hyväksymispäivä arvosana

arvostelija

Paikalliset hajautetut verkkoalgoritmit

Mika Laitinen

Helsinki 6.5.2011

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author

Mika Laitinen

Työn nimi — Arbetets titel — Title

Paikalliset hajautetut verkkoalgoritmit

Oppiaine — Läroämne — Subject

Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	6.5.2011	20 sivua + 0 liitesivua

Tiivistelmä — Referat — Abstract

Paikalliset hajautetut verkkoalgoritmit ovat algoritmeja, joissa verkon solmut kykenevät tekemään laskentaa. Verkon solmuilla on vain rajattu määrä tietoa verkon koostumuksesta, mutta algoritmit pyrkivät löytämään ratkaisun globaaliin verkkoa koskevaan ongelmaan vain paikallista tietoa käsittelemällä. Ongelmat, joita paikalliset algoritmit kykenevät ratkaisemaan, ovat tyypillisesti rajoitettuja, ja sellaisia, että ratkaisun oikeellisuus on hyvin helppoa tarkastaa vain paikallista tietoa hyödyntämällä. Paikalliset algoritmit löytävät kuitenkin ratkaisuja hyvin nopeasti sellaisiin ongelmiin, joiden ratkaisemiseen ne soveltuvat. Paikalliset algoritmit ovat lupaavia esimerkiksi tiedonkulun reitittämiseen mobiiliverkoissa, missä monet tunnetut paikalliset algoritmit ovat avuksi tiedonsiirron jouduttamisessa.

ACM Computing Classification System (CCS):
F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems
G.2.2 [Discrete Mathematics]: Graph Theory

Avainsanat — Nyckelord — Keywords

paikallinen algoritmi, hajautettu algoritmi, verkot, verkon väritys

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto

Monet tietojenkäsittelytieteessä vastaan tulevat ongelmat ovat luonteeltaan sellaisia, että yksi laskentayksikkö, jolle on annettu kaikki tieto ongelmasta, voi luontevasti löytää ratkaisun sopivalla algoritmilla. Kaikki ongelmat eivät kuitenkaan ole helppoja yksittäisten laskentayksiköiden ratkaistaviksi. Varsinkin suurikokoisia ja monimutkaisia verkkoja, kuten Internetiä tai suuria sensoriverkkoja, on vaikeaa tai mahdotonta hallita yhdellä kaikkitietävällä laskentayksiköllä [KMW06]. Mikäli verkon tietomäärää ei voida käsitellä keskitetysti, voidaan käyttää paikallisia hajautettuja verkkoalgoritmeja, joissa verkon osat ratkaisevat itseään koskevia osaongelmia vain paikallista tietoa hyödyntämällä.

Paikalliset algoritmit ovat algoritmeja, joissa jokaisella verkon solmulla on käytössään vain rajallinen määrä tietoa ympäristöstään. Paikalliset algoritmit toimivat verkoissa, joiden jokaisessa solmussa sijaitsee prosessori. Näiden verkkojen kaaret kuvaavat prosessorien välisiä välittömiä kommunikointiyhteyksiä. Verkon solmujen asteluvut ovat rajoitettuja, eli jokaisella prosessorilla saa olla vain äärellinen määrä yhteyksiä muihin prosessoreihin [NS95].

Paikalliset algoritmit poikkeavat ohjelmoinnille tyypillisestä ajatusmaailmasta — sen sijaan, että ohjelmoitaisiin yksittäisiä prosessoreita, ohjelmoidaan verkkoja. Verkoissa tehokkaasti ratkeavat ongelmat poikkeavat usein merkittävästi yksittäisillä prosessoreilla tehokkaasti ratkeavista ongelmista. Esimerkiksi pienimmän virittävän puun löytäminen on paikallisille algoritmeille mahdotonta, mikäli algoritmi saa ajonsa alussa rajallisen määrän tietoa ja kaarien painot ovat mielivaltaisia. Sen sijaan paikalliset algoritmit voivat kyetä ratkaisemaan hyvin tehokkaasti sellaisia ongelmia, joissa paikallisen ratkaisun oikeellisuuden osoittaminen ei ole hankalaa. Tällainen ongelma on esimerkiksi solmuväritys, jossa ratkaisun oikeellisuuden osoittaminen on paikallisesti helppoa, sillä on yksinkertaista tarkistaa, että kaikkien naapurisolmujen väri poikkeaa omasta väristä [NS95].

Paikallisilla algoritmeilla pyritään tyypillisesti ratkaisemaan ongelmia hyvin suurissa verkoissa, sillä paikallisten algoritmien aikavaatimukset kasvavat usein hyvin hitaasti verkon koon funktiona, ja useat ongelmat voidaan ratkaista jopa vakioajassa [Suo11]. Mikäli nopeaa tarkan vastauksen antavaa paikallista algoritmia ei ole mahdollista kehittää, on usein kuitenkin mahdollista kehittää nopea paikallinen algoritmi approksimoimaan oikeaa tulosta. Varsinkin hyvin suurten verkkojen tapauksessa hyvä approksimaatio on usein lähes yhtä arvokas kuin optimaalinen ratkaisu, jos tarkkuutta uhraamalla voidaan säästää laskenta-aikaa [WKLL11].

Usein suurissa verkoissa kommunikaatio solmujen välillä on huomattavasti hitaampaa kuin operaatiot itse solmujen sisällä. Esimerkiksi suurissa sensoriverkoissa yhteyden muodostaminen ja tiedon siirtäminen kahden sensorin välillä on hidas operaatio. Tästä syystä paikallisten algoritmien aikavaativuusanalyysissa ollaan kiinnostuneita vain prosessorien välisten kommunikaatiokierrosten määrästä [Lin92, PR01]. Perinteiseen aikavaativuusanalyysiin verrattuna ero on merkittävä, sillä kommunikaatiokierrosten väleissä prosessorit voisivat tehdä minkä tahansa äärellisen määrän työtä vaikuttamatta aikavaativuuteen.

Verkkojen ohjelmointi ajamalla samaa algoritmia jokaisessa verkon solmussa on mielenkiintoinen lähestymistapa myös siksi, että tällaista laskentamallia ei tunneta vielä kovin hyvin. Tutkimustulokset viittaavat kuitenkin siihen, että joihinkin tiettyihin ongelmiin hajautettu laskentamalli voisi soveltua paremmin kuin aiemmat olemassaolevat mallit. Näitä tuloksia tukevat myös jotkin luonnossa esiintyvät ilmiöt, esimerkiksi ihmisten aivosolut näyttävät pystyvän tekemään päätöksiä oman tilansa suhteen vain paikallisen tiedon perusteella [KMW06]. Analogia paikallisiin algoritmeihin on huomattava, sillä neuronit voivat olla yhteydessä vain rajalliseen määrään lähistöllä sijaitsevia muita neuroneita. Näiden neuroneiden täytyy keskenäisen kommunikointinsa perusteella päätyä yhteiseen lopputulokseen, joka vaikuttaa ihmisen toimintaan.

Paikallisten algoritmien tutkimus on vielä nuorta, eikä tutkimuksessa ole jumiuduttu tietyille urille, vaan alalla on vielä useita avoimia tutkimusongelmia, jotka poikkeavat parhaimmillaan hyvin paljon toisistaan. Tässä tutkielmassa esitellään kaksi hajautettua laskentamallia, jonka jälkeen keskitytään tutkimaan vahvemmassa mallissa ratkeavia ongelmia. Erityisesti keskitytään verkon väritysongelmaan, ja havainnollistetaan paikallisten algoritmien toimintaa esittelemällä verkon väritysongelmaan yksi ensimmäisistä paikallisista algoritmeista. Lopuksi käydään läpi verkon värityksen sovellutusmahdollisuuksia, sekä analysoidaan hajautetun laskentamallin tulevaisuutta ja käytännön sovellutuksia.

2 Hajautettu laskentamalli

Hajautettua laskentamallia voidaan käyttää sellaisissa verkoissa, joiden solmuissa voidaan tehdä laskentaa. Mikäli verkon jokaiseen solmuun sijoitetaan prosessori, voi jokainen solmu yrittää ratkaista paikallista tietoa hyödyntämällä solmun itsensä kannalta relevantteja kysymyksiä.

Hajautetussa laskentamallissa jokaista solmua edustaa prosessori, joka voi kommunikoida naapurisolmujensa prosessorien kanssa. Prosessorien toiminta on jaoteltu laskentaan ja kommunikaatiokierroksiin. Prosessorit vaihtelevat laskentatilan ja kommunikointitilan väleillä. Kun prosessori on päättänyt, että se on laskenut tarpeeksi, se pyrkii saamaan lisää tietoa kommunikoimalla muiden prosessorien kanssa.

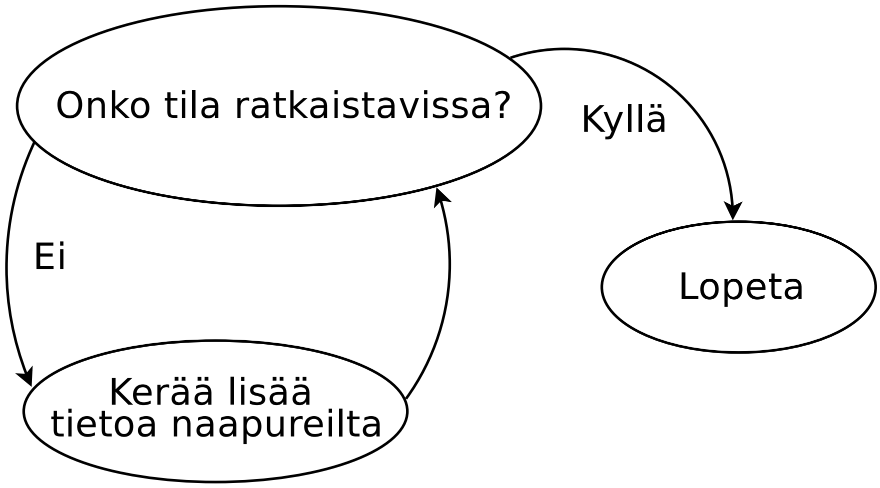
Solmut pyrkivät laskemaan omaa osaansa koko verkkoa koskevasta ongelmasta. Kun kaikki solmut ovat ratkaisseet oman osaongelmansa, voidaan muodostaa koko verkkoa koskevan ongelman ratkaisu. Solmujen omien osaongelmien luonteet vaihtelevat, mutta tyypillisesti solmu pyrkii ratkaisemaan jotain solmun (tai solmusta lähtevien kaarien) ominaisuutta algoritmin suorituksen loputtua. Esimerkiksi maksimaalista riippumatonta joukkoa[1] haettaessa jokaisen solmun osaongelma on ratkaista, kuuluuko solmu maksimaaliseen riippumattomaan joukkoon vai ei.

Hajautettu laskentamalli toimii verkoissa $G = (V, E)$, joiden jokaisessa solmussa $v \in V$ on prosessori. Solmussa x sijaitseva prosessori p_x voi kommunikoida solmussa y sijaitsevan prosessorin p_y kanssa jos, ja vain jos on olemassa $(x, y) \in E$, jossa (x, y) on järjestämätön pari. Prosessorit eivät jaa muistia keskenään, vaan ovat täysin itsenäisiä, ja saavat ajon alussa tietää vain niistä prosessoreista, joiden kanssa ne voivat kommunikoida.

Itse ongelmanratkaisu hajautetussa laskentamallissa toimii siten, että jokaiselle prosessorille annetaan sama algoritmi, jota prosessorit ajavat, kunnes ovat ratkaisseet oman paikallisen ongelmansa ja pysähtyneet. Tämä tarkoittaa sitä, että prosessorit voivat yksilöidä toimintaansa vain alussa ja algoritmin ajon aikana saadun tiedon avulla. Paikallinen ongelma, jota prosessorit ratkaisevat, on prosessorin oma lopputila. Prosessorin lopputila tarkoittaa verkon kannalta sitä tilaa, mitä ratkaisemaan prosessori on osoitettu. Esimerkiksi verkon väritysongelmissa prosessorin lopputila on prosessorin omalle solmulleen valitsema väri.

Paikalliset algoritmit laskevat saadun tiedon perusteella, onko oma tila ratkaistavissa. Jos algoritmi päättää, että omaa tilaa ei voida vielä ratkaista, se pyrkii saamaan lisää tietoa kommunikoimalla niiden prosessorien kanssa, joihin sillä on yhteys. Kommunikaatiokierrosten rooli on hajautetussa laskentamallissa hyvin suuri — itse asiassa olemme algoritmien aikavaatimuksia analysoidessamme kiinnostuneita vain

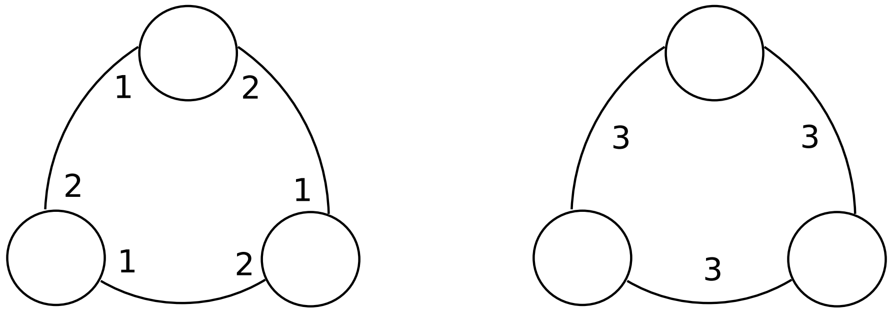
[1] Riippumaton joukko on verkon solmujen osajoukko, jossa yksikään joukkoon kuuluva solmu ei ole toisen joukkoon kuuluvan solmun naapuri. Maksimaalinen riippumaton joukko ei ole minkään riippumattoman joukon aito osajoukko.



Kuva 1: Havainnollistus paikallisten algoritmien toiminnasta. Jokaisen kommunikaatiokierroksen välissä algoritmi selvittää, onko se saanut tarpeeksi tietoa, jotta se voisi ratkaista oman paikallisen ongelmansa. Mikäli tietoa ei ole vielä tarpeeksi, pyritään seuraavalla kommunikaatiokierroksella keräämään lisää tietoa ongelmanratkaisua varten.

algoritmin käyttämien kommunikaatiokierrosten määrästä. Tämä tarkoittaa sitä, että prosessorit saavat käyttää rajoittamattoman määrän laskenta-aikaa jokaisen kommunikaatiokierroksen välissä. Lisäksi kommunikaatiokierroksen aikana siirrettävän datan määrää ei ole rajoitettu.

Koska kaikki prosessorit ajavat samaa algoritmia, keinot prosessorien yksilöintiin ovat arvokkaita — muutoin joitain ongelmia ei välttämättä voida ratkaista. Mikäli prosessorit voidaan erottaa toisistaan esimerkiksi tunnistenumeroilla, voidaan kaikki ongelmat ratkaista yhdistetyissä verkoissa ajassa $\mathcal{O}(\mathrm{diam}(G))$, jossa $\mathrm{diam}(G)$ on verkon halkaisija, eli verkon pisin polku. Tässä ajassa kaikille prosessoreille saadaan kommunikoitua koko verkon rakenne. Koska prosessorien käyttämää laskenta-aikaa ei ole rajoitettu, voi jokainen prosessori käyttää tarvittavan ajan ratkaisun löytämiseen, valita oman tilansa, ja lopettaa algoritmin suorituksen. Tästä syystä yleensä ollaan kiinnostuneita huomattavasti nopeammista algoritmeista, esimerkiksi polylogaritmisen tai vakioajan vaativista algoritmeista. Linial kutsuu ongelmaa *paikallisesti laskettavaksi*, mikäli ongelma voidaan ratkaista alle $\mathcal{O}(\mathrm{diam}(G))$-ajassa [Lin92].



Kuva 2: Vasemmalla kuva kolmisolmuisesta verkosta, jossa kaikki solmut ovat keskenään symmetrisessä tilanteessa, joten tilanne on porttinumerointimallissa ratkaisukelvoton. Kuvan kokonaisluvut kuvaavat solmujen naapureiden porttinumerointeja. Vastaavasti oikealla symmetrinen tilanne pienimmän virittävän puun ongelmassa, jossa kaikilla kaarilla on samat painot.

2.1 Porttinumerointimalli

Suomela [Suo10a] mainitsee kaksi mielenkiintoista tutkimuskohdetta hajautetussa laskentamallissa: mitä voidaan ratkaista alle $\mathcal{O}(\mathrm{diam}(G))$-ajassa, mikäli tunnistenumerot ovat käytössä, ja mitä voidaan ratkaista, jos tunnistenumeroita ei ole käytössä. Malleista, joissa prosessoreilla ei ole tunnistenumeroita, Suomela mainitsee porttinumerointimallin. Porttinumerointimallissa solmu, jonka asteluku on d, voi viitata sen naapureihin kokonaisluvuilla $1, 2, \ldots, d$. Nämä viitenumerot annetaan jokaiselle solmulle ennen algoritmin ajoa. Porttinumerointi on luonnollinen lähestymistapa siksi, että prosessorin täytyy mallin määritelmän mukaisesti voida kommunikoida muiden prosessoreiden kanssa. Koska olemme kiinnostuneita vain determinisististä malleista, täytyy naapuriprosessorit antaa prosessoreille tietyssä järjestyksessä.

Porttinumerointimalli ei ole kovin vahva malli, ja se pystyy ratkaisemaan vain tietynlaisia ongelmia. Tästä huolimatta porttinumerointimallilla voidaan ratkaista joitain epätriviaaleja verkko-ongelmia. Esimerkiksi voidaan luoda 3-approksimointialgoritmi pienimmän solmupeitteen löytämiseen [PS09], ja löytää pienin virittävä puu, jos millään kahdella kaarella ei ole samaa painoa [GHS83]. Porttinumerointimallin suurin ongelma on kuitenkin se, että on olemassa paljon ongelmia, joissa voidaan konstruoida ratkaisukelvottomia tilanteita. Tällaisissa tilanteissa porttinumerointimallissa ei ole mahdollista luoda algoritmia, joka kykenisi ratkaisemaan tilanteen.

Kuvassa 2 on kaksi verkko-ongelmaa, joissa molemmissa kaikki verkon solmut ovat keskenään symmetrisessä tilanteessa. Tällöin, kaikkien solmujen ajaessa samaa algoritmia, niiden pitäisi päätyä samaan lopputulokseen. On selvää, että esimerkiksi verkon väritysongelmassa ei ole kuitenkaan suotavaa, että kaikki verkon solmut valitsevat itselleen saman värin. Verkon väritys ei siis ole ongelma, jonka voisi kaikissa tapauksissa ratkaista porttinumerointimallissa. Kuva havainnollistaa myös samojen kaaripainojen sallimisen ongelmallisuuden pienintä virittävää puuta etsittäessä.

Symmetrian voi rikkoa satunnaisuuden avulla, mutta on olemassa myös deterministinen vaihtoehto symmetrian rikkomiseen. Uniikkeja tunnistenumeroita käyttämällä ei voida päätyä symmetriseen tilanteeseen [Lin92]. Tunnistenumeroiden käyttäminen on yksinkertainen malli, jossa voidaan ratkaista kaikki ratkaistavissa olevat ongelmat yhtenäisissä verkoissa. Tämän vuoksi suuri osa alan tutkimuksesta keskittyy juuri tunnistenumeromalliin. Myös tässä tutkielmassa keskitytään analysoimaan tunnistenumeromallissa toimivia algoritmeja.

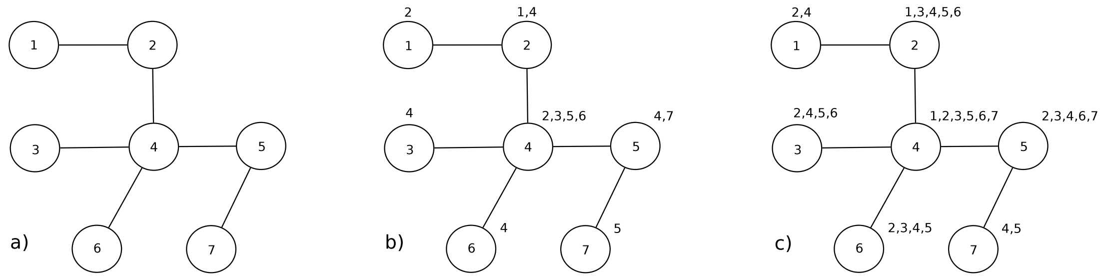
2.2 Tunnistenumeromalli

Tunnistenumeromallissa jokaiselle prosessorille annetaan uniikki tunnistenumero, jonka avulla muuten symmetrisessä tilanteessa olevat prosessorit voidaan erottaa toisistaan. Kuten aiemmin mainittiin, nyt koko verkon koostumus voidaan siirtää jokaiselle verkon solmulle, mikäli verkko on yhtenäinen. Toisin sanoen, mikäli käytetään tarpeeksi aikaa, mikä tahansa ongelma, jolla on ratkaisu yhtenäisissä verkoissa, voidaan ratkaista algoritmilla 1.

Algoritmi 1 Algoritmi kaikkien yhdistetyissä verkoissa ratkeavien ongelmien ratkaisemiseen tunnistenumeromallissa

repeat
    Kerää kaikki tieto naapuriprosessoreilta verkon rakenteesta. Muodosta tämänastinen käsitys verkon rakenteesta tunnistenumeroita hyväksikäyttämällä.
until Kierroksella ei saatu uutta tietoa verkon rakenteesta, eli verkon rakenne on onnistuneesti selvitetty.
Ratkaise annettu verkko-ongelma oman solmun näkökulmasta.
Valitse solmun lopputila ja lopeta algoritmin suoritus.

Vaikka algoritmin 1 olemassaolo ei vielä takaa, että ongelmia voitaisiin ratkaista epäyhtenäisissä verkoissa, suuri osa kiinnostavista verkko-ongelmista ei vaadi verkon yhtenäisyyttä. Esimerkiksi pienimmän virittävän puun, verkon värittämisen tai maksimaalisen riippumattoman joukon etsiminen komponentettain on järkevää.



Kuva 3: Kuvasarja algoritmin 1 toiminnasta. a) Suuntaamaton verkko ennen algoritmin ajoa. Solmut on numeroitu. b) Solmujen tietämys verkosta algoritmin ensimmäisen askeleen jälkeen listattuna solmun yläpuolelle. Esimerkiksi solmu 5 tietää oman ympäristönsä lisäksi solmujen 4 ja 7 ympäristöt. c) Solmujen tietämys verkosta algoritmin toisen askeleen jälkeen. Nyt solmu 4 tietää kaikkien verkon solmujen ympäristön, eli on kerännyt kaiken tarvittavan tiedon, jotta se voi muodostaa oikean käsityksen verkon rakenteesta.

Algoritmi 1 on hyvin hidas, ja vie pahimmillaan lineaarisen ajan verkon solmumäärään nähden. Tämän vuoksi on luontevaa keskittyä tutkimaan ongelmia, jotka ovat paikallisesti laskettavia. Usein myös approksimaatioalgoritmien löytäminen on kiinnostava tutkimuskohde, varsinkin jos ongelman tarkka ratkaisu ei ole paikallisesti laskettavissa.

Algoritmin 1 perimmäinen tarkoitus on selvittää koko verkon koostumus jokaisen verkon solmun tietoon, jotta ne voisivat ratkaista ongelman itsenäisesti. Koska jokaisella solmulla on uniikki tunnistenumero, on algoritmin helppo karsia kaikki tieto, jonka se saa useamman kerran, sekä yhdistää sen saamat tiedot oikeaan osaan verkkoa. Porttinumerointimallissa verkon koostumuksen kerääminen on kuitenkin ongelmallista. Esimerkiksi jos saamme samaa dataa uudelleen, emme voi olla varmoja päädyimmekö jo käytyyn solmuun, vai onko jossain päin verkkoa useita samanlaisia ympäristöjä nykyisen kerätyn tiedon perusteella. Symmetriaongelmat todistavat, että verkon koostumusta ei porttinumerointimallissa voi selvittää.

3 Verkon kolmiväritys

Hajautetussa laskentamallissa ongelmien ratkaisumenetelmät poikkeavat huomattavasti perinteisempien ongelmien ratkaisussa käytettävistä menetelmistä. Paikallisesti tarkastettavissa olevat ominaisuudet (engl. *locally checkable labelings*), eli esimerkiksi solmu- ja kaariväritykset sekä maksimaaliset riippumattomat joukot ovat ongelmia, joita on helppo lähestyä paikallisilla algoritmeilla. Näissä ongelmissa ratkaisun laillisuus on helppo tarkastaa nopeasti jokaisessa algoritmin vaiheessa [NS95]. Tällöin algoritmin ei tarvitse käyttää suurta määrää kommunikaatiokierroksia yksinkertaisiin ongelmiin, kuten siihen, voiko algoritmin suorituksen jo lopettaa.

Tyypillinen ongelma hajautetussa laskentamallissa on verkon solmuväritys. Solmuvärityksessä ratkaisun laillisuus on paikallisesti tarkastettavissa oleva ominaisuus, sillä on yksinkertaista ja nopeaa varmistaa, että solmun naapurit valitsevat askeleen lopuksi eri värin kuin solmu itse.

Solmuvärityksen soveltuvuudesta paikallisten algoritmien maailmaan kertoo myös se, että monet merkittävät tulokset hajautetussa laskentamallissa liittyvät juuri solmuvärittämiseen. Solmuvärittämisessä onkin monia tutkimuskohteita, esimerkiksi rajoitettujen verkkojen kolmivärittäminen tai verkon ominaisuuksista, kuten solmujen maksimiasteesta riippuvien nopeiden väritysalgoritmien löytäminen.

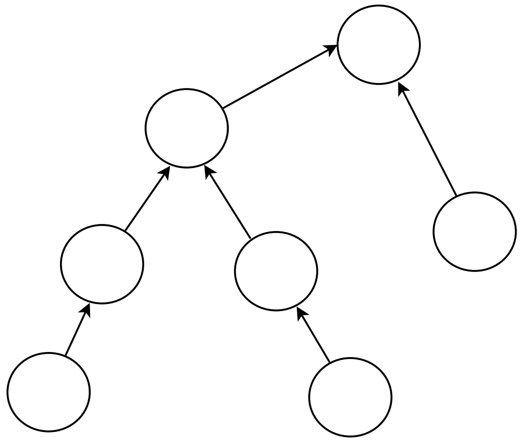
Solmuvärityksen hyödyllisyys riippuu yleensä käytettyjen värien määrästä. Värien määrän minimointi on useimmiten hyödyllistä, ja verkon väritystä voidaan käyttää muiden ongelmien ratkaisemiseen. Esimerkiksi TDMA MAC -protokolla perustuu vahvasti solmuväritykseen, ja värien vähentäminen tehostaa suoraan datan liikkuvuutta [WKLL11]. Verkon rakenteesta riippuu, kuinka montaa väriä täytyy käyttää, jotta verkko saadaan väritettyä.

Colen-Vishkinin algoritmi kykenee vähentämään värejä sellaisista suunnatuista verkoista, joissa jokaisella solmulla on korkeintaan yksi lapsi (ks. kuva 4). Tällaisissa verkoissa voidaan aina vähentää värien määrä kolmeen [GPS87]. Colen-Vishkinin algoritmilla voidaan tehokkaasti vähentää värien määrä korkeintaan kuuteen. Kuudesta väristä päästään kolmeen väriin yksinkertaisemmilla algoritmeilla, tyypillisesti sellaisilla, jotka vähentävät verkosta värejä lineaarisesti kommunikaatiokierrosten määrää kohti.

3.1 Colen-Vishkinin algoritmi

Yksi ensimmäisistä tuloksista hajautetussa laskentamallissa on Colen-Vishkinin värien vähennysalgoritmi. Tämän kappaleen esitys Colen-Vishkinin algoritmista perustuu voimakkaasti kirjallisuudessa esiintyneisiin esityksiin [CV86, Lin92, WKLL11, Suo10b].

Alun perin Colen-Vishkinin algoritmi oli tarkoitettu värien vähentämiseen linkite-



Kuva 4: Suunnattu verkko, jonka jokaisella solmulla on korkeintaan yksi lapsi. Verkon nuolet kuvaavat jälkeläissuhteita siten, että nuolen päässä oleva solmu on lähtöpisteessä olevan solmun lapsi. Lapsisolmu voi kommunikaatiokierroksen aikana siirtää tietoa isäntäsolmulleen. Solmut ovat keskenään naapureita, mikäli toinen solmuista on toisen lapsi.

tyissä listoissa hajautetusti [CV86]. Algoritmi toimii kuitenkin myös paikallisena algoritmina. Algoritmi toimii suunnatuissa verkoissa, joissa jokaisella solmulla on korkeintaan yksi lapsi, ks. kuva 4. Prosessori b voi siis lähettää tietoa prosessorille a vain, jos on olemassa $(a, b) \in E$, jossa (a, b) on järjestett pari. Kuitenkin solmut a ja b ovat *naapureita*, mikäli on olemassa $(a, b) \in E$ tai $(b, a) \in E$. Colen-Vishkinin algoritmi vaatii myös, että solmujen värit esitetään kokonaislukuina.

Colen-Vishkinin algoritmilla valmista verkon väritystä voidaan askel askeleelta vähentää siihen asti, kunnes verkossa on jäljellä enää kuusi väriä. Suunnatussa verkossa, jossa solmuilla on korkeintaan yksi lapsi, on mahdollista löytää kolmiväritys. Colen-Vishkinin algoritmin löytämästä kuusivärityksestä voidaan päästä kolmiväritykseen vakioajassa käyttämällä algoritmeja, jotka vähentävät verkosta yhden värin kommunikaatiokierrosta kohti [CV86, Suo10b].

Mikäli ongelmaverkko on suuri, emme halua käyttää algoritmeja, jotka vähentävät verkon värejä lineaarisesti kommunikaatiokierrosten määrän suhteen, ellei verkon värien määrä ole alussa hyvin vähäinen. Tunnistenumeromallissa ainoa väritys, mitä voimme alkutilanteessa käyttää, on juuri tunnistenumeroiden muodostama väritys. Alussa siis värien määrä on sama kuin verkon solmujen määrä, joten on selvää, että lineaarinen algoritmi olisi hyvin hidas, ja mikäli se olisi optimaalinen, olisi se aikavaativuudeltaan samaa luokkaa kuin algoritmi 1.

Lineaariseen aikavaativuuteen ei kuitenkaan tarvitse tyytyä. Colen-Vishkinin algoritmi toimii siten, että jokaisella kierroksella verkon suurin väri pienenee logaritmisesti. Vaikka Colen-Vishkinin algoritmissa verkossa olevien värien määrä voi kierroksen aikana jopa kasvaa, niin verkon suurin väri x pienenee varmasti, jos $x > 5$.

Määritelmä 1. *Solmu x, jonka lapsi on solmu y, valitsee uudeksi värikseen $g(x, y)$:n. Funktion n arvo määräytyy vertaamalla solmujen x ja y värien binäärimuotoja $x_n \cdot 2^n + x_{n-1} \cdot 2^{n-1} + \ldots + x_0 \cdot 2^0$ ja $y_n \cdot 2^n + y_{n-1} \cdot 2^{n-1} + \ldots + y_0 \cdot 2^0$, $x_i, y_i \in \{0, 1\}$. Tällöin $g(x, y) = f(x, y) \cdot 2 + x_{f(x,y)}$, jossa $f(x, y) = i$ siten, että $x_i \neq y_i$ ja i on mahdollisimman pieni. Mikäli solmulla x ei ole lasta, oletetaan lapseksi olematon solmu p siten, että $f(x, p) = 0$. Tällöin solmu x valitsee sen uudeksi värikseen x_0:n.*

Väite 2. *Jos verkko on laillisesti väritetty ja yksikään solmu ei ole lopettanut algoritmin ajoa, solmu a ei voi valita uudeksi värikseen väriä, jonka sen naapurisolmu b valitsee. Näin verkko säilyttää laillisen värityksen jokaisella algoritmin askeleella.*

Todistus. Oletetaan, että solmut a ja b valitsevat värikseen saman värin algoritmin askeleen lopuksi. Yleisyyttä menettämättä voidaan sopia, että solmu b on a:n lapsi. Oletetaan, että b:llä on lapsi c, $f(a, b) = i$, ja $f(b, c) = j$. Jotta solmut a ja b valitsisivat saman värin, täytyy päteä $g(a, b) = g(b, c) \Leftrightarrow f(a, b) \cdot 2 + a_{f(a,b)} = f(b, c) \cdot 2 + b_{f(b,c)}$. Edelleen seuraa $2 \cdot (f(a, b) - f(b, c)) = b_{f(b,c)} - a_{f(a,b)}$. Nyt $b_{f(b,c)} - a_{f(a,b)}$ voi saada arvoja joukosta $\{-1, 0, 1\}$. Koska $2 \cdot (f(a, b) - f(b, c))$ on aina parillinen, on oltava $a_{f(a,b)} = b_{f(b,c)}$, ja $f(a, b) = f(b, c)$, eli $a_{f(a,b)} = b_{f(a,b)}$. Määritelmän mukaan kuitenkin $a_{f(a,b)} \neq b_{f(a,b)}$, eli kyseessä on ristiriita, ja alkuperäinen väite on tosi. □

Oletetaan solmut a ja b siten, että b on a lapsi, ja a:n väri on b:tä suurempi. Tällöin määritelmän 1 mukaisesti solmu a, jonka väri on y, voi valita uudeksi värikseen korkeintaan arvon $2 \cdot h(y) + 1$, jossa $h(y) = \lfloor \log_2 y \rfloor + 1$, eli $h(y)$ on y:n bittien määrä binääriesityksessä. Koska väri x on verkon suurin, niin yläraja verkon suurimmaksi väriksi kierroksen jälkeen on $2 \cdot h(x) + 1$. Jokaisella kierroksella värien määrä vähenee siis logaritmisesti, ja algoritmin vaatimien kommunikaatiokierrosten määrä on $\mathcal{O}(\log^* k)$, jossa k on suurin väri algoritmin suorituksen alussa [WKLL11].

Algoritmi 2 Colen-Vishkinin algoritmi

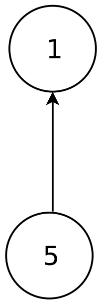
$k \leftarrow$ tarvittavien askelten määrä
$v \leftarrow$ solmu, jossa algoritmia ajetaan
for $i = 1 \ldots k$ **do**
  if v:llä on lapsi u **then**
    Odota, kunnes lapsi u kertoo värinsä $C(u)$.
    Vertaa omaa väriä $C(v)$ lapsen väriin $C(u)$.
    Määritelmän 1 mukaisesti, etsi binäärimuodossa oikeanpuolimmaisin bitti, indeksiltään $i = f(v, u)$, jossa värit poikkeavat.
    Valitse uudeksi väriksi $g(v, u) = 2 \cdot i + C(v)_i$, jossa $C(v)_i$ on nykyisen värin i:nnen bitin arvo, eli 0 tai 1.
  else
    Aseta uudeksi väriksi $C(v)_0$.
  end if
end for

Jokaisella kierroksella Colen-Vishkinin algoritmissa jokainen solmu, jolla on lapsi, kysyy lapsensa väriä. Tämän jälkeen solmu vertaa omaa väriään lapsensa väriin binäärimuodossa, ja etsii sen bitin indeksin, joka kahdessa värissä poikkeaa ensimmäisenä oikealta laskettuna. Tällainen bitti löytyy, sillä naapurisolmujen värit poikkeavat varmasti toisistaan laillisessa värityksessä. Kun bitin indeksi i löytyy, indeksoinnin alkaessa nollasta, valitsee solmu uudeksi värikseen $2 \cdot i + c_i$:n, jossa c_i on edellisen oman värin i:nnen bitin arvo, eli 0 tai 1. Mikäli solmulla ei ole lasta, se valitsee uudeksi värikseen edellisen värinsä viimeisen bitin, aivan kuin se poikkeaisi olemattoman lapsensa väristä jo bitissä 0. Algoritmin yhden kierroksen toimintaa havainnollistaa kuva 6.

Colen-Vishkinin algoritmi jatkaa suoritustaan, kunnes se voi jäädä jumiin. Jos verkon suurin väri on korkeintaan 5, on mahdollista, että algoritmi ei enää kykene pienentämään verkon suurinta väriä. Kuva 5 esittää hyvin yksinkertaista verkkoa, jossa Colen-Vishkinin algoritmi ei pysty etenemään.

Mikäli jokainen solmu tietää verkon suurimman värin arvon algoritmin alussa, voidaan algoritmin alussa laskea tarvittavien ajokierrosten määrä. Näiden ajokierrosten jälkeen voidaan olla varmoja, että verkossa on korkeintaan kuusi väriä.

On kuitenkin mahdollista, että prosessorit eivät tiedä alussa verkon suurinta arvoa. Tällöin Colen-Vishkinin algoritmi täytyy toteuttaa monimutkaisemmin, jotta voi-



Kuva 5: Esimerkki verkosta, jonka väritystä Colen-Vishkinin algoritmi ei voi enää muuttaa. Colen-Vishkinin algoritmi vertailisi värien $5 = \underline{1}01_2$ ja $1 = \underline{0}01_2$ binäärimuotoja. Tällöin algoritmi valitsisi väriä 5 kantavan solmun uudeksi väriksi $2 \cdot 2 + 1$ ja juurisolmun uudeksi väriksi värin 1, joten verkko pysyisi täysin samanlaisena.

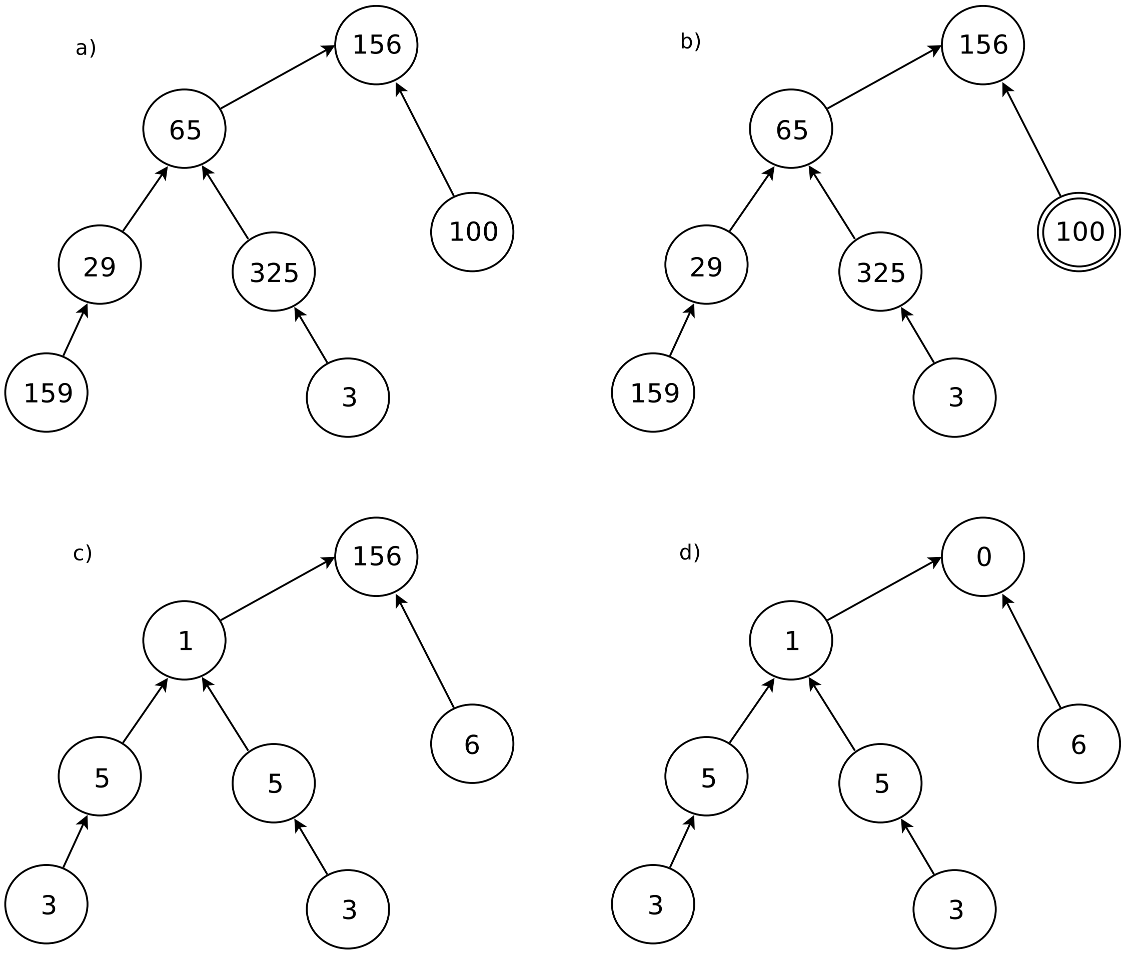
daan tietää, koska algoritmin suoritus voidaan huoletta lopettaa. Synkronoinnista täytyy pitää huolta, sillä jos jokin solmu lopettaa algoritmin ajamisen aikaisemmin, ei ole takeita värityksen laillisuuden säilyvyydestä. Tällaisen menetelmän toteuttamisesta ei kuitenkaan käydä tässä tarkemmin läpi. Colen-Vishkinin algoritmin esityksessä (ks. algoritmi 2) oletetaan, että algoritmille on esimerkiksi kerrottu verkon suurin väri, jotta tarvittavien kierrosten määrä voidaan etukäteen laskea.

Huomionarvoista on myös, että kaikissa reaalimaailman tilanteissa kierrosten määrä voidaan asettaa esimerkiksi kymmeneen. Colen-Vishkinin algoritmi pienentää värien määrää niin nopeasti, että verkon väriavaruuden täytyisi olla hyvin suuri, jotta kymmenen askelta ei riittäisi kuuden värin saavuttamiseen.

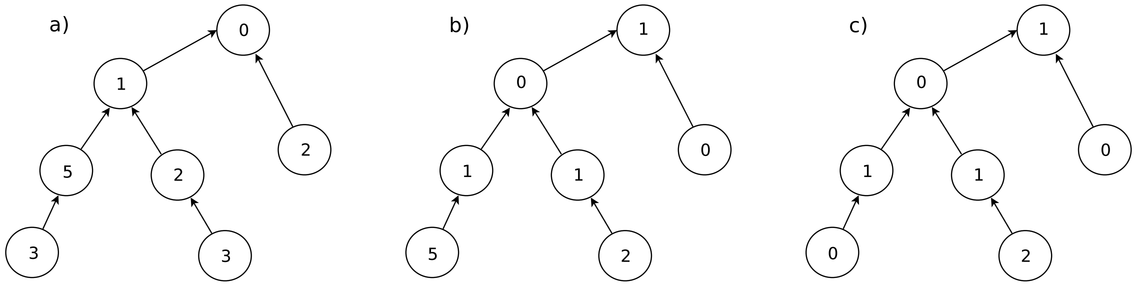
3.2 Yksinkertainen värinvähennysalgoritmi

Colen-Vishkinin algoritmilla värien määrä pystyttiin pienentämään hyvin tehokkaasti hyvin pieniin värimääriin asti. Colen-Vishkinin algoritmi ei kuitenkaan takaa, että alle kuuden värin päästäisiin. Koska kolmiväritys on kuitenkin mahdollista löytää, on mielenkiintoinen jatkokysymys, että voidaanko Colen-Vishkinin algoritmin jälkeen löytää kolmiväritys käyttämällä jotain muuta algoritmia?

Värien määrää voidaan vähentää aina kolmeen asti algoritmilla, joka vähentää yhden värin jokaista kommunikaatiokierrosta kohti. Koska Colen-Vishkinin algoritmin ajamisen jälkeen värejä on maksimissaan kuusi, ei lineaarinen aikavaatimus haittaa, sillä kommunikaatiokierroksia tulee maksimissaan kolme lisää, eli algoritmin aikavaativuus kasvaa vain vakiolla.



Kuva 6: Havainnollistus Colen-Vishkinin algoritmin yhdestä kierroksesta. a) Suunnattu verkko, jossa jokaisella solmulla on väri. b) Algoritmin suoritus kaikissa solmuissa samanaikaisesti. Solmu, jonka väri on 100, vertaa omaa väriään lapsensa väriin binäärimuodossa. Värit $100 = 110\underline{0}100_2$ ja $156 = 1001\underline{1}100_2$ poikkeavat toisistaan kolmannessa bitissä oikealta laskettuna (indeksoinnin alkaessa nollasta), joten solmu, jonka väri on 100, valitsee uudeksi värikseen $3 \cdot 2 + 0 = 6$. c) Kaikki solmut, joilla on lapsi, ovat vaihtaneet väriä b-kohdassa esitetyn menetelmän mukaisesti. d) Solmu, jolla ei ole lasta, teeskentelee poikkeavansa olemattoman lapsensa kanssa jo nollannessa bitissä, jolloin se valitsee uudeksi värikseen nykyisen värinsä viimeisen bitin, nollan. Todellisuudessa lapsetonkin solmu vaihtaa väriään samaan aikaan muiden solmujen kanssa jo c-kohdassa, ja se on erotettu omaan kuvaansa vain esityksen selkeyden vuoksi.



Kuva 7: Kuvasarja värienvähennysalgoritmin toiminnasta. a) Laillisesti väritetty suunnattu verkko. Solmuissa olevat numerot kuvaavat solmun väriä. b) Jokainen solmu on perinyt lapsensa värin. Lapsettomat solmut ovat valinneet uuden värin itselleen joukosta {0,1,2}. c) Väri 5 on eliminoitu verkosta siten, että jokainen solmu, jonka väri oli 5, valitsi sellaisen värin joukosta {0,1,2} jota millään naapurilla ei b-vaiheen jälkeen ollut.

Lineaarisessa värienvähennysalgoritmissa [WKLL11] jokaisella kierroksella päätetään poistettava väri k. Väri k poistetaan verkosta siten, että ensiksi suoritetaan värien siirto-operaatio. Värien siirto-operaatiossa jokainen väri valitsee uudeksi värikseen lapsensa nykyisen värin, ks. kuva 7. Mikäli lasta ei ole, valitaan edellisestä poikkeava väri joukosta $\{0, 1, 2\}$. Väitteen 3 mukaisesti siirto-operaatio säilyttää laillisen värityksen.

Väite 3. *Värien siirto-operaatio säilyttää laillisen värityksen.*

Todistus. Osoitetaan, että kaikki solmut valitsevat siirto-operaatiossa eri värit naapurisolmujensa kanssa. Jaetaan todistus kahteen osaan, juurisolmuun, ja muihin solmuihin.

Juurisolmun kaikki naapurit ovat sen isäntäsolmuja. Siirto-operaatiossa juurisolmun isäntäsolmut valitsevat uudeksi värikseen juurisolmun vanhan värin, ja juurisolmu valitsee itselleen uuden värin. Juurisolmun mikään naapuri ei siis valitse sen kanssa samaa väriä.

Siis kaikki juurisolmun isännät valitsevat eri värin lapsensa kanssa. Ne myös antavat vanhan värinsä kaikille isännilleen, ja ottavat lapsensa vanhan värin. Juurisolmun isännät valitsevat siis eri värin myös kaikkien omien isäntiensä kanssa, sillä juurisolmun isännän uusi väri, ja juurisolmun isännän isännän uusi väri olivat vierekkäin jo edellisessä verkossa, jonka väritys oli laillinen. Sama päättelyketju toistuu kaikissa muissakin verkon solmuissa. Verkon väritys on siis edelleen laillinen. $\square$

Kun jokainen solmu on valinnut itselleen värin, jokainen solmu, jonka väri on k, valitsee itselleen uuden värin joukosta $\{0, 1, 2\}$, säilyttäen samalla laillisen värityksen. Kaikki solmut, joiden väri on k, voivat valita itselleen uuden värin joukosta $\{0, 1, 2\}$ niin, että väritys pysyy laillisena. Ensinnäkin jokaisella solmulla on korkeintaan kahdenlaisia naapureita, isäntiä ja lapsia. Solmulla voi olla vain yksi lapsi, ja kaikilla solmun isäntäsolmuilla on sama väri, joten solmun naapurisolmuilla on yhteensä maksimissaan kahta eri väriä. Tällöin joukosta $\{0, 1, 2\}$ löytyy varmasti väri, jota naapurisolmuilla ei ole, joten väri k saadaan eliminoitua verkosta.

Ajettuamme Colen-Vishkinin algoritmin alkuperäisessä väritetyssä verkossa, tiedämme, että verkon suurin väri on korkeintaan 5. Voimme siis poistaa algoritmilla 3 verkosta ensin värin 5, jonka jälkeen värit 4 ja 3 voidaan poistaa samalla prosessilla.

Algoritmi 3 Algoritmi värien vähentämiseen kuudesta kolmeen suunnatuissa verkoissa, joissa solmuilla on korkeintaan yksi lapsi

```
for i = 5, 4, 3 do
  if solmulla on lapsi u then
    Valitse uudeksi väriksi solmun u väri.
    if solmun nykyinen väri on i then
      Valitse uudeksi väriksi väri joukosta {0, 1, 2}, joka poikkeaa solmun van-
      hasta väristä, sekä väristä, jonka solmu u valitsi uudeksi värikseen.
    end if
  else
    Valitse uusi, entisestä poikkeava, väri joukosta {0, 1, 2}.
  end if
end for
```

4 Yleisten verkkojen värittäminen ja värityksen soveltaminen

Edellisessä osiossa käytiin läpi rajoitetun verkon kolmivärittämistä. Verkon värittämisen tutkiminen on hyödyllistä, sillä verkon väritystä voidaan käyttää apuna esimerkiksi algoritmien hajauttamisessa ja muiden verkko-ongelmien ratkaisemisessa.

Colen-Vishkinin algoritmi on elegantti ja nopea algoritmi värien vähentämiseen verkoissa. Colen-Vishkinin ongelma on kuitenkin se, että se toimii vain sellaisissa suunnatuissa verkoissa, joissa solmuilla on korkeintaan yksi lapsi. Tällaiset verkot ovat vain pieni osa kaikista mahdollisista verkoista, ja reaalimaailmassa vastaan tulee usein tilanteita, joissa haluttaisiin löytää väritys yleisessä verkossa.

Yleisissä verkoissa toimivia väritysalgoritmeja on tutkittu paljon, ja useita eri lähtökohtiin perustuvia algoritmeja on julkaistu, esimerkiksi [AGLP89, Lin92, BE09]. Myös Colen-Vishkinin algoritmia voidaan yleistää siten, että se toimii kaikenlaisissa verkoissa. Yleistetty algoritmi kykenee värittämään verkon $\Delta + 1$:llä värillä, jossa Δ on verkon maksimiaste, eli suurin määrä kaaria, mitä yhdestäkään solmusta lähtee [WKLL11].

Nyt yleistetyssä tapauksessa verkko on siis suuntaamaton, eikä solmujen yhteyksissä muihin solmuihin ole samanlaisia rajoituksia kuin Colen-Vishkinin algoritmissa. Väritysidea on seuraavanlainen: jokaisella kierroksella jokainen solmu vertaa väriään jokaisen naapurinsa kanssa, ja solmut laskevat samalla tavalla uuden värinsä samoin kuin Colen-Vishkinin algoritmissa. Nyt uusia värejä tulee kuitenkin yhtä monta, kuin solmulla on naapureita, sen sijaan että värejä tulisi vain yksi. Solmu valitseekin uudeksi värikseen kaikkien näiden värien yhdistelmän, eli laittaa kaikki uudet värit peräkkäin ja valitsee sen uudeksi värikseen. Jos verkon maksimiaste Δ ei riipu verkon solmujen määrästä, vaan on vakio, tällä algoritmilla saadaan aikaan 3Δ-väritys $\mathcal{O}(\log^* n)$ askeleessa.

Yleistetyllä Colen-Vishkinin algoritmilla saavutettu 3Δ-väritys voidaan vähentää $\Delta{+}1$ -väritykseksi yksinkertaisella värinvähennysalgoritmilla $2^{3\Delta}$ askeleessa [WKLL11]. Koska Δ on vakio verkon kokoon nähden, on myös $2^{3\Delta}$ vakio, joten yleisessä verkossa värien määrä voidaan vähentää $\Delta + 1$ väriin $\mathcal{O}(\log^* n)$ askeleessa, Δ:n ollessa vakio.

Yleisen verkon värittävästä algoritmista on hyötyä myös muiden ongelmien ratkaisemisessa. Esimerkiksi maksimaalinen riippumaton joukko voidaan löytää yksinkertaisesti sen jälkeen, kun verkolle on löytynyt laillinen väritys. Tässäkin tapauksessa värien määrä vaikuttaa suoraan algoritmin aikavaativuuteen.

Maksimaalinen riippumaton joukko voidaan löytää värityksen avulla c:ssä kierroksessa, kun $\{0, \ldots, c-1\}$ ovat verkossa olevat värit [WKLL11]. Algoritmi toimii siten, että jokainen verkon väri käydään yksi kerrallaan läpi, ja jokaisella kierroksella käsittelyssä olevaa väriä olevat solmut katsovat, onko yksikään niiden naapurisolmu vielä riippumattomassa joukossa. Jos yksikään naapurisolmu ei ole vielä liittynyt riippumattomaan joukkoon, lisätään solmu riippumattomaan joukkoon.

Tämä algoritmi (ks. algoritmi 4) käyttää c kommunikaatiokierrosta maksimaalisen riippumattoman joukon löytämiseen. Koska $c = \Delta + 1$, niin tämän algoritmin ja Colen-Vishkinin yleistyksen avulla maksimaalinen riippumaton joukko voidaan löytää yleisestä verkosta $\mathcal{O}(\Delta + \log^* n)$ kommunikaatiokierroksessa.

Algoritmi 4 Algoritmi maksimaalisen riippumattoman joukon muodostamiseen c-värityksen avulla

---

$v \leftarrow$ solmu, jossa algoritmia ajetaan
for $i = 0 \ldots c - 1$ **do**
  if v:n väri on i **then**
    if Yksikään v:n naapureista ei ole mukana riippumattomassa joukossa **then**
      Lisää v riippumattomaan joukkoon.
    end if
  end if
end for

Väite 4. *Algoritmi 4 muodostaa maksimaalisen riippumattoman joukon.*

Todistus. Osoitetaan ensin, että algoritmi muodostaa riippumattoman joukon. Jotta joukko ei olisi riippumaton, täytyy olla olemassa ainakin yksi solmupari (a, b) siten, että a ja b ovat toistensa naapureita ja kuuluvat mukaan joukkoon. Oletetaan yleisyyttä menettämättä, että solmun b väri $C(b)$ on solmun a väriä $C(a)$ suurempi. Tällöin kierroksella $C(a)$ varmistaa, että yksikään sen naapureista ei kuulu vielä riippumattomaan joukkoon, ja lisää itsensä riippumattomaan joukkoon. Kun $C(b)$ myöhemmin varmistaa, että yksikään sen naapureista ei ole mukana riippumattomassa joukossa, ei se saa lupaa liittyä riippumattomaan joukkoon, koska solmu a on jo liittynyt siihen.

Jäljelle jää tapaus, jossa $C(a) = C(b)$, jolloin a ja b voisivat liittyä joukkoon samalla kierroksella, jolloin muodostuva joukko ei olisi riippumaton. Mutta koska a ja b ovat naapureita, on niillä eri värit laillisessa värityksessä, joten solmut eivät voi liittyä riippumattomaan joukkoon samalla kierroksella. Algoritmi siis muodostaa riippumattoman joukon.

Osoitetaan vielä, että joukko on maksimaalinen. Jotta joukko ei olisi maksimaalinen, täytyisi olla olemassa solmu v siten, että yksikään sen naapureista ei ole mukana riippumattomassa joukossa. Koska riippumattomasta joukosta ei kuitenkaan poistuta algoritmin ajon aikana, on varmaa, että yksikään v:n naapureista ei missään vaiheessa algoritmia kuulu riippumattomaan joukkoon. Tällöin solmu v huomaa oman värinsä kierroksella, että yksikään sen naapureista ei ole mukana riippumattomassa joukossa, ja liittyy mukaan riippumattomaan joukkoon. Solmua v ei siis voi olla olemassa, joten muodostuva joukko on maksimaalinen.

□

5 Yhteenveto

Tässä tutkielmassa kuvattu osuus hajautettujen paikallisten hajautettujen verkkoalgoritmien tutkimuksesta on suppea. Tässä tutkielmassa esitelty osuus paikallisista hajautetuista verkkoalgoritmeista on teoreettislähtöinen, vaikka osa paikallisten algoritmien tutkimuksesta on myös algoritmien tutkimista käytännön ehdoilla.

Tässä tutkielmassa on annettu esimerkkejä vain tilanteista, joissa prosessorit saavat tehdä kommunikaatiokierrosten väleillä suuren määrän työtä, eikä prosessorien toisilleen lähettämien viestien kokoja ole rajoitettu. Käytännön sovellutuksissa tällaisia myönnytyksiä algoritmeille ei kuitenkaan voi antaa. Käytännössä toimivan algoritmin olisi syytä tehdä myös kommunikaatiokierrosten väliset laskutoimitukset ripeästi, eikä tietoa voi lähettää suunnattomia määriä. Esimerkiksi algoritmissa 1 jokainen solmu kerää koko verkon rakenteen. Verkon koon kasvaessa tietomäärä voi kasvaa kuitenkin mielettömän suureksi, eikä tiedonsiirto tarvittavalla nopeudella välttämättä ole mahdollista.

Monet julkaisut paikallisten hajautettujen verkkoalgoritmien tutkimusalalla ovat esittäneet algoritmeja, joissa käytännön rajoitukset on otettu huomioon. Näissä algoritmeissa on tyypillisesti osoitettu, että lähetettävät viestit ovat kooltaan maltillisia. Mielenkiintoinen lähestymistapa on myös rajoitusten vähentäminen, sillä esimerkiksi lisäämällä alussa annetun tiedon määrää voidaan parantaa approksimaatioalgoritmien tarkkuutta.

Vaikka paikallisten algoritmien tutkimus on ollut teoreettislähtöistä, voidaan jo olemassaolevista komponenteista rakentaa käytännössä hyödyttäviä kokonaisuuksia. Esimerkiksi mobiiliverkoissa hajautettuja paikallisia algoritmeja käytetään jo nykyään [AP90], ja muissakin sensoriverkoissa jo olemassaolevia komponentteja voidaan hyödyntää. Erityisesti viestien välittämisen ja reitityksen optimointiin paikallisten algoritmien kehittäminen on tärkeää, ja useita algoritmeja tätä varten on tutkittukin [Urr07].

Vaikka paikallisten algoritmien tutkimus on vielä nuorta, vaikuttaa siltä, että paikallisille algoritmeille voi löytyä tulevaisuudessa useita käytännön sovellutuksia. Käsiteltävät tietomäärät kasvavat jatkuvasti, ja tietoa hajautetaan jatkuvasti enemmän. Tällöin paikalliset algoritmit, jotka kykenevät parhaimmillaan jopa vakioaikaiseen toimintaan, voivat olla huomattavan arvokkaita työkaluja ongelmanratkaisuun.

Lähteet

AGLP89 Awerbuch, B., Goldberg, A. V., Luby, M. ja Plotkin, S. A., Network decomposition and locality in distributed computation (extended abstract). 1989, sivut 364–369.

AP90 Awerbuch, B. ja Peleg, D., Sparse partitions (extended abstract). *In IEEE Symposium on Foundations of Computer Science*. IEEE, 1990, sivut 503–513.

BE09 Barenboim, L. ja Elkin, M., Distributed $(\Delta + 1)$-coloring in linear (in Δ) time. *STOC '09: Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. ACM, 2009, sivut 111–120.

CV86 Cole, R. ja Vishkin, U., Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70,1(1986), sivut 32–53.

GHS83 Gallager, R., Humblet, P. ja Spira, P., A distributed algorithm for minimum weight spanning trees. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5,1(1983).

GPS87 Goldberg, A., Plotkin, S. ja Shannon, G., Parallel symmetry-breaking in sparse graphs. *STOC '87 Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, sivut 315–324.

KMW06 Kuhn, F., Moscibroda, T. ja Wattenhofer, R., The price of being near-sighted. *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM Press, 2006, sivut 980–989.

Lin92 Linial, N., Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21,1(1992), sivut 193–201.

NS95 Naor, M. ja Stockmeyer, L., What can be computed locally? *SIAM Journal on Computing*, 24,6(1995), sivut 1259–1277.

PR01 Panconesi, A. ja Rizzi, R., Some simple distributed algorithms for sparse networks. *Distributed computing*, 14,2(2001), sivut 97–100.

PS09 Polishchuk, V. ja Suomela, J., A simple local 3-approximation algorithm for vertex cover. *Information Processing Letters*, 109,12(2009).

Suo10a Suomela, J., Lecture notes, 2010. `http://www.cs.helsinki.fi/u/josuomel/dda-2010/adobe/lecture-1.pdf`. [6.3.2011]

Suo10b Suomela, J., Lecture notes, 2010. `http://www.cs.helsinki.fi/u/josuomel/dda-2010/adobe/lecture-2.pdf`. [6.3.2011]

Suo11 Suomela, J., Survey of local algorithms, 2011. `http://www.cs.helsinki.fi/u/josuomel/doc/local-survey.pdf`. [6.3.2011]

Urr07 Urrutia, J., Local solutions for global problems in wireless networks. *Journal of Discrete Algorithms*, 5,3(2007), sivut 395–407.

WKLL11 Wattenhofer, R., Kuhn, F., Lenzen, C. ja Locher, T., Principles of distributed computing, 2011. `http://dcg.ethz.ch/lectures/podc_allstars/lecture/podc.pdf`