ristikko on ratkaistu
    if row == height:
        return True

    // Määritetään seuraavan ruudun rivi ja sarake
    if col + 1 == width:
```

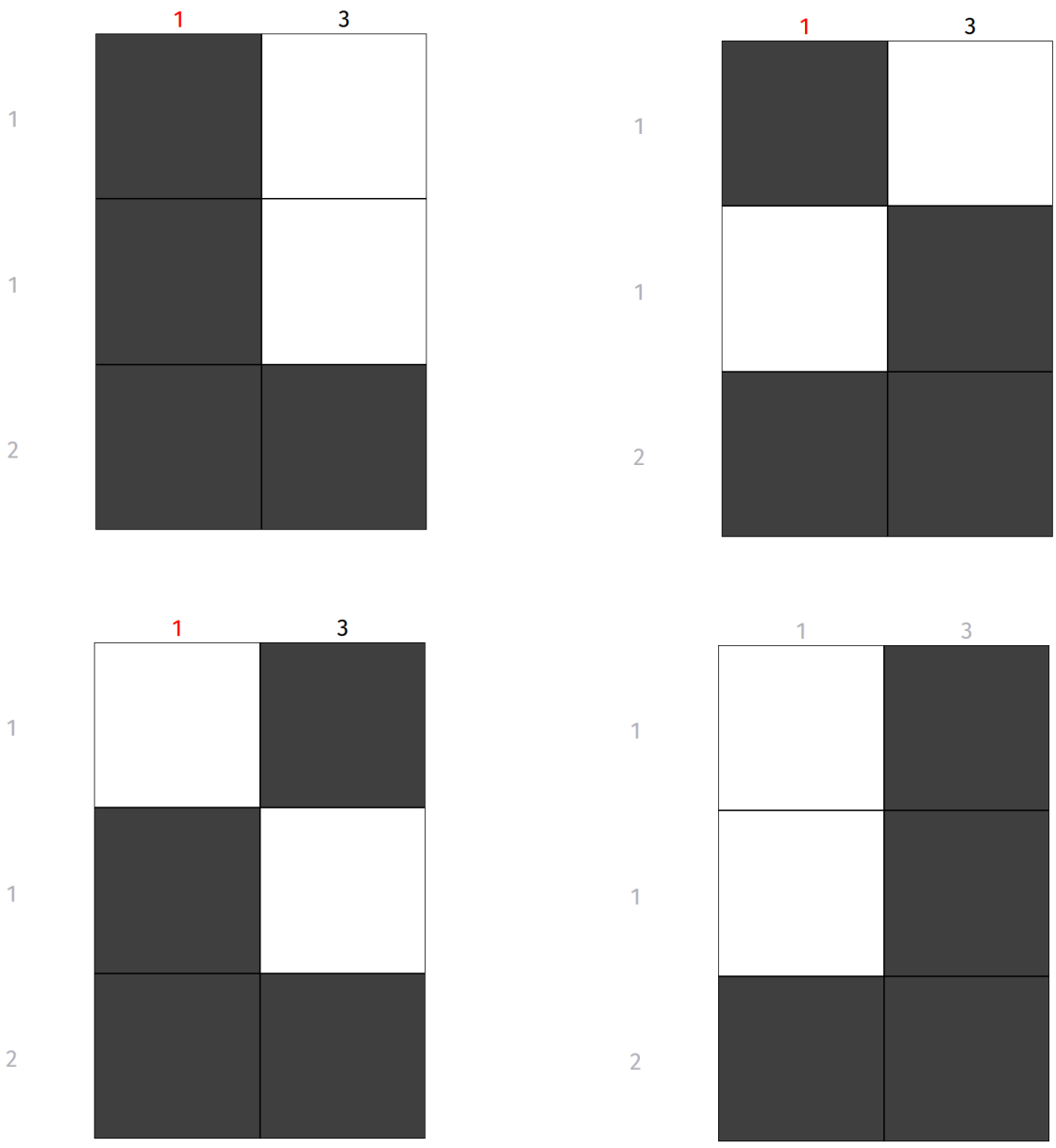
```
        nextRow = row + 1
    else:
        nextRow = row
    nextCol = (col + 1) mod width

    // Asetetaan ruutu riviltä row ja sarakkeelta
    // col mustaksi ja tarkistetaan verify-funktiolla.
    // Jos väritys on sallittu, niin kutsutaan
    // rekursiivisesti seuraavan ruudun koordinaateilla.
    board[row][col] = 1
    if verify(row, col) and findSolution(nextRow, nextCol):
        return True

    // Sama kuin yllä, mutta ruutu asetetaan
    // tyhjäksi ja tarkistetaan taas verify-funktiolla.
    board[row][col] = 0
    if verify(row, col) and findSolution(nextRow, nextCol):
        return True

    // Tänne päästään vain jos ristikolla ei ole ratkaisua
    return False
```

Edellä esitetyssä algoritmissa joudutaan pahimmassa tapauksessa käymään kaikki ruudut läpi, jolloin eri yhdistelmien lukumäärä $m \times n$-kokoiselle ristikolle on 2^{mn} (Batenburg ja Kosters 2009). Algoritmi löytää aina ratkaisun jos sellainen on olemassa, mutta on erittäin hidas suurikokoisilla ristikoilla (Jing ym. 2009).



Kuvio 6. Syvyyshaulla riviratkaisuja hyödyntäen löydetyt neljä ratkaisua

4 Muita ratkaisualgoritmeja

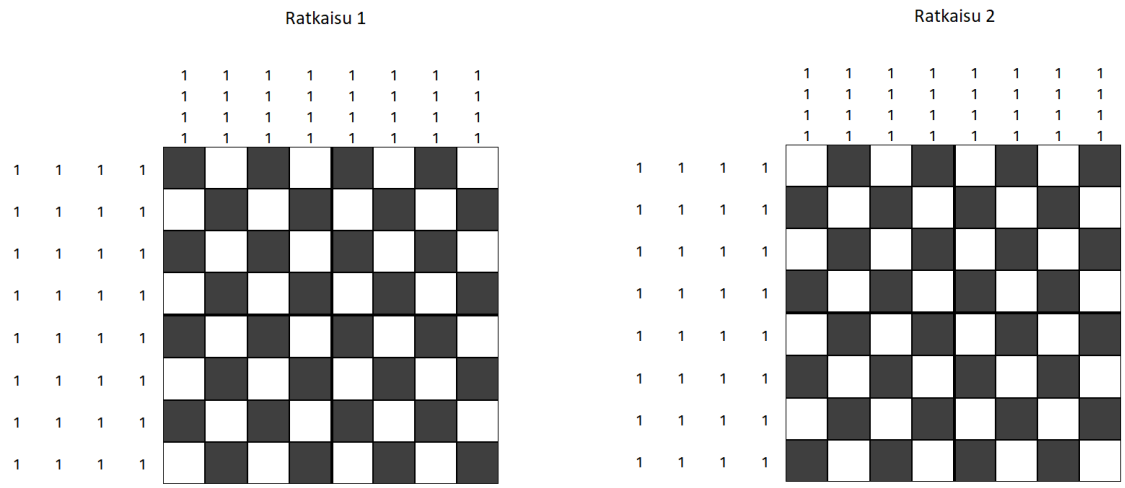
Tässä luvussa esitetään muutamia muita ratkaisualgoritmeja. Vihjeiden perusteella ratkovalle algoritmille käytetään usein englanninkielisessä kirjallisuudessa nimeä *line solver*, jolle suomenkielisenä vastineena käytetään tässä tutkielmassa nimeä *vihjeratkoja* (kirjoittajan itse keksimä, ei virallinen nimi).

4.1 Vihjeiden perusteella ratkominen

Luvussa 2.2 mainittiin, että suurin osa lehdissä ja kokoelmissa esiintyvistä ristikoista voidaan ratkaista vihjenumeroiden avulla käyttäen loogisia päättelysääntöjä. Täten on mahdollista toteuttaa ratkaisualgoritmi, joka käyttää samoja sääntöjä ristikon ratkaisemiseen. Esimerkiksi Yu, Lee ja Chen (2011) kuvaavat julkaisussaan omaa ratkaisualgoritmiaan, jossa hyödynnetään samantapaisia päättelytekniikoita kuin mitä luvussa 2.2 kuvailtiin.

Vihjeiden perusteella ratkominen on mahdollista tehdä polynomisessa ajassa (Batenburg ja Kosters 2009; Wang ja Tang 2014; Berend ym. 2014), mutta toisin kuin syvyyshaussa, vihjeiden perusteella ratkominen ei aina ole mahdollista. Tällöin joudutaan turvautumaan muihin algoritmeihin, tai esimerkiksi arvailuun. Usein tällaisia vaikeasti ratkeavia ristikoita ovat ristikot, joihin on mahdollista muodostaa useita eri ratkaisuja ja joissa on paljon pieniä, yhden tai kahden ruudun kokoisia segmenttejä (Batenburg ja Kosters 2012; Wang ja Tang 2014). Esimerkki edelläkuvatusta usean ratkaisun sisältävästä ristikosta on kuviossa 7 esitetty ristikko, jonka rivi- ja sarakevihjeet muodostavat shakkilautaa muistuttavan lopputuloksen. Kyseiseen ristikkoon on olemassa useampi kuin yksi ratkaisu eikä vihjeiden perusteella pysty yksikäsitteisesti määrittämään väritettyjä ruutuja.

Vaikka vihjeiden perusteella ratkaisu ei olisi mahdollista, niin osittaisratkaisu voi olla muodostettavissa. Tällöin osa ruuduista voi olla väritettyjä ja osa poissuljettuja. Jäljelle jääviä tyhjiä ruutuja ei voida ratkaista vihjeiden avulla, mutta ne voidaan ratkaista esimerkiksi syvyyshaulla. Yu, Lee ja Chen (2011) hyödyntävät tätä menetelmää ristikoiden ratkaisuun, kun taas Batenburg ja Kosters (2009) käyttävät yksittäistä arvausta, jonka jälkeen yritetään taas ratkoa vihjeiden avulla. Etuna jälkimmäisessä tavassa on mahdollisesti nopeampi ratkaisun



Kuvio 7. Shakkilautaa muistuttava ristikko, jolla kaksi ratkaisua

löytyminen, mutta koska arvauksia tehdään vain yksi, niin on mahdollista että yksittäinen ruutu ei tuo lisäinformaatiota vihjeiden avulla ratkaisuun, jolloin ristikkoa ei voida ratkaista pidemmälle.

4.2 Simuloitu jäähdytys

Wang ja Tang (2014) esittävät simuloitua jäähdytystä (engl. simulated annealing) käyttävän ratkaisualgoritmin, jonka tarkoituksena on ratkoa tehokkaasti ristikoita, joilla on useita ratkaisuja. Algoritmin tarkoitus ei ole kuitenkaan löytää kaikkia mahdollisia ratkaisuja, vaan yhden ratkaisun löytäminen riittää. Kuten syvyyshaussa, yhdistelmien lukumäärä voi olla erittäin suuri, joten suurilla ristikoilla algoritmin ratkaisuaika kasvaa eksponentiaalisesti ristikon koon kasvaessa (Wang ja Tang 2014).

Algoritmi laskee väritettyjen ruutujen yhteismäärän vihjeiden perusteella ja värittää kyseisen määrän satunnaisesti valittuja ruutuja. Seuraavaksi algoritmi valitsee kaksi ratkaisun suhteen eniten ristiriidassa olevaa ruutua, joista toinen on väritetty ja toinen tyhjä. Alussa lämpötila on korkea, jolloin vaihtoja tehdään paljon ratkaisun parantamiseksi, mutta algoritmin edetessä parempi ratkaisu löytyy ja lämpötila laskee, jolloin vaihtojen määrä laskee.

Ruutujen ristiriita-arvo lasketaan erilaisilla tarkistuksilla, joissa verrataan esimerkiksi ruutu-

jen lukumääriä ja väritettyjen ruutujen ketjuja. Jokaiselle ruudulle saadaan ristiriita-arvo, ja suurimmat arvot sisältävät ruudut valitaan. Vaihtoalgoritmi voi jäädä jumiin lokaaliin optimiin joka ei ole kokonainen ratkaisu, joten vaihtoalgoritmi valitsee ruutuja ristiriita-arvoista lasketuilla todennäköisyyksillä.

Kaikkien ruutujen yhteenlaskettua ristiriita-arvoa käytetään lämpötilan hallintaan. Aina kun vaihto on parannus edelliseen tilaan niin lämpötila laskee ja vastaavasti kun vaihto on huonompi kuin edellinen tila niin lämpötilaa nostetaan. Lämpötilan nosto voi johtaa huonomman vaihtoehdon valintaan, mutta sen avulla voidaan päästä pois lokaalista optimista.

5 Algoritmien suorituskykyvertailu

Tässä luvussa vertaillaan julkaisuissa esitettyjen algoritmien suorituskykyä erikokoisten ristikoiden ratkaisemisessa ja pohditaan saatuja tuloksia. Suorityskykytestit ovat artikkelien kirjoittajien suorittamia, ja niissä käytetyt testiristikot ovat myös kirjoittajien itse valitsemia. Testiristikot sekä niiden koot vaihtelevat julkaisujen välillä, joten tässä esitetty algoritmien vertailu on pääosin suuntaa antava.

Taulukossa 1 esitetään eri algoritmien suorituskykytuloksia. Yhdistelmäalgoritmit yhdistävät vihjeiden avulla ratkomista ja jotain toista algoritmia keskenään: Yun, Leen ja Chenin (2011) toteutus hyödyntää syvyyshakua ja Berendin ym. (2014) toteutus hyödyntää syvyyshaun kaltaista algoritmia. Julkaisujen testeissä käytetään useita erikokoisia ristikoita, ja suurin ristikko -sarake on taulukossa vain havainnollistamassa ristikon koon suhdetta suurimpaan ratkonta-aikaan.

Taulukko 1. Eri algoritmien suorituskyky testiristikoilla. Tähdellä (*) merkityt toteutukset eivät ratkaisseet testiristikoita kokonaan.

Algoritmi	Toteutus	Suurin ristikko	Ratkonta-aika	
			Keskimäärin	Suurin
Syvyyshaku	Jing ym. (2009)	$> 25 \times 25$	$>$ 1 min	$>$ 2 pv
	Yu ym. (2011)	$> 25 \times 25$	$>$ 1 min	$>$ 2 pv
	Więckowski ym. (2021)	20×20	$<$ 5 min	$>$ 1 t
	Wiggers ym. (2004) *	15×15	$>$ 1 t	$>$ 1 t
Vihjeratkoja	Salcedo-Sanz ym. (2007)	145×60	$<$ 1 s	$<$ 1 s
	Batenburg ym. (2009)	$> 40 \times 40$	-	$<$ 1 min
Simuloitu jäähdytys	Wang ym. (2014)	25×25	$<$ 30 s	$<$ 5 min
Yhdistelmä	Yu ym. (2011)	$> 25 \times 25$	$<$ 1 s	$<$ 5 min
	Berend ym. (2014)	$\geq 30 \times 30$	$<$ 1 s	$>$ 2 min

6 Pohdinta

Tuloksista nähdään, että syvyyshakualgoritmin suoritusaika kasvaa huomattavasti ristikon koon kasvaessa ja että suoritusajoissa on erittäin suurta vaihtelua, sillä pahimmassa tapauksessa ratkaisun löytäminen saattaa vaatia lähestulkoon kaikkien yhdistelmien läpikäynnin. Pienillä ristikoilla algoritmi toimii hyvin, mutta ristikon koon kasvaessa syvyyshakua on järkevämpää käyttää ratkaisuprosessin tukena, esimerkiksi jos vihjeiden avulla ratkomisessa jäädään jumiin ja ratkaisuprosessin eteneminen vaatii arvaamista.

Vihjeratkojien tuloksista kannattaa huomioida, että Salcedo-Sanzin ym. (2007) julkaisussa testattavat ristikot olivat vihjeiden perusteella ratkottavissa, ja Batenburgin ja Kostersin (2009) testiristikot voitiin ratkaista vihjeitä ja enintään yksittäisiä arvauksia hyödyntäen. Kyseiset ristikot voisi täten mieltää kaikkien ristikoiden helpommiksi erikoistapauksiksi. Tästä huolimatta vihjeiden avulla ratkominen on suurillakin ristikoilla tehokas ratkaisumenetelmä, sillä kyseiset ratkaisualgoritmit ratkoivat syvyyshakuun verrattuna merkittävästi suurempia ristikoita huomattavasti nopeammassa ajassa.

Algoritmien yhdistely tuotti erittäin hyviä tuloksia, etenkin Yun, Leen ja Chenin (2011) julkaisussa, jossa pahimmassa tapauksessa päästiin yli kahden päivän ratkaisuajasta alle puoleen minuuttiin. Berendin ym. (2014) ratkaisija toimi tehokkaasti, varsinkin kun vain 0,4 % julkaisussa käytetyistä satunnaisesti luoduista testiristikoista pystyttiin ratkaisemaan kokonaan pelkkien vihjenumeroiden avulla. Jälkimmäisestä kannattaa myös mainita, että ainoastaan 29 ristikon ratkomisessa yli 7000 testiristikon joukosta kesti yli kymmenen sekuntia, ja näistä 29:stä vain viiden ratkomisessa kesti yli kaksi minuuttia.

Johtopäätöksenä tuloksista voidaan sanoa, että syvyyshaun tai vastaavien menetelmien käyttö yksinään ei ole järkevää ristikoiden ratkaisussa, vaan vihjenumeroiden avulla ratkaiseminen kannattaa aina olla ensisijainen valinta. Syvyyshakua kannattaa käyttää vasta kun vihjeiden avulla on saatu useita ruutuja ratkaistua, jolloin tarkistettavien yhdistelmien lukumäärä pienenee. Pahimmassa tapauksessa vihjeiden avulla ei voida ratkoa yhtään ruutua, jolloin kaikkien yhdistelmien läpikäynti esimerkiksi syvyyshaulla on ainoa vaihtoehto.

7 Yhteenveto

Nonogram-ristikot ovat toiminnaltaan yksinkertaisia pulmapelejä, mutta yksinkertaisuudestaan huolimatta ristikoiden ratkaiseminen tietokoneella on osoittautunut haastavaksi. Ristikoiden ratkaiseminen on NP-täydellinen ongelma, joten lyhyessä ajassa ja kaikille ristikoille toimivaa ratkaisualgoritmia ei ole löydetty. Syvyyshakumenetelmä toimii hyvin pienikokoisilla ristikoilla, mutta koon kasvaessa tarkistettavien yhdistelmien lukumäärä kasvaa erittäin nopeasti. Suurin osa ristikoista voidaan ratkaista nopeasti rivi- ja sarakevihjeitä hyödyntäen, mutta kaikkia ristikoita ei tällä tavalla pystytä ratkomaan kokonaan.

Varsinkin ristikot, joilla on useita eri ratkaisuja, ovat osoittautuneet vaikeasti ratkeaviksi, sillä ristikkoa ei tällöin välttämättä voida ratkaista vihjeiden avulla tehokkaasti vaan joudutaan käyttämään hitaampia menetelmiä, esimerkiksi syvyyshakua. Näissä tapauksissa vihjeiden avulla ratkomisen ja sitä täydentävän ratkaisumenetelmän, esimerkiksi syvyyshaun, todettiin tuottavan yleisesti hyviä tuloksia jopa suuria ristikoita ratkottaessa.

Lähteet

Batenburg, K.J. ja W.A. Kosters. 2009. “Solving Nonograms by combining relaxations”. Advances in combinatorial image analysis, *Pattern Recognition* 42 (8): 1672–1683. ISSN: 0031-3203. https://doi.org/https://doi.org/10.1016/j.patcog.2008.12.003.

———. 2012. “ON THE DIFFICULTY OF NONOGRAMS”, viitattu 8. helmikuuta 2023. https://liacs.leidenuniv.nl/~kosterswa/nonodec2012.pdf.

Berend, Daniel, Dolev Pomeranz, Ronen Rabani ja Ben Raziel. 2014. “Nonograms: Combinatorial questions and algorithms”. *Discrete Applied Mathematics* 169:30–42. ISSN: 0166-218X. https://doi.org/https://doi.org/10.1016/j.dam.2014.01.004.

“Griddler Puzzles and Nonogram Puzzles -Picture Logic Puzzles”. 2017. Viitattu 9. helmikuuta 2023. https://www.puzzlemuseum.com/griddler/gridhist.htm.

Hoogeboom, Hendrik Jan, Walter A. Kosters, Jan N. Van Rijn ja Jonathan K. Vis. 2014. “Acyclic constraint logic and games”. Cited by: 4; All Open Access, Green Open Access, *ICGA Journal* 37 (1): 3–16. https://doi.org/10.3233/ICG-2014-37102.

“How to solve a Griddler or Nonogram”. 2002. Viitattu 10. maaliskuuta 2023. https://www.puzzlemuseum.com/griddler/gridins.htm.

Jing, Min-Quan, Chiung-Hsueh Yu, Hui-Lung Lee ja Ling-Hwei Chen. 2009. “Solving Japanese puzzles with logical rules and depth first search algorithm”. Teoksessa *2009 International Conference on Machine Learning and Cybernetics,* 5:2962–2967. https://doi.org/10.1109/ICMLC.2009.5212614.

Salcedo-Sanz, Sancho, Emilio G. Ortiz-Garcia, Angel M. Perez-Bellido, Antonio Portilla-Figueras ja Xin Yao. 2007. “Solving Japanese Puzzles with Heuristics”. Teoksessa *2007 IEEE Symposium on Computational Intelligence and Games,* 224–231. https://doi.org/10.1109/CIG.2007.368102.

Ueda, Nobuhisa ja Tadaaki Nagao. 1996. “NP-completeness Results for NONOGRAM via Parsimonious Reductions”. Viitattu 8. helmikuuta 2023. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1bb23460c7f0462d95832bb876ec2ee0e5bc46cf.

Wang, Wen-Li ja Mei-Huei Tang. 2014. “Simulated Annealing Approach to Solve Nonogram Puzzles with Multiple Solutions”. Complex Adaptive Systems Philadelphia, PA November 3-5, 2014, *Procedia Computer Science* 36:541–548. ISSN: 1877-0509. https://doi.org/https://doi.org/10.1016/j.procs.2014.09.052.

Więckowski, Jakub ja Andrii Shekhovtsov. 2021. “Algorithms Effectiveness comparison in solving Nonogram boards”. Knowledge-Based and Intelligent Information and Engineering Systems: Proceedings of the 25th International Conference KES2021, *Procedia Computer Science* 192:1885–1893. ISSN: 1877-0509. https://doi.org/https://doi.org/10.1016/j.procs.2021.08.194.

Wiggers, Wouter ja Willem van Bergen. 2004. “A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms”. Teoksessa *Twente student conference on IT*. Citeseer. Viitattu 11. maaliskuuta 2023. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=7f10e79106bb147def51b03e5662be08490e15f8.

Yu, Chiung-Hsueh, Hui-Lung Lee ja Ling-Hwei Chen. 2011. “An efficient algorithm for solving nonograms”. *Applied Intelligence* 35:18–31. Viitattu 14. maaliskuuta 2023. https://doi.org/https://doi.org/10.1007/s10489-009-0200-0.