MC-pelien ratkaisualgoritmeista

Timo Poranen

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Pro gradu -tutkielma
16.2.1999

Tampereen yliopisto
Tietojenkäsittelyopin laitos
Timo Poranen: MC-pelien ratkaisualgoritmeista
Pro gradu -tutkielma, 48 sivua
Helmikuu 1999

Tiivistelmä

Tämä työ perustuu pääosin Colin Stirlingin artikkeleihin *Bisimulation and Modal Logic with Fixed Points* ja *Bisimulation, Model Checking and other Games*. Molemmissa julkaisuissa käsitellään MC-pelejä, joilla mallinnetaan samanaikaisten prosessien toimintaa ja niiden välistä kommunikointia.

Ensiksi käsitellään perusteellisesti Stirlingin todistus sille, että jokaisella MC-pelillä on olemassa yksikäsitteinen voittaja. Stirlingin tulos perustuu induktiotodistukseen, joka samalla konstruoi eksponentiaalisen ratkaisualgoritmin MC-pelille ja polynomisen ratkaisun eräälle erikoistapaukselle.

Työn loppuosassa esitetään polynomisessa ajassa ratkeavia MC-pelin erikoistapauksia ja niiden ratkaisemiseen sovellettavia algoritmeja. Näitä MC-pelin erikoistapauksia ei tiettävästi ole aiemmin käsitelty kirjallisuudessa. Polynomisessa ajassa ratkeaviksi MC-pelin erikoistapauksiksi todistetaan puumainen ja syklikiinnitetty MC-peli. Puumainen MC-peli ei sisällä muita silmukoita kuin itsesilmukoita ja syklikiinnitetyn MC-pelin jokaisella syklillä on vähintään yksi yhteinen särmä. Näille kahdelle erikoistapaukselle esitetään ajassa $O(E)$ toimiva ratkaisualgoritmi. Lisäksi esitetään muutama erikoistilanne, joiden ratkeavuus ja löytyminen pelistä on riippuvainen syvyyssuuntaisesta etsinnästä.

Lopuksi ennen yleisen ratkaisualgoritmin ominaisuuksien pohdintaa esitetään tulos, joka antaa ylärajan MC-pelin eri ratkaisujoukkojen lukumäärälle. Ratkaisujoukkojen määrä kertoo niiden alipelien lukumäärän, joilla voi olla eri voittaja.

Useimmat MC-pelien erikoistapauksia ratkovat polynomiset algoritmit perustuvat syvyyssuuntaiseen etsintään, kun taas Stirlingin esittämät todistukset perustuvat pakotusjoukkojen laskentaan. Tässä työssä tarkastellaan myös näiden kahden selkeästi toisistaan eroavan menetelmän ominaisuuksia ja soveltamista MC-pelien erikoistapauksille.

Avoimeksi kysymykseksi jää edelleen, että onko MC-peleille olemassa yleisessä tilanteessa toimivaa polynomista ratkaisualgoritmia.

Sisällys

Algoritmit

Kuvat

Kiitokset

Haluan kiittää professori Jyrki Nummenmaata ohjauksesta ja lukuisista keskusteluhetkistä, joita olemme tähän työhön liittyvistä asioista käyneet. Professori Nummenmaalle kuuluu myös kiitos neljännessä luvussa käsitellyistä puumaisista MC-peleistä, joihin liittyvien apulauseiden periaatteet ovat hänen käsialaansa. Minä olen vain kirjoittanut ne formaaliin muotoon.

Kaikki tämän työn kuvat on toteutettu *XFigure*-ohjelmalla. *JSci-A science API for Java*^{TM}-ohjelmointirajapinnasta oli huomattava apu MC-peleihin liittyvien ohjelmien ja algoritmien toteutuksessa. Haluan myös esittää kiitokset henkilöille, joiden ansiosta olen voinut edellämainittuja ohjelmia käyttää.

1 Johdanto

MC-pelin (model checking game) [18] tarkoitus on mallintaa graafiteoreettisesti äärellistilaisia prosesseja ja sitä, onko jollain prosessilla jokin määrätty ominaisuus vai ei. Mikäli MC-pelin voittaja pystytään ratkaisemaan polynomisessa ajassa, pystytään myös ratkaisemaan monia äärellistilaisiin prosesseihin sekä niiden mallintamiseen liittyviä ongelmia polynomisesti.

MC-peli on nimensä mukaisesti peli, jossa on kaksi pelaajaa. Pelin tiloja (tilanteita) kuvataan solmuilla ja siirtymiä tilojen välillä särmillä. Solmut on jaoteltu pelaajan I solmuiksi ja pelaajan II solmuiksi. Jokaisesta pelin tilasta on olemassa ainakin yksi siirtomahdollisuus (solmusta lähtee aina vähintään yksi särmä). Pelin voittaa se pelaaja, joka pystyy pitämään pelin määrätyt ominaisuudet täyttävässä tilassa (tai tiloissa).

MC-pelejä on käsitelty kirjallisuudessa erittäin vähän, sillä päälähteenä käyttämäni Stirlingin artikkeli on julkaistu vasta vuonna 1997. Tätä ennen tutkimuksen kohteena ovat olleet yksinkertaiset stokastiset pelit (SSG), jotka esiteltiin vuonna 1992 [1]. Mark Jerrum [18] on osoittanut jälkeenpäin, että jokainen MC-peli voidaan muuttaa SSG:ksi polynomisessa ajassa. SSG:lle on olemassa ratkaisualgoritmi, jonka aikavaatimus on $2^{O(\sqrt{n})}$. Yksinkertaisia stokastisia pelejä käsitellään pääpiirteittäin seuraavassa luvussa.

Tässä työssä MC-pelejä tarkastellaan pelkästään graafiteoreettisena ongelmana. MC-pelien sovelluksista kiinnostuneen lukijan kannattaa tutustua tarkemmin Stirlingin artikkeleihin [17] ja [18] sekä Milnerin [11] kirjaan *Communication and Concurrency*.

Työssä on sovellettu sellaista käytäntöä, että osa määritelmistä esitetään vasta siinä yhteydessä, kun niitä tarvitaan. Tämä johtuu määritelmien pohjautumisesta aiemmin esitettyihin algoritmeihin ja lauseisiin, jolloin niiden kaikkien sijoittaminen toiseen lukuun olisi epäkäytännöllistä.

Määritelmien lisäksi toisessa luvussa käsitellään syvyyssuuntainen etsintä perusteellisesti, koska se toimii pohjana useimmille polynomisia erikoistapauksia ratkoville algoritmeille. Evenin kirjasta [3] poiketen jaetaan syvyyssuuntaisen etsinnän löytämät särmät viiteen eri luokkaan, koska itsesilmukat ovat keskeisessä osassa MC-pelin ratkaisemisessa.

Stirlingin artikkeli [18] on toiminut keskeisenä lähteenä MC-pelin määrittelevässä kappaleessa ja kolmannessa luvussa, jossa todistetaan yksikäsitteisen ratkaisun olemassaolo sekä esitetään II-yksinkertaisen MC-pelin ratkaiseminen polynomisessa ajassa.

Stirlingin todistus MC-pelin voittajan yksikäsitteisyydestä ja sen löytämisestä käsitellään perusteellisesti. Todistus perustuu induktioon MC-pelin solmujen lukumäärän suhteen, ja se muodostaa samalla eksponentiaalisen algoritmin MC-pelin voittajan ratkaisemiseen. Induktiotodistuksessa käytetään hyväksi pakotusjoukkoja, joiden avulla on myös mahdollista ratkaista muutama erikoistapaus polynomisessa ajassa. Samalla algoritmi muodostaa myös voittostrategian pelin voittajalle.

Neljännessä luvussa esitellään kaksi MC-pelin erikoistapausta: puumaiset ja vahvasti puumaiset MC-pelit. Näitä tapauksia ei ole tiettävästi käsitelty aiemmin kirjallisuudessa. Puumaiset ja vahvasti puumaiset MC-pelit eivät sisällä muita syklejä kuin itsesilmukoita. Samassa luvussa todistetaan, että puumaisen ja vahvasti puumaisen MC-pelin voittaja on mahdollista ratkaista polynomisessa ajassa. Niille esitetään myös syvyyssuuntaiseen etsintään perustuva ajassa $O(E)$ toimiva ratkaisualgoritmi, missä E tarkoittaa MC-pelin särmien lukumäärää. Lisäksi esitellään pakotusjoukkojen laskentaan perustuva polynominen algoritmi puumaisille MC-peleille.

Viidennessä luvussa käsitellään syklejä sisältäviä MC-pelin erikoistapauksia, jotka on mahdollista ratkaista polynomisessa ajassa. Taas esitellään uusi MC-pelin erikoistapaus, syklikiinnitty MC-peli. Syklikiinnitetyssä komponentissa jokaisella syklillä on vähintään yksi yhteinen särmä. Myös syklikiinnitettyjen komponenttien ratkaiseminen perustuu syvyyssuuntaiseen etsintään, jonka aikana algoritmi kerää tietoa pelistä. Voittaja selviää yleensä vasta, kun komponentin jokainen solmu on käyty läpi.

Tämän jälkeen käsitellään tilannetta, jossa löytynyt polku tai sykli on pakottava. Polun pakottavuus on riippuvainen syvyyssuuntaisen etsinnän etemisestä MC-pelissä, eikä tätä heuristiikkaa ole mahdollista käyttää aina yleisessä tilanteessa.

Viimeisessä ongelmaa käsittelevässä luvussa tutkitaan MC-pelin ratkaisemista yleisessä tilanteessa, jossa peli saa sisältää syklejä rajoittamattomasti. Tässä luvussa esitetään myös yläraja MC-pelin eri ratkaisujoukkojen määrälle, joka saadaan selville niiden syklien ja itsesilmukoiden lukumäärästä, joilla ei ole yhtään yhteisiä solmuja. Ratkaisujoukkojen määrän tunteminen auttaa pelin kompleksisuuden ymmärtämisessä ja sen konkreettisena hyötynä on saada rajattua alueita, joita ei mahdollisesti pystytä pelistä ratkaisemaan.

Samassa luvussa pyritään luonnehtimaan yleisen polynomisen ratkaisun (jos sellainen olisi) ominaisuuksia. Samoin tarkastellaan sitä tiedon määrää, joka tarvitaan solmun tai solmujoukon ratkaisemiseksi. Pystymme saamaan syvyyssuuntaisen etsinnän aikana hyvin paljon tietoa MC-pelin ominaisuuksista. Ongel-

mallista on se, mitkä graafin ominaisuudet ovat oleellisia ratkaisemisen kannalta ja kuinka niiden avulla olisi mahdollista tehdä solmujen voittajan selvittämiseen johtavia päätelmiä.

Liitteessä A esitetään ratkaisualgoritmien toteuttamista ja niiden visualisointia varten tehty graphviewer-ohjelma. Algoritmien ja graphviewer-ohjelman toteutusta ei ole laitettu erilliseksi liitteeksi, sillä niihin on lukijan mahdollista tutustua www-osoitteessa `http://www.uta.fi/~tp54752/graphviewer.html`. Ratkaisualgoritmeja ei ole toteutettu käyttöliittymätasolla. Niiden toteukset löytyvät tällä hetkellä ainoastaan MC-pelin luokkien sisältä.

MC-pelien piirtämisen ohjelmoinnissa huomattiin, että piirtotavalla on hyvin suuri merkitys graafien visualisoinnissa. MC-pelin piirtäminen toteutettiin yksinkertaisimmalla mahdollisella tavalla sijoittamalla solmut ympyrän kehälle ja piirtämällä tämän jälkeen särmät. Tämä piirtotapa ei kuitenkaan tuota kovin hyviä tuloksia solmumäärän kasvaessa yli kymmeneen tai risteävien särmien määrän ollessa suuri.

2 Määritelmiä

2.1 Graafiteorian nimityksiä ja merkintöjä

Graafiteorian nimityksien ja merkintöjen osalta pääasiallisena lähteenä on käytetty Swamyn ja Thulasiramanin kirjaa [19] sekä joidenkin suomenkielisten termien lähteenä Koiviston ja Niemistön julkaisua [6].

Graafi (graph) G on pari (V, E), missä $V \neq \emptyset$ on äärellinen joukko ja E on joukko pareja (u, v), missä $u, v \in V$. Joukon V alkioita kutsutaan *solmuiksi* (vertex, node) ja joukon E alkioita kutsutaan *särmiksi* (edge) tai *kaariksi* (arc). Jos joukon E alkiot ovat järjestettyjä pareja, kutsutaan graafia *suunnatuksi* (directed graph, digraph). Graafia, joka ei ole suunnattu, kutsutaan *suuntaamattomaksi graafiksi*.

Tässä luvussa käsitellään suuntaamattomia graafeja, ellei toisin mainita. Suuntaamattomalle graafille esitetyt määritelmät pätevät myös suunnatuille graafeille, ellei asiayhteydessä selviä toisin. Suunnattua graafia vastaava suuntaamaton graafi saadaan, kun särmien järjestysominaisuus jätetään pois.

Graafi on *yksinkertainen*, jos sen kahden solmun $v, u \in V$ välillä on korkeintaan yksi särmä. Jatkossa oletetaan ilman eri mainintaa, että käsiteltävät graafit ovat yksinkertaisia.

Olkoon (u, v) graafin $G = (V, E)$ särmä. Tällöin solmua u kutsutaan *lähtösolmuksi* ja solmua v *maalisolmuksi*. Lähtö- ja maalisolmusta käytetään usein myös nimitystä *alku-* ja *loppusolmu*. Särmää (u, v) kutsutaan myös *solmujen u ja v väliseksi kaareksi*. Jos $u = v$, niin särmää kutsutaan *itsesilmukaksi* (self-loop). Solmuja u ja v kutsutaan myös särmän *solmupariksi* tai *päätesolmuiksi*.

Graafi $H = (W, F)$ on graafin $G = (V, E)$ *aligraafi* (subgraph), jos $W \subseteq V$ ja $F \subseteq E$. Jos $W \subset V$ ja $F \subseteq E$ tai $W \subseteq V$ ja $F \subset E$, niin graafia H sanotaan graafin G *aidoksi aligraafiksi*. Olkoon $G = (V, E)$ ja $H = (V, F)$ graafin G aligraafi. Tällöin H on graafin G *virittävä* aligraafi.

Olkoon $G = (V, E)$ ja $F \subseteq E$. Tällöin $H = (W, F)$, missä $W \subseteq V$ on joukon F särmien päätesolmujen joukko, on joukon F *(särmä)indusoima* graafin G aligraafi. Tällöin merkitään $H = \langle F \rangle$. Usein puhutaan myös särmäjoukon virittämästä aligraafista.

Olkoon $G = (V, E)$ ja $W \subseteq V$. Tällöin $H = (W, F)$ on joukon W *(solmu)indusoima* graafin G aligraafi, jos $F \subseteq E$ muodostuu niistä joukon E särmistä, joiden päätesolmut kuuluvat joukkoon W. Tällöin merkitään $H = \langle W \rangle$. Solmujoukon indusoiman aligraafin sijaan voidaan puhua myös solmujoukon virit-

tämästä aligraafista.

Olkoon $G = (V, E)$ ja $H = (W, F)$ graafeja, joille on voimassa $W \subseteq V$. Tällöin graafien G ja H *erotus* on graafi $G - H = (V, E \backslash F)$.

Solmun v särmiksi sanotaan niitä särmiä, joiden solmuparissa ainakin toinen solmu on v. Suunnatun graafin särmää (u, v) sanotaan *solmuun v tulevaksi särmäksi* ja särmää (v, w) sanotaan *solmusta v lähteväksi särmäksi*. Solmun v *aste deg(v)* on solmun v särmien lukumäärä, jossa itsesilmukat lasketaan mukaan kahdesti. Suunnatun graafin solmun v *lähtöaste* $deg^+(v)$ on solmusta v lähtevien särmien lukumäärä. Suunnatun graafin solmun v *tuloaste* $deg^-(v)$ on solmuun v tulevien särmien lukumäärä.

Jos v on graafin $G = (V, E)$ solmu, niin $G - v$ on solmujoukon $V \backslash \{v\}$ indusoima graafin G aligraafi. Jos e on graafin $G = (V, E)$ särmä, niin $G - e = (V, E \backslash \{e\})$. Koska useampia särmiä ja solmuja poistettaessa ei niiden poistamisjärjestyksellä ole merkitystä, voidaan käyttää merkintöjä $G - v_1, v_2, \ldots, v_n$ ja $G - e_1, e_2, \ldots, e_n$.

*P*olku (path) P solmusta v_0 solmuun v_k on sellainen äärellinen vuorotteleva jono solmuja ja kaaria, että $P = v_0, (v_0, v_1), v_1, (v_1, v_2), v_2, \ldots, (v_{k-1}, v_k), v_k$. Solmu v_0 on polun *alkusolmu* ja solmu v_k on polun P *loppusolmu*. Jos polun P alkusolmu on v ja loppusolmu on u, niin polkua P sanotaan *solmujen u ja v väliseksi poluksi*. Solmusta on aina polku itseensä ja polkua v_0 sanotaan *tyhjäksi poluksi*. Suunnatussa graafissa polku määritellään samoin kuin suuntaamattomassakin, mutta kaaria voi kulkea vain alkusolmusta loppusolmuun. Suunnattujen graafien yhteydessä käytetään polulle joskus nimitystä *suunnattu polku*.

Olkoon P graafin $G = (V, E)$ polku. Koska jokaisen särmän lähtö- ja maalisolmu ovat yksikäsitteisiä, voidaan polku P antaa myös särmäjonona. Yksinkertaisen graafin polku voidaan antaa myös solmujonona.

Polun *pituus* on sen särmien lukumäärä. Tyhjän polun pituus on 0. Polku on *yksinkertainen*, jos mikään särmä ei esiinny siinä kahta kertaa. Polku on *suora*, jos mikään solmu ei esiinny siinä kahta kertaa. Solmujen u ja v *etäisyys* on niiden välisen lyhimmän suoran polun pituus. Mikäli kahden solmun välillä ei ole polkua, on niiden välinen etäisyys määrittelemätön.

Polku on *piiri* (circuit) jos sen alku ja loppusolmu ovat samat. Piiri on *yksinkertainen* (simple), jos jokaista särmää käytetään korkeintaan kerran. *Silmukka* on yksinkertainen piiri, jossa alku- ja loppusolmua lukuunottamatta mikään solmu ei esiinny kahta kertaa. Tässä työssä silmukalla ja piirillä tarkoitetaan aina polkua, jonka pituus on suurempi kuin 1. Muussa tapauksessa puhutaan itsesilmukasta. Silmukasta käytetään usein myös nimitystä *sykli* (cycle).

Polku voi olla myös ääretön jono solmuja ja kaaria. Tällöin sitä sanotaan *äärettömäksi* poluksi. Äärettömällä polulla ei ole loppusolmua. Koska graafi on aina äärellinen, alkaa äärettömässä polussa toistumaan ainakin yksi silmukka äärettömän usein.

Graafi on *yhtenäinen* (connected), jos minkä tahansa kahden eri solmun välillä on polku. Suunnattu graafi on *vahvasti yhtenäinen* (strongly connected), jos jokaisen solmuparin u, v välillä on polku sekä solmusta u solmuun v että solmusta v solmuun u. Jos suunnattu graafi ei ole vahvasti yhtenäinen, mutta sitä vastaava suuntaamaton graafi on yhtenäinen, on graafi *heikosti yhtenäinen*.

Graafin G yhtenäinen aligraafi H on graafin G *komponentti*, jos H ei ole minkään graafin G yhtenäisen aligraafin aito aligraafi.

Suunnatun graafin $G = (V, E)$ *vahvasti yhtenäinen komponentti* $H = (W, F)$ on graafin G maksimaalinen aligraafi, jonka jokaista solmuparia $u, v \in W$ kohti on olemassa polku solmusta u solmuun v ja solmusta v solmuun u.

Puu on suuntaamaton yhtenäinen graafi, jossa ei ole silmukoita eikä itsesilmukoita. Jos puu on graafin G aligraafi, on kyseessä graafin G *alipuu*. Graafin G *virittävä puu* on graafin G alipuu, joka sisältää kaikki graafin G solmut.

Suunnatun graafin G solmu v on *juuri*, jos solmusta v on suunnattu polku jokaiseen graafin solmuun. Suunnatussa graafissa voi olla useita juuria. Suunnattu graafi G on *juurellinen puu*, jos graafissa G on juuri ja vastaava suuntaamaton graafi on puu. Juurellisessa puussa on täsmälleen yksi juuri. Jos juurellisen puun G solmun v lähtöaste $deg^+(v) = 0$, niin solmua v sanotaan *lehdeksi* tai *lehtisolmuksi*. Muuten solmu on *sisäsolmu*.

Olkoon $G = (V, E)$ juurellinen puu. Jos $(u, v) \in E$, sanotaan, että solmu u on solmun v *vanhempi* ja v vastaavasti solmun u lapsi. Solmun u *esivanhemmat* ovat ne solmut, joista on (suunnattu) polku solmuun u. Solmun u *jälkeläiset* ovat ne solmut, joihin solmusta u on (suunnattu) polku.

Olkoon $G = (V, E)$ graafi ja olkoon |V|=n. Bijektio $num : V \to \{1, \ldots, n\}$ on graafin G solmujen *numerointi*. Jos $num(v) = i$, niin sanotaan, että i on solmun v numero.

Olkoon K graafin $G = (V, E)$ vahvasti yhtenäinen komponentti. Jos kaikilla särmille (v, u), missä $v \in K$, on voimassa $u \in K$, sanotaan komponenttia K *eristetyksi*.

Graafia, jossa on täsmälleen yksi solmu, sanotaan *triviaaliksi* graafiksi. Graafi on *täydellinen*, jos jokaisen solmuparin v, u välillä on särmä.

2.2 Algoritmeista

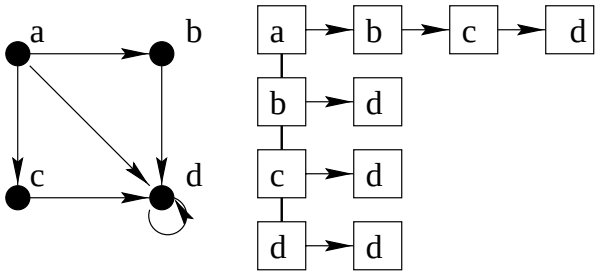
Tässä työssä algoritmien esittämiseen on käytetty Evenin [3] kirjasta tuttua esitystapaa, jossa algoritmin operaatiot on esitetty numeroituna luettelona. Algoritmin suoritus alkaa kohdasta 1 ja jatkuu seuraavaan numeroon, elleivät hyppy- ja toistolauseet ohjaa algoritmin suoritusta muihin kohtiin.

Algoritmin toimintaohjeet on esitetty luonnollisten kielten ja ohjelmointikielten välille sijoittuvalla *pseudokoodilla*. Pseudokoodi on tarvittaessa helppo muokata mille tahansa lausepohjaiselle ohjelmointikielelle, esimerkiksi *java*- tai *c*-kielille.

Jatkossa oletetaan, että lukijalla on perustiedot algoritmeista ja niiden analyysistä. Algoritmien suunnittelua, analyysia ja merkintätapoja on käsitelty perusteellisesti lähteissä [2] ja [12].

Algoritmi on *deterministinen*, jos algoritmin toimintaohjeet määräävät jokaisen suoritettavan toiminta-askeleen yksikäsitteisesti. Muussa tapauksessa algoritmi on *epädeterministinen*.

Kuva 2.1 Suunnatun graafin esittäminen piirroksena, vieruslistana ja matriisina



	a	b	c	d
a	0	1	1	1
b	0	0	0	1
c	0	0	0	1
d	0	0	0	1

G=({a,b,c,d},{(a,b),(a,c),(a,d),(b,d),(c,d),(d,d)})

Graafialgoritmit ovat graafien käsittelyyn ja muokkaamiseen tarkoitettuja algoritmeja. Näiden algoritmien tehokkuuteen vaikuttaa oleellisesti se, millaisella tietorakenteella graafi ohjelmassa esitetään. Kaksi yleisintä tapaa esittää graafi on *vieruslista* (adjacency list) ja *vierusmatriisi* (adjacency matrix). Vieruslistassa kerrotaan jokaisesta solmusta lähtevät kaaret listassa ja vierusmatriisissa ilmaistaan totuusarvoilla, onko kahden solmun välissä kaari vai ei. Vieruslistan koko on suhteessa graafin särmien lukumäärään ja vierusmatriisin koko on $|V| * |V|$. Algoritmisesti vierusmatriisin käsittely on helppoa, koska solmuihin ja särmiin voidaan viitata suoraan matriisin rivien ja sarakkeiden numeroilla. Vieruslista

vie vähemmän tallennustilaa, mutta on useimmiten vaikeampi käsitellä. Yleisesti ottaen vieruslista sopii *harvoille* (sparse) graafeille, jotka sisältävät vähän kaaria suhteessa solmujen lukumäärään. Vierusmatriisi on taas hyvä esitystapa *tiheille* (dense) graafeille, jotka sisältävät paljon kaaria.

Algoritmien keskeisiä ominaisuuksia ovat *aikavaatimus* ja tuloksen *oikeellisuus*. Aikavaatimuksella tarkoitetaan algoritmin suorittamien operaatioiden määrää suhteessa syötteeseen [2, 12]. Toisinaan puhutaan myös algoritmin *aikakompleksisuudesta* [4].

Usein kaikkien algoritmin suorittamien operaatioiden tarkka laskeminen on vaikeaa. Yleensä riittää selvittää, miten kompleksisuusfunktio kasvaa (suhteessa syötteeseen) asymptoottisesti välittämättä vakiokertoimesta. Tätä varten otamme käyttöön merkintätavan $O(g)$ [2].

Olkoon funktiot $f : \mathbb{N} \to \mathbb{R}$ ja $g : \mathbb{N} \to \mathbb{R}$. Määritellään, että funktio f on $O(g)$, jos on olemassa sellaiset positiiviset vakiot c ja n_0, että $|f(n)| \leq c|g(n)|$, kun $n \geq n_0$. Syötealkioiden lukumäärää merkitään muuttujalla n ja operaatioiden määrää suhteessa syötteeseen funktiolla $f(n)$.

Merkinnällä $O(g(n))$ tai lyhyemmin $O(g)$ tarkoitetaan siis *asymptoottista ylärajaa* algoritmin aikavaatimukselle.

Algoritmi on *polynominen*, jos sen aikakompleksisuus on $O(g(n))$, missä g on polynominen funktio. Mikäli aikavaatimusta ei pystytä rajaamaan polynomisella funktiolla, on algoritmi *eksponentiaalinen*. Mikäli algoritmin aikavaatimus on $O(n)$, on algoritmi *lineaarinen*. Graafien käsittelyssä usein esiintyvät aikavaatimukset ovat $O(|E|)$, $O(|V|)$, $O(|V| * |E|)$ ja $O(|V|!)$. Näistä kolme ensimmäistä on polynomisia, ja viimeksi mainittu on eksponentiaalinen.

Seuraavaksi esitettävä kahden algoritmien luokan määritteleminen vaatii *formaalien kielten* ja *Turingin koneen* käsitteiden tuntemista. Näiden määritelmät sivuutetaan tässä työssä, mutta niihin voi tutustua tarkemmin lähteiden [4, 5, 13, 14] avulla.

Määritellään luokat P ja NP [2, 4, 14] seuraavasti:

- $NP = \{L|$ on olemassa sellainen polynomi $p(n)$ ja sellainen epädeterministinen Turingin kone M, että M hyväksyy kielen L ajassa $p(n)\}$.

- $P = \{L|$ on olemassa sellainen polynomi $p(n)$ ja sellainen deterministinen Turingin kone M, että M hyväksyy kielen L ajassa $p(n)\}$.

Koska deterministinen Turingin kone on erikoistapaus epädeterministisestä

Turingin koneesta, on voimassa $P \subseteq NP$. Yleisesti uskotaan, että $P \neq NP$, mutta tätä ei ole vielä pystytty todistamaan [14, 4].

Kieli L on *NP-kova*, jos kaikki luokan NP kielet redusoituvat [14] siihen polynomisessa ajassa. Kieli on *NP-täydellinen*, jos se on NP-kova ja kuuluu luokkaan NP.

Olkoon L niiden kielten joukko, joiden komplementti $\overline{L} \in NP$. Tällöin L kuuluu kompleksisuusluokkaan $co - NP$. Avoimena kysymyksenä tällä hetkellä on, seuraako ominaisuudesta $L \in NP$ ominaisuus $\overline{L} \in NP$.

Kaikki polynomisilla algoritmeilla ratkeavat ongelmat kuuluvat luokkaan P. Luokan P voi siis määritellä myös niiden ongelmien joukkona, jotka ratkeavat polynomisessa ajassa. Luokan NP ongelmien epädeterministisesti arvattu ratkaisu voidaan aina tarkistaa polynomisessa ajassa.

Luokkiin NP tai P kuulumista pidetään yleisesti erotteluna laskennallisesti vaikeiden ja helppojen ongelmien välillä.

Algoritmi on *ahne* (greedy), jos se pyrkii aina päätöksentekohetkellä tekemään parhaalta vaikuttavan ratkaisun. Ahneen algoritmin antama tulos ei siis välttämättä ole ongelman optimaalinen, tai aina edes oikea, ratkaisu.

Heuristiikka (heuristic) on päättelyketju- tai menetelmä, joka ratkaisee tietyn tyyppisen ongelman.

2.3 Syvyyssuuntainen etsintä

Graafin solmujen läpikäynti on osatehtävänä monissa graafeja käsittelevissä algoritmeissa. Yleisin menettely solmujen läpikäyntiin on *syvyyssuuntainen etsintä* (depth-first search, DFS).

Solmujen läpikäyntitekniikkana syvyyssuuntainen etsintä on hyvin vanha. DFS tunnettiin jo 1800-luvulla ja se oli alunperin kehitetty labyrinttien läpikäymiseen. Syvyyssuuntaisen etsinnän perusmuoto kulkee nimellä *Tremauxin algoritmi* sen keksijän mukaan [3].

Nykyaikainen versio syvyyssuuntaisesta etsinnästä esitettiin 1970-luvulla Tarjanin [20] artikkelissa. Uutta Tarjanin toteutuksessa oli solmujen numerointi etsinnän aikana. Tämä mahdollisti muun muassa kaarien ja solmujen luokittelun niiden ominaisuuksien perusteella, mistä seuraa DFS-algoritmin käyttökelpoisuus mitä erilaisimpien graafiteoreettisten ongelmien ratkaisemisessa.

Solmua, josta etsintä alkaa, kutsutaan *aloitussolmuksi*. Syvyyssuuntainen etsintä löytää kaikki solmut, jotka ovat aloitussolmun jälkeläisiä. Käytännössä suunnatuilla graafeilla aloitussolmuna toimii useimmiten graafin juuri.

Algoritmi 1 Syvyyssuuntainen etsintä suunnatulle graafille

Muuttujat: Käsiteltävä solmu v, solmun v vanhempi $f(v)$ ja solmun v dfs-numero $k(v)$.

Syöte: Graafi G.

Tulos: Graafin G dfs-numerointi.

1: **Alustaminen**. Merkitse graafin kaikki särmät käyttämättömiksi. Sijoita kaikille solmuille $k(v) \leftarrow 0$ ja $f(v) \leftarrow -1$. Sijoita $i \leftarrow 0$ ja $v \leftarrow s$. Solmu s on aloitussolmu.

2: Sijoita $i \leftarrow i + 1$ ja $k(v) \leftarrow i$.

3: Jos solmussa ei ole käyttämättömiä särmiä niin mene kohtaan (5).

4: Valitse käyttämätön särmä (v, u). Merkitse särmä käytetyksi. Jos $k(u) \neq 0$, mene kohtaan (3). Muuten ($k(u) = 0$) sijoita $f(u) \leftarrow v$, $v \leftarrow u$ ja mene kohtaan (2).

5: **Peruutus**. Jos $f(v) \neq -1$, niin sijoita $v \leftarrow f(v)$ ja mene kohtaan (3).

6: **Lopetus**. ($f(v) = -1$) Kaikki solmut käyty läpi ja numeroitu. Lopeta.

Etsinnän kuluessa graafille saadaan virittävä puu tai virittävän puun alipuu. Syvyyssuuntainen etsintä antaa myös jokaiselle graafin solmulle yksikäsitteisen *dfs-numeron*, joka ilmoittaa, monentenako algoritmi löysi kyseisen solmun ensimmäisen kerran. Syvyyssuuntainen etsintä toimii sekä suunnatuille että suuntaamattomille graafeille.

Syvyyssuuntainen etsintä toimii siten, että käsiteltävästä solmusta v valitaan aina läpikäymätön särmä (v, u) ja merkitään se käytetyksi. Mikäli solmussa u ei olla käyty aiemmin, edetään solmuun u. Merkitään solmu u tutkituksi ja solmu v solmun u vanhemmaksi ($f(u) \leftarrow v$). Sijoitetaan solmu u käsiteltäväksi solmuksi ja tutkitaan vastaavasti sen särmiä.

Jos särmän (v, u) maalisolmussa on käyty jo aiemmin, jatketaan solmun v muiden särmien tutkimista.

Kun käsiteltävän solmun v kaikki särmät on tutkittu, peruutetaan siihen solmuun, josta solmu v ensimmäisen kerran löydettiin (solmu $f(v)$), sijoitetaan solmu $f(v)$ käsiteltäväksi solmuksi ($v \leftarrow f(v)$) ja tutkitaan solmun v särmiä, kunnes kaikki saadaan käytyä läpi. Mikäli solmulle, jonka kaikki särmät on käyty läpi, ei ole määritelty vanhempaa, on graafin kaikki solmut käyty läpi, ja etsintä voidaan lopettaa.

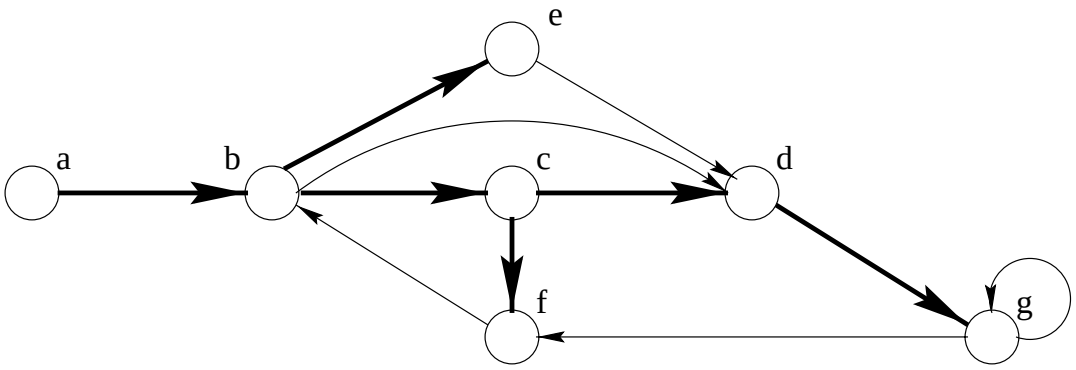
Algoritmi 1 löytää graafista kaikki aloitussolmun jälkeläiset. Mikäli alkuperäinen graafi ei ole yhtenäinen tai aloitussolmusta ei ole polkua kaikkiin muihin

solmuihin, jää osa solmuista läpikäymättä. Tällöin algoritmia 1 voi muokata löytämään graafin kaikki solmut lisäämällä kohtien 5 ja 6 väliin seuraava ohje:

> $(f(v) = -1)$ Jos on olemassa solmu u, jolle $k(u) = 0$, niin sijoita $v \leftarrow u$ ja mene kohtaan (2).

Tällöin algoritmi tarkistaa erikseen, onko jäljellä vielä tutkimattomia solmuja. Tätä lisäystä on syytä käyttää aina, jos on olemassa mahdollisuus, että aloitussolmusta ei ole polkua graafin muihin solmuihin.

Kuva 2.2 Eri kaarityyppien löytyminen syvyyssuuntaisella etsinnällä



Syvyyssuuntaisen etsinnän suunnatusta graafista löytämät särmät voidaan jakaa viiteen eri luokkaan läpikäyntihetken dfs-numeroiden perusteella:

1. *Puukaari* (tree edge): särmä (v, u) on puukaari, jos sitä käytettiin löydettäessä solmu u ensimmäisen kerran; tällöin on $k(u) = 0$ etsinnän tutkiessa särmää (v, u).

2. *Menokaari* (forward edge): särmä (v, u) on menokaari, jos $k(u) > k(v)$; tällöin solmu u oli jo löydetty aikaisemmin ja u on solmun v jälkeläinen.

3. *Itsesilmukka* (self-loop): särmä (v, u) on itsesilmukka, jos $k(u) = k(v)$; tällöin särmä johtaa takaisin lähtösolmuun $(u = v)$.

4. *Paluukaari* (backward edge): särmä (v, u) on paluukaari, jos $k(u) < k(v)$ ja solmu u on solmun v esivanhempi.

5. *Ristikkäiskaari* (cross edge): särmä (v, u) on ristikkäiskaari, jos $k(u) < k(v)$ ja solmu u ei ole solmun v esivanhempi.

Esimerkki. Kuvassa 2.2 on esimerkki syvyyssuuntainen etsinnän tuottamasta kaarien luokittelusta. Kaaret (a,b),(b,c),(c,f),(c,d),(d,g) ja (b,e) muodostavat kuvan graafin virittävän puun. Puukaret on vahvennettu. Kaari (b,d) on menokaari, (e,d) on ristikkäiskaari ja (g,g) on itsesilmukka. Kaaret (f,b) ja (g,f) ovat paluukaaria. Yllä esitellyn kaarien luokittelun lisäksi on myös muita mahdollisuuksia. Luokittelu riippuu solmujen läpikäyntijärjestyksestä.

Syvyyssuuntaisen etsinnän suoritusaika voidaan määritellä graafin särmien määrästä riippuvaksi. Tämän perusteella voidaan johtaa algoritmin aikavaatimus.

Apulause 2.1 *([20]) Algoritmin 1 aikavaatimus on $O(|E|)$.*

Todistus. Syvyyssuuntaisen etsinnän suorituksen aikana jokainen särmä $(v,u) \in E$ käydään läpi kahdesti. Ensimmäisen kerran kun särmää käytetään särmän suuntaisesti lähtösolmusta v maalisomuun u. Toisella kerralla kuljetaan särmää päinvastaiseen suuntaan. Jos maalisolmu u on löydetty jo aikaisemmin, peruutetaan heti solmusta u solmuun v. Mikäli maalisolmu u on tutkimatta, peruutetaan särmää (v,u) takaisin vasta sitten, kun solmun u kaikki särmät on tutkittu.

Jos solmun v särmä on jo merkitty tutkituksi, ei sitä koskaan käytetä toista kertaa.

□

Jos syvyyssuuntaiseen etsintään liitetään ylimääräisiä toimintoja, on algoritmin aikakompleksisuus edelleen poynominen, mikäli suoritettavat lisätoimenpiteet voi suorittaa polynomisessa ajassa. Tähän perustuu myöhemmin esitettävien DFS:n pohjalle rakennettujen ratkaisualgoritmien tehokkuus. Syvyyssuuntaisen etsinnän avulla on mahdollista toteuttaa esimerkiksi graafin syklien etsiminen ja polkujen pituuden laskenta.

Seuraavaksi esitetään todistus, jonka perusteella graafin virittävän puun muodostaminen onnistuu syvyyssuuntaisen etsinnän avulla.

Apulause 2.2 *([3]) Algoritmin 1 löytämät puukaaret muodostavat graafin G virittävän puun tai virittävän puun alipuun, ja jonka juuri on etsinnän aloitussolmu.*

Todistus. Sivuutetaan.

□

Olkoon (v,u) syvyyssuuntaisen etsinnän löytämä paluukaari ja P puukaarten muodostama polku solmusta v solmuun u. Sykliä C, joka muodostuu polusta P ja paluukaaresta (v,u), sanotaan *paluukaaren (v,u) virittämäksi sykliksi.*

Sovellettaessa syvyyssuuntaista etsintää useampia kertoja samalle graafille voidaan saada erilainen löytöjärjestys solmuille. Jos solmujen läpikäyntijärjestys muuttuu, voi myös särmien luokittelu muuttua. Tämä perustuu siihen, että valittaessa käyttämätöntä särmää algoritmin 1 kohdassa 4, ei valintaperusteita ole määritelty. Yleensä solmuilla on olemassa jokin järjestys, jonka perusteella voimme valita aina määrätyt ominaisuudet täyttävän solmun. Mikäli solmuille ei ole olemassa järjestystä, joudutaan valinta suorittamaan satunnaisesti. Tämän seurauksena voi etsinnän eri suorituskerroilla tulla erilaisia luokituksia särmille.

2.4 Vahvasti yhtenäiset komponentit

Vahvasti yhtenäisten komponenttien etsiminen suunnatusta graafista on tärkeä sovellus syvyyssuuntaiselle etsinnälle.

Yksi tapa etsiä graafista komponentteja on käyttää *low-point algoritmia* [3], joka ei kuitenkaan löydä suoraan kaikkia maksimaalisia vahvasti yhtenäisiä komponentteja.

Toinen mahdollisuus on käyttää syvyyssuuntaista etsintää komponenttien löytämiseksi, jolloin voimme löytää maksimaaliset komponentit. Syvyyssuuntainen etsintä pitää suorittaa graafin jokaiselle solmulle erikseen. Jokaisesta etsinnästä tallennetaan tieto, onko aloitussolmusta v polku solmuun u. Jos solmusta u solmuun v on polku ja solmusta v solmuun u, niin solmut v ja u kuuluvat samaan vahvasti yhtenäiseen komponenttiin. Tämä perustuu suoraan vahvasti yhtenäisen komponentin määritelmään.

Komponenttien löytämiseksi järjestetään solmut komponentteihin sen mukaan, onko solmuparien tai solmun ja aiemmin saadun solmujoukon välillä polku molempiin suuntiin. Komponentit voidaan numeroida esimerkiksi ensimmäisen syvyyssuuntaisen etsinnän antaman dfs-numeron mukaan.

Edellä kuvatulla algoritmilla tapahtuva maksimaalisten vahvasti yhtenäisten komponenttien etsiminen tapahtuu ajassa $O(V * E)$, koska syvyyssuuntainen etsintä joudutaan suorittamaan $|V|$ kertaa.

Mäkinen kuvaa kirjassaan [12] algoritmin, joka löytää suunnatun graafin vahvasti yhtenäiset komponentit kahdella syvyyssuuntaisella etsinnällä. Etsintöjen välillä on graafin kaikkien särmien suunnat muutettava päinvastaiseksi. Menetelmä on hieman monimutkaisempi, mutta tehokkaampi.

2.5 Silmukoiden etsiminen graafista

Silmukoiden etsiminen suunnatusta graafista on mahdollista tehdä käyttäen hyväksi syvyyssuuntaista etsintää. Graafista voi silmukoita löytyä hyvin suuri määrä, jos graafi on tiheä. Täydellisessä graafissa silmukoita on eksponentiaalinen määrä suhteessa solmujen määrään. Graafi, jossa on n solmua ja $n(n-1)$ särmää, sisältää $\binom{n}{i}(i-1)!$ silmukkaa, joiden pituus on i. Yhteensä silmukoita löytyy $\sum_{i=2}^{n}\binom{n}{i}(i-1)!$ kappaletta [15].

Lueteltaessa graafin silmukoita voidaan silmukoita $C_1 = v_0, v_1, v_2, \ldots, v_n, v_0$ ja $C_2 = v_1, v_2, \ldots, v_n, v_0, v_1$ pitää samoina.

Tarkastellaan silmukoiden muodostumista suhteessa syvyyssuuntaisen etsinnän löytämiin kaarityyppeihin. Aina löydettäessä paluukaari (v, u) voidaan kaaren (v, u) virittämä sykli lisätä graafin silmukoiden joukkoon. Meno- ja ristikkäiskaarten käsittely on hieman vaikeampaa, sillä niiden löytäminen voi kasvattaa silmukoiden määrää kaksinkertaiseksi aiemmin löydettyihin silmukoihin verrattuna.

Kun löytyy menokaari (v, u), tarkistetaan, onko menokaaren molemmat solmut osallisena johonkin aiemmin löydettyyn silmukkaan $C_i = v_0, v_1, v_2, \ldots v, \ldots, u, \ldots, v_n, v_0$. Nyt saadaan jokaista aiemmin löytynyttä silmukkaa, josta löytyy menokaaren solmut, kohti uusi silmukka $C_i' = v_0, v_1, v_2, \ldots, v, u, \ldots, v_n, v_0$.

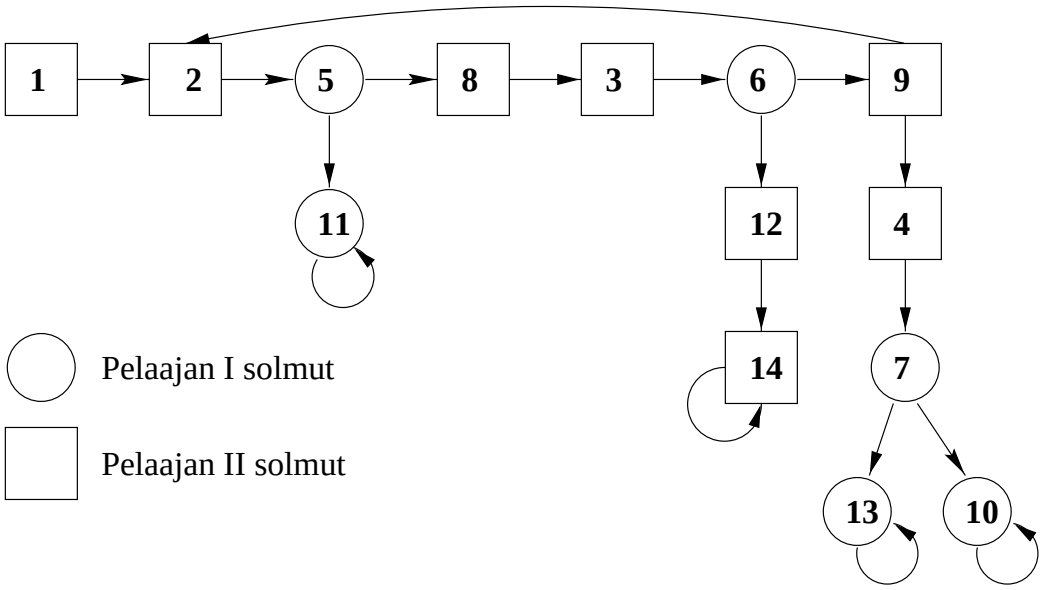
Löydetäessä ristikkäiskaari (v, u) tilanne on hieman monimutkaisempi. Olkoon $P = u_0, u_1, u_2, \ldots, v$ puukaarten muodostama polku syvyyssuuntaisen etsinnän aloitussolmusta solmuun v. Jos solmu u ja solmu $u_i \in P : u_i \neq u$ kuuluvat johonkin aiemmin löydettyyn silmukkaan $C_i = v_0, v_1, v_2, \ldots, u_i, \ldots, u, v_k, \ldots, v_n, v_0$, niin on löytynyt taas uusi silmukka. Olkoon polku $P' = u_i, u_{i+1}, \ldots, v \subseteq P$ puukaarista koostuva polku. Nyt saadaan jokaista silmukkaa C_i kohti uusi silmukka $C_i' = v_0, v_1, v_2, \ldots, u_i, u_{i+1}, \ldots, v, u, v_k, ..., v_n, v_0$.

Edellä esitetty menetelmä suunnatun graafin kaikkien silmukoiden löytämiseksi on alunperin esitetty hieman erilaisessa muodossa Reingoldin, Nievergeltin ja Deon kirjassa [15].

2.6 MC-peli

MC-peli (MC-game) on pari $\mathcal{G} = (G, token)$, missä $G = (V, E)$ on suunnattu graafi, jonka solmujen joukko on $V = \{1, \ldots, n\}$ ja särmien joukko on E. Bijektio $token$ on kuvaus joukolta V joukolle Pl, missä $Pl = \{I, II\}$. Lisäksi MC-pelin jokaisen solmun lähtöaste $deg(v)_+ \geq 1$ ja solmu 1 on graafin G juuri.

Kuva 2.3 Esimerkkipeli



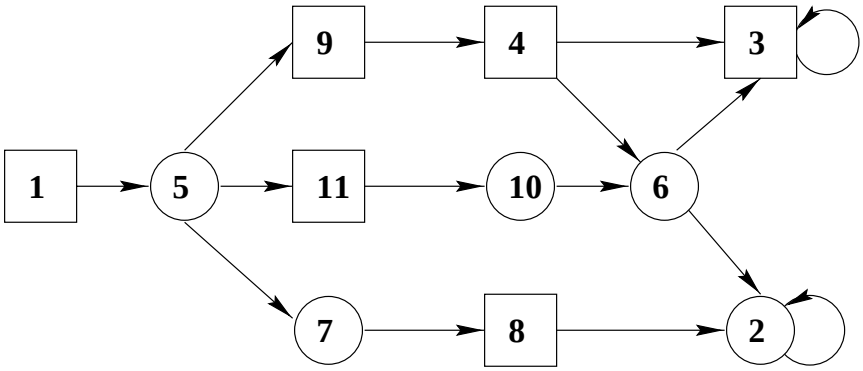
Joukkoa Pl kutsutaan *pelaajien* joukoksi. Bijektio $token$ nimeää MC-pelin $\mathcal{G}$ jokaisen solmun joko pelaajalle I tai II. Olkoon $token(v) = a$. Tällöin sanotaan, että solmu v on *pelaajan a solmu* tai pelaajan a *omistama* solmu.

MC-pelistä $\mathcal{G} = (G, token)$ puhuttaessa tarkoitetaan usein graafia G. Haluttaessa korostaa, että tarkoitetaan täsmällisesti MC-pelin $\mathcal{G}$ graafia G, voidaan puhua *MC-pelin $\mathcal{G}$ graafista*. Jatkossa kuitenkin käytetään MC-peliä ja graafia eräänlaisina synonyymeinä. Sanottaessa, että käytetään syvyyssuuntaista etsintää MC-pelille, tarkoitetaan itseasiassa graafille G suoritettavaa syvyyssuuntaista etsintää. Myöskin sanottaessa MC-pelin sisältävän syklejä tarkoitetaan, että MC-pelin graafi G sisältää syklejä. Muita syvyyssuuntaista etsintää ja graafiteoriaa koskevia määritelmiä käytetään vastaavasti.

MC-pelin piirtämisessä on omaksuttu lähteen [18] tapa, jossa solmut piirretään eri tavalla riippuen solmun omistajasta. Solmu v piirretään ympyränä, jos sen omistaja $token(v) = I$ ja jos $token(v) = II$, niin neliönä.

MC-peli alkaa solmusta numero 1 ja etenee siten, että oltaessa solmussa v on siirtovuoro pelaajalla $token(v)$, joka on solmun omistaja. Solmun omistaja valitsee aina särmän, jota pitkin peliä jatketaan. Tätä kutsutaan *siirroksi* tai *siirtämiseksi*. Yksi *peli* koostuu äärettömän pitkästä polusta, jonka *aloitussolmu* on 1. *Pelin voittaja* on pelaaja, joka omistaa numeroltaan pienimmän solmun k, joka esiintyy pelissä äärettömän usein.

Kuva 2.4 Puumainen MC-peli



Tässä työssä tarkastellaan pääasiassa niitä tilanteita, joissa molemmat pelaajat pyrkivät pelaamaan optimaalisesti. Olemme siis kiinnostuneita pelin voittajasta suhteessa koko graafiin, emmekä niinkään yksittäisen pelin voittajasta.

Määritellään seuraavaksi MC-pelin ja solmujen erotus, jonka avulla pystymme myös määrittelemään MC-pelin *alipelin*. Olkoon MC-peleille $\mathcal{G} = \{(V, E), token\}$ ja $\mathcal{G}' = \{(V', E'), token'\}$ voimassa $V' = V - X$, missä $X \subseteq V$ ja $E' = E \cap (V' \times V')$. Jos vielä kuvaukselle $token'$ on voimassa $token'(v) = token(v)$, missä $v \in V'$, niin $\mathcal{G}' = \mathcal{G} - X$. Mikäli $V' \neq \emptyset$ ja jokaiselle solmulle $v \in V'$ on voimassa $deg_+(v') \geq 1$, niin $\mathcal{G}'$ on MC-pelin $\mathcal{G}$ *alipeli*.

Merkitään, että $\mathcal{G}(i)$ on MC-peli $\mathcal{G}$, jonka aloitussolmu on i. Tällöin $\mathcal{G}(1) = \mathcal{G}$ ja $\mathcal{G}(i)$ on aina MC-pelin $\mathcal{G}(1)$ alipeli. Usein puhutaan myös solmun i virittämästä alipelistä. Alipeli $\mathcal{G}(i)$ muodostuu siis solmun i kaikkien jälkeläisten joukon I virittämästä graafista $\langle I \rangle$. Alipelin $\mathcal{G}(i)$ voittaja on pelaaja, joka voittaa solmusta i alkavan pelin.

Olkoon MC-pelin $\mathcal{G}(i)$ voittaja a. Tällöin voimme merkitä, että solmun i voittaja $win(i) = a$. Olkoon MC-pelin solmulle v voimassa $token(v) = a$. Tällöin pelaajan a *vastustaja* O on pelaaja I, jos a on II, ja vastaavasti II jos a on I.

Olkoon $\mathcal{G} = \{(V, E), token\}$ MC-peli ja olkoon $win(v) = a$, missä $a \in \{I, II\}$ ja $v \in V$. Tällöin sanotaan, että pelaajalla a on *voittostrategia* σ_a^v solmussa v. Nyt pelaajalla a on olemassa pelitapa, jota käyttämällä hän voittaa aina pelin $\mathcal{G}(v)$. Jos $token(v) = a$, niin voittostrategia koostuu ohjeesta, joka yksiselitteisesti kertoo, mitä solmusta v lähtevää särmää käyttämällä pelaaja a voittaa. Jos $token(v) = O$, niin pelaaja a voittaa riippumatta vastustajan siirrosta.

Voittostrategia on *historiaton* (history free) [8, 21], jos aiemmilla siirroilla ei ole merkitystä solmun v voittajan määrittelemisessä. Usein puhutaan myös *muis-*

tittomasta strategiasta (no-memory strategy). Kaikki MC-pelin voittostrategiat ovat historiattomia. Jatkossa myös oletetaan, että käsiteltävät voittostrategiat ovat historiattomia, vaikkei sitä erikseen mainittaisi.

Olkoon pelaajalla a voittostrategia σ_a^v kaikille solmuille $v \in V' \subseteq V$. Tällöin voimme merkitä, että joukon $V' = \{v_0, v_1, \ldots, v_n\}$ voittostrategia on $\sigma_a^{V'} = \sigma_a^{v_0} + \sigma_a^{v_1} + \ldots + \sigma_a^{v_n}$.

MC-peli $\mathcal{G}$ on *vahvasti puumainen*, jos sitä vastaava suuntaamaton graafi ei sisällä muita silmukoita kuin itsesilmukoita. MC-peli on *puumainen*, jos se ei sisällä silmukoita. MC-peli on *triviaali*, jos se koostuu täsmälleen yhdestä solmusta. Triviaalissa MC-pelissä on aina itsesilmukka.

Kuvan 2.3 MC-pelistä saa vahvasti puumaisen poistamalla solmujen 9 ja 2 välillä olevan särmän. Kuvassa 2.4 on puumainen MC-peli.

2.7 MC-pelin yleistäminen ja yksinkertaiset stokastiset pelit

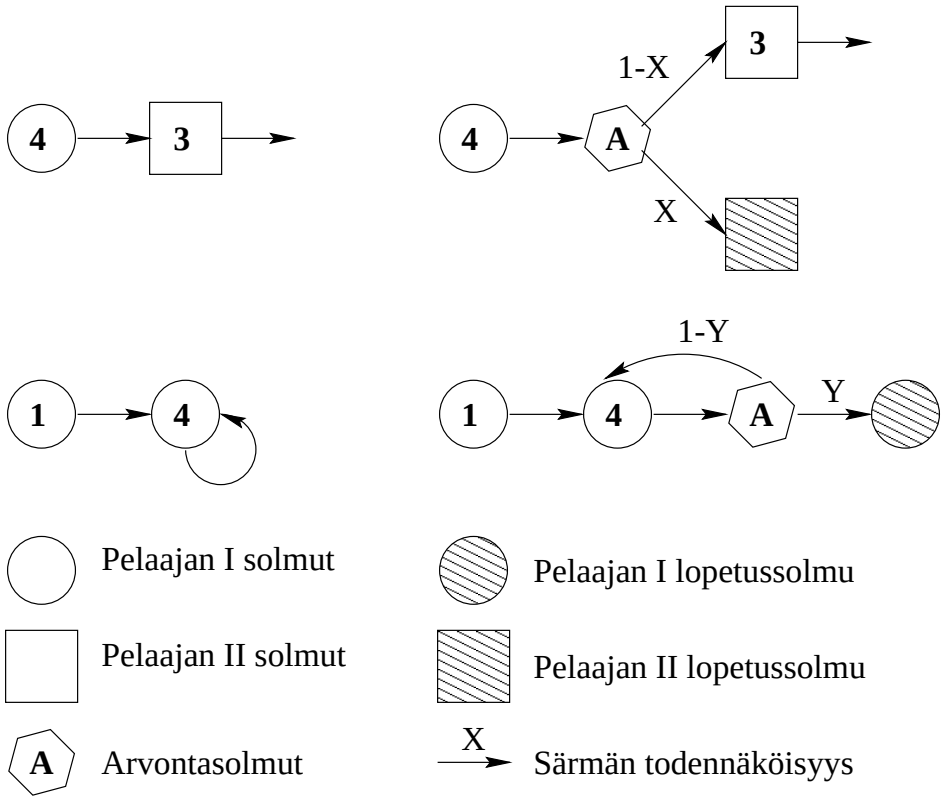
MC-pelit voidaan yleistää *laskettaviin äärellisiin peleihin* $\mathcal{G} = ((V, E), token, \mathcal{P})$, missä $\mathcal{P} = \{I, II, \ldots, N\}$ on äärellinen pelaajien joukko [18]. Jokainen solmu $v \in V$ kuuluu täsmälleen yhdelle pelaajalla $p \in \mathcal{P}$. Nyt bijektio *token* on kuvaus joukolta V joukolle $\mathcal{P}$. Pelin $\mathcal{G}$ voittaja on pelaaja, joka omistaa pieninumeroisimman solmun, jossa käydään äärettömän usein.

Jos pelaajia on enemmän kuin kaksi, pelin voittamiseen vaikuttaa toisten pelaajien siirrot. Olkoon pelaajan p_i omistamasta solmusta v kaksi lähtevää särmää (v, u') ja (v, u''). Jos särmän (v, u') pelaaminen johtaa pelaajan p_j voittoon ja särmän (v, u'') pelaajan p_k voittoon, on pelaajalla p_i mahdollisuus vaikuttaa pelin voittajaan tilanteessa, jossa hän valitsee solmusta v pelattavan särmän.

Toinen laajennus graafeilla mallinnettaviin peleihin on ottaa mukaan todennäköisyydet solmusta lähtevien särmien valintaan. *Yksinkertainen stokastinen peli* (simple stochastic game, SSG) [1] on äärellinen peli, jossa on kolmenlaisia solmuja: pelaajien I ja II solmuja sekä arvontasolmuja A (average). Lisäksi on olemassa kahdenlaisia päätössolmuja: *I-lopetus* ja *II-lopetus* (I/II-sink). Päätössolmujen lähtöaste on 0. Kaikkien muiden solmujen lähtöaste on vähintään yksi ja kaikilla arvontasolmuilla $v \in A$ lähtöaste $deg_+(v) = 2$. Arvontasolmuissa arvotaan pelin etenemissuunta siten, että molempien särmien todennäköisyys on $\frac{1}{2}$.

Tämäkin voidaan yleistää [17] niin, että särmien todennäköisyydet ovat muotoa $\frac{p}{q}$, missä $0 \geq p \geq q \geq 2^m$, missä $m \in \mathbb{N}$ on peliin liittyvä vakio, ja todennäköisyyksien summa on 1.

Kuva 2.5 MC-pelin muuttaminen SSG:ksi



Peli loppuu, kun saavutaan lopetussolmuun v. Jos v on pelaajan I lopetussolmu, niin pelaaja I voittaa, ja jos v on pelaajan II lopetussolmu, niin pelaaja II voittaa pelin. Mikäli peli ei joudu koskaan lopetussolmuun, vaan kiertää ikuisesti jotain sykliä, voittaa pelaaja I.

SSG:n päätösongelmana on selvittää, onko pelaajan II voittotodennäköisyys suurempi kuin $\frac{1}{2}$. Tällekään ongelmalle ei tällä hetkellä tunneta polynomista ratkaisualgoritmia. Condon [1] on osoittanut, että SSG:n päätösongelma kuuluu luokkaan $NP \cap co-NP$ ja Ludwig [9] on esittänyt ongelmalle eksponentiaalisen ratkaisualgoritmin, jonka aikavaatimukseksi tulee $2^{O(\sqrt{n})}$.

Jerrum [18] on osoittanut, että jokainen MC-peli voidaan muuttaa polynomisessa ajassa SSG:ksi. MC-pelin $\mathcal{G} = ((V, E), token)$ muuttaminen yksinkertaiseksi stokastiseksi peliksi tapahtuu seuraavasti. Lisätään molemmille pelaajille lopetussolmut. Lisätään jokaista solmua $u \in V$, johon tulee vähintään yksi särmä suurempinumeroisesta solmusta, kohden arvontasolmu A_u. Poistetaan ne särmät (v, u), joille $v \geq u$ ja lisätään poistettujen särmien lähtösolmusta särmä solmuun

A_u. Olkoon $token(v) = a$. Lisätään solmuun A_u kaksi lähtevää särmää, toinen pelaajan a lopetussolmuun End_a ja toinen särmä solmuun u. Merkitään särmän (A_u, End_a) todennäköisyydeksi k^{-u} ja särmän (A_u, u) todennäköisyydeksi $1-k^{-u}$, missä k on peliin liittyvä vakio. Arvontasolmusta lähtevien särmien kokonaistodennäköisyys on yksi.

Esimerkki. Kuvassa 2.5 on muunnettu MC-pelin kaksi tilannetta yksinkertaiseksi stokastisen pelin tilanteiksi. Vasemmalla on lähtötilanne ja oikealla tilanne SSG:ksi muuttamisen jälkeen. Särmille saadaan seuraavat todennäköisyydet: ylemmässä tilanteessa $prob(A_3, End_{II}) = \frac{1}{k^3}$ ja $prob(A_3, 3) = 1 - \frac{1}{k^3}$ sekä alemmassa tilanteessa $prob(A_4, End_I) = \frac{1}{k^4}$ ja $prob(A_4, 4) = 1-\frac{1}{k^4}$. Solmusta lähtevien särmien kokonaistodennäköisyys on aina yksi.

3 MC-PELIN EKSPONENTIAALINEN RATKAISUALGORITMI

3.1 Eksponentiaalinen ratkaisualgoritmi

MC-pelin eksponentiaalinen ratkaisualgoritmi perustuu Stirlingin [18] artikkelissa esitettyyn todistukseen. Stirling todistaa, että jokaiselle MC-pelille on olemassa yksikäsitteinen voittaja. Todistus esitetään tarkasti sen keskeisyyden takia. Tulosta tarvitaan myös jatkossa erikoistapausten ratkaisemisessa sekä eksponentiaalisen ratkaisualgoritmin konstruoimisessa.

Tulosta varten joudumme määrittelemään *pakotusjoukot* (force set) [18] molemmille pelaajille sekä apulauseen, jonka avulla MC-pelistä voidaan poistaa pakotusjoukon määrittelemiä solmujoukkoja. Apulause osoittaa, että poistettaessa MC-pelistä $\mathcal{G}$ pakotusjoukko X, jää jäljelle joko alipeli tai tyhjä joukko.

Pakotusjoukko $Force_P(X)$ on niiden solmujen joukko, joista pelaaja P voi pakottavasti pelata solmujoukkoon X vastustajan siirroista riippumatta.

Olkoon $G = (V, E, token)$ peli ja $X \subseteq V$. Määritellään

1. $Force_P^0(X) = X$, missä $P \in \{I, II\}$.

2. $Force_I^{i+1}(X) = Force_I^i(X) \cup \{v \in V : token(v) = I \text{ ja } \exists u \in Force_I^i(X) : (v,u) \in E\} \cup \{v \in V : token(v) = II \text{ ja } \forall (v,u) \in E : u \in Force_I^i(X)\}$.

3. $Force_{II}^{i+1}(X) = Force_{II}^i(X) \cup \{v \in V : token(v) = II \text{ ja } \exists u \in Force_{II}^i(X) : (v,u) \in E\} \cup \{v \in V : token(v) = I \text{ ja } \forall (v,u) \in E : u \in Force_{II}^i(X)\}$.

4. $Force_P(X) = \bigcup\{Force_P^i(X) : i \geq 0\}$, missä $P \in \{I, II\}$.

Siis, jos $v \in Force_P(X)$, niin pelaaja P voi toisesta pelaajasta riippumatta pelata pelin solmusta v solmujoukkoon X. Solmun *sijoitus* (rank) joukossa $Force_P(X)$ on pienin indeksi i, jolle on voimassa $v \in Force_P^i(X)$. Solmun sijoitus kertoo siirtojen lukumäärän, joka tarvitaan pelaamiseen solmujoukkoon X. Samalla solmulle saatu sijoitus antaa ohjeet voittostrategian konstruoimiseen, kuten myöhemmin todistetaan.

Esimerkki. Lasketaan kuvan 2.4 MC-pelin solmulle 2 pakotusjoukot molemmille pelaajille. Tulokseksi saadaan $Force_I(\{2\}) = \{1, 2, 5, 6, 7, 8, 10, 11\}$, missä esimerkiksi solmun 5 sijoitus on 3 sekä $Force_{II}(\{2\}) = \{2, 7, 8\}$, missä solmun 7 sijoitus on 2.

$Force_I^0(\{2\}) = \{2\}$
$Force_I^1(\{2\}) = \{2\} \cup \{6, 8\}$
$Force_I^2(\{2\}) = \{2, 6, 8\} \cup \{7, 10\}$
$Force_I^3(\{2\}) = \{2, 6, 8, 7, 10\} \cup \{5, 11\}$
$Force_I^4(\{2\}) = \{2, 5, 6, 7, 8, 10, 11\} \cup \{1\}$
$Force_I^5(\{2\}) = \{1, 2, 5, 6, 7, 8, 10, 11\} \cup \{\emptyset\}$
$Force_I(\{2\}) = \{1, 2, 5, 6, 7, 8, 10, 11\}$

$Force_{II}^0(\{2\}) = \{2\}$
$Force_{II}^1(\{2\}) = \{2\} \cup \{8\}$
$Force_{II}^2(\{2\}) = \{2, 8\} \cup \{7\}$
$Force_{II}^3(\{2\}) = \{2, 8, 7\} \cup \{\emptyset\}$
$Force_{II}(\{2\}) = \{2, 8, 7\}$

Seuraava tulos osoittaa, että poistettaessa MC-pelistä pakotusjoukko jää jäljelle joko alkuperäisen pelin alipeli tai tyhjä joukko.

Apulause 3.1 *([18]) Jos peli $\mathcal{G} = ((V, E), token)$ ja $X \subseteq V$, niin joko $Force_P(X) = V$ tai $\mathcal{G}' = \mathcal{G} - Force_P(X)$ on pelin $\mathcal{G}$ alipeli.*

Todistus. Oletetaan, että $X \subseteq V$ ja $Force_P(X) \neq V$. Olkoon $\mathcal{G}' = \mathcal{G} - Force_P(X)$. Pari $\mathcal{G}' = ((V', E'), token')$ ei ole MC-peli, eikä siis pelin $\mathcal{G}$ alipeli, jos on olemassa sellainen solmu $v \in V'$, että siitä ei lähde yhtään särmää $(v, u) \in E'$. Olkoon v tällainen solmu. Selvästi jokainen solmu u, jolle on voimassa $(v, u) \in E$, kuuluu pakotusjoukkoon $Force_P(X)$. Tällainen solmu u on olemassa, koska $\mathcal{G}$ on MC-peli. Tällöin kaikilla solmuilla $u \in Force_P^i(X)$, on olemassa pienin indeksi i, jolla se kuuluu pakotusjoukkoon. Nyt $v \in Force_P^{i+1}(X)$, mistä seuraa, että $v \in Force_P(X)$. Tämä johtaa ristiriitaan oletuksen $v \in \mathcal{G}'$ kanssa.

□

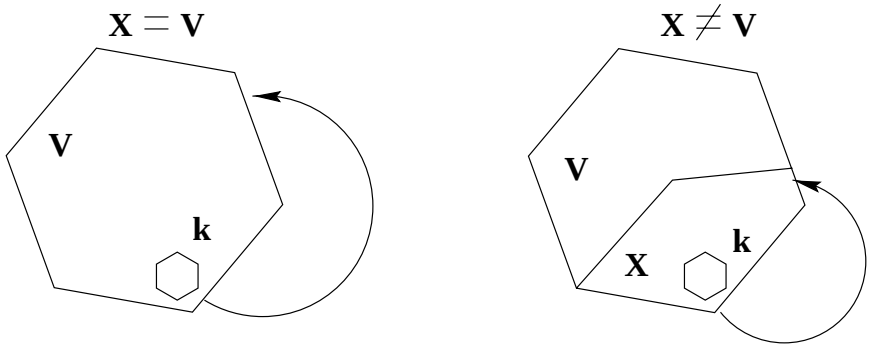
Tämän jälkeen tarkastellaan pakotusjoukkojen ominaisuuksia ja niiden käyttöä MC-pelin voittajan ratkaisemiseen. Seuraavassa kahdessa apulauseessa käsitellään pakotusjoukkojen laskentaan perustuvat tilanteet, joissa pystytään voittaja ratkaisemaan. Nämä tilanteet muodostavat eksponentiaalisen todistuksen perustapaukset.

Ensimmäinen apulause osoittaa, että jos pelin pienimmälle solmulle k muodostettu pakotusjoukko $Force_{token(k)}(k)$ käsittää pelin kaikki solmut, niin pelaajalla $token(k)$ on olemassa voittostrategia pelin kaikille solmuille.

Apulauseessa on käytetty merkintää $Min(X)$, jolla tarkoitetaan joukon X pienintä alkiota solmujen numeroinnin suhteen. Myöhemmin käytetään myös merkintää $Max(X)$, jolla tarkoitetaan joukon X suurinta alkiota.

Apulause 3.2 *Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli ja olkoon $Min(V) = k$ ja $token(k) = a$. Olkoon $X = Force_a(k)$. Jos $X = V$, niin pelaajalla a on pelissä $\mathcal{G}(v)$ voittostrategia kaikilla solmuilla $v \in V$.*

Kuva 3.1 Apulauseiden 3.2 ja 3.3 tilanteet



Todistus.

Voittostrategia koostuu seuraavista säännöistä: solmusta k valitse kaari (k, u), missä solmulla u on pienin sijoitus joukossa $X = Force_{token(k)}(\{k\})$. Kaikille muille solmuille $u' \in X$, joille pätee $token(u') = token(k)$, voittostrategia on valita kaari (u', u''), missä solmulla u'' on pienin sijoitus. Tämä on voittostrategia, koska jokaisen pelin täytyy kulkea solmun k kautta, joka on pienin joukosta V. Särmä solmusta k joukkoon X on olemassa, sillä muuten $\mathcal{G}$ ei olisi MC-peli.

□

Toisessa apulauseessa tarkastellaan tilannetta, jossa muodostettu pakotusjoukko $X = Force_{token(k)}(k)$ on joukon V aito osajoukko ja missä $Min(X) = k$. Jos tällöin on olemassa kaari solmusta k joukkoon X, niin pelaajalla $token(k)$ on olemassa voittostrategia kaikille joukon X solmuille.

Apulause 3.3 *Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli ja $X \subset V$. Olkoon $Min(X) = k$ ja $token(k) = a$. Jos $Force_a(k) = X$ ja on olemassa särmä (k, u), missä $u \in X$, niin pelaajalla a on voittostrategia pelille $\mathcal{G}(v)$ kaikilla $v \in X$.*

Todistus.

Nyt pelaajalle $token(k)$ on olemassa voittostrategia, joka koostuu seuraavista säännöistä: solmussa k pelaa särmää (k, u), missä solmun u sijoitus on pienin mahdollinen joukossa X. Kaikille muille solmuille $u' \in X$, joille pätee $token(u') = token(k)$, valitse särmä, joka johtaa solmuun, jonka sijoitus on pienin. Tämä on voittostrategia, koska jokaisen pelin täytyy kulkea solmun k kautta, joka on pienin joukosta X.

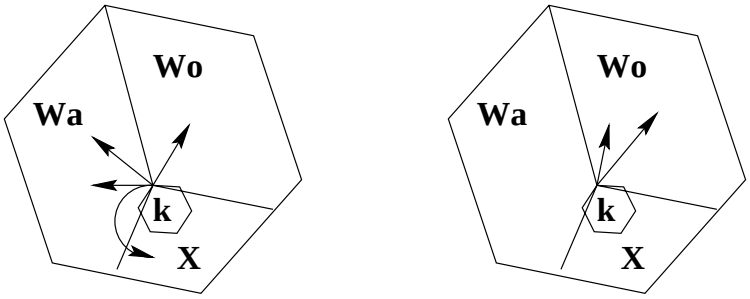
□

Laskettaessa pakotusjoukkoja tulee jokaiselle pakotusjoukkoon kuuluvalle solmulle sijoitus, joka kertoo monennellako iteraatiokierroksella solmu saadaan kuulumaan pakotusjoukkoon. Näiden sijoitusten perusteella saadaan aina määriteltyä voittostrategia, kuten edellä olleissa todistuksissa esitettiin. On siis aina siirryttävä solmuun, jolla on pienempi sijoitus kuin solmussa, jossa ollaan.

Seuraavaksi todistetaan, että jokaisella MC-pelillä on olemassa yksikäsitteinen voittaja. Todistus perustuu induktioon solmujen lukumäärän suhteen. Perustapauksina toimivat triviaali MC-peli ja edellä esitettyihin apulauseisiin 3.2 ja 3.3 perustuvat tilanteet.

Induktion käyttö on mahdollista pakotusjoukkojen laskennan avulla, jolloin saamme aina pienennettyä tarkasteltavan pelin kokoa tai ratkaistua osan solmuista.

Kuva 3.2 Lauseen 3.1 induktiotodistuksen tapaukset 1 ja 2



Lause 3.1 *([18]) Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli. Tällöin jokaiselle MC-pelille $\mathcal{G}(i)$, missä $i \in V$, on joko pelaajalle I tai II voittostrategia.*

Todistus.

Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli. Käsitellään ensin perustapaus, jossa $|V| = 1$, jolloin $V = \{1\}$. Koska $\mathcal{G}$ on MC-peli, niin on olemassa kaari $(1, 1) \in E$. Pelaajan $token(1)$ voittostrategia on valita solmussa 1 kaari $(1, 1)$.

Olkoon solmu $k = Min(V)$, $token(k) = a$ ja olkoon pakotusjoukko $X = Force_a(\{k\})$. Jos $X = V$, niin pelaajalla a on voittostrategia pelissä $\mathcal{G}(i)$ kaikille $i \in V$ apulauseen 3.2 perusteella (kuva 3.1, vasen).

Muuten $X \neq V$. Muodostetaan alipeli $\mathcal{G}' = \mathcal{G} - X$. Alipeli $\mathcal{G}'$ on nyt pienempi kuin alkuperäinen peli. Induktio-oletuksen perusteella jokaisella solmulla $u \in V'$ pelaajalla $p \in \{I, II\}$ on voittostrategia σ_p^u pelille $\mathcal{G}'(u)$. Jaetaan alipelin $\mathcal{G}'$ solmut induktio-oletuksen perusteella joukkoon W_I, jotka pelaaja I voittaa ja

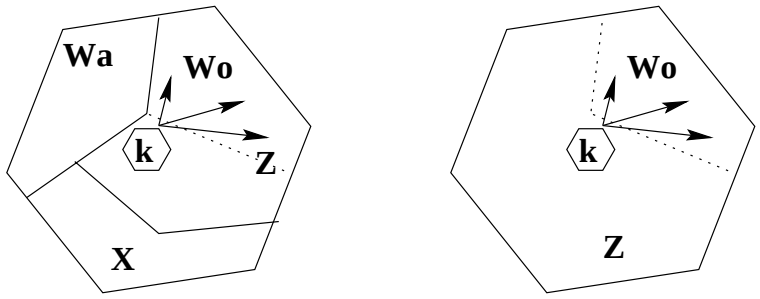
joukkoon W_{II}, jotka pelaaja II voittaa. Todistus koostuu nyt kahdesta eri tapauksesta riippuen joukosta $Y = \{u : (k, u) \in E\}$.

Tapaus 1: (Kuva 3.2, vasen) $Y \cap (W_a \cup X) \neq \emptyset$. Nyt solmusta k lähtee kaari joko takaisin pakotusjoukkoon $Force_a(\{k\})$ tai solmuun, jossa pelaajalla a on voittostrategia. On siis olemassa kaari $(k, w) \in E$, missä $w \in X$ tai $w \in W_a$. Pelaajalla a on nyt voittostrategia pelille $\mathcal{G}(i)$ kaikilla $i \in X \cup W_a$. Olkoon σ_a^{X-k} pelaajan a strategia, joka pakottaa jokaisen pelin joukosta X solmuun k. Olkoon σ_a^k stategia valita solmusta k aina särmä (k, w). Merkitään myös, että pelaajan a voittostrategia joukossa W_a on $\sigma_a^{W_a}$.

Jos $w \in X$, niin pelaajan a voittostrategia muodostuu seuraavasti: $\sigma_a^X = \sigma_a^{X-k} + \sigma_a^k$ on voittostrategia pelille $\mathcal{G}(i)$ kaikilla $i \in X$.

Jos $w \in W_a$ niin strategia $\sigma_a^{X \cup W_a} = \sigma_a^{X-k} + \sigma_a^k + \sigma_a^{W_a}$ on voittostrategia pelille $\mathcal{G}(i)$ kaikilla $i \in X \cup W_a$.

Kuva 3.3 Induktiotodistuksen tapaus 2, $Z \subset V$ ja $Z = V$



Tapaus 2: (Kuvat 3.2, oikea ja 3.3) $Y \cap (W_a \cup X) = \emptyset$. Nyt kaikki solmusta k lähtevät kaaret johtavat pelaajan a häviöön. Tämä tarkoittaa, että jokaiselle solmulle w, johon tulee kaari (k, w), on pelaajan a vastustajalla O voittostrategia σ_O^w pelille $\mathcal{G}'(w)$. Olkoon $Z = Force_O(\{k\})$ suhteessa koko graafiin. Olkoon strategia σ_O^{Z-k}, joka pakottaa pelin joukosta Z solmuun k, σ_O^k strategia solmussa k ja $\sigma_O^{W_O}$ joukon W_O solmuille määritelty voittostrategia. Nyt kaikille solmuille $i \in Z \cup W_O$ on pelaajalla O voittostrategia pelille $\mathcal{G}(i)$.

Jos $Z \cup W_O = V$, niin strategia $\sigma_O^V = \sigma_O^{Z-k} + \sigma_O^k + \sigma_O^{W_O}$ on voittostrategia pelille $\mathcal{G}(i)$ kaikilla $i \in V$ (kuva 3.3, oikea). Nyt pelin kaikkien solmujen voittostrategiat on ratkaistu.

Jos $Z \subset V$, niin strategia $\sigma_O^{Z \cup W_O} = \sigma_O^{Z-k} + \sigma_O^k + \sigma_O^{W_O}$ on voittostrategia pelille $\mathcal{G}(i)$ kaikilla $i \in Z \cup W_O$. Muodostetaan alipeli $\mathcal{G}'' = \mathcal{G} - Z$. Alipeli $\mathcal{G}''$ on nyt pienempi kuin alkuperäinen peli. Induktio-oletuksen perusteella jokaisella

solmulla $j \in V''$ on pelaajalla $p \in \{I, II\}$ voittostrategia σ_p^j pelille $\mathcal{G}'(j)$. Jaetaan alipelin $\mathcal{G}''$ solmut joukkoon W_I', jotka pelaaja I voittaa ja joukkoon W_{II}', jotka pelaaja II voittaa.

Jos $j \in W_a'$, niin σ_a^j on voittostrategia pelaajalle a pelissä $\mathcal{G}(j)$.

Muuten $j \in O$, jolloin pelaajalla O on voittostrategia pelille $\mathcal{G}(j)$. Pelaaja O käyttää strategiaa σ_O^j kunnes, jos koskaan, pelaaja a pelaa pelin solmujoukkoon $Z \cup W_O$, missä tapauksessa pelaaja O käyttää siellä määriteltyä voittostrategiaa $\sigma_O^{Z \cup W_O}$.

□

Lauseen 3.1 todistus muodostaa Stirlingin [18] mukaan eksponentiaalisen algoritmin MC-pelin voittajan ratkaisemiseen. Pelin, jonka koko on n, voittajan ratkaisemiseksi tarvitaan korkeintaan kaksi rekursiivista ratkaisukutsua peleille, joiden koko on pienempi kuin n. Ensimmäinen kutsu tehdään alussa, kun muodostetaan MC-peli $\mathcal{G}'$ ja toinen tapauksessa 2, jossa muodostetaan peli $\mathcal{G}''$. Todistuksesta konstruoitu algoritmi muodostaa samalla myös voittostrategian pelille $\mathcal{G}(v)$ kaikilla $v \in V$. Tämän perusteella eksponentiaalisen ratkaisualgoritmin aikavaatimukseksi saadaan $O(2^n)$, missä n on MC-pelin $\mathcal{G}$ solmujen lukumäärä.

Mikäli tarkastellaan vain pelin $\mathcal{G}(1)$ voittajaa, eikä kaikkien solmujen voittajia, saamme tehokkuutta lisää tarkistamalla algoritmin eri suoritusvaiheissa, kuuluuko solmu 1 ratkaistuihin solmuihin. Tarkistuksista huolimatta aikavaatimus säilyy eksponentiaalisena.

3.2 II-yksinkertaiset MC-pelit

Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli ja olkoon kaikille pelaajan II omistamille solmuille voimassa $deg_+(v) = 1$. Tällöin sanotaan, että peli on *II-yksinkertainen*. *I-yksinkertainen* MC-peli määritellään vastaavasti.

Apulause 3.4 *([18]) Olkoon $\mathcal{G} = ((V, E), token)$ II-yksinkertainen MC-peli. Tällöin MC-pelin voittaja pystytään ratkaisemaan polynomisessa ajassa.*

Todistus.

Polynomisuus perustuu siihen, ettei tarvitse missään tilanteessa muodostaa peliä $\mathcal{G}''$ induktiotodistuksen kohdassa 2.

Jos käsiteltävästä solmusta k lähtee kaari solmuun u, jossa pelaajalla $token(k)$ on voittostrategia, saamme ratkaistua pelin muodostamalla alipelin $\mathcal{G}' = \mathcal{G} - X$. Nyt joudutaan tarkastelemaan vain induktiotodistuksen tapausta 1.

Jos kaikki solmusta k lähtevät kaaret johtavat vastustajan voittoon, muodostetaan pakotusjoukko $Z = Force_O(k)$. Nyt pakotusjoukko Z koostuu pelin kaikista solmuista. Tämä perustuu vastustajan solmujen lähtöasteeseen, joka on aina 1, mikäli joudumme käsittelemään induktiotodistuksen tapausta 2.

□

Esimerkki.

Kuvassa 3.3 vasemmalla joukko Z kattaa kaikki ne solmut, jotka kuuluvat joukkoon W_O ja osan joukon X solmuista. Nyt joudumme muodostamaan pelin $\mathcal{G}''$. Kuvan oikenpuoleisessa tilanteessa käsitellään II-yksinkertaista MC-peliä. Emme joudu nyt muodostamaan peliä $\mathcal{G}''$, vaan saamme ratkaistua tilanteen välittömästi.

Apulause 3.5 *([18]) MC-pelin päätösongelma kuuluu luokkaan $NP \cap co - NP$.*

Todistus.

Arvataan voittostrategia σ_{II} pelaajalle II. Poistetaan pelistä kaikki ne pelaajan II solmuista lähtevät särmät, jotka eivät kuulu voittostrategiaan σ_{II}. Tuloksena saadaan II-yksinkertainen MC-peli, jonka voittaja on ratkaistavissa polynomisessa ajassa. Tämä osoittaa, että päätösongelma kuuluu luokkaan NP.

Määritellään MC-pelin komplementti vaihtamalla solmujen omistajaa. Jos $\mathcal{G} = ((V, E), token)$, niin $\mathcal{G}' = ((V, E), token')$, missä $token'(i) = I$, jos $token(i) = II$ ja $token'(i) = II$, jos $token(i) = I$. Nyt pelaajalla on voittostrategia pelissä $\mathcal{G}(i)$, jos hänen vastustajallaan on voittostrategia pelissä $\mathcal{G}'(i)$.

□

4 Puumaisten MC-pelien ratkaiseminen

4.1 Ratkaisemisen yleinen toimintaperiaate

Useimmat tässä työssä esitettävät MC-pelin erikoistapausten polynomiset ratkaisualgoritmit perustuvat syvyyssuuntaiseen etsintään. Ainoa poikkeus on pakotusjoukkojen laskentaan perustuva puumaisten MC-pelien ratkaisualgoritmi, joka esitetään tämän luvun viimeisessä kappaleessa.

Syvyyssuuntaiseen etsintään perustuvat algoritmit käyvät useimmissa tapauksissa läpi pelin kaikki solmut ja särmät. Algoritmit keräävät etsinnän aikana informaatiota solmujen ja särmien ominaisuuksista ja tekevät peruuttamisen yhteydessä tarvittavat ratkaisut solmujen ja särmien voittajan selvittämiseksi. Kaikki tässä luvussa esitetyt lauseet ja apulauseet ovat voimassa myös syklejä sisältävissä MC-peleissä, mutta niitä ei tällöin ole aina mahdollista käyttää.

Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli. Pelin $\mathcal{G}$ solmu v on *lopullisesti ratkaistu*, kun tiedetään MC-pelin $\mathcal{G}(v)$ voittaja. Solmun voittajaa merkitään $win(v) = a$, missä $v \in V$ ja $a \in \{I, II\}$. MC-pelin särmän (v, u) voittaja $win(v, u)$ on pelaaja, joka voittaa pelin $\mathcal{G}(u)$.

Solmun arvottamisella tarkoitetaan peruutuksen yhteydessä tapahtuvia ratkaisualgoritmiin liittyvien numeeristen arvojen päivittämistä.

Särmän valinnalla tarkoitetaan solmun v arvottamisen yhteydessä valittua särmää (v, u), jonka perusteella solmun voittaja ratkaistiin. Valitut särmät muodostavat voittostrategian solmussa v pelaajalle $win(v)$.

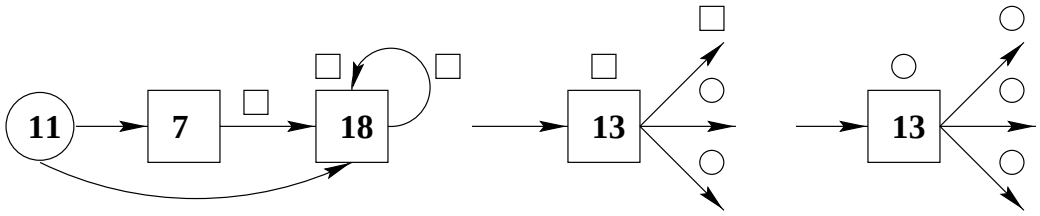
4.2 Vahvasti puumaisen MC-pelin ratkaiseminen

Tässä kohdassa todistetaan, että kaikki vahvasti puumaiset MC-pelit ovat ratkaistavissa polynomisessa ajassa. Asian todistamiseksi esitetään neljä apulausetta, jotka käsittelevät kaikki vahvasti puumaisissa MC-peleissä vastaan tulevat tilanteet.

Ensimmäinen apulause tarkastelee tilannetta, jossa käsiteltävällä solmulla on itsesilmukka. Tällöin solmun voittaja on helppo ratkaista. Esimerkkitilanne löytyy kuvan 4.1 vasemman puoleisesta alipelistä. Kuvassa pienellä piirretyt ympyrät ja neliöt kuvaavat särmien ja solmujen voittajia.

Apulause 4.1 *Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli, jonka solmulla v on itsesilmukka $(v, v) \in E$ ja olkoon solmun v omistaja $token(v) = a$. Tällöin särmän (v, v) voittaja $win(v, v)$ on pelaaja a ja solmun v voittaja $win(v)$ on pelaaja a.*

Kuva 4.1 Apulauseiden 4.1-4.4 käyttö puumaisen MC-pelin ratkaisemisessa



Todistus.

Olkoon solmussa v itsesilmukka. Pelaajan $token(v) = a$ voittostrategia on aina valita särmä (v, v), joka johtaa takaisin solmuun v. Tällöin solmun v omistaja omistaa pienimmän solmun, jossa käydään äärettömän usein. Särmälle (v, v) on siis voimassa $win(v, v) = a$ ja solmulle v on voimassa $win(v) = a$.

□

Seuraavassa apulauseessa tarkastellaan tilannetta, jossa olemme jo aiemmin löytäneet solmun v ja ratkaisseet sen voittajan. Nyt kaikki solmuun v tulevat särmät johtavat pelaajan $win(v)$ voittoon.

Apulausetta käytetään algoritmissa 2 kahdessa tilanteessa. Kun etsintä peruuttaa solmusta v solmuun $f(v)$, voimme sijoittaa särmän $(f(v), v)$ voittajaksi solmun v voittajan. Voimme käyttä sitä myös aina, kun löydämme solmun, joka on jo ratkaistu.

Kuvassa 4.1 vasemman puoleisessa alipelissä näkyy tilanne, jossa etsintä on ratkaissut solmun 18 ja peruutamme sen vanhempaan. Nyt pystymme ratkaisemaan särmän $(f(18), 18)$ voittajan. Samassa kuvassa apulausetta on mahdollista käyttää myös siinä vaiheessa, kun algoritmi tutkii särmää $(11, 18)$. Tällöin käsiteltävän särmän maalisolmu on jo ratkaistu. Jälkimmäistä tilannetta ei esiinny vahvasti puumaisissa MC-peleissä.

Apulause 4.2 *Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli, jonka solmun $v \in V$ voittaja $win(v) = a$. Tällöin jokaiselle särmälle $(v', v) \in E$ on voimassa $win(v', v) = a$.*

Todistus.

Olkoon solmulle v voimassa $win(v) = a$. Jokainen särmä, joka tulee solmuun v, johtaa pelaajan $win(v)$ voittoon, koska solmussa v pelaajalle $win(v)$ on voittostrategia.

□

Apulauseet 4.3 ja 4.4 tuovat mieleen lauseen 3.1 todistuksessa läpikäydyt tilanteet, joissa solmusta k lähti vähintään yksi särmä solmun k omistajan voittamaan solmuun (tapaus 1) tai kaikki solmusta k lähtevät särmät johtivat vastustajan voittoon (tapaus 2). Nyt nämä eksponentiaalisen ratkaisun tilanteet määritellään konkreettisemmin. Esimerkit kahden seuraavan apulauseen käytöstä löytyy kuvasta 4.1 keskeltä ja oikealta.

Apulause 4.3 *Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli, jonka solmusta $v \in V$ jokaiselle lähtevälle särmälle $(v, v') \in E$ on voimassa $win(v, v') = a$. Tällöin solmun v voittaja $win(v) = a$.*

Todistus.

Jos jokainen solmusta v lähtevä särmä johtaa pelaajan a voittoon, voittaa pelaaja a solmun v omistajan siirrosta riippumatta solmun v, koska jokaisella solmun v jälkeläisellä on pelaajalla a voittostrategia.

□

Apulause 4.4 *Olkoon $\mathcal{G} = ((V, E), token)$ MC-peli, jonka solmusta $v \in V$ vähintään yhdelle lähtevälle särmälle $(v, v') \in E$ on voimassa $win(v, v') = token(v)$. Tällöin on $win(v) = token(v)$.*

Todistus.

Olkoon $token(v) = win(v, u)$. Nyt siis solmussa u on pelaajalla $win(v, u)$ olemassa voittostrategia $\sigma^u_{token(v)}$. Koska pelaaja $win(v, u)$ omistaa solmun v, voi hän aina solmusta v valita särmän (v, u). Nyt pelaajan $token(v)$ voittostrategia solmussa v on (v, u).

□

Algoritmi 2 Vahvasti puumaisen MC-pelin ratkaisualgoritmi

Muuttujat: Käsiteltävä solmu v, solmun v vanhempi $f(v)$, solmun v dfs-numero $k(v)$, solmun v voittaja $win(v)$, solmun v omistaja $token(v)$, särmän (v, u) voittaja $win(v, u)$.

Syöte: Vahvasti puumainen MC-peli $\mathcal{G}$ ja aloitussolmu s.

Tulos: MC-pelin $\mathcal{G}(s)$ voittaja.

1: **Alustaminen**. Merkitse graafin kaikki särmät käyttämättömiksi. Sijoita kaikille solmuille $k(v) \leftarrow 0$, $win(v) \leftarrow 0$, $f(v) \leftarrow -1$ ja särmille $win(v, u) \leftarrow 0$. Sijoita $i \leftarrow 0$ ja $v \leftarrow s$. Solmu s on aloitussolmu.

2: Sijoita $i \leftarrow i + 1$, $k(v) \leftarrow i$.

3: Jos solmusta v lähtee särmä, jolle on voimassa $token(v) = win(v, u)$, niin mene kohtaan arvottaminen.

4: Jos solmun kaikki särmät on käyty läpi, niin mene kohtaan arvottaminen.

5: Valitse käyttämätön särmä (v, u). Merkitse särmä käytetyksi. Jos $k(u) = 0$, niin sijoita $f(u) \leftarrow v$, $v \leftarrow u$ ja mene kohtaan 3.

6: $(u = v)$ Sijoita $win(v, v) \leftarrow token(v)$ ja mene kohtaan 3.

7: **Arvottaminen**. Jos solmusta v jokaiselle lähtevälle särmälle (v, v') on voimassa $win(v, v') = a$, niin sijoita $win(v) \leftarrow a$. Mene kohtaan peruutus.

8: Jos solmusta v lähtee särmä, jolle on voimassa $token(v) = win(v, v')$, niin sijoita $win(v) \leftarrow token(v)$. Mene kohtaan peruutus.

9: **Peruutus**. Jos $f(v) \neq -1$, niin sijoita $win(f(v), v) \leftarrow win(v)$, $v \leftarrow f(v)$ ja mene kohtaan 3.

10: **Lopetus**. $(f(v) = -1)$. MC-pelin $\mathcal{G}(s)$ voittaja on pelaaja $win(v)$. Lopeta.

Lause 4.1 *Algoritmi 2 ratkaisee vahvasti puumaisen MC-pelin $\mathcal{G}$ voittajan ajassa $O(E)$.*

Todistus.

Särmän arvottaminen tapahtuu ensimmäisen kerran syvyyssuuntaisen etsinnän löydettyä itsesilmukan. Aina itsesilmukan (v, v) löydyttyä voimme käyttää apulausetta 4.1, jonka perusteella saamme itsesilmukallisen solmun voittajan ratkaistua. Koska solmu on nyt ratkaistu, peruuttaa algoritmi käsitellyn solmun vanhempaan. Voimme merkitä särmän $(f(v), v)$ voittajaksi solmun v voittajan (apulause 4.2) ja peruuttaa solmuun $f(v)$ sekä sijoittaa $v \leftarrow f(v)$. Mikäli solmun v ratkaisua ei peruutuksen jälkeen voida tehdä apulauseiden 4.3 tai 4.4 avulla, on solmussa v tutkimattomia särmiä. Nyt algoritmi alkaa tutkimaan läpikäymättömiä särmiä, kunnes solmu v saadaan ratkaistua.

Kun solmun kaikki lähtevät särmät on käyty läpi, algoritmi pystyy tekemään ratkaisun solmun voittajasta ja peruuttamaan solmun v vanhempaan ja sijoittamaan taas solmun $f(v)$ käsiteltäväksi solmuksi.

Tätä jatketaan, kunnes etsintä pystyy ratkaisemaan aloitussolmun.

□

Itsesilmukallisten solmujen olemassaolo perustuu siihen MC-pelin ominaisuuteen, että jokaisesta solmusta on lähdettävä vähintään yksi särmä. Tutkittaessa läpikäymättömiä särmiä ei tule ikinä vastaan tilannetta, jossa maalisolmu olisi jo ratkaistu. Tämä perustuu käsiteltävän pelin vahvasti puumaisuuteen.

4.3 Puumaisen MC-pelin ratkaiseminen

Mikäli MC-peli on puumainen, voi syvyyssuuntainen etsintä löytää siitä sellaisia solmuja, jotka on jo ratkaistu. Käytännössä tämä tarkoittaa sitä, että syvyyssuuntainen etsintä voi löytää menokaaria sekä ristikkäiskaaria puu- ja paluukaarten lisäksi. MC-peli pysyy kuitenkin edelleen syklittömänä itsesilmukoita lukuunottamatta.

Lause 4.2 *Algoritmi 2, jonka kohta 6 on muutettu seuraavanlaiseksi:*

(6) Jos $u = v$, niin sijoita $win(v, v) \leftarrow token(v)$ ja mene kohtaan 3. Jos $k(u) > k(v)$, niin sijoita $win(v, u) \leftarrow win(u)$ ja mene kohtaan 3. ($k(u) < k(v)$ ja $win(u) \neq 0$) Sijoita $win(v, u) = win(u)$ ja mene kohtaan 3.

ratkaisee puumaisen MC-pelin $\mathcal{G} = ((V, E), token)$ voittajan ajassa $O(E)$.

Todistus.

Olkoon (v, u) syvyyssuuntaisen etsinnän MC-pelistä löytämä ristikkäis- tai menokaari. Koska syvyyssuuntainen etsintä on peruuttanut aiemmin solmusta u, on solmun u voittaja jo ratkaistu. Voimme siis käyttää aina särmän (v, u) arvottamiseen apulausetta 4.2.

Muissa tilanteissa ratkaiseminen tapahtuu kuten lauseen 4.2 todistuksessa esitetään.

□

4.4 Pakotusjoukkojen käyttö ratkaisemisessa

Puumaisen MC-pelin ratkaiseminen on mahdollista tehdä myös käyttäen pakotusjoukkoja. Algoritmin toiminta perustuu itsesilmukallisten solmujen ratkeavuu-

teen ja niiden perusteella muodostettujen pakotusjoukkojen poistamiseen tarkasteltavasta pelistä.

Algoritmi 3 Puumaisen MC-pelin ratkaisualgoritmi

Muuttujat: Käsiteltävä solmu v, solmun v pakotusjoukko $Force_{token(v)}(v)$ pelaajalle $token(v)$ ja aloitussolmu s.

Syöte: Puumainen MC-peli $\mathcal{G}$.

Tulos: MC-pelin $\mathcal{G}(s)$ voittaja.

1: Olkoon k MC-pelin $\mathcal{G}$ solmu, jossa on itsesilmukka.
2: Muodosta pakotusjoukko $X = Force_{token(k)}(k)$.
3: Jos solmu $s \notin X$, niin muodosta $\mathcal{G}' = \mathcal{G} - X$ ja ratkaise $\mathcal{G}'$.
4: Nyt $s \in X$. Pelin voittaja on pelaaja $token(k)$. Lopeta.

Lause 4.3 *Olkoon $\mathcal{G} = ((V, E), token)$ puumainen MC-peli. Tällöin algoritmi 3 ratkaisee pelin $\mathcal{G}$ voittajan polynomisessa ajassa.*

Todistus.

Koska itsesilmukallisen solmun v voittaja on pelaaja $token(v) = a$ (apulause 4.1), niin pelaaja a voittaa ne solmut, joista hän pystyy pakottamaan pelin etenemään tähän itsesilmukalliseen solmuun. Voimme siis muodostaa pakotusjoukon $X = Force_a(\{v\})$. Jos pelin aloitussolmu kuuluu tähän pakotusjoukkoon, on peli ratkaistu. Muussa tapauksessa voimme tarkastella pienempää peliä $\mathcal{G}' = \mathcal{G} - X$. Jatkamalla pakotusjoukkojen muodostamista itsesilmukallisille solmuille tulee aikanaan vastaan tilanne, jossa aloitussolmu kuuluu muodostettuun pakotusjoukkoon. Tällöin saamme pelin voittajan selville.

□

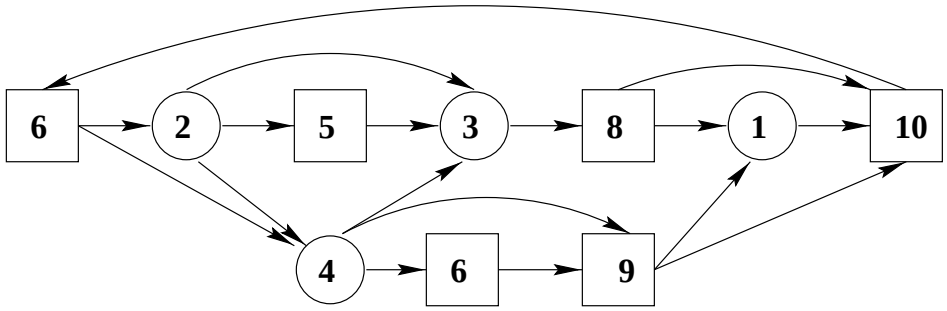
5 Polynomisessa ajassa ratkeavia syklisiä erikoistapauksia

5.1 Syklikiinnitettyjen komponenttien ratkaiseminen

Olkoon K MC-pelin $\mathcal{G}$ maksimaalinen vahvasti yhtenäinen itsesilmukaton eristetty komponentti, jonka jokainen sykli kulkee särmän (v, u) kautta. Tällöin komponenttia K sanotaan *syklikiinnitetyksi*.

Esimerkki syklikiinnitetystä komponentista K on kuvassa 5.1. Komponentin K jokainen sykli kulkee särmän $(10, 6)$ kautta.

Kuva 5.1 Syklikiinnitetty komponentti



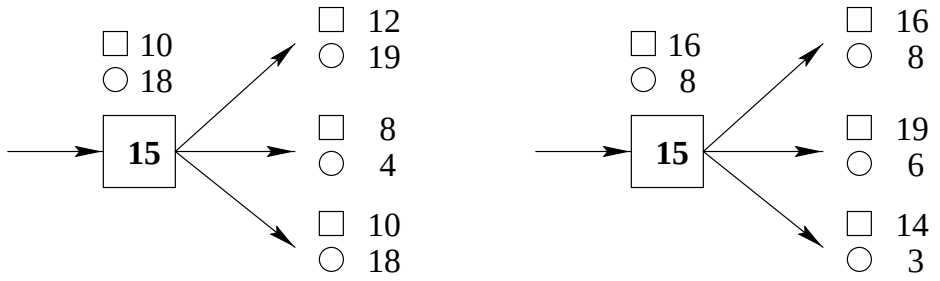
Apulause 5.1 *Olkoon K syklikiinnitetty komponentti ja olkoon (v, u) kaari, joka kuuluu komponentin jokaiseen sykliin. Tällöin solmusta u alkavan syvyyssuuntaisen etsinän löytämä ainoa paluukaari on (v, u).*

Todistus. Tehdään vastaoletus, että syvyyssuuntainen etsintä löytää useampia paluukaaria. Olkoon särmä $(v', u') \neq (v, u)$ komponentista löytynyt toinen paluukaari. Nyt voimme muodostaa syklin C, johon ei sisälly kaarta (v, u). Sykli C muodostuu puukaarista solmusta u' solmuun v' ja paluukaaresta (v', u'). Tämä johtaa ristiriitaan syklikiinnitetyn komponentin määritelmän kanssa.

□

Nyt jokainen sykli joutuu kulkemaan löydetyn paluukaaren (v, u) kautta. Komponentista löydetyt puu-, ristikkäis- ja menokaaret eivät kierrä löydettyä paluukaarta. Ne ainoastaan ohittavat solmuja paluukaaren virittämästä syklistä. Komponentin ratkaisemiselle ei ole merkitystä sillä, mitä kautta ainoa paluukaari löydettiin.

Kuva 5.2 Etuminimin laskeminen



Syklikiinnitettyjen komponenttien voittajan ratkaiseminen perustuu peruutuksen yhteydessä tapahtuvaan tiedon keräämiseen ja käsittelyyn. Etsinnän aikana otetaan muistiin molempien pelaajien pieninumeroisimmat solmut, jotka sijaitsevat edessäpäin suhteessa syvyyssuuntaiseen etsintään. Näiden tietojen perusteella pystytään tekemään särmien valinnat ja ratkaisemaan komponentin voittaja.

Solmun v *etuminimi* pelaajalle I on pienin solmu, joka on solmun v jälkeläinen tai komponentin ainoan paluukaaren maalisolmu, johon pelaajan I on mahdollista pelata ja jonka pelaaja I omistaa. Etuminimi pelaajalle II määritellään vastaavasti. Etuminimistä käytetään merkitää $min_P(v)$, missä $P = \{I, II\}$. Särmän (v, u) etuminimi $min_P(v, u) = min_P(u)$ on maalisolmun u etuminimi.

Tarkastellaan etuminimien laskemista seuraavalla sivulla esitettävän algoritmin 4 mukaisesti. Lähtötilanteessa kaikkien solmujen etuminimit on alustettu nolliksi. Etuminimien kerääminen alkaa siinä vaiheessa, kun syvyys-suuntainen etsintä löytää ensimmäistä kertaa solmun, jossa ollaan jo käyty. Tämä tapahtuu paluukaaren (v, u) löydyttyä. Olkoon solmulle u voimassa $token(u) = a$. Nyt sijoitetaan pelaajan a etuminimiksi $min_a(v, u) \leftarrow Min(min_a(u), u)$ ja pelaajan O etuminimiksi sijoitetaan $min_O(v, u) \leftarrow min_O(u)$.

Peruutuksen yhteydessä joudutaan valintatilanteeseen. Jos solmulla on useita jälkeläisiä, minkä särmän pelaaminen on optimaalisinta solmun omistajalle? Tässä vaiheessa tiedämme jokaisen solmusta v lähtevän särmän etuminimin, koska syvyyssuuntainen etsintä on käynyt läpi kaikki solmun v jälkeläiset.

Nyt pelattavan särmän valinta perustuu kahteen heuristiikkaan, jotka esitellään seuraavaksi. Näiden heuristiikkojen toimivuus tässä tilanteessa perustuu siihen, että löydettäessä komponentin ensimmäinen paluukaari, ei siinä tilanteessa voitu tehdä muuta valintaa. Tämän jälkeen jokainen sykli joutuu käymään paluukaaren maalisolmussa. Kun peruutetaan ja löydetään uusia paluukaaren virit-

Algoritmi 4 Syklikiinnitetyn komponentin ratkaisualgoritmi

Muuttujat: Käsiteltävä solmu v, solmun v vanhempi $f(v)$, solmun v dfs-numero $k(v)$, solmun v etuminimit $min_I(v)$ ja $min_{II}(v)$ pelaajalla I ja pelaajalla II, solmun v omistaja $token(v)$ sekä särmän (v,u) etuminimit $min_p(v,u)$, missä $p \in \{I, II\}$.

Syöte: MC-pelin vahvasti yhtenäinen 1-paluukaarinen itsesilmukaton komponentti ja särmä (t,s), joka kuuluu komponentin jokaiseen sykliin.

Tulos: MC-pelin $\mathcal{G}(s)$ voittaja.

1: **Alustaminen**. Merkitse graafin kaikki särmät käyttämättömiksi. Sijoita kaikille solmuille $k(v) \leftarrow 0$, $f(v) \leftarrow -1$, $min_I(v) \leftarrow 0$, $min_{II}(v) \leftarrow 0$ ja särmille $min_I(v,u) \leftarrow 0$, $min_{II}(v,u) \leftarrow 0$. Sijoita $i \leftarrow 0$ ja $v \leftarrow s$.

2: Sijoita $i \leftarrow i+1$ ja $k(v) \leftarrow i$.

3: Jos solmun kaikki särmät on käyty läpi, mene kohtaan arvottaminen.

4: Valitse käyttämätön särmä (v,u). Merkitse särmä käytetyksi. Jos $k(u) = 0$, niin sijoita $f(u) \leftarrow v$, $v \leftarrow u$ ja mene kohtaan 3.

5: $(k(u) \neq 0)$ Olkoon $token(u) = a$. Sijoita $min_a(v,u) \leftarrow Min(min_a(u), u)$ ja $min_O(v,u) \leftarrow min_O(u)$. Mene kohtaan 3.

6: **Arvottaminen**. Olkoon $E_I = \{(v,u_i) : 0 < min_I(v,u_i) < min_{II}(v,u_i) \cup (v,u_i) : min_{II}(v,u_i) = 0\}$ ja $E_{II} = \{(v,u_i) : min_I(v,u_i) > min_{II}(v,u_i) > 0 \cup (v,u_i) : min_I(v,u_i) = 0\}$.

7: Olkoon $token(v) = a$. Jos $E_a = \emptyset$, niin mene kohtaan 8. Olkoon $(v,v') : Min\{min_a(v,v') : (v,v') \in E_a\}$. Sijoita $min_a(v) \leftarrow min_a(v,v')$ ja $min_O(v) \leftarrow min_O(v,v')$. Mene kohtaan peruutus.

8: $(E_a = \emptyset)$ Olkoon $(v,v') : Max\{min_O(v,v') : (v,v') \in E_O\}$. Sijoita $min_a(v) \leftarrow min_a(v,v')$ ja $min_O(v) \leftarrow min_O(v,v')$. Mene kohtaan peruutus.

9: **Peruutus**. Jos $f(v) = -1$, niin mene kohtaan lopetus. Olkoon $token(v) = a$. Jos $min_a(v) \neq 0$, niin sijoita $min_a(f(v),v) \leftarrow Min\{min_a(v), v\}$, muuten sijoita $min_a(f(v),v) \leftarrow min_a(v)$. Sijoita $min_O(f(v),v) \leftarrow min_O(v)$. Sijoita $v \leftarrow f(v)$. Mene kohtaan 3.

10: **Lopetus**. Jos $0 < min_I(v) < min_{II}(v)$, niin pelin $\mathcal{G}(s)$ voittaja on pelaaja I, muuten pelaaja II. Lopeta.

tämää sykliä kiertäviä polkuja, voivat nämä *ohituspolut* parantaa tai huonontaa sen pelaajan tilannetta, jolla on mahdollisuus valita ohituspolun käyttö.

Kun syvyyssuuntainen etsintä on käynyt solmun v kaikki särmät läpi, muodostetaan etuminimien perusteella joukot $E_I = \{(v,u) : f(u) = v : 0 < min_I(v,u) < min_{II}(v,u) \cup (v,u_i) : min_{II}(v,u_i) = 0\}$ ja $E_{II} = \{(v,u) : f(u) = v : min_I(v,u) > min_{II}(v,u) > 0 \cup (v,u_i) : min_I(v,u_i) = 0\}$. Joukko E_I koostuu nyt niistä särmistä, joissa pelaajalla I on pienempi etuminimi kuin pelaajalla II. Joukko E_{II} on muodostettu vastaavasti.

Heuristiikka 5.1 *Jos $E_{token(v)} \neq \emptyset$, niin valitse särmä (v,v'), jolle on voimassa $(v,v') = Min\{min_{token(v)}(v,v') : (v,v') \in E_{token(v)}\}$.*

Heuristiikka 5.2 *Jos $E_{token(v)} = \emptyset$, niin valitse särmä (v,v'), jolle on voimassa $(v,v') : Max\{min_O(v,v') : (v,v') \in E_O\}$.*

Valintatilanteessa on luonnollisinta valita särmä, joka johtaa solmun v omistajalle $token(v) = a$ parhaimpaan tilanteeseen. Jos $E_a \neq \emptyset$, valitaan sellainen särmä $(v,u) \in E_a$, jolla on pienin etuminimi pelaajalle a. Nyt vastustajan on mahdollisimman vaikeaa alittaa valitun särmän etuminimiä.

Jos kaikilla särmillä on vastustajalla pienempi etuminimi, pitää valita sellainen särmä, jolle vastustajan etuminimi on mahdollisimman suuri. Nyt pelaajalla a on suurimmat mahdollisuudet alittaa vastustajan etuminimi.

Lause 5.1 *Olkoon K MC-pelin $\mathcal{G}$ syklikiinnitetty komponentti. Tällöin algoritmi 4 ratkaisee komponentin voittajan ajassa $O(E)$.*

Todistus.
Sivuutetaan.

□

Syklikiinnitetty komponentti määriteltiin siten, että komponentin jokaisella syklillä oli vähintään yksi yhteinen särmä (v,u). Tällä hetkellä on avoimena kysymyksenä, että olisiko mahdollista määritellä syklikiinnitetty komponentti yhden, kaikille sykleille yhteisen, solmun u avulla.

Määrittelemällä syklikiinnittyneisyys suhteessa solmuun, eikä särmään, saataisiin hieman laajempi ratkeavien MC-pelien joukko.

5.2 Pakottava polku ja sykli

Olkoon v käsiteltävänä oleva solmu ja olkoon P sellainen suora polku syvyyssuuntaisen etsinnän aloitussolmusta solmuun v, että se koostuu pelkästään puukaaristä. Tällöin polkua P kutsutaan *syvyyssuuntaisen etsinnän määräämäksi poluksi.*

Olkoon $P = v_0, v_1, v_2, \ldots, v_{n-1}, v_n$ MC-pelin $\mathcal{G}$ polku ja V_a niiden polun P solmujen joukko, joiden omistaja on a ja V_O pelaajan a vastustajan omistamien solmujen joukko. Mikäli $V_a = \emptyset$ tai kaikilla niillä särmillä (v', u'), missä $v' \in V_a$ ja $(v', u') \notin P$, on voimassa $win(v', u') = O$, sanotaan polkua P *pakottavaksi pelaajalle* a. Polku voi olla pakottava myös molemmille pelaajille. Polun pakottavuus merkitsee sitä, että toisen pelaajista on pakko pelata sitä pitkin, koska muut pelitavat johtavat häviöön. Määrittely eroaa luvussa 3 esitetystä pakotusjoukkojen määritelmästä siten, että jos on pystytty jo todistamaan vastustajan omistamien solmujen särmien johtavan häviöön, voimme jättää niiden pelaamisen huomioimatta.

Polun pakottavuus on aina riippuvainen syvyyssuuntaisen etsinnän etenemisestä ja sen tarkistaminen ei aina tuo samaa tulosta samalle MC-pelille suoritettavilla etsinnöillä.

Apulause 5.2 *Olkoon MC-pelin $\mathcal{G}$ polku $P = v_0, v_1, v_2, \ldots, v_{n-1}, v_n, v_0$ pakottava pelaajalle a ja olkoon $win(v_n) = O$. Tällöin $win(v) = O$ polun P kaikilla solmuilla v.*

Todistus.

Todistus perustuu peruuttamisen yhteydessä tapahtuvaan solmujen arvottamiseen ja aiemmin esitettyjen apulauseiden käyttöön. Kun käsiteltävänä oleva solmu kuuluu pelaajalle a, voidaan käyttää apulausetta 4.3 ja oltaessa pelaajan O solmussa apulausetta 4.4.

□

Apulause 5.3 *Olkoon MC-pelin $\mathcal{G}$ sykli $C = v_0, v_1, v_2, ..., v_{n-1}, v_n, v_0$ pakottava pelaajalle a, $Min(C) = k$ ja $token(k) = O$. Tällöin $win(v) = O$ syklin C kaikilla solmuilla v.*

Todistus.

Pelaajan a on aina pakko pelata sykliä C pitkin, koska kaikki muut pelitavat johtavat häviöön. Pelaaja O omistaa kuitenkin syklin C pieninumeroisimman solmun, jossa käydään äärettömän usein. Pelaajalla O on siis olemassa kaikille

solmuille $v_i \in C$ voittostrategia. Solmussa $v_i \in C$ pelaa solmuun v_{i+1} tai, jos $i = n$, pelaa solmusta v_n solmuun v_0.

□

Apulause 5.3 mahdollistaa siis paluukaarien ratkaisemisen silloin, kun löytynyt sykli on pakottava. Polun pakottavuuden tarkistaminen paluukaaren (v, u) löydyttyä tapahtuu tarkistamalla syvyyssuuntaisen etsinnän määräämän polun solmut alkaen solmusta u solmuun v asti.

6 Syklisten MC-pelien ongelmatilanteet

6.1 Usean syklin ongelma

Edellisessä luvussa tutkittiin tilannetta, jossa komponentin jokaisella syklillä oli vähintään yksi yhteinen särmä. Tällöin komponentin voittaja pystyttiin ratkaisemaan etuminimien laskennan avulla.

Samoin ratkaisun tekeminen oli mahdollista, mikäli syvyyssuuntaista etsintää käytettäessä löytyneen paluukaaren virittämä sykli on pakottava.

Yleisessä tilanteessa komponentin sykleillä ei välttämättä ole yhtään yhteisiä solmuja tai särmiä. Tällöin komponentille suoritettu syvyyssuuntainen etsintä voi löytää useita paluukaaria komponentin sisältä.

Tällä hetkellä ei siis ole olemassa mitään yleisesti toimivaa heuristiikkaa, jonka avulla pystyisimme arvottamaan paluukaaret välittömästi niiden löytymisen jälkeen.

Paluukaaren ratkaiseminen olisi mahdollista myös tehdä jälkikäteen, vastaavasti kuten syklikiinnitettyjen komponenttien ratkaisualgoritmissa. Tällöin pitäisi käydä läpi ratkaisemisen kannalta riittävä määrä solmuja ja ratkaista vasta niiden perusteella voittaja. Seuraavassa ratkaisujoukkoja käsittelevässä kohdassa hahmotellaan riittävän solmumäärän ja ratkeavien alipelien käsitteitä.

Oikean toimintatavan löytäminen tähän tilanteeseen on ongelma. On mahdollista, että oikean ratkaisun tekeminen käytettävissä olevalla informaatiolla on mahdotonta. Tämä johtaisi siihen, että ongelman ratkaiseminen puhtaasti syvyyssuuntaiseen etsintään perustuvilla algoritmeilla ei olisi mahdollista.

6.2 Ratkaisujoukot

Olkoon K MC-pelin $\mathcal{G}$ maksimaalinen vahvasti yhtenäinen komponentti ja olkoon kaari $(v, u) \in K$ syvyyssuuntaisella etsinnällä löytynyt paluukaari. Mikäli kaaren (v, u) ratkaisemista ei voida tehdä välittömästi peruutettaessa kaarta (v, u) pitkin, niin milloin kaaren ratkaiseminen on mahdollista, tai milloin se viimeistään pitäisi ratkaista?

On ilmeistä, että komponentin K jokainen solmu pitää saada ratkaistuksi viimeistään silloin, kun komponentin kaikki solmut on käsitelty ja peruutetaan pois komponentista.

Maksimaalinen komponentti voi sisältää kuitenkin useita eri solmujoukkoja, joilla voi olla eri voittajat. Siksi tarkastelemme MC-pelin $\mathcal{G} = ((V, E), token)$ *ratkaisujoukkojen* määrää. Määritellään, että solmujoukko $V' \subseteq V$ on ratkaisu-

joukko, jos graafia $\langle V' \rangle$ vastaava suuntaamaton graafi on yhtenäinen ja kaikilla joukon V' solmuilla v on voimassa $win(v) = a$. Ratkaisujoukon jokaisella solmulla on siis sama voittaja.

Ratkaisujoukkojen lukumäärä ei anna suoraan apua pelin voittajan ratkaisemiseen. Se antaa kuitenkin mahdollisuuksia ymmärtää ongelmien luonnetta ratkaistaessa syklisiä MC-pelejä. Ratkaisujoukkojen lukumäärän tietäminen auttaa rajaamaan alueita, joiden voittaja tulee selvittää. Pystymme siis tunnistamaan alueita, joita ei ehkä pystytä ratkaisemaan, ja huomioimaan ne ratkaisualgoritmeissa.

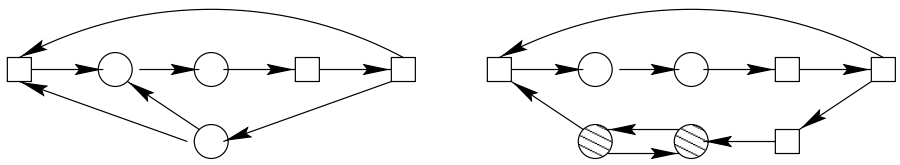
Ongelmana ratkaisujoukkojen määritelmässä on niiden sitoutuneisuus graafin syklien lukumäärään. Ja kuten kappaleessa 2.5 todettiin, voi syklejä olla graafissa eksponentiaalinen määrä.

Apulause 6.1 *Olkoon MC-pelissä $\mathcal{G}$ yhteensä n kappaletta sellaisia silmukoita tai itsesilmukoita, joilla ei ole yhtään yhteisiä solmuja. Tällöin pelissä $\mathcal{G}$ on eri ratkaisujoukkoja korkeintaan n kappaletta.*

Todistus. Sivuutetaan. □

Apulauseen 6.1 ilmoittama yläraja saadaan etsimällä ensin MC-pelin $\mathcal{G}$ kaikki silmukat sekä itsesilmukat. Silmukat $v_0, v_1, ..., v_n, v_0$ ja $v_1, v_2, ..., v_n, v_0, v_1$ voidaan luonnollisesti tulkita samoiksi silmuikoiksi. Tämän jälkeen tutkitaan, kuinka saamme valittua pelistä $\mathcal{G}$ mahdollisimman monta silmukkaa siten, ettei niillä ole yhtään yhteisiä solmuja.

Kuva 6.1 Esimerkki ratkaisujoukkojen lukumäärästä



Esimerkki.

Esimerkkejä ratkaisujoukkojen sijoittumisesta peliin on kuvassa 6.1. Vasemmanpuoleisesta pelistä löytyy 4 solmua, jonka kautta pelin jokainen sykli kulkee. Oikeanpuoleisesta pelistä löytyy korkeintaan kaksi ratkaisujoukkoa. Sille on mahdollista konstruoida vain kaksi sellaista sykliä, joilla ei ole yhtään yhteisiä solmuja.

6.3 Voittostrategian suhde syvyyssuuntaiseen etsintään

Käytettäessä syvyyssuuntaista etsintää MC-pelin ratkaisemiseen pitäisi algoritmin pystyä antaamaan tieto voittajasta läpikäytyjen solmujen perusteella. Vaikka kaikkia pelin solmuja ei vielä olisi käyty läpi, pitäisi algoritmin pystyä antamaan arvio voittajasta myös suhteessa läpikäytyihin solmuihin ja särmiin.

Läpikäydyt solmut eivät aina viritä MC-peliä. Seuraavassa apulauseessa esitetään tilanteet, jolloin tutkittujen solmujen virittämä graafi on MC-peli.

Apulause 6.2 *Olkoon MC-pelille $\mathcal{G}$ suoritettavan syvyyssuuntaisen etsinnän viimeksi läpikäymä särmä (v, u) ja olkoon V' etsinnän tähän mennessä löytämien solmujen joukko. Jos särmä (v, u) on itsesilmukka tai meno-, ristikkäis- tai paluukaari, niin $\langle V' \rangle$ on MC-peli.*

Todistus. Olkoon V' niiden solmujen joukko, jotka syvyyssuuntainen etsintä on tähän mennessä käynyt läpi ja olkoon etsinnän viimeksi läpikäymä särmä (v, u). Jos (v, u) on puukaari, on solmun u lähtöaste graafissa $\langle V' \rangle$ nolla. Kyseessä ei siis voi olla MC-peli.

Jos läpikäyty särmä ei ole puukaari, on silloin kaikkien tähän mennessä läpikäytyjen solmujen virittämän graafin $\langle V' \rangle$ jokaisen solmun lähtöaste ≥ 1.

□

Seuraavaksi tarkastellaan syvyyssuuntaisen etsinnän läpikäymien solmujen suhdetta voittostrategian konstruoimiseen. Syklikiinnitetyssä komponentissa oli aina mahdollista peruutuksen yhteydessä valita solmun omistajalle optimaalinen särmä. Yleisessä tilanteessa voittostrategian muodostaminen peruutettaessa ei ole mahdollista.

Apulause 6.3 *Olkoon MC-pelille $\mathcal{G}$ suoritettavan syvyyssuuntaisen etsinnän viimeksi löytämä solmu $v \in K$, missä K on vahvasti yhtenäinen maksimaalinen komponentti. Tällöin voittostrategia solmulle v on pahimmassa tapauksessa mahdollista muodostaa vasta, kun kaikki komponentin K solmut on käyty läpi.*

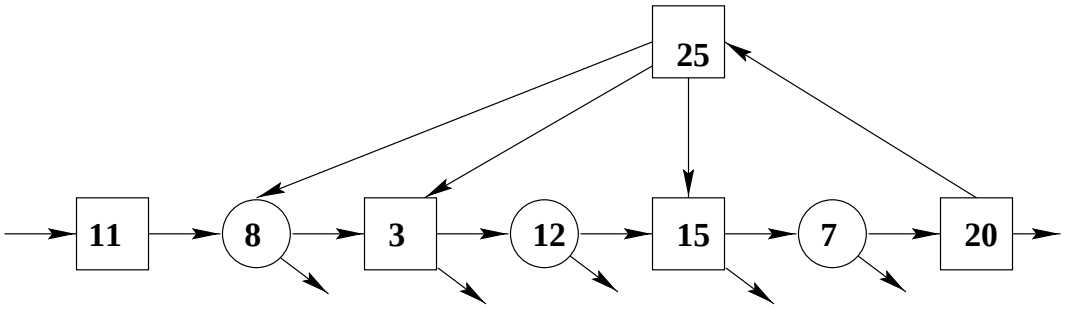
Todistus. Esitetään tilanne, jossa peruutettaessa solmusta v vanhempaan $f(v)$ ei voida konstruoida optimaalista pelitapaa solmun v omistajalle. Tilanne esiintyy kuvan 6.2 MC-pelissä. Oikea pelitapa solmusta 25 voidaan päättää vasta, kun tiedetään solmujen $8, 3$ ja 15 voittajat.

□

Vaikka voittostrategian konstruointi syvyyssuuntaisen etsinnän peruuttaessa ei ole mahdollista yleisessä tilanteessa, ei se vielä kumoa syvyyssuuntaisen

etsinnän käyttöä ratkaisualgoritmien pohjana. Mikäli etsinnän keräämät tiedot voidaan käsitellä komponentin kaikkien solmujen läpikäynin jälkeen, olisi kenties voittostrategioiden muodostaminen mahdollista myös jälkeenpäin.

Kuva 6.2 Särmän valinta peruutettaessa



Esimerkki.

Tarkastellaan vielä kuvan 6.2 tilannetta, jossa MC-pelin solmusta on löytynyt useampia paluukaaria. Olkoon V' syvyyssuuntaisen etsinnän tutkimien solmujen joukko ja 25 käsiteltävänä oleva solmu. Kaikki solmusta 25 lähtevät särmät on jo tutkittu. Tiedämme vielä, että solmuista lähtee tutkimattomia särmiä, sillä muuten sykli olisi pakottava.

Tässä tilanteessa ei voida päättää, minkä särmän pelaaminen on optimaalisinta solmun 25 omistajalle tai hänen vastustajalleen. Komponentissa on vielä läpikäymättömiä särmiä, jotka voivat vaikuttaa voittajan ratkaisemiseen. Yleisessä tilanteessa voittostrategian määrittely ei onnistu ennen komponentin kaikkien solmujen läpikäyntiä.

Mikäli tarkastelemme vain syvyyssuuntaisen etsinnän läpikäymiä solmuja kuvan 6.2 alipelissä, sekä niiden virittämää peliä $\mathcal{G}' = (\langle V' \rangle, token)$, pystymme tässä tilanteessa kertomaan pelin $\mathcal{G}'$ voittajan. Mutta jos etsintä löytää lisää kaaria, ei läpikäytyjen solmujen virittämä peli ole enää välttämättä ratkaistavissa. Algoritmin ominaisuuksiin pitäisi myös kuulua läpikäytyjen solmujen virittämän alipelin ratkaiseminen.

7 Yhteenveto

Tällä hetkellä MC-pelin voittajan ratkaisemiseen ei tunneta kuin eksponentiaalisia algoritmeja, joten polynomisten erikoistapausten etsiminen ja yleisen polynomisen algoritmin ominaisuuksien hahmottelu oli keskeinen osa työtä.

Tutkielmassa tarkasteltiin pääasiassa MC-pelin ratkaisemista kahdella eri menetelmällä. Ensimmäinen perustui pakotusjoukkojen laskentaan ja toinen syvyyssuuntaiseen etsintään.

Stirling [18] on todistanut pakotusjoukkojen laskentaan ja induktioon perustuvan todistuksen avulla, että aina joko pelaajalla I tai II on olemassa voittostrategia. Todistuksen perusteella konstruoitu algoritmi on eksponentiaalinen. Stirling on myös todistanut I/II-yksinkertaisen MC-pelin ratkeavan polynomisessa ajassa.

Tässä työssä esitettiin muutamaan MC-pelin erikoistapaukseen, puumaiseen ja syklikiinnitettyyn MC-peliin, ajassa $O(E)$ toimiva ratkaisualgoritmi. Näitä algoritmeja ei tiettävästi ole aiemmin käsitelty kirjallisuudessa. Lisäksi esitettiin pakotusjoukkojen laskentaan perustuva algoritmi puumaisille MC-peleille.

Ongelmaksi MC-pelien ratkaisemisessa osoitettiin pelistä löytyvät syklit. Mikäli MC-peli ei sisältänyt muita syklejä kuin itsesilmukoita, oli peli helppo ratkaista polynomisessa ajassa. Kun peliin lisättiin syklejä, pystyttiin myös näitä tilanteita ratkomaan tietyin rajoituksin. Jos MC-pelin vahvasti yhtenäisestä komponentista löytyy yksikin sellainen särmä, joka on osallisena komponentin jokaiseen sykliin, pystyttiin komponentti ratkaisemaan.

Työssä osoitettiin vielä, että eri ratkaisujoukkojen määrällä on yläraja. Ratkaisujoukkoja on korkeintaan yhtä paljon kuin pelissä on sellaisia syklejä tai itsesilmukoita, joilla ei ole yhtään yhteisiä solmua.

Loppuosassa työtä pohdittiin ja hahmoteltiin mahdollista yleisessä tapauksessa toimivaa polynomista algoritmia sekä esitettiin ehtoja sen olemassaololle.

Yleistä polynomista ratkaisua ongelmalle ei vielä ole löytynyt ja sen olemassaolo jää edelleen avoimeksi kysymykseksi. On mahdollista, ettei polynomista ratkaisualgoritmia ole olemassa, mutta toisaalta ongelmaa ei ole vielä todistettu NP-täydelliseksi.

Viiteluettelo

[1] A Condon. The complexity of stochastic games. *Information and Computation*, 121:203–224, 1992.

[2] T.H Cormen, C.E Leiserson, and R.L Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[3] S Even. *Graph Algorithms*. Computer Science Press, 1979.

[4] M.R Garey and D.S Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1981.

[5] J.E Hopcroft and J.D Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[6] P Koivisto ja R Niemistö. *Graafiteoria*. Matemaattisten tieteiden laitos, Tampereen yliopisto, 1998.

[7] K Koskimies. *Pieni oliokirja*. Suomen ATK-kustannus, 1997.

[8] H Lescow. On polynomial-size programs winning finite-state games. *Lecture Notes in Computer Science*, 939:239–252, 1995.

[9] W Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117:151–155, 1995.

[10] R McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[11] R Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[12] E Mäkinen. *Algoritmien suunnittelu ja analyysi*. Raportti C-1996-3. Tietojenkäsittelyopin laitos, Tampereen yliopisto, 1996.

[13] E Mäkinen. *Laskennan teorian perusteet*. Raportti C-1996-2. Tietojenkäsittelyopin laitos, Tampereen yliopisto, 1996.

[14] M Penttonen. *Johdatus algoritmien suunnitteluun ja analysointiin*. Otatieto, 1997.

[15] E Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms*. Prentice Hall, 1977.

[16] H Siirtola. Verkkojen piirtoalgoritmit. Pro gradu -tutkielma, Tampereen yliopiston tietojenkäsittelyopin laitos, 1988.

[17] C Stirling. Bisimulation and modal logic with fixed points. Manuscript, 1998.

[18] C Stirling. Bisimulation, model checking and other games. Notes for Mathfit Instructional Meeting on Games and Computation, University of Edinburgh, June 23-24, 1997. HTTP://www.dcs.ed.ac.uk/home/cps/mathfit.ps.

[19] M.N.S Swamy and K Thulasiraman. *Graphs, Networks and Algorithms*. John Wiley and Sons, 1981.

[20] R Tarjan. Depht-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[21] W Thomas. On the synthesis of strategies in infinite games. *Lecture Notes in Computer Science*, 900:239–252, 1995.

A Ratkaisualgoritmien toteutuksesta ja MC-pelien piirtämisestä

A.1 Ratkaisualgoritmien ohjelmoinnista

Tässä työssä esiteltyjen MC-pelin ratkaisualgoritmit on ohjelmoitu java-kielellä. Algoritmeja varten toteutettiin myös tarvittavat luokat graafien esittämiseen.

Graafien ohjelmoimiseen käytettiin matriiseja niiden helppouden takia. Matriisien esittämisessä käytettiin valmiiksi toteutettua *JSci-luokkakirjastoa*. JSci toteuttaa monia matematiikan, fysiikan sekä kemian luokkia. JSci on kopioitavissa osoitteesta `http://fourier.dur.ac.uk:8000/~dma3mjh/jsci/index.html`. Lähdekoodien kääntämiseen JSci on välttämätön.

Alla olevassa listassa esitellään toteutetut luokat pääpiirteittäin:

- Graph
  - Datajäsenenä graafin matriisi
  - Perusmetodit (muun muassa lisää/poista solmu)
- DirectedGraph
  - Perii Graph-luokan
  - Syvyyssuuntainen etsintä
  - Vahvasti yhtenäisten komponenttien löytäminen
- MCGraph
  - Perii DirectedGraph-luokan
  - Datajäsenenä token-bijektio, joka ilmoittaa solmun omistajan
  - Pakotusjoukkojen laskenta
  - Puumaisten MC-pelien ratkaisualgoritmi
  - Muita ratkaisuheuristiikkoja
- GraphMath, RandomGraphs
  - Satunnaisten MC-pelien generointi
  - Muita graafien tarvitsemia matemaattisia operaatioita

- GraphTester

    - Testiluokka, luokkien ja algoritmien testaus.

Luokkien lähdekoodit sekä MC-pelien piirtämistä varten toteutettu ohjelma on saatavilla osoitteesta `http://www.uta.fi/~tp54752/graphviewer.html`.

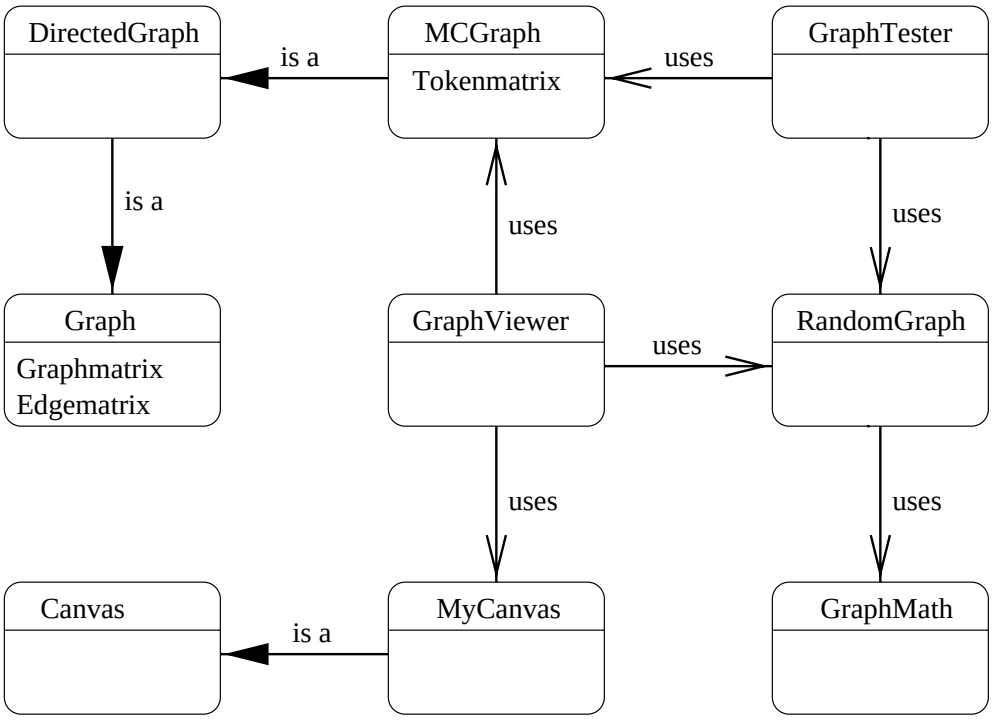
A.2 MC-pelien piirtämisestä

Graafien tasoesityksen [16] muodostaminen algoritmisesti on yksi teoreettisen tietojenkäsittelyopin keskeisistä tutkimusalueista.

MC-pelien piirtämistä varten toteutettiin ohjelma, joka piirtää käyttäjän antamien parametrien mukaisia MC-pelejä.

GraphViewer-ohjelma piirtää MC-pelin solmut ympyrän kehälle siten, että pelaajan I solmut piirretään ympyrällä ja pelaajan II solmut neliöllä.

Kuva A.1 Ohjelmoitujen luokkien luokkakaavio



Luokkakaavion piirtämisessä on käytetty OMT-notaatiota (Object Modeling Technique)[7]. Luokkien väliset assosiaatiot on piirretty tavallisilla ja perintäsuhteet vahvennetuilla nuolilla.

Suunnatun ja suuntaamattoman graafin perintäsuhde on hieman ongelmallinen, sillä molemmat voidaan kuvata ja määritellä toistensa avulla. Mutta käytännössä ei ole kovin suurta merkitystä kumpi on toisen aliluokka.

Ohjelma on suoraan ajettavissa javaa tukevilla selaimilla osoitteesta `http://www.uta.fi/~tp54752/graphviewer.html`.

GraphViewer käyttää edellisessä luvussa esiteltyjä luokkia graafien kooditason esitykseen sekä algoritmien toteuttamiseen. MyCanvas luokkaa GraphViewer käyttää graafien piirtämiseen. MyCanvas-luokka perii Canvas-luokan, joka on javassa valmiina oleva luokka piirto-operaatioita varten.

GraphViewer-ohjelma toimii nopesti hyvinkin suurilla solmumäärillä, mutta piirretyn kuvan selkeys katoaa viimeistään noin 20 solmun kohdalla riippuen särmien määrästä. Tällöin särmien leikkauspisteet menevät useimmiten liikaa ristiin keskenään, mikä vaikeuttaa peligraafin hahmottamista.

Tässä vaiheessa ei GraphVieweriin ole vielä toteutettu Force-joukkojen laskemista eikä muitakaan ratkaisualgoritmeihin liittyviä toimintoja ole integroitu sen käyttöliittymään. Näitä on kuitenkin mahdollista testata tekemällä testiohjelma näitä toimintoja varten. Esimerkkejä löytyy luokasta *GraphTester*.