hyväksymispäivä arvosana

arvostelija

Paikalliset hajautetut verkkoalgoritmit

Mika Laitinen

Helsinki 11.4.2011

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
	11.4.2011	16 sivua + 0 liitesivua

Tiivistelmä — Referat — Abstract

Avainsanat — Nyckelord — Keywords

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 alueella on käytössään vain rajallinen määrä tietoa ympäristöstään. Paikalliset algoritmit toimivat verkoissa, joiden jokainen solmu esittää prosessoria. 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 sen alussa saama tieto on rajallista. Sen sijaan paikalliset algoritmit voivat kyetä ratkaisemaan hyvin tehokkaasti sellaisia ongelmia, joissa paikallisen ratkaisun oikeellisuuden osoittaminen ei ole hankalaa. Esimerkiksi solmuvärityksen 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 toisinaan 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.

Usein suurissa verkoissa kommunikaatio solmujen välillä on huomattavasti hitaam-

paa 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]. Perinteiseseen aikavaativuusanalyysiin verrattuna ero on merkittävä, sillä kommunikaatiokierrosten väleissä prosessorit voisivat teoriassa 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. On kuitenkin olemassa viitteitä, että joihinkin tiettyihin ongelmiin hajautettu laskentamalli voisi soveltua paremmin kuin aiemmat olemassaolevat mallit. 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 hajautettu laskentamalli, jossa ratkeavista ongelmista erityisesti keskitytään verkon väritysongelmaan. Havainnollistamistarkoituksessa esitetään 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 informaatiota 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 kom-

munikointitilan 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. Esimerkiksi maksimaalista itsenäistä joukkoa[1] haettaessa jokaisen solmun osaongelma on ratkaista, kuuluuko solmu maksimaaliseen itsenäiseen joukkoon vai ei.

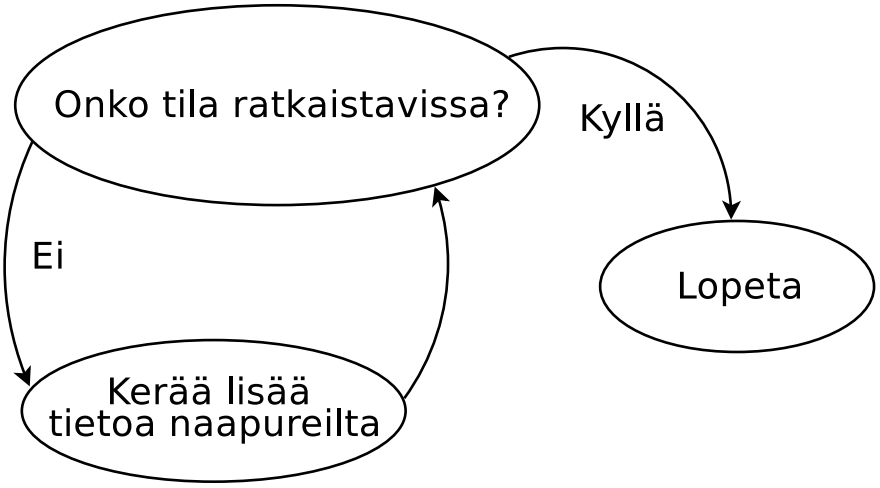
Hajautettu laskentamalli toimii verkoissa $G = (V, E)$, joiden jokaisessa solmussa $v \in V$ on prosessori. Solmussa x sijaitseva prosessori v_x voi kommunikoida solmussa y sijaitsevan prosessorin v_y kanssa jos, ja vain jos on olemassa $e \in E$ siten, että $e = (x, y)$. Prosessorit eivät jaa muistia 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 paikallinen algoritmi, jota prosessorit ajavat, kunnes ovat ratkaisseet oman paikallisen ongelmansa ja pysähtyneet. Paikallinen ongelma, jota prosessorit ratkaisevat, on prosessorin oma lopputila. Prosessorin lopputila tarkoittaa verkon kannalta sitä tilaa, mitä ratkaisemaan prosessori on osoitettu. Tämä tila on tyypillisesti esimerkiksi sen solmun tila, johon prosessori on sijoitettu. 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 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.

Mikäli prosessorit voidaan erottaa toisistaan esimerkiksi tunnistenumeroilla, voidaan kaikki ongelmat ratkaista yhdistetyissä verkoissa ajassa $O(\mathrm{diam}(G))$, jossa $\mathrm{diam}(G)$ on verkon halkaisija. Tässä ajassa kaikille prosessoreille saadaan kommunikoitua ko-

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

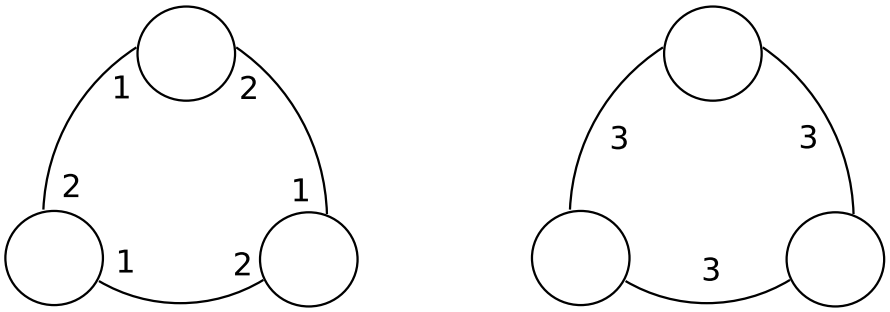


Kuva 1: Havainnollistus paikallisten algoritmien toiminnasta. Jokaisen kommunikaatiokierroksen välissä algoritmi selvittää, onko se saanut tarpeeksi tietoa, jotta se voisi ratkaista oman paikallisen ongelmansa. Mikäli tietoa ei ole vielä tarpeeksi, pyritään seuraavalla kommunikaatiokierroksella keräämään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

ko 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 $O(\mathrm{diam}(G))$-ajassa [Lin92].

2.1 Porttinumerointimalli

Suomela mainitsee kaksi mielenkiintoista tutkimuskohdetta hajautetussa laskentamallissa: mitä voidaan ratkaista alle $O(\mathrm{diam}(G))$-ajassa, mikäli tunnistenumerot ovat käytössä, ja mitä voidaan ratkaista, jos tunnistenumeroita ei ole käytössä [Suo10a]. 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.



Kuva 2: Vasemmalla kuva kolmisolmuisesta verkosta, jossa kaikki solmut ovat keskenään symmetrisessa 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.

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 symmetrisessa 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.

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. 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 symmetrisessa 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.

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 itsenäisen joukon etsiminen komponentettain on järkevää.

Algoritmi 1 on hyvin hidas, ja vie pahimmillaan lineaarisen ajan verkon kokoon 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.

Algorithm 1 Algoritmi kaikkien yhdistetyissä verkoissa ratkeavien ongelmien ratkaisemiseen tunnistenumeromallissa

repeat
  Kerää kaikki tieto naapuriprosessoreilta.
until Kierroksella ei saatu uutta tietoa verkon rakenteesta.
Ratkaise ongelma oman solmun näkökulmasta.
Valitse solmun lopputila ja lopeta algoritmin suoritus.

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 itsenäiset 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.

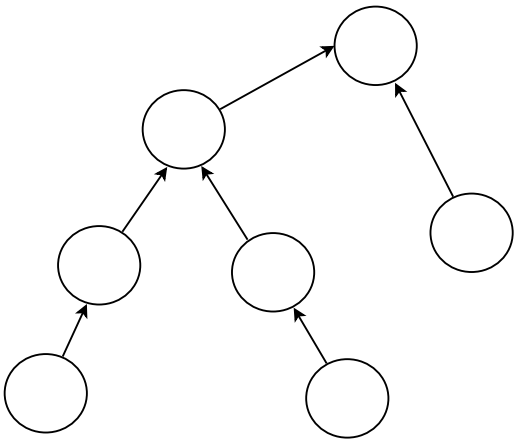
Tyypillinen ongelma hajautetussa laskentamallissa on verkon solmuväritys. Solmuvärityksessä ratkaisun laillisuus on paikallisesti tarkastettavissa oleva ominaisuus, sillä on yksinkertaista ja nopeaa varmistaa, että solmun naapurit valitsevat askeleen lopuksi eri värin kuin solmu itse.

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. 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 jälkeläinen (ks. kuva 3). Tällaisissa verkoissa voidaan aina vähentää värien määrä kolmeen. Colen-Vishkinin algoritmilla voidaan tehokkaasti vähentää värien määrä korkeintaan kuuteen. Kuudesta väristä päästään kolmeen väriin yksinkertaisemmilla algoritmeilla, tyypillisisesti sellaisilla, jotka vähentävät verkosta värejä lineearisesti kommunikaatiokierrosten määrää kohti.

3.1 Colen-Vishkinin algoritmi

Yksi ensimmäisistä tuloksista hajautetussa laskentamallissa on Colen-Vishkinin värien vähennysalgoritmi. Alun perin Colen-Vishkinin algoritmi oli tarkoitettu värien vähentämiseen linkitetyissä listoissa hajautetusti [CV86]. Algoritmi toimii kuiten-



Kuva 3: Suunnattu verkko, jonka jokaisella solmulla on korkeintaan yksi jälkeläinen. Verkon nuolet kuvaavat jälkeläissuhteita siten, että nuolen päässä oleva solmu on lähtöpisteessä olevan solmun jälkeläinen. Jälkeläissolmu voi kommunikaatiokierroksen aikana siirtää tietoa isäntäsolmulleen. Solmut ovat keskenään naapureita, mikäli toinen solmuista on toisen jälkeläinen.

kin myös paikallisena algoritmina. Algoritmi toimii suunnatuissa verkoissa, joissa jokaisella solmulla on korkeintaan yksi jälkeläinen, ks. kuva 3. Prosessori a voi siis keskustella prosessorin b kanssa vain, jos on olemassa $e \in E$ siten, että $e = (a, b)$, jossa (a, b) on suunnattu 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 jälkeläinen, 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 $x > 5$. Määritelmän 1 mukaisesti solmu, jonka väri on y on, voi valita uudeksi värikseen korkeintaan arvon $2 \cdot b(y) + 1$, jossa $b(a) = \lfloor \log_2 a \rfloor + 1$, eli $b(y)$ on y:n bittien määrä binääriesityksessä. Näin yläraja verkon suurimmaksi väriksi kierroksen jälkeen on $2 \cdot b(x) + 1$.

Algorithm 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 jälkeläinen u **then**
    Odota, että jälkeläinen u kertoo värinsä $C(u)$.
    Vertaa omaa väriä $C(v)$ jälkeläisen väriin $C(u)$.
    Etsi binäärimuodossa oikeanpuolimmaisin bitti, indeksiltään i, jossa värit poikkeavat.
    Valitse uudeksi väriksi $2 \cdot i + C(v)_i$, 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

Määritelmä 1. *Solmu x, jonka jälkeläinen on solmu y, valitsee uudeksi värikseen $n(x, y)$: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 $n(x, y) = f(x, y) \cdot 2 + x_{f(x,y)}$, jossa $f(x, y) = i$ siten, että $x_i \neq y_i$ ja i on mahdollisimman pieni. Mikäli solmulla x ei ole jälkeläistä, oletetaan jälkeläiseksi 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 jälkeläinen. Oletetaan, että b:llä on jälkeläinen c, $f(a,b) = i$, ja $f(b,c) = j$. Jotta solmut a ja b valitsisivat saman värin, täytyy päteä $n(a,b) = n(b,c) \Leftrightarrow f(a,b) \cdot 2 + a_{f(a,b)} = f(b,c) \cdot 2 + b_{f(b,c)}$. 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. □

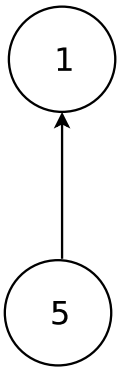
Jokaisella kierroksella Colen-Vishkinin algoritmissa jokainen solmu, jolla on jälkeläinen, kysyy jälkeläisensä väriä. Tämän jälkeen solmu vertaa omaa väriään jälkeläisensä väriin binäärimuodossa, ja etsii sen bitin indeksin, joka kahdessa värissä poikkeaa ensimmäisenä oikealta laskettuna. Tällainen bitti 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 bitin i arvo, eli 0 tai 1. Mikäli solmulla ei ole jälkeläistä, se valitsee uudeksi värikseen edellisen värinsä viimeisen bitin, aivan kuin se poikkeaisi olemattoman jälkeläisensä väristä jo bitissä 0.

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 4 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 voidaan 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.

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



Kuva 4: 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.

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?

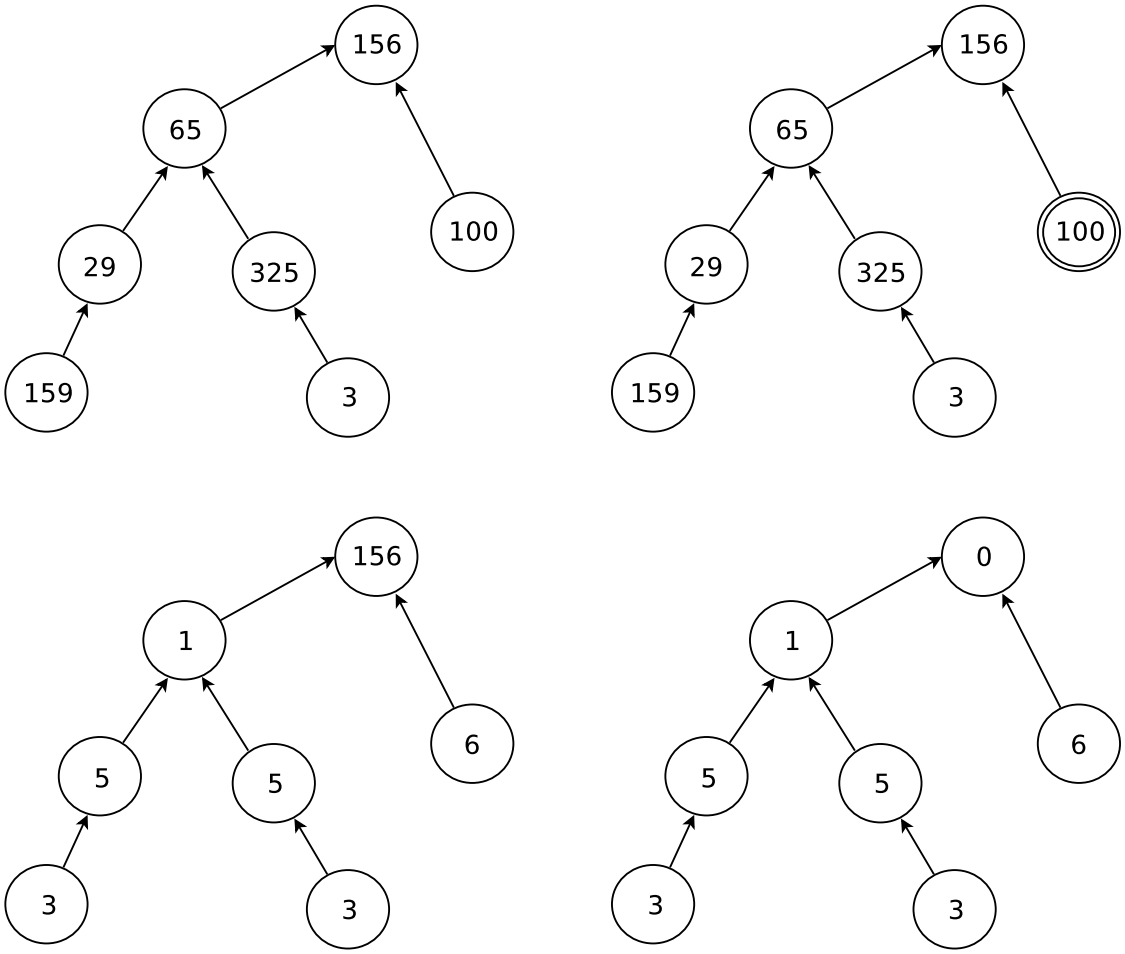
Värien määrää voidaan vähentää aina kolmeen asti algoritmilla, joka vähentää yhden värin jokaista kommunikaatiokierrosta kohti. Koska Colen-Vishkinin algoritmin ajamisen jälkeen värejä on maksimissaan kuusi, ei lineaarinen aikavaatimus haittaa, sillä kommunikaatiokierroksia tulee maksimissaan kolme lisää,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

Lineaarisessa värienvähennysalgoritmissa 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 jälkeläisensä nykyisen värin, ks. kuva 6. Mikäli jälkeläistä ei ole, valitaan edellisestä poikkeava väri joukosta $\{0, 1, 2\}$. Väitteen 3 mukaisesti siirto-operaatio säilyttää laillisen värityksen.

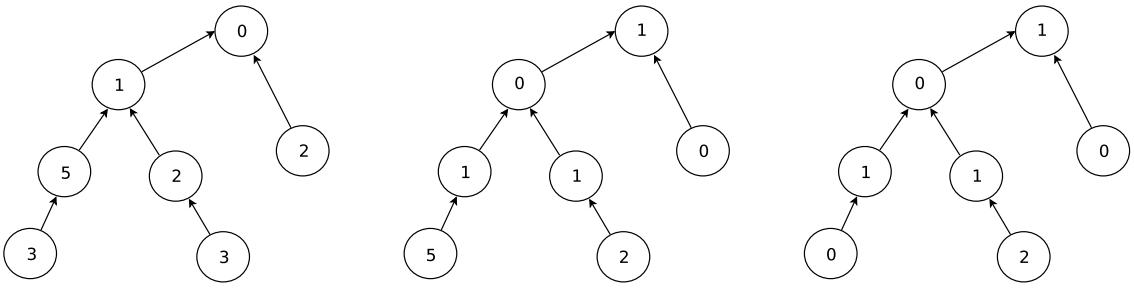
Väite 3. *Värien siirto-operaatio säilyttää laillisen värityksen.*

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



Kuva 5: Kuvasarja havainnollistamaan Colen-Vishkinin algoritmin yhtä kierrosta. a) Suunnattu verkko, jossa jokaisella solmulla on väri. b) Algoritmin suoritus kaikissa solmuissa samanaikaisesti. Solmu, jonka väri on 100, vertaa omaa väriään jälkeläisensä väriin binäärimuodossa. Värit $100 = 110\underline{0}100_2$ ja $156 = 1001\underline{1}100_2$ poikkeavat toisistaan kolmannessa bitissä oikealta laskettuna, joten solmu, jonka väri on 100, valitsee uudeksi värikseen $3 \cdot 2 + 0 = 6$. c) Kaikki solmut, joilla on jälkeläinen ovat vaihtaneet väriä b-kohdassa esitetyn menetelmän mukaisesti. d) Solmu, jolla ei ole jälkeläistä, teeskentelee poikkeavansa olemattoman jälkeläisensä kanssa jo nollannessa bitissä, jolloin se valitsee uudeksi värikseen värinsä viimeisen bitin, nollan.



Kuva 6: Jokainen solmu valitsee uudeksi värikseen jälkeläisensä vanhan värin. Mikäli solmulla ei ole jälkeläistä, valitsee se värin joukosta {0,1,2} niin, että sen väri vaihtuu. Lopuksi eliminoidaan kaikki väriä 5 olevat solmut siten, että ne valitsevat joukosta $\{0, 1, 2\}$ värin, jota niiden naapureilla ei ole.

samaa väriä.

Siis kaikki juurisolmun isännät valitsevat eri värin jälkeläisensä kanssa. Ne myös antavat vanhan värinsä kaikille isännilleen, ja ottavat jälkeläisensä 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. □

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 jälkeläisiä. Solmulla voi olla vain yksi jälkeläinen, 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.

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

for $i = 5, 4, 3$ **do**
  if solmulla on jälkeläinen 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 vanhasta 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 Värityksen sovellutuksia

TODO: kirjoita maksimaalisesta itsenäisestä joukosta ja maksimipariutuksesta.

5 Tutkimuskohteita ja päätelmiä

Tässä tutkielmassa kuvattu osuus hajautettujen paikallisten hajautettujen verkkoalgoritmien tutkimuksesta on suppea. Vaikka tässä tutkielmassa on esitelty paikallisia hajautettuja verkkoalgoritmeja teoreettisesta näkökulmasta, ei tutkimusalalla ole varsinaista konsensusta algoritmitutkimuksen teoreettisuusasteesta.

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. Lisäksi approksimaatioalgoritmeissa kasvattamalla alussa annetun tiedon määrää voi usein parantaa saavutetun tuloksen approksimaatiotarkkuutta.

Vaikka paikallisten algoritmien tutkimus on ollut teoreettislähtöistä, voidaan jo olemassaolevista komponenteista rakentaa käytännössä hyödyttäviä kokonaisuuksia. Esimerkiksi mobiiliverkoissa hajautettuja paikallisia algoritmeja käytetään jo nykyään, ja muissakin sensoriverkoissa jo olemassaolevia komponentteja voidaan hyödyntää [Urr07]. Yksinkertaisena esimerkkinä voidaan mainita esimerkiksi pienimmän virittävän puun laskenta sensoriverkossa siten, että kaaren paino on sitä pienempi, mitä vähemmän energiaa kuluu lähetettäessä tietoa kaaren yli. Näin voidaan ratkaista, mitä yhteyksiä sensoreiden välillä kannattaa käyttää.

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. Aivan kaikkiin ongelmiin paikallisten algoritmien yleistymistä ei kuitenkaan liene realistista olettaa, sillä prosessoriverkon luominen on usein aivan liian suuri prosessi, ellei prosessoriverkkoa ole jo valmiiksi käyttövalmiina.

Lähteet

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).

KMW06 Kuhn, F., Moscibroda, T. ja Wattenhofer, R., The price of being nearsighted. *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.

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.