Maria Zudina

NP-TÄYDELLISYYS PELEISSÄ

Tietotekniikan pro gradu -tutkielma

11. toukokuuta 2022

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2022

TIIVISTELMÄ

Tekijä: Maria Zudina

Yhteystiedot: zudinamaria@gmail.com

Ohjaajat: Antti Valmari

Työn nimi: NP-täydellisyys peleissä

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 69+0

Tiivistelmä: Tässä työssä tarkastellaan sitä, mikä on NP-täydellisyys, miten se esiintyy peleissä sekä millaisia todistusmenetelmiä on käytetty esimerkkipelien NP-täydellisyyden todistuksissa. Ensimmäiseksi avataan NP-täydellisyyden käsitettä muiden ongelmien vaikeusluokkien ohella, sitten tarkastellaan muutamaa konkreettista käyttöesimerkkiä pelien yhteydessä ja lopuksi pohditaan sekä ilmiön että työn merkittävyyttä. Pohdintaosiossa käydään läpi muun muassa se, miksi NP-täydellisyys esiintyy niin monessa pelissä ja millaista aiheen opiskelu on opiskelijan näkökulmasta. Työn tavoite on selventää NP-täydellisyyden käsitettä ja sen ilmenemistä pelien yhteydessä ensisijaisesti yliopistotason opiskelijoille, joiden opintoihin aiheen opiskelu sisältyy.

Avainsanat: NP-täydellisyys, pelit, vaikeusluokat, P, NP, NP-kovuus

ABSTRACT

Author: Maria Zudina

Contact information: zudinamaria@gmail.com

Supervisor: Antti Valmari

Name of the publication: NP-completeness in games

Subject: Master's thesis

Major: Mathematical Information Technology

Pages: 69+0

Abstract: In this thesis we will go through what is NP-completeness and how does it show in games. We start by introducing the concept of NP-completeness along with other complexity classes to the reader, after which we go a bit deeper by analyzing some existing examples of proving NP-completeness in games, giving the reader tools and references to continue research on the subject. In the analysis part of the thesis, we will also touch upon why is NP-completeness present in so many games and what's it like to learn about the subject from the student's perspective. The goal of this thesis is to clarify the concept of NP-completeness and its presence in games, primarily targeting university students, who are learning about the subject through their curriculum.

Keywords: NP-complete, games, complexity classes, P, NP, NP-hard

Termiluettelo

3-Partition	Yksi NP-täydellinen ongelma, esitellään luvussa 2.
3-SAT	Yksi NP-täydellinen ongelma, esitellään luvussa 2.
Aikavaatimus/Aikaluokka	Kertoo siitä, kuinka kauan algoritmin suoritus saa enintään kestää ongelman kokoon nähden. Eng. "*Time Complexity*".
DTM	Eng. Deterministic Turing Machine, suomeksi deterministinen Turing-kone.
Ei tehokkaasti ratkaistavissa	Eng. "*Intractable*" eli ongelma, jolle ei ole olemassa polynomiajassa toimivaa algoritmia.
Eksponentiaalinen aika	Aikavaatimus, jonka yläraja nousee eksponenttiin. Esim. $O(2^n)$, $O(n^n)$.
Eksponentiaalinen tila	Algoritmin käyttämän muistin määrä, jonka yläraja nousee eksponenttiin saakka. Vrt. "Eksponentiaalinen aika".
EXPSPACE	Päätösongelmien joukko, jotka ovat ratkaistavissa deterministisen Turing-koneen avulla eksponentiaalista muistitilaa käyttäen.
EXPTIME	Päätösongelmien joukko, jotka ovat ratkaistavissa deterministisen Turing-koneen avulla eksponentiaalisessa ajassa.
Instanssi	Instanssi on ongelman yksittäinen ilmentymä. Jos ongelmana on esimerkiksi Sudoku, instanssina toimisi yksi tietty esitäytetty Sudoku-kenttä.
Klausuuli	Totuusarvoisten muuttujien äärellinen joukko. Käytetään mm. SAT ja 3-SAT yhteydessä. Esim. $(x_1, y_3, \neg z_{15})$
Komplementti	Päätösongelma on toisen päätösongelman komplementti, jos ja vain jos sen "kyllä" ja "ei" vastaukset on käännetty ympäri.

	Esimerkiksi ”Onko tämä ongelma ratkaistavissa?” -ongelman komplementti olisi muotoa ”Onko tämä ongelma ratkeamaton?”.
Literaali	Muuttujan arvo.
Muuntaminen	Eng. ”*Reduction*”. Keino muuntaa yksi ongelma toiseksi. Ongelma on onnistuneesti muunnettu, jos ja vain jos kaikki sen ”kyllä” ja ”ei” -instanssit vastaavat toisen ongelman vastaavia instansseja.
NDTM	Eng. “*Nondeterministic Turing Machine*”, suomeksi epä-deterministinen Turing-kone.
NP-kova	NP-luokan vaikeimpien tehtävien joukko. Kaikki NP-luokan ongelmat ovat muunnettavissa NP-kovaksi ongelmaksi polynomiajassa.
NP-täydellinen	Ongelma on NP-täydellinen jos ja vain jos se on NP-kova ja kuuluu NP-vaikeusluokkaan.
NP-vaikeusluokka	Tähän luokkaan kuuluvat ne päätösongelmat, jotka ovat ratkaistavissa polynomiajassa epädeterministisen Turing-koneen avulla. Näille ongelmille ei välttämättä ole olemassa polynomiajassa toimivia ratkaisualgoritmeja, mutta niiden ”kyllä”-vastaus voidaan tarkistaa polynomiajassa. Lyhenne englannin sanoista “*Nondetermeistic polynomial time*”, suomeksi epädeterministinen polynomimuotoinen aika. (Termin suomentaja Kimmo Pietiläinen, Kultainen pääsylippu 2014)
Partition (problem)	Yksi NP-täydellinen ongelma, esitellään luvussa 2.
Polynomiaika	Aikavaatimus, jonka yläraja ei nouse eksponenttiin saakka. Esim. $O(1), O(5), O(n), O(n \log n)$. Meidän jaossa tämän termin alle kuuluvat myös mm. lineaarinen ja vakioaika.

Polynomitila	Algoritmin käyttämän muistitilan määrä, jonka yläraja ei nouse eksponenttiin saakka. Vrt. ”Polynomiaika”.
PSPACE	Päätösongelmien joukko, jotka voidaan ratkaista Turing-koneen avulla polynomimääräistä muistitilaa käyttäen.
P-vaikeusluokka	Tähän luokkaan kuuluvat ne ongelmat, joille on olemassa polynomiajassa toimiva algoritmi, joka ratkaisee ne. Lyhenne englannin sanoista ”*Polynomial time*”, suomeksi polynomimuotoinen aika. (Termin suomentaja Kimmo Pietiläinen, Kultainen pääsylippu 2014)
Päätösongelma	Ongelma, jonka vastaus on muotoa ”kyllä/ei” saamansa syötteen perusteella.
SAT	Yksi NP-täydellinen ongelma, esitellään luvussa 2.
Syötteen pituus	Eng. ”*Input length*” on sen merkkijonon pituus, jolla saadaan kuvattua ongelman instanssi.
Tehokkaasti ratkaistavissa	Eng. ”*Tractable*” eli ongelma, jolle on olemassa polynomi-ajassa toimiva ratkaisualgoritmi.
Unaarimuoto	Unaari esitysmuoto, jossa asiat voi esittää unaarissa, eli yhtä merkkiä käyttäen. Esimerkiksi luvut 1, 2, 3 voidaan unaarimuodossa esittää muodossa 1, 11, 111.
Vaikeusluokka	Tapa luokitella ongelmia sen mukaan, kuinka tehokkaita algoritmeja niiden ratkaisemiseksi on olemassa.

Kuvat

Kuva 1: NP ja co-NP visualisointi 17
Kuva 2: Muunnos A → B, jotta ratkaisematon A voi käyttää B:n ratkaisua. 19
Kuva 3: Muunnos A→B, kun A on todistetusti ilman tehokasta ratkaisua. 19
Kuva 4: Muunnos B→A, ratkaistaan B käyttäen A:n ratkaisua. 20
Kuva 5: Vaikeusluokkien visualisointi, mikäli P ≠ NP 25
Kuva 6: Ongelmien luokittelun visuaalisuusero, riippuen siitä, onko P = NP 27
Kuva 7: Icosian game, pohjautuu Hamiltonian Circleen, esimerkkiratkaisuineen. 29
Kuva 8: The Hamiltonian Circuit -peli, tehtävänä laatia yhtenäinen tie, joka kulkee joka ikisen ruudun kautta. Lähde: GoodThinkingGames Ltd. 29
Kuva 9: Bark and Fester LLC:n RoTopo, tehtävänä juosta kaikkien ruutujen läpi kerran ja päästä loppuun. 30
Kuva 10: Tetrominot, eli Tetriksen pelinappulat. Huomataan, että jokainen muodostuu tasan 4 palikasta, näin ollen pinta-ala on kaikilla sama. 32
Kuva 11: Tetris-pelialueen rakenne mukautettuna 3-partition -ongelman muotoon. Kuvan lähde: *Tetris is Hard, Made Easy*, suomennettu. 34
Kuva 12: Todistuksen käyttämä tetrominosekvenssi. Lähde: *Tetris is Hard, Made Easy*, suomennettu 37
Kuva 13: Vaon täyttö tetrominosekvenssiä käyttäen. Lähde: *Tetris is Hard, Made Easy*, suomennettu. 39
Kuva 14: Vakojen täyttäminen tasaisiksi. Lähde: *Tetris is Hard, Made Easy* 39
Kuva 15: Täytetyt vaot ja täyttöalueen täyttöä havainnollistava I-tetrominon malliasento 40
Kuva 16: Vaihtoehtoisten sijaintien aiheuttamat saavuttamattomat kohdat. Lähde: *Tetris is Hard, Made Easy* 42
Kuva 17: Aloituspalan vaihtoehtosijoitukset. Lähde: *Tetris is Hard, Made Easy* 42
Kuva 18: Keskikohta-palojen vaihtoehtosijoitukset. Lähde: *Tetris is Hard, Made Easy* ... 43
Kuva 19: Loppupalojen vaihtoehtosijoitukset Lähde: *Tetris is Hard, Made Easy* 43
Kuva 20: Yleinen runko 3-SAT -ongelman muunnokseen. Lähde: *Classic Nintendo Games are (Computationally) Hard*, suomennettu. 47
Kuva 21: Esimerkkipolku pelikentän läpi, vihreästä punaiseen. 49
Kuva 22: Vasen: Super Mario Bros. -pelin alkutila. Oikea: palkintopalikka sisältää sienen, johon osumalla Mario kasvaa isoksi. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard* 51
Kuva 23: Super Mario Bros. -pelin lopputila. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard* 51
Kuva 24: Super Mario Bros. -pelin muuttujaristeys. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard* 52
Kuva 25: Super Mario Bros. -pelin klausuuli. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard* 53
Kuva 26: Super Mario Bros. -pelin risteyskohta. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard* 53
Kuva 27: Wikipedian mielipide NP-täydellisyydestä. Kuvakaappaus otettu 19.4.2022 59

Taulukot

Taulukko 1: Eri aikaluokkien vertailutaulukko. Lähde: Garey ja Johnson 1979, 7. 13
Taulukko 2: Parannetun konetehon vaikutus eri aikaluokkiin kuuluvien algoritmien suoritustehokkuuteen. Lähde: Garey ja Johnson 1979, 8. 14

Sisältö

TIIVISTELMÄT 1

TERMILUETTELO 3

KUVAT 65

KAAVAT 7

1 JOHDANTO 9

2 ONGELMIEN VAIKEUSLUOKAT 11
2.1 P-vaikeusluokka 15
2.2 NP-vaikeusluokka 16
2.3 Co-NP 16
2.4 NP-täydellisyys 18
2.4.1 SAT ja 3-SAT 21
2.4.2 3-Partition 23
2.5 Muut vaikeusluokat 23
2.6 P = NP 25

3 NP-TÄYDELLISYYS PELEISSÄ 28
3.1 Tetris 31
3.1.1 Säännöt 32
3.1.2 Alkuasetelma 33
3.1.3 Todistus 37
3.1.3.1 Kyllä-instanssi 38
3.1.3.2 Ei-instanssi 40
3.1.4 Todistuksen laajennos 44
3.2 Super Mario Bros. 45
3.2.1 Pelin säännöt 46
3.2.2 Tasohyppelypelien runko 47
3.2.3 Super Mario Bros. muunnos rungon avulla 50
3.3 Sudoku 55

4 POHDINTA 57

5 YHTEENVETO 62

LÄHTEET 65
Kuvien lähteet: 68

1 Johdanto

Tämän työn tarkoitus on käsitellä, mikä on NP-täydellisyys ja miten se esiintyy peleissä. Kyseessä on teoreettinen työ ja osa käytössä olevasta materiaalista on kirjallisuuden lisäksi itse käsiteltävänä olevat pelit ja näiden mekaniikkojen lähempi tutkiminen. Muita työn puitteissa käsiteltyjä aiheita ovat ongelmien vaikeusluokat yleisesti ja yhtenä käsiteltävistä lähestymiskohdista on opiskelijan näkökulma aiheen opiskeluun.

Teoriaosuuden materiaali on valittu sen mukaan, miten hyvin se käsittelee NP-täydellisyyttä tai laskennallisten vaikeusluokkien välisiä yhteyksiä. Tämä rajaus tehdään siksi, että saadaan työn aihe esitettyä mahdollisimman tiiviisti ja selkeästi ja yritetään muotoilla helpommin ymmärrettävään muotoon. Laskennallisuuden laajemmista aiheista on jo olemassa runsas määrä teoreettista kirjallisuutta, eikä sen toistamista tämän työn puitteissa nähdä tarpeelliseksi. Työssä esiintyy kuitenkin tarpeellinen joukko NP-täydellisyyden ulkopuolelle jääviä vaikeusluokkia, joten kirjallisuuden rajaus pelkästään NP-täydellisyyteen ei ollut mielekäs valinta.

Toinen käytettävälle kirjallisuudelle valittu rajaus oli ehto siitä, että lähde on vapaasti saavutettavissa opiskelijalle. Näin ollen, mikäli lähdettä ei pysty saamaan käsiinsä kirjastopalvelujen tai internetin kautta ilmaiseksi, se jää tämän työn puitteissa käsittelyn ulkopuolelle. Tämä rajaus on perusteltu budjettisyillä sekä sillä, että vapaasti saavutettavissa olevaa materiaalia on riittävästi työn tarkoitusta varten. Lähdeluettelossa on kuitenkin muutama maksullinen lähde, joiden oleellinen tietolainaus on ollut internetin kautta ilmaiseksi luettavissa, esimerkiksi esikatselun muodossa.

Esimerkkipelit on valittu edustamaan mahdollisimman erilaisia tutkimuskohteita NP-täydellisyyden esiintymisen kannalta, jotta lukija saa mahdollisimman monipuolisen esimerkkikirjon NP-täydellisyyden esiintymisestä peleissä. Valitut artikkelit kattavat pelitutkimusten osalta sekä NP-täydellisyyden todistuksen, NP-kovuuden todistuksen sekä muunnosten hyödyntämisen NP-kovien ongelmien ratkomiseksi. Pelit on myös valittu kirjoittajan henkilökohtaisen mieltymyksen mukaan, sillä kohdepelin logiikan tunteminen on koettu hyödyksi todistuksia lukiessa. Kaikki esimerkkipelit ovat myös mahdollisimman tunnettuja edellä mainitusta syystä.

Työn päätoiminen tarkoitus on esittää erilaisia lähestymistapoja NP-täydellisyyden tutkimiseen peleissä ja tarjota riittävä tietopohja siihen, että samoja periaatteita noudattaen NP-täydellisyyttä pystyy etsimään myös esimerkkipelien ulkopuolelle jäävistä peleistä, lukijan omien mieltymysten mukaan. Toinen tavoite on tarjota opiskelijan näkökulmasta helpommin lähestyttävä esittely NP-täydellisyyden ja ongelmien vaikeusluokkien käsittelyn pariin, sillä aihe kuuluu opintosuunnan kurssitarjontaan ja se on monen opiskelijan toimesta todettu hankalaksi.

Luvussa 2 esitellään laskennallisten ongelmien vaikeusluokkia, muunnoksia sekä keskitytään tarkemmin NP-täydellisyyden rooliin ja määritelmään. Tämä luku selittää muun muassa sellaiset termit kuin muunnos, P, NP ja NP-täydellisyys, niiden väliset yhteydet sekä esittelee P = NP -ongelman ja sen merkittävyyden. Myös muita vaikeusluokkia mainitaan. Luvun tarkoitus on vastata työn ensimmäiseen tutkimuskysymykseen, eli ”Mikä on NP-täydellisyys?”

Luvussa 3 tarkastelemme joukkoa esimerkkipelejä ja niistä tehtyjä vaikeusluokka-tutkimuksia. Luvun tarkoitus on havainnollistaa NP-kovuuden löytämisen ja todistamisen erilaisten pelien yhteydessä, sekä antaa lukijalle suuntaa ja työkaluja jatkaa aiheeseen perehtymistä oman mieltymyksen mukaan. Tarkemmassa tarkastelussa ovat esimerkkipeleinä Tetris ja Super Mario Bros., joiden todistuksissa käytettävät NP-täydelliset ongelmat 3-Partition ja 3-SAT esitellään luvussa 2. Luku antaa käsityksen työn toiseen tutkimuskysymykseen: ”Miten NP-täydellisyys esiintyy peleissä?”

Pohdintaosiossa käymme läpi sitä, kuinka yleistä NP-täydellisyys on peleissä, kuinka lähestyttävä se on aiheena ja mitä käytännön merkitystä pelien vaikeusluokilla on. Pohdimme myös tämän työn ohessa heränneitä kysymyksiä NP-täydellisyydestä sekä työn saavutettua hyötyä. Yhteenvetoluvussa niputamme yhteen kaiken työn aikana saamamme tiedon ja esitämme työn kulussa esiin nousseita kysymyksiä, joita voisi tutkia eteenpäin.

2 Ongelmien vaikeusluokat

Kun eteemme tulee ongelma, joka kuuluisi ratkaista, mistä tiedämme, kuinka sitä kuuluisi parhaiten lähestyä? Mikä olisi tehokkain tapa ratkaista tämä ongelma, vai onko ongelma ylipäätään ratkaistavissa? Voiko tietää etukäteen, kannattaako uhrata aikaa tehokkaan ratkaisun löytämiseen, vai onko sellainen ratkaisu ylipäätään mahdollinen? Koska kaikki ongelmat eivät ole yhtä vaikeita, niiden käsittelyn helpottamiseksi ongelmat on päätetty jaotella eri ryhmiin.

Ongelmien luonteesta on totuttu sanomaan, että sellainen ongelma, joka voidaan ratkaista järkevällä työmäärällä, on "*tractable*", eli tehokkaasti ratkaistavissa oleva. Kaikki muut ongelmat, joiden ratkaisu on joko liian monimutkainen tai epätehokas, sekä ne ongelmat, jotka ovat ratkeamattomia jopa teoriassa, ovat "*intractable*", eli ei tehokkaasti ratkaistavissa olevia. (Davis ja Weyuker 1983, Garey ja Johnson 1979)

Laskennalliselta kannalta ongelmat voidaan ryhmitellä eri *vaikeusluokkiin*. Yksi keino jakaa ongelmat eri vaikeusluokkiin on tarkastella niiden ratkaisemiseksi laadittavien algoritmien ajan tai muistin enimmäiskulutusta huonoimman tapauksen kohdalla. Muistin käytön suhteen tehty jako tarkastelee, paljonko muistitilaa ongelman ratkaisualgoritmi voi enintään vaatia pahimman tapauksen kohdalla. Ajan suhteen tehty jako taas tarkastelee, kuinka kauan algoritmilla voi kulua enintään aikaa ratkaista ongelma huonoimman tapauksen kohdalla, ja se esitetään *aikaluokkana*. Toisin sanoen voimme sanoa, että ongelma kuuluu tiettyyn vaikeusluokkaan sen perusteella, kuinka tehokas algoritmi sen ratkaisemiseksi on olemassa. (Garey ja Johnson 1979)

Voidaan yleisesti hyväksyä, että mikäli on olemassa algoritmi, joka ratkaisee ongelman sen pahimmankin variaation kohdalla keskimääräisesti nopeammin kuin toinen, tämä algoritmi on parempi ratkaisu kyseiseen ongelmaan. Käytännössä tämä ei kuitenkaan aina pidä paikkaansa, sillä kaikilla ongelmilla on omat esiintymisympäristönsä ja ne ratkaisevilla algoritmeilla omat käyttötarkoituksensa. Joillakin ongelmilla huonoimman tapauksen esiintyminen on niin marginaalinen tilanne, että käytännön suorituksessa vaativampikin algoritmi saattaa toimia paljon tehokkaammin niiden ratkaisemiseksi, kuin resursseiltaan vaatimattomampi vastine. Tässä tapauksessa mukaan laskuihin tulevat myös sellaiset asiat

kuin syötteen suurin sallittu koko, muistin käyttö ja tilanteen hyväksymät suoritusajat, mutta keskittykäämme käsittelemään ensin pelkkää aikaluokkajakoa.

Polynomiajassa toimiva algoritmi määritellään sellaiseksi algoritmiksi, jonka aikavaatimus on luokkaa $O(\mathrm{p}(n))$ jollekin polynomifunktiolle p, jossa n merkitsee syötteen kokoa. (Garey ja Johnson 1979, 6) Tämä kuvastaa sitä maksimiaikaa, joka algoritmilla kuluu suoriutua tehtävästään, eli ongelman ratkaisuun ei huonoimmankaan tapauksen kohdalla saa kulua enempää aikaa, mutta aikavaatimusta nopeammin saa suoriutua. Syötteen koolla tarkoitamme ongelman *instanssin*, eli yksittäisen esiintymän koodatun muodon kokoa. (Garey ja Johnson 1979)

Kaikkia niitä algoritmeja, jotka eivät täytä näitä kriteereitä, kutsumme *eksponentiaalisen* ajan algoritmeiksi. Tämän määritelmän mukaan eksponentiaaliseen ryhmään kuuluu myös joukko ei-polynomiajassa toimivia algoritmeja, joita ei yleensä myöskään lasketa ekspontiaalisiksi algoritmeiksi, kuten $O(n^{log_n})$. Nämä kyseiset algoritmit ovat sellaisia, jotka voivat käyttää huonoimman tapauksensa kohdalla enemmän aikaa, kuin polynomiaikaiset algoritmit, mutta vähemmän, kuin aidosti eksponentiaaliset. (Garey ja Johnson 1979)

Polynomiaikaisten ja eksponentiaaliaikaisten algoritmien ero kasvaa merkitseväksi varsinkin silloin, kun käsiteltävän ongelman koko kasvaa. Ongelman syötteen pituus voi helposti kasvaa sellaisiin mittoihin, että polynominen aika ei yksinkertaisesti riitä tehokkaaseen ratkaisuun. Taulukosta 1 voimme nähdä esimerkkejä siitä, miten syötteen pituus eli ongelman koko vaikuttaa eri aikaluokkiin kuuluvien tyypillisten esimerkki-algoritmien suoritusnopeuteen mikrosekunneissa mitattuna. Voidaan huomata, että syötteen pituuden kasvaessa ero polynomiaikaisien ja eksponentiaalisten algoritmien välillä on valtava. (Garey ja Johnson 1979)

Taulukon ylärivissä nähdään syötteen pituus (*n* arvo) ja pystypalkissa algoritmin aikaluokka.

Syötteen pituus→ Aikaluokka ↓	10	20	30	40	50	60
n	0.00001 sekuntia	0.00002 sekuntia	0.00003 sekuntia	0.00004 sekuntia	0.00005 sekuntia	0.00006 sekuntia
n^2	0.0001 sekuntia	0.0004 sekuntia	0.0009 sekuntia	0.0016 sekuntia	0.0025 sekuntia	0.0036 sekuntia
n^3	0.001 sekuntia	0.008 sekuntia	0.027 sekuntia	0.064 sekuntia	0.125 sekuntia	0.216 sekuntia
n^5	0.1 sekuntia	3.2 sekuntia	24.3 sekuntia	1.7 minuuttia	5.2 minuuttia	13.0 minuuttia
2^n	0.001 sekuntia	1.0 sekuntia	17.9 minuuttia	12.7 päivää	35.7 vuotta	366 vuosisataa
3^n	0.059 sekuntia	58 minuuttia	6.5 vuotta	3855 vuosisataa	$2\ x\ 10^8$ vuosisataa	$1.3\ x\ 10^{13}$ vuosisataa

Taulukko 1: Eri aikaluokkien vertailutaulukko. Lähde: Garey ja Johnson 1979, 7.

Koska algoritmien suoritus tapahtuu pääosin tietokoneiden avulla, on tärkeää huomata, miten tietokoneiden tehojen kehittyminen vaikuttaa ongelmien laskettavuuteen. Voisi luulla, että näillä eri aikaluokkiin kuuluvien algoritmien suoritusnopeuksilla ei enää ole niin paljon eroa, kunhan ne suoritetaan vähän nopeammalla koneella? Taulukosta 2 voimme kuitenkin nähdä, että vaikka tietokoneet olisivat tuhansia kertoja nykyisiä nopeampia suoritukseltaan, polynomifunktioiden kohdalla samassa ajassa suurin ratkaistava ongelma muuttuu moninkertaiseksi, siinä missä eksponentiaalifunktioiden kohdalla se ei edes tuplaannu. (Garey ja Johnson 1979)

Ongelmainstanssin suurin mahdollinen koko (N), jonka algoritmi voi ratkaista 1 tunnin aikana

Algoritmin aikaluokka	Nykytietokone	100 kertaa nopeampi tietokone	1000 kertaa nopeampi tietokone
n	N_1	100 N_1	1000 N_1
n^2	N_2	10 N_2	31.6 N_2
n^3	N_3	4.64 N_3	10 N_3
n^5	N_4	2.5 N_4	3.98 N_4
2^n	N_5	N_5 + 6.64	N_5 + 9.97
3^n	N_6	N_6 + 4.19	N_6 + 6.29

Taulukko 2: Parannetun konetehon vaikutus eri aikaluokkiin kuuluvien algoritmien suoritustehokkuuteen. Lähde: Garey ja Johnson 1979, 8.

Tästä johtuen polynomiajassa toimivat algoritmit ovat tyypillisesti paljon halutumpia, vaikka käytännössä pienillä syötteillä eksponentiaaliset algoritmit voivatkin suoriutua nopeammin. Esimerkiksi Simplex -niminen algoritmi luokitellaan eksponentiaaliseksi, mutta se on silti todistettu toimivaksi melkein aina erittäin nopeasti. (Spielman, Teng 2008) Vallassa on kuitenkin yleinen mielipide siitä, että ongelmaa ei voi sanoa ”hyvin ratkaistuksi”, ellei siihen ole löydetty nimenomaan polynomiajassa toimivaa algoritmia. (Garey ja Johnson 1979)

Ennen, kuin voimme seuraavissa alaluvuissa perehtyä tarkemmin vaikeusluokkien jakoon, on aika määritellä tarkemmin, millaisia ongelmia aiomme käsitellä. Ongelmia on hyvin monimuotoisia ja jokainen vaatii ratkaisuunsa omat menetelmänsä. Tämän työn puitteissa rajaamme ongelmien joukon koskemaan nimenomaan *päätösongelmia*, joiden tunnuspiirteenä on, että ne vastaavat yksinkertaiseen kysymykseen “kyllä/ei”. (Garey ja Johnson 1979) Tällä rajauksella pyritään yksinkertaistamaan tekstiä ja sen väitteitä sekä helpottamaan NP-vaikeusluokan käsittelyä. Joitakin työssä esiintyviä teorioita voi laajentaa

koskemaan muitakin ongelmaluokkia joko sellaisenaan tai kohtuullisella lisätyöllä, mutta jatkamme olettamalla, että tekstissä esiintyvät ongelmat ovat nimenomaan päätösongelmia.

Toinen tarkennus koskee syötteitä. Jotta tuloksemme ovat mielekkäitä, meidän on sallittava rajaton syötejoukko, jolloin käsiteltävät ongelmat voivat olla miten suuria tahansa. Tämä tehdään siksi, että mikäli syötejoukon koolla on kiinteä yläraja, kaikki nämä tapaukset ovat ratkaistavissa vain luomalla järjettömän suuri joukko vastauksia, joihin annettua syötettä verrataan ja näin saadaan toteutettua polynomiaikainen algoritmi, jota ei voi kumminkaan nimittää tehokkaaksi ratkaisuksi ongelmaan. Rajaton syötejoukko heijastuu myös rajattomaan syötteen pituuteen, jolloin yllä osoitettujen tulosten varjolla ero polynomi- aikaisten ja eksponentiaalisten algoritmien välillä kasvaa. Meidän tapauksessamme rajaton ei kuitenkaan tarkoita ääretöntä, vaan mielivaltaisen suurta.

Pelien kohdalla tämä rajattomuus tarkoittaa sitä, että pelin säännöt ja mekaniikat pysyvät ennallaan, mutta pelin ”koko” muuttuu. Koko voi tässä yhteydessä tarkoittaa esimerkiksi pelilautaa, pelikenttää tai muuta avaruutta, jonka puitteissa peli toimii. Meidän on siis sallittava peli-instanssin olevan kuinka iso tahansa, että voimme tutkia sen laskettavuutta.

2.1 P-vaikeusluokka

Vaikeusluokka P on sellaisten ongelmien joukko, joille on olemassa polynomiajassa toimiva ratkaisualgoritmi. Siksi termi on saanutkin nimensä “Polynomial time algorithm”:in perusteella. (Garey ja Johnson 1979) Joissain lähteissä tarkennetaan, että P-luokkaan kuuluakseen ongelma on ratkaistava polynomiajassa deterministisen Turing-koneen avulla, mutta tämän gradun puitteissa emme sen tarkemmin keskity Turing-koneiden esittelyyn. Sekä deterministinen että epädeterministinen Turing-kone nousee kyllä aina välillä esiin terminä määritelmien yhteydessä, mutta näiden koneiden toimintatapoihin tarkempi perehtyminen ei ole välttämätöntä väitteiden ymmärryksen kannalta.

Esimerkkinä P-luokkaan kuuluvasta ongelmasta toimii palindromin tarkistus ja päätösongelman muodossa kysymys olisi muotoa ”Onko tämä merkkijono palindromi?”. Ongelma ratkaistaan kääntämällä syöte ympäri ja vertaamalla itseensä. Mikäli se pysyy

samana, tulos on ”kyllä”, mikäli ei, vastaus on ”ei”. Saippuakauppias on yksi tällainen toimiva palindromi. Kuten voi huomata, tällaisen ongelman ratkaisemiseksi ei hyvinkään pitkän syötteen kohdalla kulu eksponentiaalisesti aikaa, sillä välivaiheita ei yksinkertaisesti tarvita enempää. Voimme siis sanoa polynomiajan riittävän tämän ongelman ratkaisemiseksi, sillä sellainen ratkaisu on löytynyt.

2.2 NP-vaikeusluokka

Lyhenne NP tulee sanoista “Non-deterministic polynomial time”, mikä viittaa epädeterministiseen Turingin koneeseen, eli luokan yksi määritelmä on johdettu siitä, että tämän vaikeusluokan päätösongelmat ovat ratkaistavissa polynomiajassa epä-deterministisellä Turing-koneella. (Garey ja Johnson 1979) Helpommin sanottuna NP-luokkaan kuuluvat ne laskettavissa olevat päätösongelmat, joiden ”kyllä”-ratkaisun tarkistus onnistuu polynomiajassa. Se, miten tehokkaasti itse ratkaisu on saatu aikaan, on tälle luokalle toissijaista.

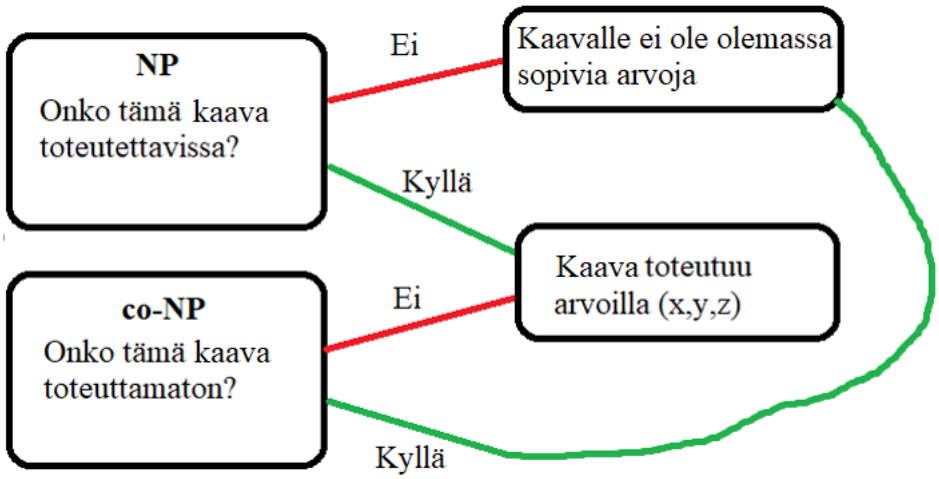
Tämän luvun loppupuolella esiteltävä P=NP -kysymys koskeekin pitkälti sitä, että voiko kaikki nämä helposti tarkistettavissa olevat ongelmat myös ratkaista helposti, vai ei. Näin ollen saamme käsityksen siitä, että kaikki P-luokan ongelmat ovat myös NP-joukon osajoukko, mutta ovatko nämä joukot täsmälleen samankokoisia vai ei, on kiistanalaista. (Goldreich 2010)

Toinen tärkeä piirre NP-luokassa on tarkistuksen epäsymmetrisyys. Jokaiselle ”kyllä”-vastaukselle on olemassa polynomiaikainen todiste, mutta ”ei”:n kohdalla se ei välttämättä ole niin. (Kleinberg ja Tardos 2006, 496) Tarkemmin tämä epäsymmetria esitetään seuraavassa alaluvussa, mutta tällä hetkellä tärkeää on huomata, että tämä epäsymmetria on tärkeä asia, koska se on monien tosimaailman tehtävien olennainen piirre. (Valmari 2020)

2.3 Co-NP

Co-NP on vaikeusluokka, joka koostuu NP-vaikeusluokkaan kuuluvien ongelmien *komplementeista*. Ongelman komplementti saadaan aikaan, kun muutetaan saman ongelman

kyllä-vastaus ei-vastaukseksi. Esimerkiksi ongelman ”Onko tämä propositiologiikan kaava toteutettavissa?” komplementtiongelma olisi muotoa ”Onko tämä propositiologiikan kaava toteuttamaton?”. Voimme huomata, että vastaus ”kyllä” saadaan ensimmäisen ongelman kohdalla löytämällä totuusarvot, jotka muuttavat kaavan todeksi. Jälkimmäisessä esimerkissä kaavan toteuttavien sopivien totuusarvojen löytyminen taas tuottaa vastauksen ”ei”. Kummassakin tapauksessa ei voida tietää, onnistuuko toisen vastauksen ”Kaavalle ei ole olemassa sopivia arvoja” saaminen yhtä helposti, vai ei. (Kleinberg ja Tardos 2006, 496)



Kuva 1: NP ja co-NP visualisointi

Kuten kuvasta 1 voimme nähdä, käytännössä nämä kaksi ongelmaa ja niiden ratkaisumetodi toimivat aivan samalla tavalla, mutta kysymyksen asettelu vaikuttaa siihen, haetaanko ”kyllä” vai ”ei” -vastauksen tarkistusta. NP-luokan ”kyllä”-vastaus sekä co-NP luokan ”ei”-vastauksen tarkistus on toteutettavissa polynomiajassa heti, kun löydämme kaavan toteuttavat arvot. Vastausten alternatiivin kohdalla taas emme voi tietää, kuinka nopeasti tämä tulos voidaan saavuttaa. Mikäli kaavasta näkyy heti, että se on mahdotonta toteuttaa, vastaus saadaan nopeasti, mutta mikäli esimerkiksi annettu kaava on merkittävän pitkä, algoritmilla voi kestää hyvinkin pitkään käydä kaikki vaihtoehdot läpi, ennen kuin voi julistaa tuloksen pitäväksi. NP-luokkaan kuuluukin vaatimuksena ”kyllä”-vastausten polynomiaikainen tarkistus ja ”ei”-vastausten polynomiaikaisuus kuuluu co-NP -luokkaan.

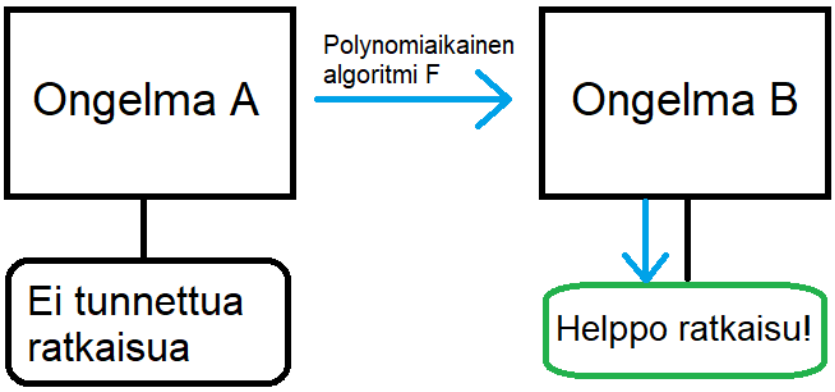
Se, onko NP = co-NP on toinen toistaiseksi ratkeamaton kysymys, kuten P=NP. (Kleinberg ja Tardos 2006, 496)

2.4 NP-täydellisyys

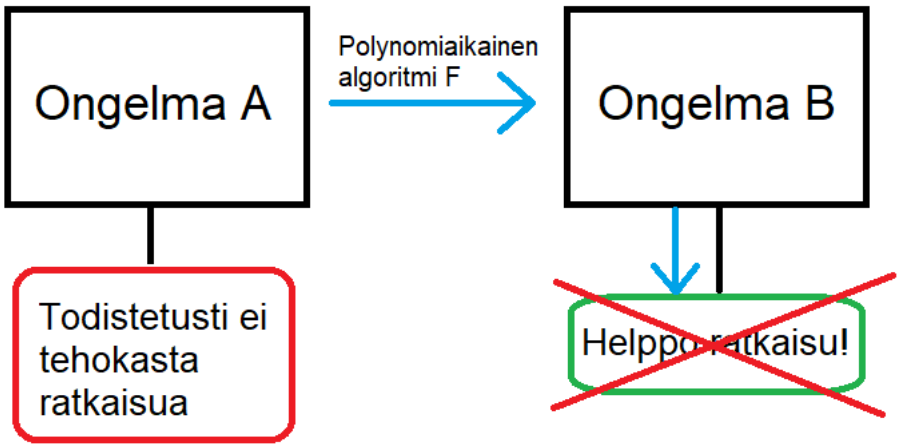
NP-täydellisyyden käsitteen pohja on ensimmäisen kerran laadittu Stephen Cookin vuoden 1971 artikkelissa "The Complexity of Theorem - Proving Procedures" (Cook 1971). Cook korosti artikkelissaan "polynomial time reductibilityn" merkitystä, mikä tarkoittaa sitä, että mikäli ongelma A voidaan *muuntaa* polynomiajassa ongelmaksi B ja ongelma B voidaan ratkaista polynomiajassa, ongelma A voidaan myös ratkaista polynomiajassa käyttäen ongelman B ratkaisualgoritmia. Käytännössä tämä tapa muuntaa ongelma toiseksi luo meille "yhtä vaikeiden" ongelmien joukon, sillä kaikki nämä onnistuneesti toisikseen muunnetut ongelmat on mahdollista ratkaista samoilla metodeilla. (Garey ja Johnson 1979)

NP-kovien ongelmien joukko on niiden ongelmien joukko, joihin jokainen NP-joukon ongelma voidaan muuntaa polynomiajassa. NP-kovuus ei siis automaattisesti tarkoita, että ongelma kuuluisi NP-luokkaan, mutta NP-luokan ongelmista ne ovat luokan vaikeimpia. Ne ongelmat, jotka ovat NP-kovia ja kuuluvat NP-luokkaan, kutsutaan *NP-täydellisiksi***.** Ongelma on siis NP-täydellinen **jos ja vain jos se kuuluu NP-luokkaan ja on NP-kova**. (Valmari 2020, Garey ja Johnson 1979)

Mikäli meillä on polynomiajassa toimiva algoritmi *F*, joka voi muuntaa ongelman A ongelmaksi B, ja tiedämme polynomiaikaisen ratkaisukeinon ongelmalle B, voimme ratkaista myös ongelman A polynomiajassa, muuntamalla se ensin B:ksi ja ajaen sitten B:n ratkaisualgoritmin (Kuva 2). Toisin päin sanottuna, mikäli tiedämme, että ongelmalle A ei ole olemassa polynomiaikaista ratkaisualgoritmia ja saamme ongelman A polynomiajassa muunnettua ongelmaksi B, myös B:lle ei voi olla olemassa polynomiaikaista ratkaisualgoritmia (Kuva 3). Koska lähtökohtana tällöin on, että helppoa ratkaisua ongelmalle A ei voi olla olemassa, ongelman B helppo ratkaisu on todennäköisesti virheellinen, tai muunnos on tehty väärin.

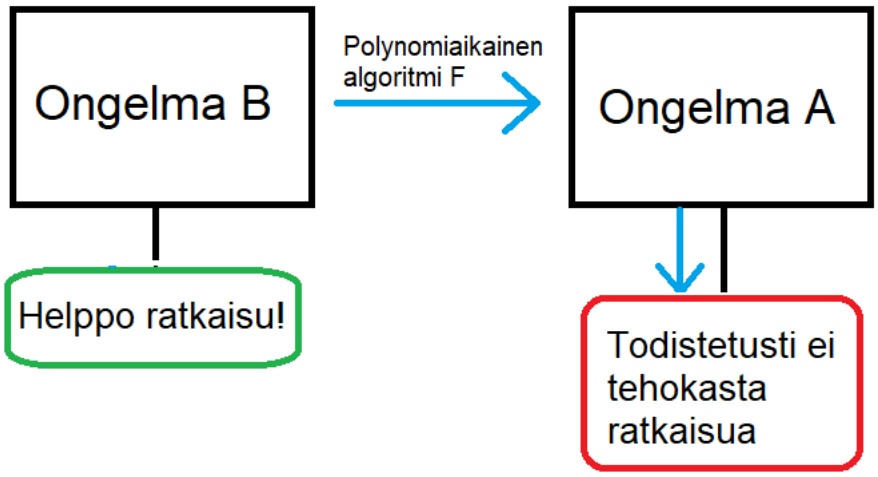


Kuva 2: Muunnos A → B, jotta ratkaisematon A voi käyttää B:n ratkaisua.



Kuva 3: Muunnos A→B, kun A on todistetusti ilman tehokasta ratkaisua.

Tästä näemmekin sen, että muunnoksen suunnalla on väliä sen kannalta, mikä meidän tarkoituksemme on. Vaikka kaikki NP-kovat ongelmat ovatkin määritelmän mukaan muunnettavissa polynomiajassa toisikseen, se johtuu juuri siitä, että ne ovat kaikki ”vähintään yhtä vaikeita”. Toisin sanottuna yhdellekään niistä ei olla löydetty polynomiaikaista ratkaisualgoritmia, eli tilanne vastaa kuvaa 3. Toki mikäli niin haluamme, voimme muuntaa polynomiajassa ratkeavan ongelman B NP-kovaksi ongelmaksi A, kuten kuvassa 4, sillä vaikka B:llä olisikin polynomiaikainen ratkaisu olemassa, se voidaan ratkaista myös vaikeampaa menetelmää käyttäen. Tämä piirre voidaan nähdä esimerkiksi siinä, että kaikki NP-luokkaan kuuluvat ongelmat, P-luokka mukaan lukien, voidaan muuntaa polynomiajassa NP-koviksi ongelmiksi, mutta ei toisinpäin.



Kuva 4: Muunnos B→A, ratkaistaan B käyttäen A:n ratkaisua.

Yksi NP-kovien ongelmien mielenkiintoisista tekijöistä on se, että koska ne voidaan aina muuntaa polynomiajassa toisikseen, niin mikäli yhdellekin niistä löytyy polynomiajassa toimiva ratkaisualgoritmi, se ratkaisee samalla kaikki NP-kovat ongelmat. Sama pätee myös NP-täydellisille ongelmille. Vuosien saatossa tutkijat ovat täydentäneet tätä NP-täydellisten ongelmien listaa löytämillään esimerkeillä, mutta yhdellekään niistä ei olla vieläkään löydetty toimivaa polynomiaikaista ratkaisualgoritmia.

Cookin esimerkin jälkeen Richard Karp julkaisi kokoelman tuloksia NP-täydellisiksi kokemistaan ongelmista, joihin kuului muun muassa 3-Satisfiability Problem (3-SAT), 3-Dimensional Matching, Vertex Cover, Clique, Hamilton Circuit ja Partition Problem. (Karp 1972, Garey ja Johnson 1979)

Nämä ongelmat ovat muodoltaan mahdollisimman pelkistettyjä matemaattisia ongelmia, ja niiden tarkoitus on toimia apuna uusien NP-täydellisten ongelmien löytämisessä. Mikäli siis haluaa todistaa, että jokin ongelma on NP-täydellinen, se on tehtävissä tuomalla esiin polynomiajassa toimiva tapa muuntaa yksi näistä tunnetuista NP-täydellisistä ongelmista tämän uuden ongelman muotoon ja osoittamalla, että ongelma kuuluu NP-luokkaan. Tilanne vastaa tällöin taas kuvaa 3. Näin voimme todistaa, että uusi ongelma on vähintään yhtä vaikea, kuin tunnettu NP-täydellinen ongelma, eikä sille siis ole tiedossa olevaa helppoa ratkaisua. Karpin ensimmäisenkin muutaman ongelman joukko on keskenään hyvin erilainen ja kattaa alleen sekä graafisia, että numeraalisia ongelmia, ja lista on niiltä ajoilta

vain paisunut huimaa vauhtia. Tästä saamme johtopäätöksen sille, että NP-täydellisyyttä esiintyy hyvin laajalti ympäristössämme, vaikka sitä ei välttämättä huomaisikaan. (Karp 1972, Garey ja Johnson 1979)

Luvussa 3 tulemme huomaamaan, että on olemassa runsas määrä artikkeleita, jotka käsittelevät pelejä, NP-täydellisyyttä ja muunnoksia. Näihin artikkeleihin törmätessä varsinkin aloittelijan on tärkeää kiinnittää huomiota siihen, kumpaan suuntaan muunnosta tehdään. Mikäli peli-instanssi muunnetaan tunnetuksi NP-täydelliseksi ongelmaksi, todistus pyrkii vain siihen, että tämän pelin ratkaisemiseen voidaan käyttää NP-täydellisen ongelman ratkaisualgoritmeja, kuten kuvassa 4. Se siis osoittaa vain sen, että peli kuuluu NP-luokkaan, mutta ei kerro sitä, onko se NP-kova vai ei. Tällöin todistus NP-täydellisyydestäkään ei päde. Mikäli taas muunnos tehdään NP-täydellisestä ongelmasta peli-instanssiksi, sillä todistetaan, että kyseiselle peli-instanssille ei voi olla olemassa helppoa ratkaisua, koska lähtökohtana on, että NP-täydellisellä ongelmalla ei sellaista ole, mikäli $P \neq NP$. Tämä vuorostaan vastaa kuvan 3 tilannetta ja todistaa pelin NP-kovuuden. Mikäli lisäksi on osoitettu kuuluminen NP-luokkaan, todistus osoittaa NP-täydellisyyttä.

Kysymykseksi saattaa herätä se, miksi kukaan haluaisi muuntaa kuvan 4 kaltaisesti pelejä sellaiseen muotoon, josta tiedetään, ettei tehokasta ratkaisua voi olla olemassa? Entä, jos pelille sellainen olisikin, ja päädymme silti käyttämään tuota NP-täydellisen ongelman vaikeampaa ratkaisua? Vastaus on siinä, että NP-täydellisiä ongelmia on niin moninainen joukko, että monelle niistä on olemassa omat, eri tehokkuusasteen ratkaisualgoritmit: onhan joitakin niistä tutkittu jo vuosikymmeniä. Joskus on helpompaa ratkaista ongelma olemassa olevia algoritmeja käyttäen, kuin kehitellä itse, mahdollisesti turhaan, tehokkaampaa vaihtoehtoa. Esimerkiksi Sudoku on kerännyt osakseen varsin suuren määrän muunnoksia erilaisiksi NP-täydellisiksi ongelmiksi, kun siihen on haluttu kehittää erilaisia ratkaisuohjelmia.

2.4.1 SAT ja 3-SAT

Satisfiability Problem, eli SAT-ongelma oli ensimmäinen Cookin esittämä NP-täydellinen ongelmia, jota Karp myöhemmin laajensi 3-SAT -variaatiolla 21 NP-täydellisen ongelman

listauksessaan. (Cook 1971, Karp 1972) Yksinkertaisuutensa takia se on varsin suosittu NP-täydellisyyttä koskevien todistusten piirissä: tulemme kohtaamaan sen myös luvussa 3 esitettyjen esimerkkipelien yhteydessä, joten käykäämme perusteet tässä.

SAT koostuu joukosta *klausuuleja*, eli äärellisistä, totuusarvomuuttujista ja niiden negaatioista modostuvista joukoista. 3-SAT on tämän ongelman variaatio sillä erotuksella, että jokaisessa klausuulissa on aina tasan 3 *literaalia*, eli muuttujan arvoa. Ongelma on päätösongelma, jossa syötteenä annetaan klausuulirimpsu ja tarkistetaan, onko olemassa sellaiset muuttujien arvot, jotka toteuttavat sen. (Martiny 2012)

Perinteisessä SAT-ongelmassa ei siis rajoituta kolmen literaalin joukkoihin, vaan klausuulit voivat olla myös yksittäisiä literaaleja, literaalipareja tai muita variaatioita. Esimerkkiongelma voi näyttää esimerkiksi tältä:

$$(\mathrm{A} \vee \mathrm{B}) \wedge (\mathrm{A} \vee \mathrm{D}) \wedge (\neg \mathrm{B} \vee \mathrm{C} \vee \mathrm{E}) \wedge (\mathrm{B} \vee \mathrm{E} \vee \mathrm{A} \vee \neg \mathrm{D}) \wedge \neg \mathrm{B}$$

Yllä esitetylle SAT-ongelmalle sopiva ratkaisu olisi A = true, B = false, C = true, D = false, E = true. 3-SAT -muotoinen ongelma voisi vastaavasti näyttää tältä:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee x_4)$$

Voimme huomata, että erilaisia muuttujia voi olla enemmänkin, kuin kolme. Yllä mainitussa esimerkissä niitä on neljä, mutta tärkeä rajaus on siinä, että jokaisessa klausuulissa esiintyy vain kolme literaalia, eli tasan kolme muuttujan arvoa on käsittelyssä ”kerrallaan”. (Martiny 2012) SAT-ongelmille yleinen esitystapa on CNF (Conjuctive Normal Form), joka koostuu *k* määrästä klausuuleja, jotka ovat kaikki yhteydessä keskenään loogisella “ja”:lla. Joissain lähteissä tätä esitystapaa noudattavaa ongelmaa nimitetään CNFSAT:ksi, määrittäen SAT-ongelmaksi paljon vapaamman esitystavan. Tässä työssä käytämme termiä SAT, kun puhumme CNFSAT-muotoisesta Satisfiability ongelmasta.

Koska kumpikin ongelma on todistetusti NP-täydellinen Cookin, Levinin ja Karpin töillä, se tarkoittaa sitä, että SAT-muotoinen klausuulirimpsu voidaan polynomiajassa muuntaa 3-SAT -rimpsuksi ja toiseen suuntaan muunnettaessa ei tarvitse tehdä mitään, sillä 3-SAT on

SAT:in osajoukko itsessään. Luvussa 3 tulemme näkemään, miten 3-SAT -ongelma muunnetaan tasohyppelypelin instanssiksi, näin liittäen sen NP-kovien ongelmien piiriin.

2.4.2 3-Partition

3-partition on NP-täydellinen ongelma, jossa on tarkoitus tarkistaa, voiko joukon positiivisia kokonaislukuja jakaa kolmen luvun osajoukoiksi niin, että näiden osajoukkojen summat ovat kaikilla samat. (Breukelaar ym., 2003) Toisin kuin 3-partitionissa, yleisessä partition-ongelmassa on tarkoituksena jakaa positiivisista kokonaisluvuista koostuvan joukon tasan kahdeksi osajoukoiksi niin, että niiden summan arvo on sama.

Tarkemmin määriteltynä 3-Partition ongelma voidaan esittää näin:

Oletetaan, että meillä on positiivisten kokonaislukujen muodostama lukujono *A*, joka koostuu arvoista $a_1, \dots, a_{3s}$ ja positiivinen kokonaislukuarvo *T* niin, että $T/4 < a_i < T/2$ kaikilla $1 \leq i \leq 3s$ ja $\sum_{i=1}^{3s} a_i = sT$.

Kysymys: voiko *A* jakaa *s* määräksi erillisiä osajoukkoja $B_1, \dots, B_s$ niin, että $\sum_{a_i \in B_j} a_i = T$ kaikilla $1 \leq j \leq s$? Mikäli jako onnistuu, saamme kyllä-instanssin ja muussa tapauksessa ei:instanssin.

Huomataan, että koska $T/4 < a_i < T/2$ kaikilla $1 \leq i \leq 3s$, ”kyllä”-instansissa $|B_j| = 3$ kaikilla $1 \leq j \leq s$.

Toisin sanoen tehtävämme on jakaa joukko *A* määräksi *s* kolmen luvun osajoukkoja niin, että jokaisen osajoukon lukujen summa on tasan *T*. Esimerkiksi jos *A* = {4, 5, 5, 5, 5, 6}, voimme jakaa sen osajoukoiksi {4, 5, 6}, {5, 5, 5} jolloin jokaisen osajoukon summa on *T* = 15.

2.5 Muut vaikeusluokat

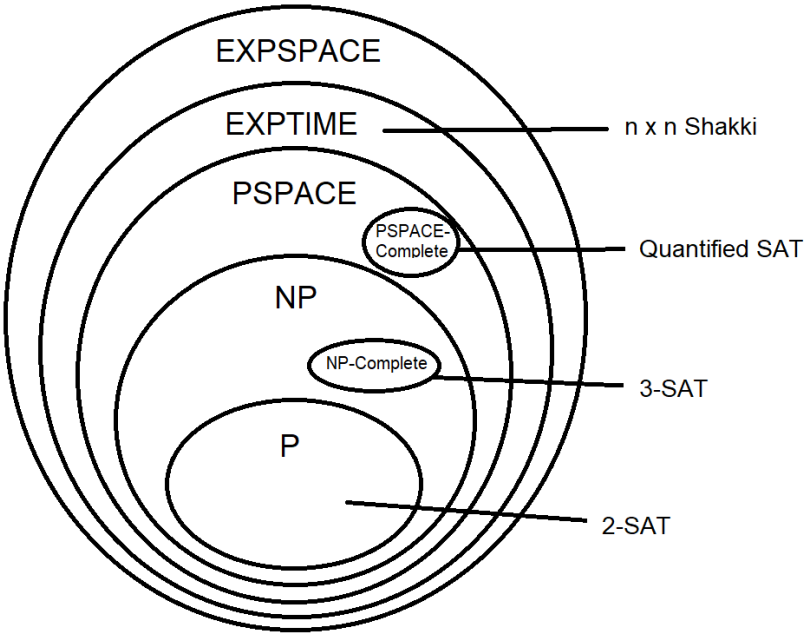
NP-koviksi ja NP-täydellisiksi luokitellut ongelmat eivät kuitenkaan välttämättä ole laskennalliselta kannalta ongelmista aina niitä haastavimpia, vaikka puhummekin niistä vaikeina ja ei tehokkaasti ratkaistavissa olevina. Kuten aiemmin määrittelimme, NP-luokan

ongelmille on tunnusomaista se, että niiden ”kyllä”-vastausratkaisu on mahdollista tarkistaa polynomiajassa. On kuitenkin olemassa myös joukko ongelmia, joille tämä ehto ei päde, eli vaikka tietäisimme ratkaisun, meidän on hyvin vaikeaa päätellä helposti, onko se oikea, vai ei.

Yhtä näistä ongelmien joukoista nimitetään **EXPTIME**-joukoksi, joka tulee sanoista “exponential time problems”. Joukon tunnuspiirteenä on se, että nämä päätösongelmat ovat ratkaistavissa deterministisellä Turing-koneella eksponentiaalista aikaa käyttäen, näin ollen sisällyttäen NP-luokan lisäksi itseensä myös niitä päätösongelmia, joiden tarkistus ei onnistu polynomiajassa. Voimme kuitenkin huomata kuvasta 5, että tämä luokka sisältää myös kaikki määritelmäänsä sopivat helpommatkin ongelmat, eli esimerkiksi koko P-luokan, joten kaikki EXPTIME-luokkaan kuuluvat ongelmat eivät automaattisesti ole vaikeita. EXPTIME-vaikeusluokkaan kuuluvat sellaiset ongelmat kuin $n \times n$ laudalla pelattavat shakki, tammi ja Go. (Fraenkel ja Lichtenstein 1981, Robston 1981)

PSPACE on niiden ongelmien vaikeusluokka, jotka ovat ratkaistavissa polynomimääräistä *muistia* käyttäen. Nyt vaatimus ei siis koske suoritusaikaa, kuten P ja NP, vaan nimenomaan muistitilaa. PSPACE-täydellinen ongelma on PSPACE:ssa oleva, PSPACE-kova ongelma. (Viglietta 2020)

EXPSPACE on vastaavasti niiden päätösongelmien vaikeusluokka, jotka ovat ratkaistavissa deterministisellä Turing-koneella eksponentiaalista muistitilaa käyttäen. Tämä luokka sisällyttää itseensä muiden vaikeusluokkien lisäksi myös laskennalliselta kannalta erittäin raskaita ongelmia sekä tilan että ajan käytön suhteen. Voimme nähdä näiden luokkien suhteet visualisoituna kuvassa 5.



Kuva 5: Vaikeusluokkien visualisointi, mikäli P ≠ NP

Vaikeusluokkien suhteet toisiinsa ovat yhä osittain hämärän peitossa, joten emme voi suoraan sanoa, ovatko mitkä luokat keskenään samankokoisia vai toistensa aitoja osajoukkoja. Voimme siis sanoa, että $P \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{EXPSPACE}$, mutta tiedetään myös, että $\text{P} \subsetneq \text{EXPTIME}$, joten jossain vaiheessa ketjua joukot eivät ole saman kokoisia. Muistin käytön suhteen on olemassa myös Savitchin teoreema, joka kertoo esimerkiksi sen, miksi PSPACE = NPSPACE. (Savitch 1970)

2.6 P = NP

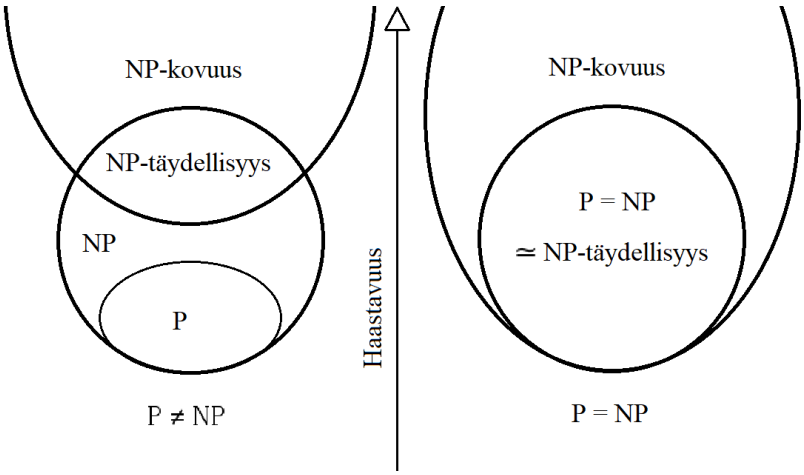
P = NP -ongelma alkoi tulla puheeksi jo 1950-luvun puolivälissä, mutta suurempaan julkisuuteen ongelman toivat kuitenkin toisistaan riippumatta kylmän sodan eri puolilla työskentelevät Steve Cook ja Leonid Levin vasta 1970-luvulla. Pian tämän jälkeen Richard Karp onnistui muuntamaan Cookin esittämän NP-täydellisen ongelman toiseksi ja todistamaan, että on olemassa “vähintään yhtä vaikeita” ongelmia, jotka kaikki ovat muunnettavissa toisiinsa. Tämän jälkeen P = NP -ongelman merkitys on vain kasvanut ja

nykyään sitä havaitaan vahvasti tietojenkäsittelytieteen lisäksi myös muilla aloilla, kuten biologiassa, lääketieteessä, taloustieteessä ja fysiikassa. (Fortnow 2013)

Miksi P = NP ongelma on niin merkitsevä? No, mikäli ajatellaan, että P = NP pitää paikkansa, niin se tarkoittaisi sitä, että suurelle osalle haastaviksi luokittelemillemme ongelmille on olemassa helppo, tehokas algoritmi, joka ratkaisee ne. Toisin sanoen, suuri osa nykyajan ongelmista voitaisiin ratkaista helposti ja tehokkaasti. Täydellisten aikataulujen laatiminen olisi helppoa, tuottavuuden maksimoiminen sujuisi hetkessä ja lääkärit tietäisivät heti, miten kannattaa hoitaa potilasta parhaiten. Melkein kaikki olisi laskettavissa koneiden avulla, kunhan vain hyödynnetään edellä mainittua nopeaa ja tehokasta algoritmia. (Fortnow 2013)

Tällä hetkellä tällaista algoritmia ei ole kuitenkaan löydetty, ja sen löytyminen vaikuttaa epätodennäköiseltä. Miksi emme siis voi vain todistaa, että P ≠ NP? Koska se puolestaan vaatisi meitä todistamaan, että tällaista algoritmia ei voi olla olemassa, ja sen todistaminen on matemaattisesti hyvin haastavaa. Kukaan ei ole tähän mennessä onnistunut sitä tekemään, mutta yleinen mielipide on kallistunut vahvasti siihen, että P ≠ NP. Tämänhetkinen konsensus tuntuu olevan se, että ihmisillä ei ole vielä käsissään riittäviä todistustekniikoita tällaisten todistusten laatimiseen, joten P = NP kysymys todennäköisesti pysyy avonaisena niin kauan, kunnes nämä tekniikat kehitetään. (Fortnow 2013)

P = NP tilanne heijastuisi myös tällä hetkellä yleisesti vallassa oleviin ongelmien luokitusmalleihin esimerkiksi kuvan 6 lailla. Koska kaikki NP-kovat ongelmat olisivat ratkaistavissa muunnosten ansiosta samalla ratkaisualgoritmilla, myös P-luokan ongelmat olisivat NP-kovien ongelmien osajoukko. Mikäli taas P ≠ NP, se tarkoittaisi, että meillä olisi NP-luokkaan kuuluvia ongelmia, jotka eivät kuulu luokkaan P, mutta eivät ole myöskään NP-kovia. Tästä aiheesta on kirjoittanut oman näkemyksensä Richard Ladner. (Ladner, 1975)



Kuva 6: Ongelmien luokittelun visuaalisuusero, riippuen siitä, onko P = NP

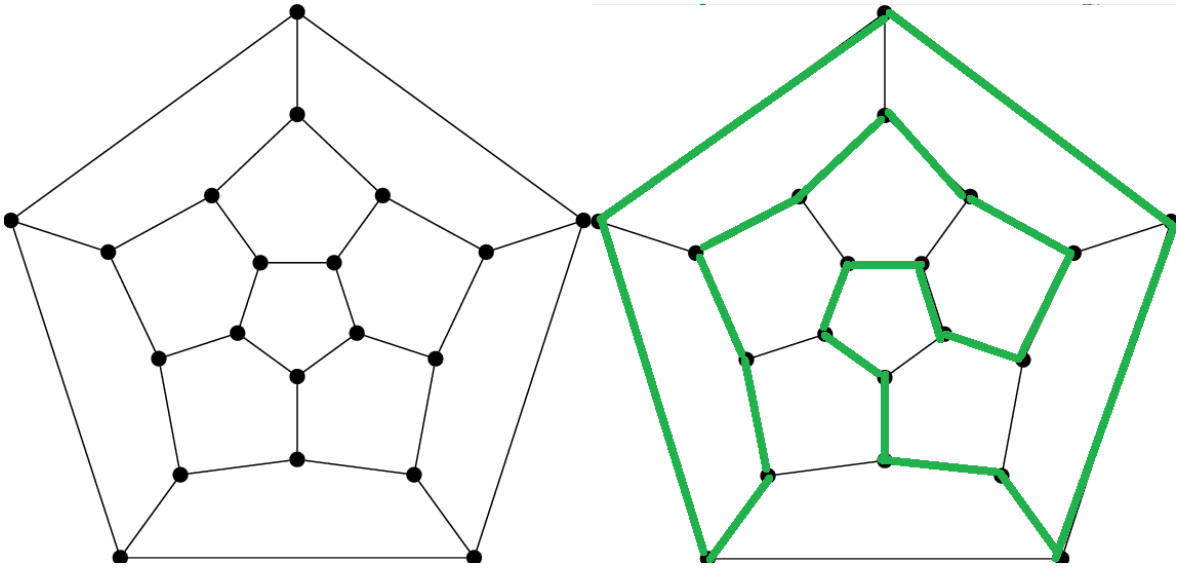
P = NP ongelma on kuitenkin edelleen kiistanalainen aihe, samoin kuin NP = co-NP. Mikäli tämän työn lukijaa kiinnostaa lukea tästä aiheesta enemmän, materiaalia ei ole lainkaan vaikeaa löytää. Tämän työn fokus ei kuitenkaan ole tässä kiistassa, vaan NP-täydellisyydestä peleissä, joten jatkamme valloillaan olevan oletuksen mukaan, että P ≠ NP ja siirrymme katsomaan, millä tavalla pelien vaikeusluokkia on tutkittu ja millaisia todistusmenetelmiä niiden yhteydessä on käytetty.

3 NP-täydellisyys peleissä

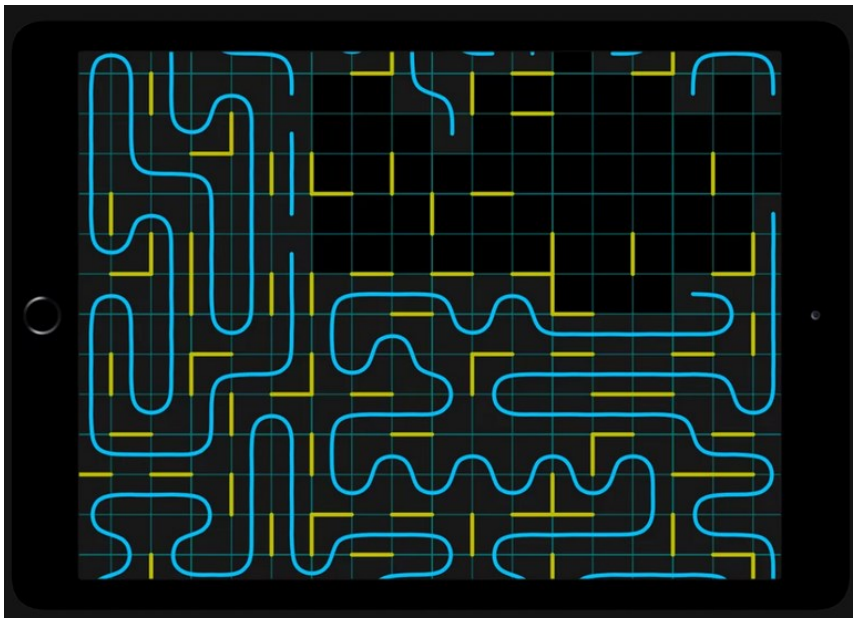
Esittelimme luvussa 2 muutaman erilaisen NP-täydellisen ongelman, joita hyödynnämme kohta peliesimerkkien läpikäymisessä. Voimme nopeasti huomata, että moni NP-täydellisistä ongelmista ilmenee joko itsessään pelinä tai voimme helposti löytää olemassa olevista peleistä samoja piirteitä. Esimerkiksi voimme ottaa vaikkapa graafin 3-väriongelman: erilaisten graafien värittämistä kolmella värillä on oman kokemukseni mukaan pelillistetty älypelien ystävien keskuudessa, mutta sen suurempaa näkyvyyttä ongelma ei ole tietääkseni saanut.

Sen sijaan jos otamme askeleen ylöspäin ja katsomme Kenneth Appelin ja Wolfgang Hakenin todistamaa neliväriteoreemaa, jonka mukaan minkä tahansa kartan alueet voi värittää vain neljää väriä käyttäen niin, etteivät mitkään kaksi vierekkäistä aluetta päädy samanvärisiksi, siitä on tehty moniakin pelivariaatioita. (Appel ja Haken, 1977) Nämä pelit tulivat ainakin minulle hyvin usein vastaan Flash-pelien kulta-aikana, mutta pelitrendien muuttuessa niiden suosio on selvästi vähentynyt. Edelleen hakemalla Googlesta hakusanoilla ”Four color theorem game” voi löytää teoreemaa hyödyntäviä pelejä eri alustoille. Vaikka neliväriteoreema ei itsessään liitykään NP-täydellisyyteen, niin se, ”Onko haluttu kartta väritettävissä kolmella värillä?” on NP-täydellinen ongelma. (Stockmayer 1973)

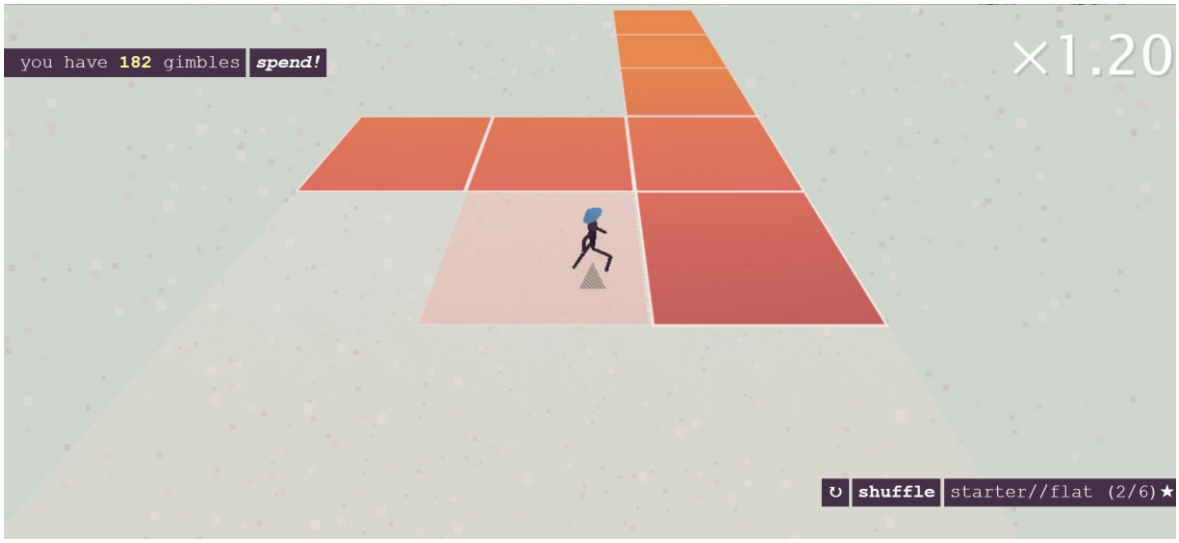
Myös esimerkiksi luvussa 2 ohimennen mainittu NP-täydellinen Hamiltonian cycle -ongelma on varsin tunnetusti pelillistettynä, sillä ongelma itsessään on kuin peli. Sitä on jopa myyty fyysisenä pelinä nimellä Icosian game. (Dalgety, 2017) Pelissä tarkoituksena on löytää yhtenäinen tie verkoksi levitetyn 12-sivuisen geometrisen kuvion läpi, kulkemalla jokaisen sen pisteen kautta tasan kerran. Visualisoinnin tästä voimme nähdä kuvassa 7. Peliä on sittemmin laajennettu myös isommille verkoille ja se on inspiroinut useampia variaatioita, kuten esimerkiksi Good Thinking Games Ltd:n The Hamiltonian Circuit (kuva 8) ja Bark and Fester LLC:n roTopo (kuva 9). Varsinkin jälkimmäisestä seikkailupelien ystävät voivat helposti nähdä, että vastaavia tienhakuelementtejä esiintyy monien muidenkin pelien yhteydessä, sillä tien löytäminen lattian kadotessa alta on varsin suosittu troupe muun muassa seikkailupeleissä.



Kuva 7: Icosian game, pohjautuu Hamiltonian Circleen, esimerkkiratkaisuineen.



Kuva 8: The Hamiltonian Circuit -peli, tehtävänä laatia yhtenäinen tie, joka kulkee joka ikisen ruudun kautta. Lähde: GoodThinkingGames Ltd.



Kuva 9: Bark and Fester LLC:n RoTopo, tehtävänä juosta kaikkien ruutujen läpi kerran ja päästä loppuun.

Mutta miksi meitä kiinnostaa tietää, mihin vaikeusluokkaan peli kuuluu? Mitä väliä sillä on, onko peli esimerkiksi P vai NP luokassa? Onko tiedosta kenellekään mitään käytännön hyötyä?

Ehkä selkein erottaja P ja NP -luokkaisten pelien välillä on niiden ”ratkaistavuus” peleinä. Mikäli peli kuuluu P-vaikeusluokkaan, se viittaa siihen, että siihen on olemassa suhteellisen nopea ratkaisualgoritmi, jonka löydyttyä peliin ei jää mitään ”ratkaistavaa”. (Eppstein 2021) Pelimaailmassa tämä näyttäytyisi suurena määränä botteja, joita vastaan oikeat pelaajat kyllästyvät kilpailemaan ja siirtyvät muiden pelien pariin. Sama toimii myös yksin pelatessakin: heti, kun löytää aina toimivan tavan ratkaista ongelma systemaattisesti, ongelmanratkaisun sijaan peliin jää ainoastaan pelkkä nopeusaspekti, kun sen pelaaminen muuttuu vain tavaksi toistaa ratkaisun vaiheet mahdollisimman nopeasti. Mikäli taas peli kuuluu NP-vaikeusluokkaan, mitään parasta ”temppua” pelin ratkaisemiseksi ei ole olemassa. (Eppstein 2021) Ongelmanratkaisuaspekti ei siis poistu, vaikka peliä pelaisi kuinka pitkään.

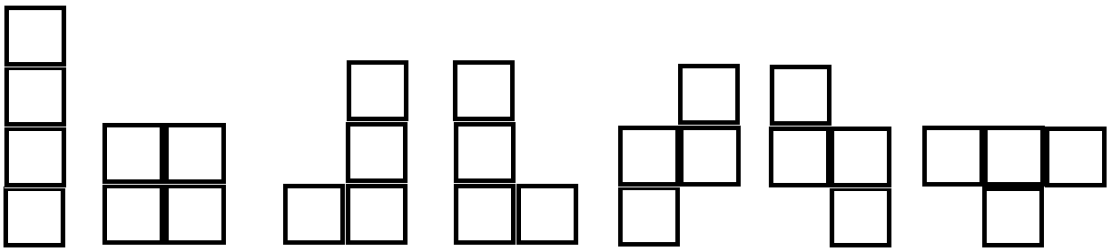
On kuitenkin hyvä muistaa, että edellä mainitut väitteet ovat vain yleistyksiä, jotka koskevat luokkiin kuuluvia pelejä ryhminä, eivätkä viittaa yksittäisiin peleihin. On esimerkiksi olemassa joukko määritelmältään NP-koviksi luokiteltuja pelejä, joille on kuitenkin

olemassa “tarpeeksi tehokkaita” ratkaisualgoritmeja, jotka löytävät ratkaisun keskimääräisesti nopeasti, jääden tehottomiksi vain tietyissä rajatapauksissa. Vastaavasti P-luokan pelin tehokkaan ratkaisun löytäminen ei välttämättä vie pelin mielekkyyttä täysin, mikäli sen ratkaisuprosessi koetaan itsessään mielekkäänä. Pelin kuuluminen P-luokkaan ei siis automaattisesti kerro sen olevan tylsä, eikä NP-luokkaan kuuluvuus tee pelistä automaattisesti vaikeaa. Näihin luokkiin kuuluu hyvin monenainen kirjo ongelmia, joihin mahtuu sekä helppoja että vaikeita variaatioita. Pelit myös harvoin koostuvat näistä ongelmista itsessään, vaan ongelman lisäksi pelin osana voi olla paljon muutakin, kuten sosiaalinen vuorovaikutus, kilpailuaspekti tai itseilmaisun keinot. Tämä pätee erityisen vahvasti digitaalisiin peleihin, jotka ovat keskimääräisesti paljon mutkikkaampia kokonaisuuksia, kuin perinteiset, klassiset pelit.

Kun tämän työn puitteissa siis väitämme, että joku peli on NP-täydellinen, väite ei välttämättä tarkoita sitä, että peli sellaisenaan kuuluisi NP-luokkaan ja olisi NP-kova, sillä käsittelemämme pelit eivät ole päätösongelmia. Väitteellä viitataan pikemminkin siihen, että kyseisen pelin tarjoamat säännöt mahdollistavat NP-täydellisten päätösongelmien toteutuksen, eli peli ns. ”sisältää” NP-täydellisiä päätösongelmia. Nämä kyseiset ongelmat esitellään jokaisen pelin käsittelyn yhteydessä ja yleensä ne koskevat pelin joitain voitto- tai häviämiskriteerejä. Lukijan on siis hyvä ottaa huomioon, että jos muualla kirjallisuudessa törmää teoksiin nimityylillä ”Peli X on NP-täydellinen”, viittaus ei sielläkään kohdistu itse peliin, vaan johonkin sen sisältämään osaan. Tämän väitteen ei ole tarkoitus olla harhaanjohtava, vaan se on vain likimääräinen, helpompi ilmaisutapa jollekin asialle, minkä avaaminen muuten olisi liian monimutkaista, varsinkin otsikon roolissa.

3.1 Tetris

Tetris on neuvostoliittolaisen ohjelmoija-pelisuunnittelijan Alexey Pazhitnovin 1980-luvun puolivälissä tekemä peli, jossa pelaajan on tarkoitus sijoitella ylhäältä putoilevia palikoita eli tetrominoja (kuva 10) niin, että pelitilan rivit täyttyvät tavalla, joka tuottaa mahdollisimman paljon pisteitä. Pelin vaikeutta säätelee tetrominojen putoamisnopeus sekä se, miten täynnä pelitila on pelin alkaessa.



Kuva 10: Tetrominot, eli Tetriksen pelinappulat. Huomataan, että jokainen muodostuu tasan 4 palikasta, näin ollen pinta-ala on kaikilla sama.

3.1.1 Säännöt

Perinteisessä Tetriksessä tetrominojen järjestys on arvottu sattumanvaraiseksi, eli pelaajalla ei voi olla ennakkoon tietoa siitä, missä järjestyksessä tetrominot tulevat. Pelaajan annetaan nähdä pelin aikana enintään heti seuraavana tulovuorossa oleva tetromino. Modernissa Tetriksessä tästä säännöstä on poikettu ja on olemassa erilaisia variaatioita siitä, millä logiikalla tulevat tetrominot arvotaan ja minkä määrän tulevista tetrominoista pelaaja saa nähdä ennakkoon pelin aikana. Esimerkiksi on yleistä, että modernissa Tetriksessä pelaaja tietää, että tetrominot tulevat seteittäin, eli jokainen seitsemästä eri tetrominosta tulee satunnaisessa järjestyksessä pelikentälle, jonka jälkeen alkaa uuden seitsemän palan setti omalla arvotulla järjestyksellään. Näin ollen kahden samanlaisen tetrominon väli voi olla enintään 12 tetrominoa. Perinteisessä Tetriksessä taas tetrominojen järjestys ei noudata tällaista setti-järjestelyä, vaan kahden samanlaisen tetrominon väli voi olla käytännössä miten suuri tai pieni tahansa.

Tetriksellä ei ole varsinaista voittokriteeriä, eli peliä ei voi voittaa. Pelin häviämiskriteeri on, että pelikenttä täyttyy, eikä seuraavana tulevalle tetrominolle ole enää tilaa generoitua. Pelin tuloksen ratkaisee pelaajan pelin aikana saavuttama pistemäärä, joka siis määrittää sen, kuinka hyvin tämä pelasi. Näin ollen pidämme pelin tavoitteena mahdollisimman korkean pistemäärän saavuttamisen.

Koska tavoitteemme on tutkia Tetriksen NP-täydellisyyttä, meidän on muutettava peli päätösongelmaksi. Seuraavan todistuksen yhteydessä teemme sen ottamalla Tetriksen tavoitteista käsittelyymme pelikentän tyhjenemisen, eli päätösongelmamme on muotoa

”Voidaanko annettu esitäytetty pelikenttä tyhjentää käyttämällä annettu rajallinen tetrominosekvenssi niin, että sen loputtua kenttä on täysin tyhjä?”. Koska pelikentän tyhjeneminen saa meidän kauemmaksi pelin häviämiskriteeriltä, eli pelikentän täyttymiseltä, päätösongelma vastaa pelin tavoitteita.

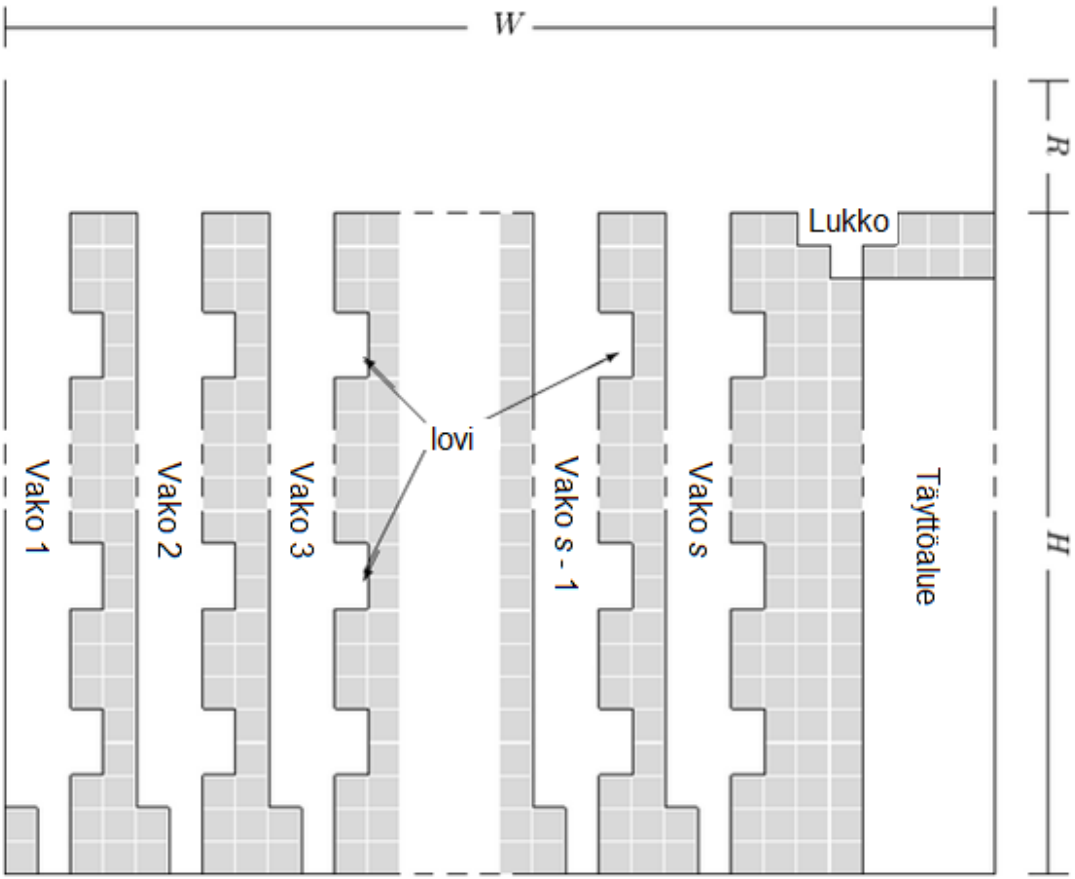
3.1.2 Alkuasetelma

Tässä alaluvussa esitetyt todistukset on poimittu Erik D. Demainen, Susan Hohenbergerin ja David Liben-Nowellin työstä “Tetris is Hard, Even to Approximate” (Demaine, Hohenberger ja Nowell 2003) sekä sen pohjalta Ron Breukelaarin, Hendrik Jan Hoogeboomin ja Walter A. Kostersin tekemästä yksinkertaistetummasta työstä ”Tetris is Hard, Made Easy” (Breukelaar, Hoogeboom ja Kosters 2003). Ensimmäisessä työssä käsitellään Tetriksen eri tavoitteiden NP-täydellisyyttä, siinä missä jälkimmäisessä avataan tämän todistuksen pohjaa yksinkertaistetummalla mallilla, keskittyen vain yhteen Tetriksen sisältämään päätösongelmaan. Tarkempia yksityiskohtia varten suosittelenkin lukijoita vilkaisemaan nämä artikkelit kokonaisuudessaan läpi. Tämän työn puitteissa pyrimme vain esittämään todistuksen ydinidean, joten pohjaudumme jälkimmäiseen artikkeliin, jättäen ensimmäisen artikkelin laajennokset pois.

Demainen, Hohenbergerin ja Nowellin artikkelissa todistetaan Tetriksen NP-täydellisyys muuntamalla yksi tunnetuista NP-täydellisistä ongelmista “3-partition” Tetriksen ongelmaksi. Esittelimme kyseisen NP-ongelman pikaisesti jo luvussa 2. Breukelaar, Hoogeboom ja Kosters sen sijaan yksinkertaistivat omassa työssään Demainen esittämän todistuksen rajaamalla pelikenttää ja tulevaa tetrominojen jonoa, osoittaen silti samat tulokset pitäväksi yhden päätösongelman kohdalla. Pelikentän rajaus koostui heidän työssään 2 palikan levyisistä alueista, jolloin muotonsa vuoksi tetrominoja ei pysty kääntelemään, poistaen näin laskuista kaikki erikoiset kääntelyvariaatiot ja yksinkertaistaen todistuksia. (Braukelaar, Hoogeboom ja Kosters 2003)

Kumpikin artikkeli käyttää todistusmenetelmänsä pohjalla 3-partitionia, joka saadaan muunnettua Tetris-ongelmaksi muokkaamalla Tetris-pelialue vastaamaan samanlaista ongelman asettelua. Tähän graduun on poimittu kahdesta muunnoksesta se

yksinkertaistetumpi malli, eli Braukelaarin, Hoogeboomin ja Kosterin muunnos. Voimme nähdä tämän alkuasetelman kuvassa 11. Aloituskentälle rakennetaan *s* määrä *vakoja*, joissa jokaisessa on joukko *lovia*, minkä lisäksi käytössä on myös tyhjä, rivien ennenaikaista katoamista estävä *täyttöalue*, johon pääsyä estää *lukko*. (Braukelaar, Hoogeboom ja Koster 2003)



Kuva 11: Tetris-pelialueen rakenne mukautettuna 3-partition -ongelman muotoon. Kuvan lähde: *Tetris is Hard, Made Easy*, suomennettu.

NP-täydellisyyden tutkimisen kannalta huomaamme taas, että tämä asettelu mahdollistaa meille mielivaltaisen kokoisen pelikentän, joka ei ole rajattu pelin normaaliin esiintymiseen. Sekä vakoja voi olla miten paljon tahansa, että niiden korkeus voi vaihdella, kunhan rakenne pysyy samana. Näin ollen asetelmamme pystyy simuloimaan miten isoja ongelma-instansseja tahansa, pelin sääntöjen pysyessä samana.

3-Partitionin määritelmä:

Oletetaan, että meillä on positiivisten kokonaislukujen muodostama lukujono *A*, joka koostuu arvoista $a_1, \ldots, a_{3s}$ ja positiivinen kokonaislukuarvo *T* niin, että $T/4 < a_i < T/2$ kaikilla $1 \leq i \leq 3s$ ja $\sum_{i=1}^{3_s} a_i = sT$.

Kysymys: voidaanko *A* jakaa *s* määräksi erillisiä osajoukkoja $B_1, \ldots, B_s$ niin, että $\sum_{a_i \in B_j} a_i = T$ kaikilla $1 \leq j \leq s$? Mikäli jako onnistuu, saamme kyllä-instanssin ja muussa tapauksessa ei-instanssin.

Tässä esimerkissä, jossa pyrimme vastaamaan kysymykseen ”*Voidaanko annettu esitäytetty pelikenttä tyhjentää käyttämällä annettu rajallinen tetrominosekvenssi niin, että sen loputtua kenttä on täysin tyhjä?*”, kokonaislukuja vastaa tetrominot ja haluttu tasainen summa *T* tarkoittaa pelikentän täydellistä tyhjenemistä. Jokainen vako on keskenään identtinen ja tehty lovien ansiosta sellaisiksi, että ne on mahdollista täyttää vain yhdellä tavalla. Näin voimme varmistaa, että tämän Tetris-asetelman pystyy ratkaisemaan arvolla “kyllä” ainoastaan silloin, kun myös 3-partitionin vastaava instanssi saa vastauksen “kyllä”. Kaikki muut variaatiot johtavat vastaukseen “ei”, jolloin kenttä ei tyhjene. Pelikentän tyhjeneminen vastaa arvoa *T* siksi, että mikäli yksikin palikka jää kentälle, kyseessä on joko yli- tai alijäämä määrätystä arvosta. Käytämme termiä ”palikka” vastaamaan kuvan 11 yhtä ruutua, joka toimii myös yksikkönä laskiessamme kentän muita arvoja. Kuten kuvassa 10 kävi ilmi, jokainen tetromino koostuu tasan neljästä palikasta.

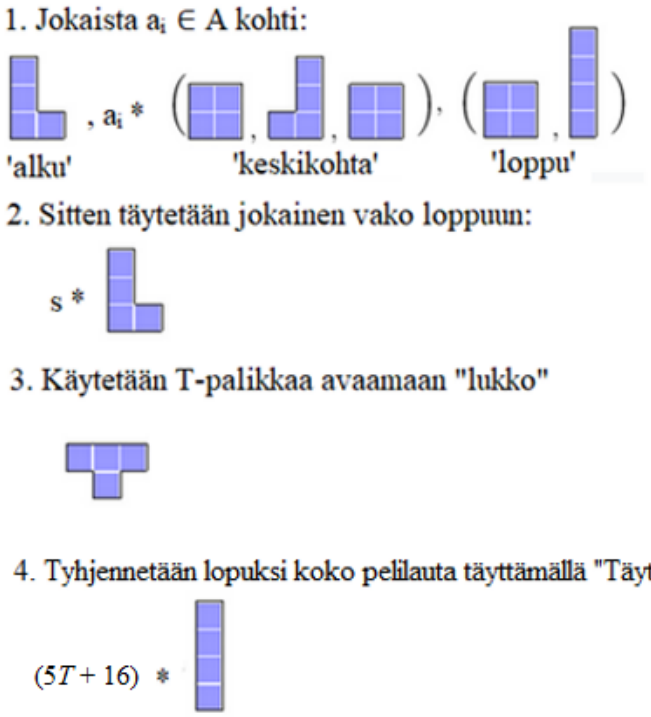
Kuvassa 11 voimme nähdä, että vakoja voi olla mielivaltainen määrä (*s*) pelikentän halutusta koosta riippuen. Jokaisessa vaossa on *T* + 3 lovea niiden oikeassa laidassa. Pelikentän leveys *W* muodostuu kokoon 4*s* + 6, sillä jokaisen vaon muodostamiseen tarvitaan 4 palikan leveyttä, täytealue on myös 4 palikan levyinen ja 2 ylimääräistä pylvästä ovat lukon toteutusta varten viimeisen vaon ja täytealueen välissä.

H on kentän esitäytetyn alaosan korkeus, joka meidän on saatava tyhjennettyä kokonaisuudessaan. *H* on 5*T* + 18. Kentän yläpuolelle jäävä tila R on laskettu riittäväksi tilaksi siirtää ja tarvittaessa käännellä putoava tetromino haluttuun paikkaan vaossa, joten sen tarkempi koko ei vaikuta laskuihimme. Kentän oikealle laidalle jätetään täyttöalueeksi

tässä tapauksessa neljän palikan levyinen suora, tyhjä alue, joka on helppo täyttää lopussa pelkkiä I- tai neliötetrominoja käyttäen. Periaatteessa tämä tila voi olla millainen tahansa, kunhan sen korkeus on H-2 ja se on helposti täytettävissä jokaisella muunnoksella. Täyttöalue on olemassa sitä varten, ettei rivien poistuminen tapahdu kesken täyttämisen, vaan se vaatii ensin tätä tilaa lukitsevan lukon täyttämisen. Kuten kuvista 10 ja 11 voi huomata, lukko on tehty vastaamaan tasan yhtä mahdollista palikkavaihtoehtoa ja sitä on mahdotonta täyttää minkään muun palikan avulla sillä tuloksella, että pelikenttä tyhjenee kokonaisuudessaan.

Tässä välissä on tärkeä huomata, että kyseinen pelialue on rakennettavissa polynomiajassa, sillä ongelmaesityksen muuttujat pystytään antamaan unaarimuodossa vahvan NP-täydellisyyden ansiosta. Ilman sitä todistus ei olisi pätevä, sillä yksi NP-kovuuden todistuksen kriteereistä on muunnettavuus polynomiajassa.

Jokainen alkuasetelmamme vako vastaa yhtä 3-partitionin osajoukkoa, ja niitä on *s* määrä. Breukelaar, Hoogeboom ja Kosters käyttävät todistuksessaan kuvan 12 mukaista palikkasekvenssiä, Demainen, Hohenbergerin ja Nowellin käyttäessä omassa vastaavassa todistuksessaan toista vaihtoehtoa. Tämän gradun puitteissa käsittelemme Breukelaarin ym. tiivistetympää todistusta, mutta lukijaa ohjataan perehtymään myös toiseen versioon, joka laajentaa todistuksen koskemaan myös muita Tetriksessä esiintyviä päätösongelmia, eli todistetaan muidenkin pelin tavoitteiden NP-täydellisyys.



Kuva 12: Todistuksen käyttämä tetrominosekvenssi. Lähde: *Tetris is Hard, Made Easy*, suomennettu

3.1.3 Todistus

Todistaaksemme muunnoksen 3-partitioninista tähän Tetris-asetelmaan, meidän on todistettava kaksi asiaa:

1. 3-partitionin “kyllä”-instanssi muuntautuu polynomiajassa sellaiseksi Tetris-instanssiksi, että pelilaudan tyhjeneminen on mahdollista.

2. 3-partitionin “ei”-instanssi muuntautuu polynomiajassa sellaiseksi Tetris-instanssiksi, että pelilaudan tyhjeneminen on mahdotonta.

Kun kumpikin väite toteutuu, osoitamme, että valitsemamme Tetris-ongelma on vähintään yhtä vaikea, kuin 3-partition. Koska 3-partition on todistetusti NP-täydellinen ja muunnos tehdään siitä eikä toisin päin, Tetris-asetelmamme on myös NP-kova. Seuraavaksi käymme läpi Breukelaarin, Hoogeboomin ja Kostersin esitykset kunkin väitteen toteutumiseen heidän

antamallaan laudan alkuasetelmalla ja tetrominosekvenssillä. Kuvat ovat mukana havainnollistamassa tapahtumia.

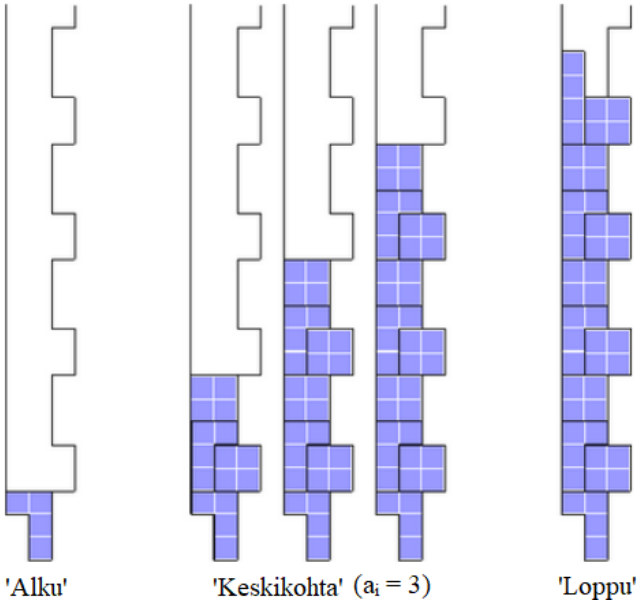
3.1.3.1 Kyllä-instanssi

Kuvassa 12 käytettävä tetrominosekvenssi vastaa arvoja $a_i \in A$. Koska *A* on osa kyllä-instanssia, se voidaan jakaa *s* määrään osajoukkoja, joiden kaikkien summa on *T*. Mikäli *A* ei olisi jaettavissa näin, meillä olisi heti alkuun ei-instanssi 3-partitionin puolelta. Pelilaudan täyttämiseksi meidän on lisättävä arvoja eli tetrominoja oikeisiin vakoihin, joka on visualisoitu kuvassa 13, jossa $a_i = 3$. Alkutetrominona toimii kuvan 12 'Alku'-osion tetromino, jota seuraa a_i määrä 'Keskikohta'-osion tetrominoja ja lopuksi käytetään 'Loppu'-osion tetrominot. Voimme huomata, että sekvenssin lopussa vako jää täysin samaan asetelmaan, kuin alussa: sen jatkamiseen tarvitaan taas 'Alku'-tetrominoa. Tämä tarkoittaa sitä, että samaan vakoon voi lisätä useampi 'arvo' peräkkäin. Tästä voimme laskea vaon korkeuden olevan:

$$\sum_{a_i \in B_j} \{(\text{"}Alun\text{" korkeus}) + a_i \times (\text{"}Keskikohdan\text{" korkeus}) + (\text{"}Lopun\text{" korkeus})\}$$

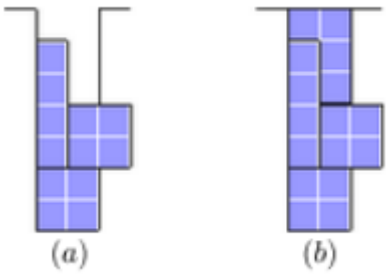
$$= \sum_{a_i \in B_j} 3 + a_i \times 5 + 2 = \sum_{a_i \in B_j} 5a_i + 5$$

Huomataan, että "Alku" ja "Loppu" -osien korkeus on osittain päällekkäin, 2 palikan verran. Tämä korkeus lasketaan yllä esitetyssä kaavassa "Alkuun", eikä lisätä enää erikseen "Loppuun". Koska $\sum_{a_i \in B_j} a_i = T$ ja $|B_j| = 3$ kaikilla $1 \leq j \leq s$ kyllä-instanssin kohdalla, korkeus on $5T + 3 \times 5 = 5T + 15$, johon lisätään "lopun" 2, jolloin saadaan $5T + 17$. Tämä on vähemmän, kuin *H* (= 5T + 18), jolloin vaon korkeus on riittävä.



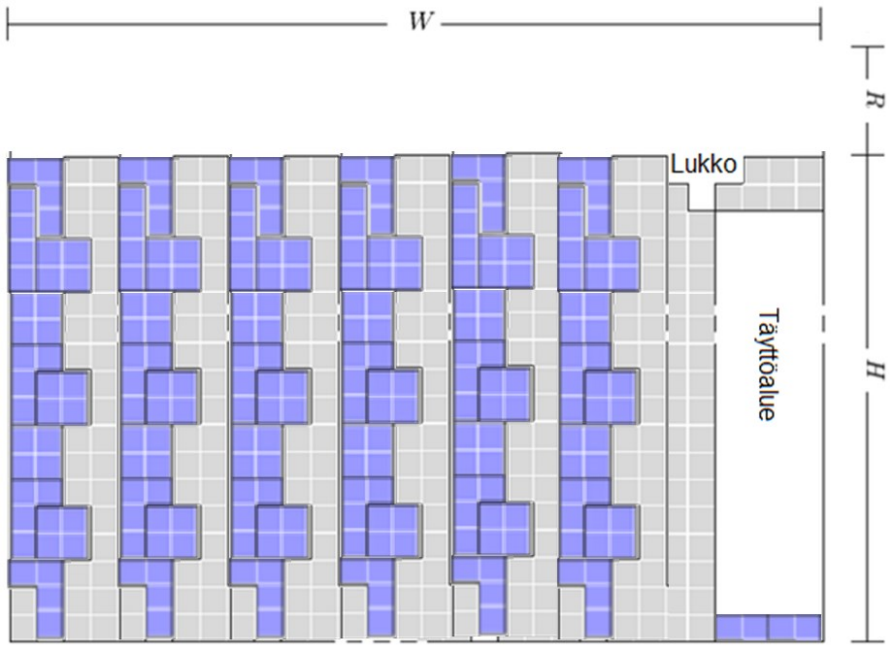
Kuva 13: Vaon täyttö tetrominosekvenssiä käyttäen. Lähde: *Tetris is Hard, Made Easy*, suomennettu.

Sen jälkeen, kun kaikki vaot on täytetty edellä esitetyn esimerkin ja kuvan 13 mukaisesti, meillä on jäljellä s määrä vakoja, jotka näyttävät kuvan 14 (a) -kohdan mukaisilta. Ne voidaan täyttää muotoon (b) käyttäen s määrää ‘osajoukon täyttäjiä’.



Kuva 14: Vakojen täyttäminen tasaisiksi. Lähde: *Tetris is Hard, Made Easy*

Tämän toimenpiteen jälkeen meillä on täysin aukottomasti täytetty pelilauta, lukkopalaa ja täyttöaluetta lukuun ottamatta. Seuraavana askeleena onkin siis lukon avaaminen T-tetrominolla, joka poistaa kaksi ylintä riviä pelikentältä, jonka jälkeen täytämme täyttöalueen valitsemallamme tavalla, tämän todistuksen puitteissa $4 \times (5T + 16)$ kokoinen tyhjiö täytetään $5T + 16$ määrällä I-tetrominoja, kuten kuvassa 15. Näin saamme kentän täysin tyhjäksi, joten todistimme valitsemamme 3-partitionin kyllä-instanssin vastaavan Tetris-asetelman kyllä-instanssia.



Kuva 15: Täytetyt vaot ja täyttöalueen täyttöä havainnollistava I-tetrominon malliasento

3.1.3.2 Ei-instanssi

Käytämme ei-instanssin todistukseen joukkoa lemmoja, eli apulauseita todistuksen eri osa-alueiden käsittelemiseksi.

Lemma 1: *Jos yksikin tetromino laitetaan 5T + 18 yläpuolelle, kenttää ei voida tyhjentää.*

Todistus: Jokaisella tetrominolla on sama pinta-ala (4 palikkaa) ja kuten äsken todistimme, niiden avulla voimme täyttää pelikentän 5T + 18 riviä aukottomasti, kun *A* vastaa “kyllä”-

instanssia. Tetrominojen määrä ei riipu siitä, onko A kyllä- vai ei-instanssi, sillä $\sum_{a_i \in A} a_i = sT$. Mikäli tetromino sijoitetaan $5T$ + 18 rivin ulkopuolelle, tulemme tarvitsemaan enemmän, kuin $5T + 18$ riviä pelikentän tyhjentämiseksi, ja se on mahdotonta. Pelin säännöistä johdamme myös sen, että tetrominoja ei voi siirtää niiden osuessa ensimmäistä kertaa johonkin, vaikka niiden alle jäisikin tyhjiä paikkoja.

***Lemma 2**: Jotta pelikenttä voidaan tyhjentää, lukkoa ei voida avata millään muulla, kuin lukkopalalla.*

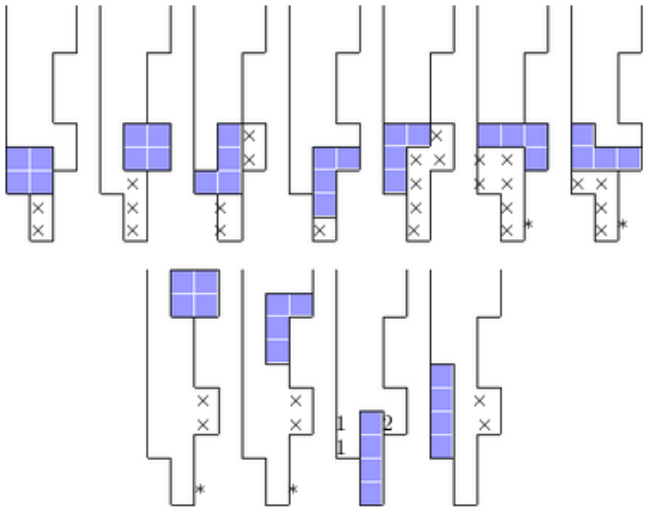
Todistus: Tetriksessä kaikki tetrominot ovat keskenään eri muotoisia ja kaikilla on sama pinta-ala (kuva 10), joten ainoastaan yksi tetromino voi sopia lukkoon, menemättä yli $5T +$ 18 rajan.

***Lemma 3**: Mikäli yhdenkään tetrominon lopullinen sija ennen lukkopalaa luo vakoihin paikkoja, joihin mikään muu tetromino ei yllä, kenttää ei voida tyhjentää.*

Todistus: Yksikään rivi ei pysty katoamaan kentältä ennen lukon avaamista täyttöalueen ansiosta, ja lukon voi avata ainoastaan lukkopalalla, kuten todistettu lemmassa 2. Käytössä olevien tetrominojen yhteenlaskettu pinta-ala riittää juuri sopivasti täyttämään kaikki vaot aukottomasti, joten mikäli vakoihin jää yksikin tyhjä tila, jokin menee väistämättä $5T$ + 18 rajan yläpuolelle, jolloin pelilaudan tyhjentäminen on mahdotonta.

***Lemma 4**: Jos mikä tahansa tetromino laitetaan väärään vakoon aloituspalan tilalle, kenttää ei voida tyhjentää.*

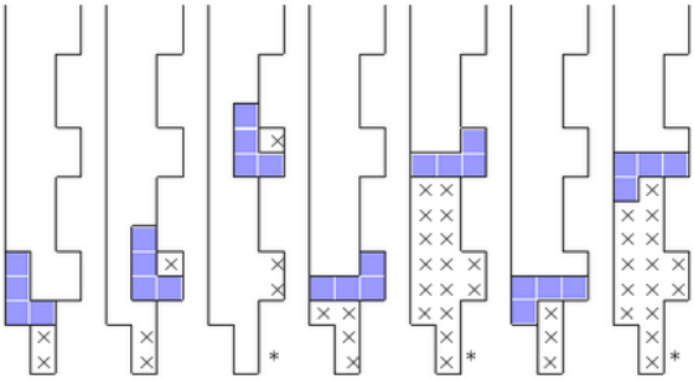
Todistus: Kuvassa 16 on visualisoitu, miten jokainen väärin asetettu tetromino aiheuttaa osia, joita on mahdotonta täyttää. Jokainen X on kohta, johon on mahdoton yltää, ja kohdat 1 ja 2 ovat sellaisia kohtia, joiden kohdalla toisen täyttyessä, toinen päätyy ulottumattomiin.



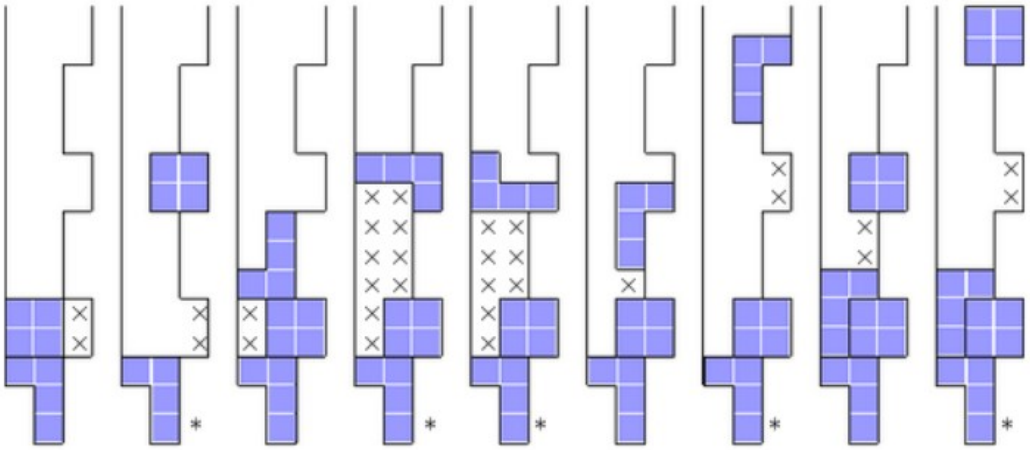
Kuva 16: Vaihtoehtoisten sijaintien aiheuttamat saavuttamattomat kohdat. Lähde: *Tetris is Hard, Made Easy*

***Lemma 5**: Jotta kenttä saadaan tyhjennettyä, jokainen tetromino pitää asettaa omaan vakoon juuri sille paikalle, mikä sille on määrätty (kyllä-instanssin esimerkissä).*

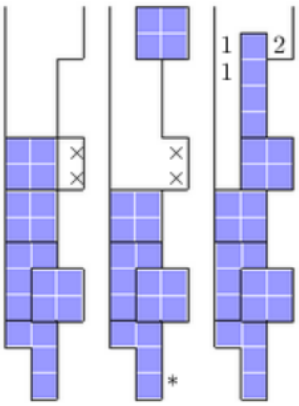
Todistuksena toimii joukko visualisointeja (kuvat 17-19) kaikista mahdollisista tetrominojen asetteluvariaatioista, joista huomaamme, että vakoon jää virheellisen asettelun jälkeen koloja, joihin ei enää ylety, näin ollen kenttää ei voida tyhjentää.



Kuva 17: Aloituspalan vaihtoehtosijoitukset. Lähde: *Tetris is Hard, Made Easy*



Kuva 18: Keskikohta-palojen vaihtoehtosijoitukset.
Lähde: *Tetris is Hard, Made Easy*



Kuva 19: Loppupalojen vaihtoehtosijoitukset
Lähde: *Tetris is Hard, Made Easy*

***Lemma 6**: Jotta kenttä tyhjenisi, jokaisen vaon pitää sisältää tarkalleen kolme 'arvoa' ja näiden arvojen summa pitää olla tasan T.*

Todistus: Jokaisessa vaossa on $T + 3$ lovea. Koska jokainen 'arvo' on sijoitettava samaan vakoon (kts. lemma 4) ja on vain yksi sopiva tapa sijoittaa 'arvo' vakoon (lemma 5), 'arvo' a_i tulee aina täyttämään $a_i + 1$ lovea: yksi per keskikohta ja yksi yhteenlaskettu alku- ja loppukohdista. Tämä tarkoittaa, että jokaisessa vaossa B on $\sum_{a_i \in B} a_i + |B|$ määrä täytettäviä

lovia. Pelikentän tyhjentämiseksi pitää täyttää tasan $T + 3$ lovea jokaisessa vaossa, eli $\sum_{a_i \in B} a_i + |B| = T + 3$. Jos $|B| < 3$, saamme $\sum_{a_i \in B} a_i \geq T + 1$ mikä on mahdotonta, sillä $a_i < T/2$ kun $1 \leq i \leq 3s$. Jos taas $|B| > 3$, saamme $\sum_{a_i \in B} a_i \leq T - 1$, mikä on mahdotonta, koska $T/4 < a_i$ kun $1 \leq i \leq 3s$. Näin päädymme siihen, että $|B| = 3$ ja $\sum_{a_i \in B} a_i = T$.

***Lemma** 7: Mikäli käytetään muunnoksessa 3-partitionin ei-instanssia, se johtaa siihen, ettei Tetris-kenttää voida tyhjentää.*

Todistus: Jotta kenttä voidaan tyhjentää, kaikki sen *s* vakoa pitää sisältää tasan kolme arvoa (kts. lemma 6) ja näiden arvojen summa on oltava tasan *T*. Tämä tilanne onnistuu ainoastaan silloin, kun *A* on kyllä-instanssi. Näin ollen pelikenttää ei voida tyhjentää, mikäli *A* on ei-instanssi.

3.1.4 Todistuksen laajennos

Äsken käsittelemämme todistus koskee Tetriksen optimaalista pelaamista, jossa muunnettiin 3-partitionin instanssit Tetriksen muotoon. Muunnos on toteutettu polynomiajassa, sillä pelikenttä on mahdollista luoda instanssin koon puitteissa. Tarkemmassa muodossa käsittelemämme kysymys oli ”Voidaanko annettu esitäytetty pelikenttä tyhjentää käyttäen annettu rajallinen tetrominosekvenssi niin, että sen loputtua kenttä on täysin tyhjä?”. (Breukelaar, Hoogeboom ja Kosters 2003)

Demaine, Hohenberger ja Nowell laajentavat työssään todistuksensa koskemaan myös muita Tetriksen tavoitteita, joista NP-täydellisyyden täyttävien päätöstehtävien muotoon on muutettu sellaiset Tetriksen tavoitteet, kuin poistettujen rivien maksimoiminen, pelattujen tetrominojen määrän maksimoiminen ennen pelin loppumista, tetristen (4 rivin yhtäaikainen poistaminen) määrän maksimoiminen ja pelikentän korkeuden minimoiminen pelin aikana. (Demaine, Hohenberger ja Nowell 2002. Breukelaar, Hoogeboom ja Kosters 2003)

Demainen, Hohenbergerin ja Nowellin todistukset käyttävät siis samaa pohjaa 3-partitionin muunnoksen puolesta ja ottavat vain enemmän asioita huomioon, vastaten ehkä lähemmin

itse pelin todellista muotoa. Tämän gradun tavoitteena oli tuoda esiin todistuksen ydin mahdollisimman selkeällä tasolla, josta lukija voi halutessaan jatkaa perehtymistään Demainen, Hohengergin ja Nowellin työhön valmiilla ymmärryspohjalla. Näiden artikkelien lukemisen yhteydessä on myös hyvä pitää silmällä NP-kovuuden ja NP-täydellisyyden eroja, sillä on tyypillistä, että todistuksissa ilmenee ainoastaan NP-kovuutta koskeva osa, ja päätösongelman kuulumista NP-luokkaan ei erikseen selitetä. Näin ollen myös käsittelemässämme Breukelaarin, Hoogeboomin ja Kostersin artikkelin lopussa hypätään teoreemaan ”Tetris is NP-complete” heti sen jälkeen, kun ongelman NP-kovuus on todistettu. (Breukelaar, Hoogeboom ja Kosters 2003, s.9)

3.2 Super Mario Bros.

Vuonna 2015 Greg Aloupis, Erik D. Demaine, Alan Guo ja Giovanni Viglietta käsittelivät artikkelissaan “*Classic Nintendo Games are (Computationally) Hard*” useamman klassisen Nintendopelin vaikeusluokkia. Työssä esiintyvät tietyt pelit sarjoista Super Mario, Donkey Kong, Legend of Zelda, Metroid ja Pokemon. Ensisilmäyksellä pelikirjo tuntuu yllättävän laajalta, mutta ne kaikki käyttävät samoja todistuspohjia, pienillä pelimekaniikkojen pakottamilla muutoksilla. Osa peleistä todistetaan kuuluvan NP-kovien ongelmien luokkaan ja osa saa luokakseen PSPACE. (Aloupis ym., 2015)

Tässä työssä käymme tarkemmin läpi pelin Super Mario Bros. -osion, jonka perusteella tasoloikkapelin NP-kovuuden todistuksen idea tulee lukijalle selväksi. Esimerkit siitä, miten todistusta adaptoidaan muihin peleihin sekä NP-kovien ja PSPACE-vaikeiden pelien eron voi lukea suoraan “*Classic Nintendo Games are (Computationally) Hard*”-artikkelista. (Aloupis ym., 2015)

Todistuksessa on käytetty pelien laajennettuja muotoja, eli pidämme pelin kaikki säännöt ja ominaisuudet ennallaan, laajentaen ainoastaan pelikenttää mielivaltaisen suureksi. Päätöskysymykseksi asetamme Super Mario Bros. -pelin kohdalla: “Annetusta pelin tilanteesta, onko pelaajan mahdollista päästä kentän loppuun?”. Tehtäväksemme jää siis muodostaa tämä lähtötilanne ja pystyä tarkistamaan, toteutuvatko kentän läpäisyn ehdot vai

ei. Koska varsinkin Super Mario Bros. -pelille kentän läpäisy eli loppuun pääseminen on sen voittokriteeri, vastaamme annetulla kysymyksellä myös siihen, onko peli voitettavissa annetusta lähtökohdasta vai ei.

Käsittelemämme peli on Nintendon julkaisema ja kaikki työn puitteissa esitetyt pelihahmot ja muut peliin liittyvät visuaaliset esitykset ovat Nintendon oikeuksien alaisia. Kuvakaappaukset Super Mario Bros. -pelistä ovat mukana työssä havainnollistamassa todistuksen vaatimaa kentän rakennetta, eivätkä kuulu pelissä julkaisuvaiheessa oleviin kenttiin tai niiden osiin. Kenttien rakenne ja itse kuvat on siteerattu käsittelyn kohteena olevasta Aloupiksen artikkelista. (Aloupis ym., 2015)

3.2.1 Pelin säännöt

Super Mario Bros. on vuonna 1985 Nintendon julkaisema 2-ulotteinen tasohyppelypeli Nintendo Entertainment System (NES) -alustoille. Pelissä pelaajalla on hallussaan Mario-niminen pelihahmo, joka pystyy liikkumaan sivuille ja hyppäämään 4-5 palikan korkeuteen. Marion tehtävänä on navigoida tiensä pelikenttien läpi ja kentissä esiintyy vaaroja, kuten vihollisia, tappavia kuiluja ja esteitä. Pelaajalle on saatavilla myös erikoiskykyjä, kuten sieni, joka muuttaa Marion isoksi ja antaa tälle voiman murtaa tiilipalikoita sekä tähti, joka tekee Mariosta kuolemattoman lyhyeksi aikaa. (Aloupis ym., 2015)

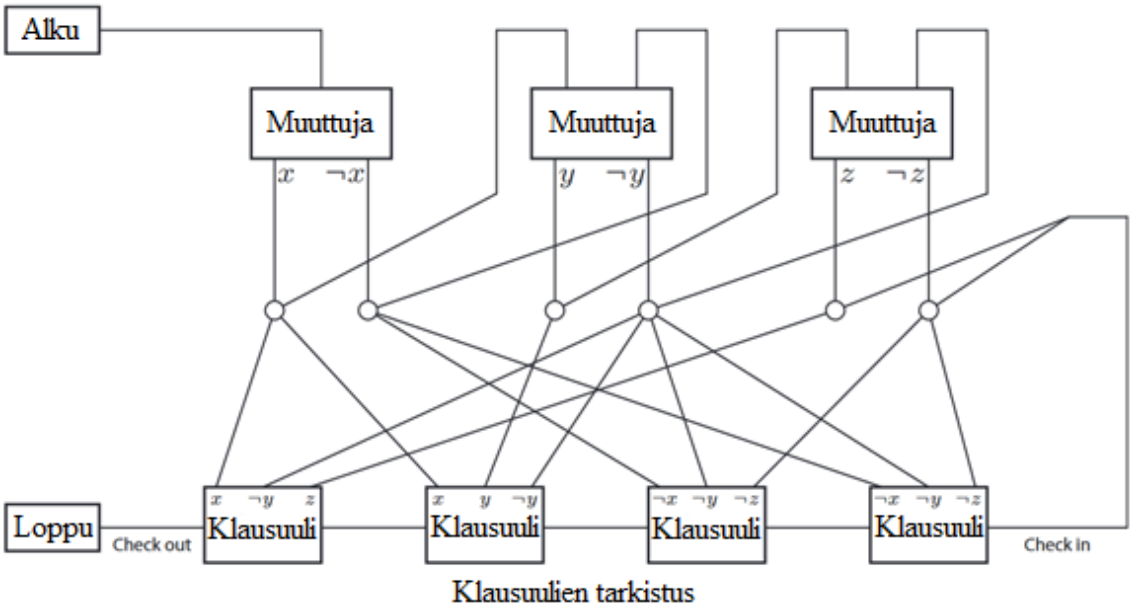
Ison ja pienen Marion erona on se, että iso Mario ei pysty menemään pienistä raoista, mutta pystyy murtamaan tiilipalikoita. Vastaavasti pieni Mario mahtuu menemään ahtaista raoista, mutta ei kykene murtamaan tiilipalikoita. Mario aloittaa pelin aina pienessä muodossa, ja mikäli koko kentässä ei ole kasvuvoimaa antavaa sientä, on turvallista olettaa, että mitään kentän palikoista ei pysty tuhoamaan. Isosta Mariosta voi muuttua pieneksi ottamalla vahinkoa vihollisilta, ja näin ollen, mikäli koko kentässä ei ole yhtäkään vihollista, voimme päätellä, että isosta Mariosta ei ole mahdollista muuttua pieneksi. (Aloupis ym., 2015)

Pelin kentät on rakennettu kolmen laatuisista palikoista; peruspalikat, jotka ovat kiinteitä, palkintopalikat, joita lyömällä Mario voi saada palkintoja sekä tiilipalikoita, jotka ovat ison Marion tuhottavissa. Pelin viholliset tekevät Mariolle vahinkoa osuessaan häneen, mutta

niiden päälle hyppääminen on Mariolle harmitonta ja toimii vihollisten päihityskeinona. Vihollisten lisäksi käytämme todistuksessa myös tuliheilureita, jotka ovat kiinteitä, pyöriviä esteitä, joihin osuessaan Mario saa vahinkoa, eikä tällä ole keinoja tuhota niitä. (Aloupis ym., 2015)

3.2.2 Tasohyppelypelien runko

Greg Aloupis, Erik D. Demaine, Alan Guo ja Giovanni Viglietta käyttävät artikkelissaan esiintyville peleille yleistä tasohyppelypelien runkoa, jolla 3-SAT -ongelma saadaan muunnettua tasohyppelypelien muotoon ja jonka perusteella todistukset toimivat. Runko on esitetty kuvassa 20, ja tähän muotoon soveltuessaan peli osoittaa sisältävänsä NP-kovan päätöstehtävän. Runko on johdettu ja vastaa luvussa 2 esitetyn 3-SAT -ongelman muotoa, jonka muunnos kohdepelin sisältämäksi päätösongelmaksi osoittaa tämän NP-kovuuden. (Aloupis ym., 2015)



Kuva 20: Yleinen runko 3-SAT -ongelman muunnokseen. Lähde: *Classic Nintendo Games are (Computationally) Hard*, suomennettu.

Rungossa esiintyville osille “Alku”, “Muuttuja”, “Klausuuli” ja “Loppu” on siis löydettävä pelistä omat vastineensa ja niiden väliset reitit vastaavat rungon rakennetta, jonka jälkeen voimme sanoa pelin noudattavan rungon logiikkaa. Toisin sanoen peli-instanssi vastaa 3-SAT -ongelman kyllä-instanssia siinä tapauksessa, kun kaikki tämän kaavan logiikka toteutuu peli-instanssin aikana onnistuneesti.

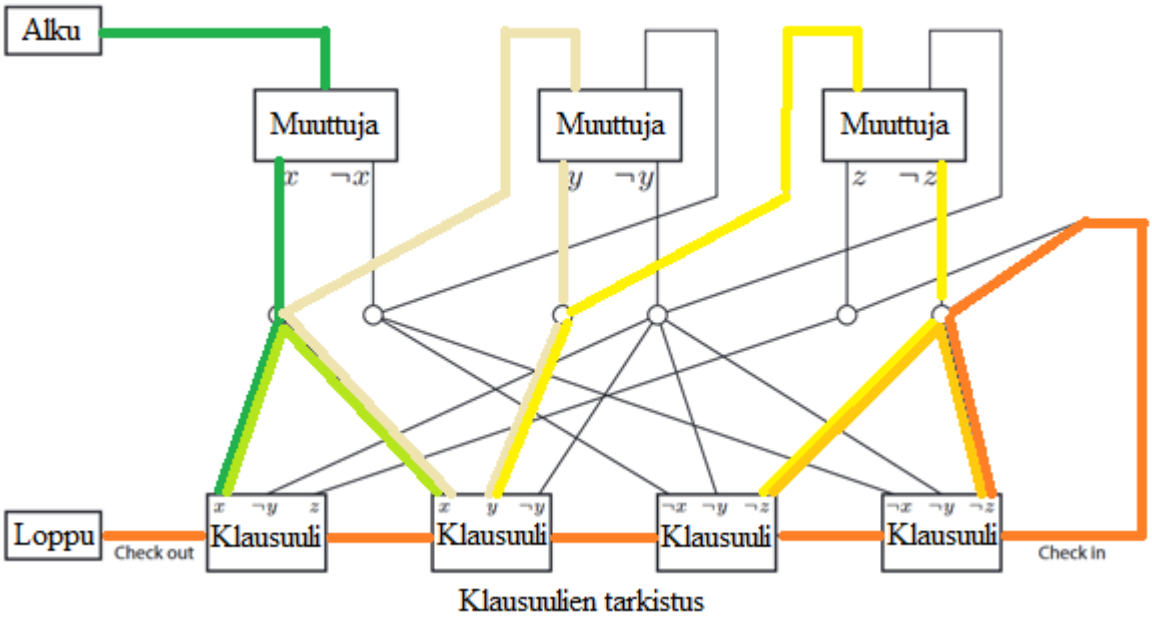
Alku-palikalla eli alkutilalla tarkoitamme sitä tilaa, mistä pelin tarkastelumme alkaa. Super Mario Bros. -pelissä tämä tarkoittaa hetkeä, jolloin kenttä alkaa, tai meidän laajennetussa tapauksessamme se hetki, josta lähdemme tarkastelemaan kentän läpäisymahdollisuutta. Käytännössä tässä vaiheessa meillä on esiluotuna kiinteän kokoinen kenttä ja Marion ohjaus on annettu pelaajan käsiin. Tämän hetken jälkeen kenttä muuttuu ainoastaan Marion toimesta.

Muuttuja-palikkana toimivat ne tasohyppelypelin ilmenemät, kun pelaajan on tehtävä valinta ja valittava kahdesta reitistä toinen. Valinnan tehtyään pelaaja ei enää pääse muuttamaan sitä saman pelikerran aikana, eli valinta on pysyvä. Kuhunkin muuttujaan voi päästä ainoastaan edellisen muuttujan kautta, ja niitä ei voi ohittaa tai jättää valitsematta arvoa.

Klausuuli on 3-SAT -ongelmasta tuttu joukko totuusarvoja eli literaaleja, jonka vastaavaa elementtiä pyritään hakemaan tasohyppelypelin instanssista. Kuten kuvasta 20 näemme, meille muodostuu alareunaan näistä klausuuleista yhtenäinen ketju, jonka läpi pääsee “Loppu”-palikkaan ainoastaan silloin, kun kaikki klausuulit ovat tyydyttyneitä.

Rungon suhteen on tärkeä huomata, että kaikki sen yhteydet on laadittu niin, että mistään kohdasta ei pääse palaamaan ja muuttamaan muuttujan arvoa toiseksi, mikäli se on kerran jo valittu. Itse pelissä yhteydet voivat ohittaa toisensa, mutta nämä ohituskohdat on laadittu niin, ettei niistä pääse vaihtamaan suuntaa keskenään. Klausuulissa käydessään pelaaja voi “vapauttaa” klausuulin pysyvästi, mutta ei pääse jatkamaan mitään muita klausuuliin johtaneita teitä pitkin.

Pelaaja kulkee siis polkua pitkin, käyden kaikki muuttujat läpi, jonka lopuksi jatkaa klausuuliketjuun tarkistaakseen, pääseekö “Loppu”-tilaan. Mikäli jossain klausuulissa ei olla käyty, se pysyy lukittuna, jolloin pelaaja ei voi saavuttaa lopputilaa, eli kenttä ei ole läpäistävissä. Esimerkkiketju pelin onnistuneesta läpäisystä ilmenee kuvassa 21. Kuvassa polut vaihtavat väriään pelaajan etenemisen myötä vihreästä punaiseen.



Kuva 21: Esimerkkipolku pelikentän läpi, vihreästä punaiseen.

Voimme huomata, että esimerkiksi muuttujien arvoilla x = TRUE (kaavassa x-reitti) ja $z = FALSE(kaavassa \neg z - reitti)$ voidaan avata useampi klausuuli, sillä pelissä on myös risteyksiä, joihin pystyy palaamaan ja valitsemaan toisen reitin. Muuttujaristeykset ovatkin tähän poikkeuksena niin, että literaalin valittuaan muuttujien valintakohtiin ei pysty palaamaan. Useamman kerran samassa klausuulissa käyminen ei vaikuta klausuulin arvoon, sillä se pitää avata vain kerran ja ylimääräinen “tien raivaus” ei enää vaikuta sen totuusarvoon.

Kuvan 21 tilanteessa pelaaja siis aloittaa Alkutilasta, valitsee muuttujan x literaaliksi x, jonka avulla pääsee aktivoimaan sekä klausuulin 1 että 2. Tämän jälkeen hän jatkaa tietään muuttujaan y, jonka valittu literaali ei oikeastaan tuota pelaajalle etenemistä klausuulien

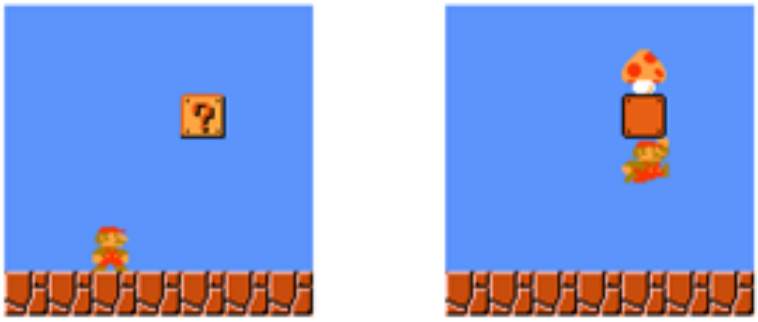
näkökulmasta, sillä klausuuli 2 on jo avattu literaalilla x. Mikäli pelaaja olisi tässä kohdassa valinnut literaalin $\neg y$, hän olisi sen avulla pystynyt avaamaan kaikki klausuulit kerralla. Z literaaliksi pelaaja valitsee kuitenkin $\neg z$, joka onneksi avaa hänelle jäljelle jääneet klausuulit 3 ja 4, joten loppuklausuulitarkistuksessa kaikki klausuulit ovat avattuina ja pelaaja pääsee onnistuneesti loppun. 3-SAT -ongelmaksi käännettynä tämä tarkoittaa, että literaalijoukko $\{x, y, \neg z\}$ toteuttavat klausuulirimpsun $(x \vee \neg y \vee z) \wedge (x \vee y \vee \neg y) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z)$.

3-SAT -ongelmien muiden instanssien kohdalle voidaan rakentaa vastaavanlainen kaava, vetämällä vain jokaisen muuttujan literaalia oikeaan kohtaan, vastaamaan annettujen klausuulien arvoja. Sekä muuttujia että klausuuleja voi lopullisessa instanssin muunnoksessa olla paljon enemmänkin, sillä 3-SAT -muoto rajautuu siihen, että jokaisessa klausuulissa on tasan kolme literaalia.

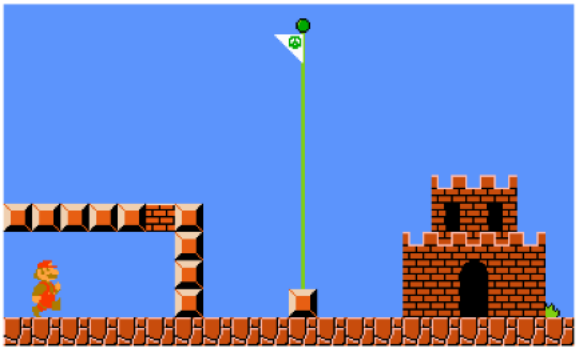
3.2.3 Super Mario Bros. muunnos rungon avulla

Nyt katsomme, kuinka saamme kuvassa 20 esitetyn rungon osat toteutettua Super Mario Bros. -pelissä, pitäen kaikki pelin säännöt alkuperäisinä ja muuttamalla ainoastaan kenttien rakennetta. Marion liikkeet, viholliset ja eri palikoiden ominaisuudet pysyvät siis ennallaan, ja ainoastaan kentän rakenne muuttuu mielivaltaiseksi, jotta todistuksemme NP-kovuudesta pätee.

Alkutilana käytämme tyhjää, kuvan 22 esittämää turvallista tilaa, jossa pelaajalla on saatavilla palkintopalikka, jonka sisällä on sieni. Sieni on tärkeä, sillä se muuttaa Marion ominaisuuksia ja määrittää tämän liikkeitä ja mahdollisuuksia. Voimme pakottaa pelaajan ottamaan sienen esimerkiksi laittamalla lopputilan tiilipalikoiden taakse kuten kuvassa 23, jolloin kenttä ei ole läpäistävissä ilman ison Marion tiilentuhousvoimaa.

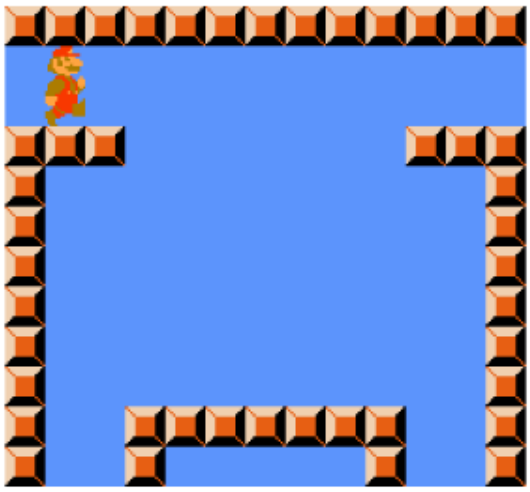


Kuva 22: Vasen: Super Mario Bros. -pelin alkutila. Oikea: palkintopalikka sisältää sienen, johon osumalla Mario kasvaa isoksi. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard*



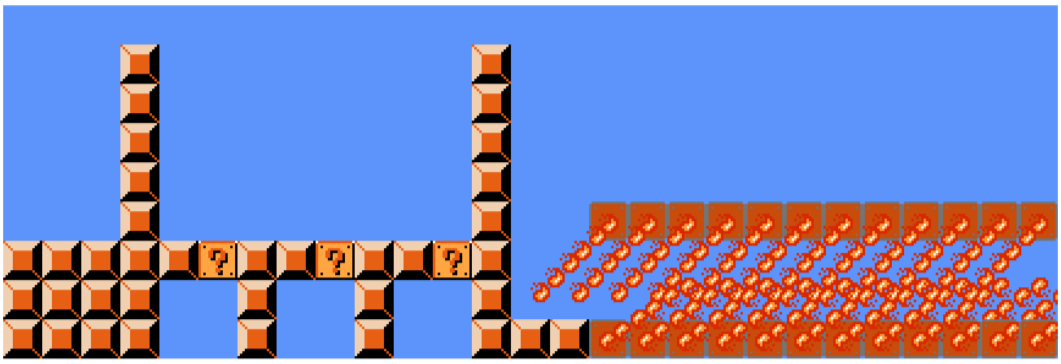
Kuva 23: Super Mario Bros. -pelin lopputila. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard*

Seuraavaksi simuloimme muuttujaristeyksen, jota havainnollistaa kuva 24. Tällä risteyksellä on kaksi erillistä sisäänkäyntiä (yläreitit), joista kukin vastaa edellisen muuttujan valittua arvoa. Jos siis kyseisen risteyksen muuttuja on x_i, sisäänkäynnit vastaavat literaaleja $x_{(i-1)}$ ja $\neg x_{(i-1)}$. Risteyksen korkeus on määritelty niin, että kun Mario hyppää alas, hän ei enää pääse takaisin ylös, eli jatkaminen toista literaalia pitkin on mahdotonta. Samoin risteyksellä on tasan kaksi uloskäyntiä, jotka vastaavat muuttujan x_i literaaleja x_i ja $\neg x_i$.



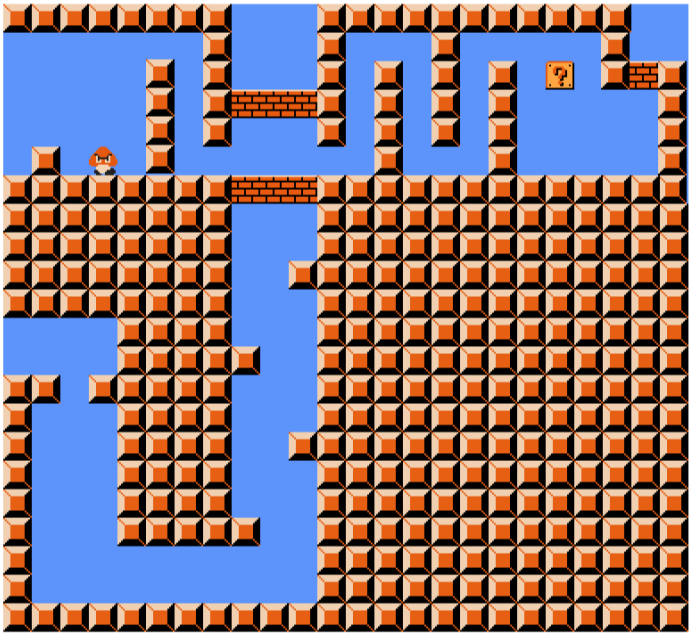
Kuva 24: Super Mario Bros. -pelin muuttujaristeys. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard*

Klausuuli voidaan esittää esimerkiksi kuvan 25 muodossa, jossa eteneminen on tuliheilureiden takia mahdotonta, mikäli yhtäkään kolmesta palkintopalikasta ei oltu aktivoitu. Kolmeen palkintopalikkaan pääsee käsiksi ainoastaan alareittejä pitkin ja ne vastaavat kunkin muuttujan haluttua arvoa tässä klausuulissa. Palkintopalikan aktivoiduttua sen yläpuolelle muodostuu tähti, joka pysyy aktiivisena, kunnes Mario hakee sen saapuessaan klausuuliin "tarkistuskierroksella" vasemmalta. Korkeilla reunapylväillä estetään tähteä karkaamasta omille teilleen sinä aikana, kun Mario palaa klausuuliin ja varmistutaan siitä, että se on sekä Marion ulottuvissa, että oikealla paikallaan klausuulin tarkistushetkellä. Täten simuloimme rungon klausuulin, joka ilman aktivointia estää Marion pääsyn lopputilaan, ja se on mahdollista aktivoida valitsemalla sopivat literaalit kentän aikana. Kuten huomataan, useampi tähti ei tässä tapauksessa vaikuta lopputulokseen, eli sama klausuuli voidaan aktivoida useamman kerran, muuttamatta tulosta.



Kuva 25: Super Mario Bros. -pelin klausuuli. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard*

Viimeisenä simuloimme risteyskohdan, joka näkyy kuvassa 26. Kuvassa on kaksi reittiä, yksi vasemmalta oikealle ja toinen alhaalta ylös. Tärkeintä tässä risteyksessä on estää pelaajaa vaihtamasta toiselle reitille, eli esimerkiksi alhaalta oikealle tai vasemmalta ylöspäin. Kuvan rakenteesta voimme kuitenkin huomata, että tämä mahdollisuus on poistettu, sillä vaakasuuntainen reitti vaatii matalien tunnelikohtien ansiosta Marion pientä muotoa, johon tämä voi muuttua osumalla vasemmalla olevaan viholliseen, jolle pääsy keskikohdasta on myöskin estetty matalalla tunnelikohdalla. Pystysuora reitti taas on tiilipalikoiden suojaama, joten siitä pääsee kulkemaan ainoastaan ison Marion muodossa.



Kuva 26: Super Mario Bros. -pelin risteyskohta. Kuvan lähde: *Classic Nintendo Games are (Computationally) Hard*

Näin ollen itse risteyskohdassa, eli kuvan keskellä kulkusuunta on riippuvainen Marion muodosta, eikä pelaaja voi muuttaa sitä siinä tilanteessa. Jotta Mario ei jää pieneksi risteyksen jälkeen kulkiessaan vasemmalta oikealle, vaan pystyy yhä läpäisemään lopputilan vaatiman tiilikohdan, oikealle uloskäynnille on sijoitettu uusi palkintopalikka, josta pelaaja saa uuden sienen ja voi muuttua takaisin isoksi Marioksi.

Nyt, kun olemme onnistuneet toistamaan kaikki rungon osat pelkästään pelin kenttärakennetta muuttamalla, olemme samalla luoneet 3-SAT -ongelman muunnoksen edellytykset tähän peliin. Mikäli luomamme kentät noudattavat rungon rakennetta ja sisältävät kaikki sen palaset, voimme muuntaa minkä tahansa 3-SAT -ongelman instanssin vastaavaksi Super Mario Bros. -pelin kentäksi. Näin kenttä on läpäistävissä ja vastaa kyllä-instanssia ainoastaan siinä tapauksessa, että sitä vastaava 3-SAT -instanssi on kyllä-instanssi ja mikäli kenttää ei saa läpäistyä, vastaavilla arvoilla saa myös 3-SAT ongelman ei-instanssin.

Todistus koskee Super Mario Bros. -peliä, jossa sen säännöt ja mekaniikat mahdollistavat NP-kovan päätösongelman luomisen. Todellisuudessahan kaikki alkuperäisen pelin sisältämät kentät ovat todistetusti läpäistävissä, sillä ne on rakentanut ja testannut ihan oikea ihminen ennen pelin julkaisua. Tässä vaiheessa on myös tärkeää huomata, että muunnos on tehtävissä polynomiajassa, sillä tehtävämme on vain generoida kenttä annettujen rungon osien ympärille ja se onnistuu syötteen avulla.

Generisoinnin yhteydessä laajennamme pelikenttää vastaamaan haluttua 3-SAT -instanssia, jolloin sen pitää todistuksen mielekkyyden nimissä olla miten suuri tahansa. Koska kyseessä on kuitenkin videopeli, jolloin näytön koko ei voi olla miten suuri tahansa, generisointi tapahtuu vain pelikentälle, ruudun pysyessä vakiona. Ruudun koon ollessa vakiona, Marion pääsy lopputilaan on tarkistettavissa polynomiajassa: tila-avaruus on polynomikokoinen, joten voimme käydä sen polynomiajassa läpi ja tarkistaa maalin pääsy. (Aloupis ym., 2015)

Tämän alaluvun päätehtävänä oli esitellä tasohyppelypelien runko ja tapa, miten 3-SAT -ongelma voidaan muuntaa Super Mario Bros. -pelin instanssiksi, mikä onnistuessaan voi toimia todisteena sen sisältämän päätösongelman NP-kovuudesta. Lisäesimerkkejä siitä, miten myös muut klassiset Nintendon pelit sopivat runkoon, voi lukea artikkelista ”*Classic*

Nintendo Games are (Computationally) Hard” ja lukija voi itsekin koittaa sovittaa tuntemiaan pelejä vastaamaan rungon osia. (Aloupis ym., 2015)

3.3 Sudoku

Sudoku on tällä hetkellä yksi maailman tunnetuimmista loogisista numeropeleistä. Sudoku pohjautuu sveitsiläisen matemaatikon 1700-luvulla kehittämään peliin nimeltä “Latin Squares”, joka sai nykyisen 9 x 9 ruudukon variaation 1970-luvulla New Yorkissa nimellä “Number Place” ja on sittemmin popularisoitu japanilaisen kustantajan toimesta 1980-luvulla nimellä “Sudoku”. (Smith 2005, Haythorpe 2016)

Yleisimmin näkemämme Sudoku on 9 x 9 ruudukko, joka on jaettu yhdeksään 9 ruudun neliöön, mutta se ei suinkaan ole ainoa pulmapelin variaatio. Sudokun sääntöjä voi vapaasti laajentaa pienemmille ja isommillekin pelikentille, jolloin sen pelikentän yleismuoto voi olla esimerkiksi muotoa $n \times n$ tai $n^2 \times n^2$. NP-kovuuden kannalta meidän on olennaista pitää kenttä rajattoman kokoisena, sillä mikäli ongelman instanssilla on kiinteä yläraja, ongelma voidaan ratkaista jopa lineaarisessa ajassa vain kirjoittamalla muistiin kaikki sopivat ratkaisuvaihtoehdot ja vertaamalla syötteitä tähän listaan. (Valmari 2020)

Ensimmäisen kerran Sudokun NP-kovuuteen viitattiin vuonna 2003 Yato Takayukin toimesta, joka työssään ”*Complexity and completeness of finding another solution and its application to puzzles*” käsitteli muiden aiheiden rinnalla Latin Squares -ongelman muuntamista Sudokuksi. (Takayuki 2003) Itse Latin Squaresin NP-täydellisyyden on osoittanut Charles J. Colbourn vuonna 1984. (Colbourn 1984) Takayukin työhön viitataan monessa Sudokun vaikeutta käsittelevissä töissä, mutta se on myös herättänyt tarpeita tarkentaa siinä esiintyviä todistuksia. Tähän päivään mennessä Sudokun vaikeutta on tutkittu jo monien artikkelien verran ja yritetty muunnosten avulla löytää siihen vaihtoehtoisia ratkaisutapoja. (Hoexum 2020)

Vuonna 2020 Eline Sophie Hoexum avasi työssään tarkemmin Yato Takayukin työtä, keskittämällä sen fokuksen koskemaan nimenomaan Sudokua ja tarkentamalla sen NP-täydellisyyden todistusta. (Hoexun 2020) Inês Lynce ja Joël Ouaknine esittivät työssään

"*Sudoku as a SAT Problem*", kuinka Sudoku saadaan muutettua SAT-ongelmaksi, vaikka käytännössä SAT-muodon klausuulien määrä kasvaa nopeasti järjettömän suureksi. (Lynce, Ouaknine 2006) Michael Haythorpe selvitteli generisoidun Sudoku-ongelman muunnosta Hamilton Cycleksi, joka on myös yksi tunnetuista NP-täydellisistä ongelmista. (Haytrophe 2016) Lisäksi Sudoku on muunnettu myös Constraint Satisfaction -ongelmaksi Helmut Simoniksen toimesta (Simonis 2005) ja Integer Programming -ongelmaksi Andrew C. Bartlettin, Timothy P. Chartierin, Amy N. Langvillen ja Timothy D. Rankinin työssä "An Integer Programming Model for the Sudoku Problem". (Barlett ym. 2008)

Emme käy tässä työssä läpi kaikkia näitä todistuksia, mutta lukija voi halutessaan perehtyä jokaiseen työhön tarkemmin lähdeviittausten kautta. Tärkeää on huomata, että generisoitu Sudoku on NP-täydellinen ongelma ja siihen kohdistuvat muunnokset voivat toimia konkreettisena esimerkkinä siitä, kuinka NP-täydelliselle ongelmalle voidaan etsiä hyvin moninaisia ratkaisualgoritmeja muista NP-kovista ongelmista. Muunnoksia Sudokusta eri NP-täydellisiksi ongelmiksi käytetään siis apuna uusien ratkaisualgoritmien soveltamisesta Sudokun ratkaisemiseksi, eikä niitä tule laskea todisteeksi Sudokun NP-kovuudesta. Ainoastaan muunnokset NP-kovasta ongelmasta Sudokuun ajavat tämän asian, joten lähdemateriaalia etsiessä kannattaa aina varmistua siitä, kumpaan suuntaan muunnos tehdään.

4 Pohdinta

Tämän työn ensisijainen tarkoitus oli tutkia ja avata NP-täydellisyyden käsitettä ja sen ilmentymistä peleissä. Tutkimuskysymyksenä oli siis: “Mikä on NP-täydellisyys ja miten se näkyy peleissä?” Luvun 2 tarkoitus oli vastata kysymyksen ensimmäiseen osaan ja luku 3 käsitteli jälkimmäistä. NP-kovien piirteiden näkeminen oli itselleni varsinkin alussa hyvin vaikeaa, joten gradun tarkoitus olikin oppia tämä taito ja auttaa työn avulla muitakin hahmottamaan pelien vaikeusluokkia paremmin, sekä tuoda koko tutkimusalaa parempaan tietoisuuteen, ainakin suomalaisten opiskelijoiden keskuudessa.

Työn etenemisen myötä tutkitusta materiaalista kävi yhä vahvemmin esiin se, että yllättävän suuri osa suosituiksi osoittautuneista peleistä mahdollistaa NP-kovien päätösongelmien muodostamisen sääntöjensä puitteissa. Vaikka peli normaalissa esiintymismuodossaan saattaisikin näyttää erittäin yksinkertaiselta, kuten esimerkiksi Tetris, pinnan alta asiaa selvittäessä suosion takaa paljastuukin yllättävä laskennallinen haastavuus. Harmikseni en ehtinyt tutkia tarpeeksi laajaa pelikirjoa tämän aiheen parissa, joten esimerkiksi se, *miten NP-kovuus vaikuttaa pelien mielekkyyteen tai suosioon*, on mielestäni erittäin mielenkiintoinen aihe jatkotutkimusten kannalta.

Huomiona tämän koetun mielekkyyden kannalta voimme nostaa myös sen, että P-vaikeusluokalle tunnusomainen kiinteän ratkaisualgoritmin löytäminen vie pelistä löytämisen ja ongelman ratkaisun ilon heti, kun tämän ratkaisun löytää. Peli muuttuu tämän jälkeen vain nopeustestiksi, eikä enää juuri tuota pelaajalleen haastetta: miksi edes etsiä uusia keinoja ratkaista ongelma, mikäli on jo tiedossa sopivan helppo tapa? Avainsanana tässä väitteessä on kuitenkin ”helppo”, sillä algoritmin pelkkä polynomiaikaisuus ei välttämättä tarkoita, että se olisi pelaajan helppo toistaa. Mikäli esimerkiksi mainitsemamme palindromin tarkistus olisi peli, on olemassa ihmisiä, jotka ovat harjoitelleet tätä taitoa hyvin pitkälle ja voivat jopa puhua sujuvasti takaperin, jolloin palindromin tarkistusalgoritmi ei tuota heille enää peliksi luokiteltua haastetta.

NP-luokan ongelmat ovatkin ehkä mielekkäitä juuri siksi, että ratkaisun löytäminen on suhteellisen vaikeaa, mutta sen jälkeen on helppo tarkistaa, onnistuiko siinä vai ei. Mikäli myös onnistumisen tarkastus on työlästä ja aikaa vievää, se saattaa jopa vähentää pelin

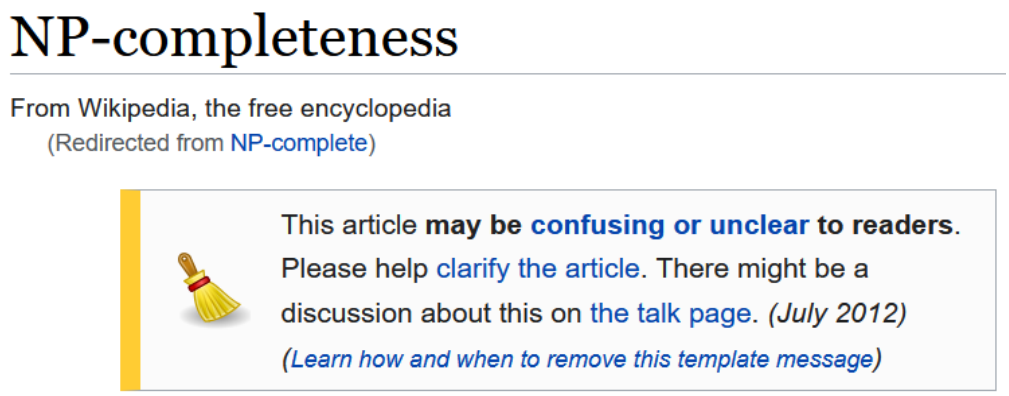
mielekkyyttä, eli ongelman vaikeusasteen nostaminen ei enää paranna peliä, vaan saattaa jopa karsia pelaajakuntaa. Tämän valossa laskisin NP-luokan todennäköisesti sopivimmaksi suosittujen pelien vaikeusluokaksi keskimääräisen pelaajan kannalta, joskin kuten aiemmin mainitsin, tämän väitteen tueksi tarvittaisiin jatkotutkimusta.

Myös erot yksin- ja moninpeleissä tuottavat takuulla omaa maustetta koettuun mielekkyyteen ja pelien vaikeusluokkajakoon, kun pelin tulos ei enää riipu vain yhden toimijan ratkaisusta tai ratkaisualgoritmista, vaan se joutuu työskentelemään alati muuttuvaa pelitilannetta vastaan, vastustajan tehdessä jokaisella askeleella omia muutoksiaan. Tämä yksin- ja moninpelijaon tutkiminen pelien mielekkyyden ja vaikeusluokkien suhteen on toinen mielenkiintoinen kysymys, johon suosittelisin jatkotutkimusta. Itse pelaajana olen aina suosinut nimenomaan yksinpelejä, joten olen tähänkin työhön suosinut esimerkkeinä nimenomaan yksinpelejä. Kuten tämän työn puitteissa parissa kohtaa mainittiinkin, kaksin pelattavat pelit myös tuppaavat esiintymään paljon NP-luokkaa vaativimmissa vaikeusluokissa, joihin ei tässä työssä paneuduttu enempää.

Tämän työn aloittaessani minulla oli vain hyvin pintapuolinen käsitys NP-täydellisyydestä, enkä oikein osannut varautua sen syvyyteen. Varsinkin itse määritelmää ja P=NP ongelmaa koskeva kirjallisuus oli varsin raskasta luettavaa opiskelijalle, joka ei ole erikoistunut matemaattisiin todistuksiin ja vaihtoehtoisia lähteitä aiheeseen tuntui olevan niukasti sekä kyseenalaisella luotettavuudella akateemisella mittapuulla. Suuri osa akateemisesti valideista lähteistä on myös ajoitettu 80-luvulle ja sen saavutettavuus digitaalisessa muodossa oli ongelmallista. Tämä on tehnyt aiheeseen perehtymisestä tarpeettoman vaikeaa, vaikka idean sisäistäessä itse asia ei enää tunnu niin monimutkaiselta.

Näin työn kirjoittamisen jälkeen olen edelleen sitä mieltä, että aiheesta saatavilla oleva materiaali tekee siitä paljon vaikeamman oloisen, kuin mitä sen pitäisi olla. Nykyaikaisen tutkimuksen ongelmana näen sen, että kaikki ”helpommassa muodossa” esitetty moderni materiaali ei enää vastaa akateemisia standardeja, vaan tyydytään siihen, että aiheesta voi lukea Wikipediasta tai selittää se YouTube-videolla. Huvittavana välihuomautuksena haluaisin nostaa esiin aiheen selkeyttämisen tarvetta, sillä jopa Wikipedian mittapuulla aihe

on nähtävästi koettu tarpeettoman hankalaksi sen tarpeeksi selkeään muotoon saamiseksi, kuten voimme nähdä kuvassa 27.



Kuva 27: Wikipedian mielipide NP-täydellisyydestä. Kuvakaappaus otettu 19.4.2022

Nykyaikana, kun ohjelmointi ja varsinkin pelit ovat paljon suuremman käyttäjäkunnan käsissä, aihe todennäköisesti kiinnostaa muitakin, kuin puhtaita matemaatikoita. Mikäli vaikeusluokat ja NP-täydellisyys esiintyisivät yleisemmin ja nykyaikaisemmassa, mutta luotettavassa muodossa, voisimme saada paljon enemmän uusia, mielenkiintoisia peliongelmia ja tutkimusta P=NP saralle. Tällä hetkellä juuri se pohja-askel tuntuu puuttuvan, joka estää suurempaa yleisöä käymästä aiheen kimppuun.

En toki väitä, että pelien ja NP-täydellisyyden tutkiminen olisi mitenkään pysähtynyt tai vähäistä. Työn ohella näin suuren määrän tiedeartikkeleita, joista välittyi sellainen tunne, että melkein kaikkien pelien osalta joku on jo tutkinut sen vaikeusluokkaa. Kaikki nämä artikkelit ovat kuitenkin jo soveltavia artikkeleita, joissa esitetään pelin muunnosta "jo tunnettuun" NP-kovaan ongelmaan, ilman sen tarkempaa käsitteiden tai metodiikan selittämistä. Näin ollen, jotta niistä olisi lukijalle mitään tolkkua, tämän olisi jo valmiiksi tiedettävä tarpeeksi hyvin NP-täydellisyyden erikoispiirteet, jotta ylipäätään ymmärtäisi, mitä työssä tehdään, ja miksi se on tai ei ole pätevä todistus. Omasta kokemuksesta voin suoraan sanoa, että työn alkumetreillä en saanut näistä artikkeleista juuri mitään irti, mutta loppua kohti ne olivat jo suhteellisen ymmärrettävää luettavaa. Mikäli aihe on siis jo

ennestään tuttu, voi olla vaikeaa nähdä tai ymmärtää, miten se voisi olla vaikealukuista tai -selkoista muille, kun omasta mielestä teksti on ihan selkeää ja johdonmukaista.

Loppupäätelmäni tähän onkin juuri se, että se ensimmäinen askel aiheen pariin on tarpeettoman hankala ja olisi hyvä, jos siihen tulisi muutos. Mikäli sen askeleen onnistuu taistelemaan läpi ja kiipeämään yli, asiat käyvät heti paljon selkeämmäksi. Toivonkin siis, että graduni olisi avuksi tähän ongelmaan ja kohderyhmänäni ovatkin päätoimisesti opiskelijat, joille tämä ensimmäinen askel tuntuu liian suurelta. Pyrin gradussani tarkoituksella käyttämään mahdollisimman paljon sanallisia ilmaisuja ja suomennoksia, joiden avulla voi saada itse ideasta kiinni. Toimikoon se pohjatietona auttamaan tulkitsemaan muita lähteitä, jotka ovat melkein kokonaisuudessaan englanniksi ja täynnä matemaattisia kaavoja.

Koska kuitenkin olen itsekin vasta opiskelija, suosittelen lukijoita ristitarkistamaan tämän gradun puitteissa esitettyjä väitteitä ja määritelmiä tekemieni huolellisuusvirheiden varalta. Saatavilla on paljon tarkemmin vuosien saatossa tarkistettuja lähteitä, jotka osaavat selittää asiat paljon yksityiskohtaisemmalla tarkkuudella ja asiantuntijuudella, kuin mihin itse olen tämän gradun teon aikana päässyt.

Koska gradun sivumäärä alkoi tulla täyteen ja mielestäni sen puitteissa esitetyt esimerkit olivat riittäviä välittämään sen idean, olemme rajoittuneet vain muutamaan peliesimerkin läpikäyntiin. Olisin mielelläni avannut niitä lisääkin, mutta se ei olisi enää tuonut työlle lisäarvoa, sillä sen tarkoitus on toimia ensiaskeleena aiheen pariin. Suosittelenkin lukijoita jatkamaan aiheesta lukemista itsenäisesti, sillä jo näillä tiedoilla pitäisi olla huomattavasti mielekkäämpää vain kirjoittaa internettiin “NP-Complete games” ja selailla kymmenien artikkeleiden valikoimaa suhteellisella varmuudella siitä, että niistä ymmärtääkin jotain.

Mikäli kuitenkin tuntuu, että tavoite ei ole saavutettu, voi aloittaa tässä työssä käsiteltyjen lähdeartikkeleiden lukemisella, sillä silloin tietää jo suunnilleen, mistä on kyse, voi yhdistää termit englanninkielisiin vastineisiinsa ja nähdä todistusten laajennoksia kaikkine yksityiskohtineen. Suosittelen kuitenkin pitäytymään tiedeartikkeleissa aiheen käsittelyn kanssa, sillä rinnalla on myös suuri määrä villiä, perustelematonta keskustelupalstaväittelyä siitä, kuuluuko jokin peli tiettyyn vaikeusluokkaan vai ei. Vaikka pitkät todistukset

saattavatkin tuntua työläältä lukea, niistä saa lopulta paljon paremman käsityksen, kuin keskustelupalstojen typistetyt väitteet.

Loppupäätelmänä tehdystä työstä haluaisin toistaa sen, että teoreettisena työnä aihe oli haastava, aiheen itseopiskelusta, saatavilla olevasta materiaalista ja sen selitysten muodosta johtuen. Teorialähteet olivat yllättävän vaikealukuisia itseopiskelun näkökulmasta, ilman varsinaisia alternatiiveja. Selityksiä on haettu erittäin syvältä teoreettiselta kannalta, lähtien liikkeelle Turingin koneista ja kielten määritelmistä, joista aloin itsekin tämän gradun teoriaosuutta kirjoittamaan, kunnes päädyin jättämään koko aiheen pois ja keskittämään idean välittämisen yleisemmällä tasolla, ns. helpommin ymmärrettäviä määritelmiä käyttäen. Näiden lähteiden keskuudessa varsinkin P=NP osio on käsitelty hyvin vaihtelevasti ja tyypillisesti vain toisiinsa viitaten. Se on kuitenkin ymmärrettävää, sillä kyseessä on suuri, hyvin pitkään auki oleva matemaattinen kysymys, johon jokaisella asiaansa perehtyneellä matemaatikolla on takuulla sanottavaa. Suosittelisin siis aloittamaan aiheen parissa jonkun kurssin tai opettajan ohjauksen alla, jotta määritelmät eivät mene heti alussa sekaisin.

Päästyäni viimein yli teoriaosuudesta vastassani taas oli aivan päinvastainen ongelma: pelien NP-todistuksia koskevia artikkeleita oli valtava määrä ja oli vaikea valita työhön ne pelit, joita lähtisi avaamaan. Kaikkia ei kuitenkaan voinut ottaa mukaan ja välillä pohdinkin työn fokuksen vaihtamista listaukseksi näistä artikkeleista, eli työ olisi toiminut kokoelmana pelitutkimuksia ja muuttanut muotoaan systemaattiseksi kirjallisuuskatsaukseksi, mutta päätin pidättäytyä ideasta. Mielestäni se ei olisi ajanut työn tarkoitusta tai ollut kenellekään tarpeeksi hyödyllinen, sillä mielestäni suurin ongelma aiheen suhteen on nimenomaan se ensimmäinen askel asian ymmärtämiseksi, eikä niinkään vaikeus löytää pelitutkimus-artikkeleita. Työn fokus pysyi siis parin valitun esimerkin avaamisessa ja aiheen tutuksi tekemisessä lukijoille, joille NP-kovuus ei ole vielä kovin tuttu aihe.

5 Yhteenveto

Olemme tämän työn puitteissa käsitelleet vaikeusluokkien eroja, keskittyen NP-täydellisyyteen. Tavoitteena on ollut käsitteen avaaminen helpommin ymmärrettävään muotoon ja sen ilmentymien esiin tuominen esimerkkien avulla. Peliesimerkkien osalta olemme esitelleet muutaman erilaisen pelin ja lähestymistavan, jotta NP-täydellisyyden ilmentymä tulee mahdollisimman monipuolisesti esiin.

Tetriksen kohdalla käsittelimme 3-partitionin muuntamista Tetriksen muotoon muuttamalla pelikenttää, mutta pitämällä pelin yleiset säännöt alkuperäisinä. Näimme perusteet siitä, miten pelin sääntöjä noudattaen voimme luoda tilanteen, joka on suoraan muunnettavissa tunnetusta NP-täydellisestä ongelmasta. Työn puitteissa esiin tuodun todistuksen lisäksi on annettu viitteet laajemmille todistuksille, jotka koskevat pelin muita osia, jotta lukija voi halutessaan perehtyä tarkemmin, miten samoja todistuskeinoja voi laajentaa todistamaan muita pelin mahdollistamia päätösongelmia.

Luvun 3.2 Super Mario Bros. -esimerkin yhteydessä on myös esitetty tasohyppelypeleille soveltuva runko, joka toimii apukeinona 3-SAT -instanssin muuntamisessa tasohyppely-pelien instansseiksi. Tämä runko soveltuu NP-kovuuden todistukseen, mikäli pelin sääntöjä ei muunnoksen yhteydessä muuteta, vaan ainoa muuttuva tekijä on pelikentän koko ja rakenne. Samaa runkoa voi siis käyttää muihinkin peleihin, ja mikäli onnistuu pelin sääntöjä noudattamalla rakentamaan sen instanssiin runkoa vastaavat palikat, tulee samalla todistaneeksi kyseisen peli-instanssin mahdollistaman päätösongelman NP-kovuuuden.

Sudokun esimerkissä näimme, että NP-täydelliselle pelille on olemassa monia eri ratkaisuvaihtoehtoja, jotka voi saavuttaa muuntamalla pelin instanssi erilaisiksi NP-täydellisiksi ongelmiksi ja käyttää näiden olemassa olevia ratkaisualgoritmeja pohjana pelin ratkaisulle. Sudoku on tämän suhteen hyvin suosittu kohde, ja siksi sitä onkin muunneltu moneksi eri ongelmaksi.

Samoin käsittelemämme tasohyppelyrunko ei ole ainoa laatuaan, vaan siitäkin on variaatioita. Esimerkiksi Erik D. Demaine, Martin L. Demaine ja Joseph O'Rourke esittävät

työssään "*PushPush and Push-1 are NP-hard in 2D*" kyseisten pelien NP-kovuuden vastaavan rungon avulla. (Demaine, Demaine ja O'Rourke 2000)

NP-kovien ongelmien muunnos peli-instansseiksi saattaa esimerkkien ja varsinkin luvussa 3.2 esitetyn selkeän runkorakenteen käsittelyn pohjalta tuntua yksinkertaiselta, mutta se on mielestäni sellainen aihe, mihin on vaikea lähteä suoriltaan. Yhteyttä pelin näkyvällä osalla ja äärimmilleen pelkistetyillä matemaattisilla ongelmilla on vaikea nähdä, ellei ole ensin nähnyt muutaman esimerkin todistusten tyylistä. Tämän työn tavoitteena onkin toimia siltana niille, jotka haluavat perehtyä aiheeseen tarkemmin ja oppia itse sekä näkemään NP-kovuuden piirteitä peleissä, että todistamaan sitä.

P=NP -ongelman ollessa edelleen ratkaisematta, uusien NP-kovien ongelmien löytyminen on suosittua ja hyödyllistä, sillä mikäli ennen pitkää löytyy yksikin NP-kova ongelma, johon on tiedossa helppo ratkaisu, P=NP ongelma voi ratketa. Toivottavasti tämä työ antaa hyvät alkukohdat ja inspiraatiota lukijoilleen syventyä aiheeseen ja jatkaa tutkimusta sen osalta.

Muut luvussa 4 esiin tuodut jatkotutkimusongelmat koskevat vaikeusluokkien vaikutusta pelien suosioon, jossa kartotettaisiin tarkemmin, mitä vaikeusluokia esiintyy eniten suosituissa sekä epäsuosituissa peleissä. Jatkotutkimuksena samalle aiheelle voi olla myös tarkempi analyysi siitä, miten kovasti tämä vaikeusluokka vaikuttaa pelin koettuun mielekkyyteen, eli onko jokin hyvin vaikea peli samalla yleisesti epäsuosittu, mutta vaikeista ongelmista pitävien pelaajien mielestä äärettömän mielekäs ja vetoavatko eri vaikeusluokan pelit erilaiseen pelaajakuntaan ylipäätään. Edustavatko esimerkiksi klassiset lasten pelit enimmäkseen helpompia vaikeusluokkia ja ovatko vaikeiden vaikeusluokkien pelit pääosin älypelejä vai jotain muuta genreä?

Toinen luvussa 4 esitetty jatkotutkimuskysymys koski yksin-, kaksin- ja moninpeliaspektin vaikutusta sekä vaikeusluokkaan, että pelin koettuun mielekkyyteen. Alustavan käsitykseni mukaan toisen pelaajan mukaan ottaminen automaattisesti nostaisi pelin vaikeustasoa, sillä pelin kenttä ja asetelma ei enää ole kiinteänä alkusyötteessä koko pelin ajan, vaan vastapelaaja muokkaa sitä joka vuorollaan uuden näköiseksi. Näin ollen ratkaisualgoritmin pitäisi jo alussa varautua paljon suurempaan määrään vaihtoehtoisia tilanteita, kuin kiinteän pelikentän kohdalla.

Koska kyseessä on kumminkin toinen pelaaja, laskuihin on otettava myös tämän mukanaan tuomat sosiaaliset aspektit. Mikäli kaksinpelattava peli koetaan vaikeudestaan huolimatta mielekkääksi, pysyykö tämä mielekkyys samalla tasolla, kun tämä sosiaalinen aspekti poistetaan, eli tilalle laitetaan botti? Sosiaalinen aspekti voi vaikuttaa hyvinkin suuresti siihen, olisiko peli ilman sitä koettu mielekkääksi vai ei.

Tämän työn puitteissa olemme kuitenkin käsitelleet ainoastaan yksinpelejä, vaikka esimerkiksi Tetris on lukuisilla variaatioillaan tarjonnut myös moninpelimahdollisuuksia vuosien saatossa. Emme ole kuitenkaan keskittyneet näihin variaatioihin, vaan rajasimme aiheen sekä peliasetelman tutkimaan nimenomaan klassisen asetelman NP-täydellisyyttä.

Voimme siis nähdä, että aiheen parissa on suuri määrä toistaiseksi avoimia tutkimuskysymyksiä, joiden parista monella lukijalla voisi löytyä itselle mielekäs gradun aihe. NP-täydellisyyden teoriaosuus saattaa tuntua monelle kuitenkin haastavalta, mutta mikäli tälle saralle saadaan ennen pitkää riittävästi sopivan tasoista tukimateriaalia, se ei enää tuntuisi niin uhkaavalta.

6 Lähteet

Aloupis, Greg, Erik D. Demaine, Alan Guo ja Giovanni Viglietta. 2015. “Classic Nintendo Games are (Computationally) Hard”. *Theoretical Computer Science 586,* 135-160.

Appel, Kenneth ja Wolfgang Haken. 1977. “The Solution of the Four-Color-Map Problem.” *Scientific American* 237, no. 4: 108–21

Bartlett, Andrew C., Timothy P. Chartier, Amy N. Langville ja Timothy D. Rankin. 2008. “An Integer Programming Model for the Sudoku Problem.” *Journal of Online Mathematics and its Applications* 8.1

Breukelaar, Ron, Hendrik Jan Hoogeboom ja Walter A. Kosters. 2003. “Tetris is Hard, Made Easy.” *Leiden Institute of Advanced Computer Science*. Universiteit Leiden.

Colbourn, Charles J. 1984. "The complexity of completing partial latin squares." *Discrete Applied Mathematics* 8.1: 25-30.

Cook, Stephen A. 1971. “The Complexity of theorem – proving procedures.” *Proceedings of the third annual ACM symposium on Theory of computing* 151-158.

Dailey, David P. 1980. “Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete.” *Discrete Mathematics* 30.3: 289-293.

Dalgety, James. 2017. "The Icosian Game." *The Puzzle Museum*. Lainattu 09.05.2022. http://puzzlemuseum.com/month/picm02/200207icosian.htm.

Davis, Martin D. ja Elaine J. Weyuker. 1983. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science.* New York: Academic Press.

Demaine, Erik D., Susan Hohenberger ja David Liben-Nowell. 2003. “Tetris is Hard, Even to Approximate.” *International Computing and Combinatorics Conference*. Springer, Berlin, Heidelberg.

Demaine, Erik D., Martin L. Demaine ja Joseph O'Rourke. 2000. "PushPush and Push-1 are NP-hard in 2D." *arXiv preprint cs/0007021*

Eppstein, David. *Computational complexity of games and puzzles*. Luentomateriaali. Lainattu: 12.11.2021. https://www.ics.uci.edu/~eppstein/cgt/hard.html.

Fortnow, Lance. 2013. *Kultainen pääsylippu*: *P, NP ja mahdottoman tavoittelu,* suom. Kimmo Pietiläinen. Helsinki: Hakapaino Oy.

Fraenkel, Aviezri S. ja David Lichtenstein. 1981. "Computing a perfect strategy for n x n chess requires time exponential in n." *International Colloquium on Automata, Languages, and Programming*. Springer, Berlin, Heidelberg.

Garey, Michael R. ja David S. Johnson. 1979. *Computers and Intractability: A guide to the theory of NP-Completeness.* A Series of Books in the Mathematical Sciences. San Francisco, Calif.: W. H. Freeman and Co.

Goldreich, Oded. 2010. *P, NP, and NP-completeness: the basics of computational complexity.* Cambridge University Press.

Haythorpe, Michael. 2016. "Reducing the generalised Sudoku problem to the Hamiltonian cycle problem." *AKCE International Journal of Graphs and Combinatorics* 13.3: 272-282.

Hoexum, Eline. 2020. "Revisiting the proof of the complexity of the sudoku puzzle." Bachelor's Thesis, University of Groningen. https://fse.studenttheses.ub.rug.nl/ 22745/1/ bMATH_2020_HoexumES.pdf.pdf

Karp, Richard Manning. 1972. "Reducibility among combinatorial problems." *Complexity of Computer Computations*. 85-103. Springer, Boston, MA.

Klee, Viktor ja George Minty. 1972. "How Good is the Simplex Algorithm." *Inequalities* 3.3: 159-175.

Kleinberg, Jon ja Éva Tardos. 2006. *Algorihm Design*. Sivut 464, 495-496. ISBN 0-321-37291-3. lainattu 18.4.2022

Ladner, Richard E. 1975. "On the structure of polynomial time reducibility". *Journal of the ACM (JACM)* 22, 155-171.

Lynce, Inês ja Joël Ouaknine. 2006. "Sudoku as a SAT Problem." *ISAIM* 11.1: 6-13.

Martiny, T. Ian. 2012. *Complexity Theory and the 3-Satisfiability Problem*. https://ianmartiny.us/interests/3SAT.pdf

Robson, John Michael. 1984. "N by N Checkers is Exptime Complete." *SIAM Journal on Computing* 13.2: 252-267.

Savitch, Walter J. 1970. "Relationships between nondeterministic and deterministic tape complexities." *Journal of computer and system sciences* 4.2: 177-192.

Simonis, Helmut. 2005. "Sudoku as a Constraint Problem." *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*. Vol. 12. Citeseer.

Smith, David (15.5.2005). "So you thought Sudoku came from the Land of the Rising Sun …" *The Observer*. Lainattu 13.03.2022.

Spielman, Daniel A. ja Shang-Hua Teng. 2004. "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time". J. ACM 51(3): 385-463

Stockmeyer, Larry. "Planar 3-colorability is polynomial complete." *ACM Sigact News*, 5.3, 19-25

Takayuki, Yato. 2003. "Complexity and completeness of finding another solution and its application to puzzles." Master's Thesis, University of Tokyo.

Valmari, Antti. 2020. *Diskreetit rakenteet*. Luentomateriaali, Jyväskylän Yliopisto. Lainattu 02.02.2022. http://users.jyu.fi/~ava/TIEP1020.pdf

Viglietta, Giovanni. 2020. *The Complexity of Video Games.* Luentomateriaali, lainattu 21.03.2022. http://www.jaist.ac.jp/~ogata/lecture/i628e/lesson11.pdf

Kuvien lähteet:

Bark and Fester LLC. 2022. “RoTopo”. Kuvakaappaus online-pelistä, otettu 03.04.2022. https://rotopo.com/

Good Thinking Games Ltd. 2022. “The Hamiltonian Circuit”. Pelin promokuva, lainattu 03.04.2022. www.goodthinkinggames.com/the-hamiltonian-circuit/