hyväksymispäivä arvosana

arvostelija

Paikalliset hajautetut verkkoalgoritmit

Mika Laitinen

Helsinki 3.4.2011

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
	3.4.2011	10 sivua + 0 liitesivua

Tiivistelmä — Referat — Abstract

Tässä esseessä kerrotaan paikallisista hajautetuista verkkoalgoritmeista. Esseessä esitellään paikallisen hajautetun laskentamallin vaikutuksia, ja ongelmia, jotka ratkeavat mallissa luontevasti. Esseessä pyritään myös havainnollistamaan lukijalle hajautettujen paikallisten verkkoalgoritmien toimintaa käytännössä, sekä esitellään mallin mahdollisia sovelluksia käytännön tarkoituksia varten.

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, *PHV*:eitä, joissa verkon osat ratkaisevat itseään koskevia osaongelmia vain paikallista tietoa hyödyntämällä.

Paikalliset hajautetut verkkoalgoritmit ovat algoritmeja, joissa jokaisella verkon alueella on käytössään vain rajallinen määrä tietoa ympäristöstään. PHV:t toimivat suuntaamattomissa 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 hajautetut verkkoalgoritmit 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 PHV:eille mahdotonta, mikäli sen alussa saama tieto on rajallista. PHV:t sen sijaan 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].

PHV:eillä pyritään tyypillisesti ratkaisemaan ongelmia hyvin suurissa verkoissa, sillä PHV:eiden aikavaatimukset kasvavat usein hyvin hitaasti verkon koon funktiona, ja toisinaan ongelmat voidaan ratkaista jopa vakioajassa [Suo11]. Mikäli nopeaa tarkan vastauksen antavaa PHV:tä ei ole mahdollista kehittää, on usein kuitenkin mahdollista kehittää nopea PHV 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 suuri operaatio. Tästä syystä PHV:eiden aikavaativuusanalyysissa ollaan kiinnostuneita vain prosessorien välisten kommunikaatiokierrosten määrästä [Lin92]. Perinteiseseen aikavaativuusanalyysiin verrattuna ero on merkittävä, sillä kommunikaatiokierrosten väleissä prosessorit voivat 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 PHV:eihin on huomattava, sillä neuronit voivat olla yhteydessä vain rajalliseen määrään lähistöllä sijaitseviin muihin neuroneihin. Näiden neuroneiden täytyy keskenäisen kommunikointinsa perusteella tulla yhteiseen lopputulokseen, joka vaikuttaa ihmisen toimintaan.

Paikalliset hajautetut verkkoalgoritmit on tutkimusalana vielä nuori, eikä tutkimus ole jumiutunut tietyille urille, vaan alalla on vielä useita avoimia tutkimusongelmia, jotka poikkeavat parhaimmillaan hyvin paljon toisistaan. Tässä esseessä annetaan yleiskuva PHV:eillä ratkeavista ongelmista sekä hankalista ongelmista. Lisäksi pyritään havainnollistamaan PHV:eiden toimintaa esittelemällä PHV:eitä yksinkertaisiin ongelmiin.

2 Hajautettu laskentamalli

Hajautettu laskentamalli toimii suuntaamattomassa verkossa $G = (V, E)$, jonka 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 $e = \{x, y\}$, 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 PHV, jota prosessorit ajavat, kunnes ovat ratkaisseet oman lokaalin ongelmansa ja pysähtyneet. Lokaali ongelma, jota prosessorit ratkaisevat, on

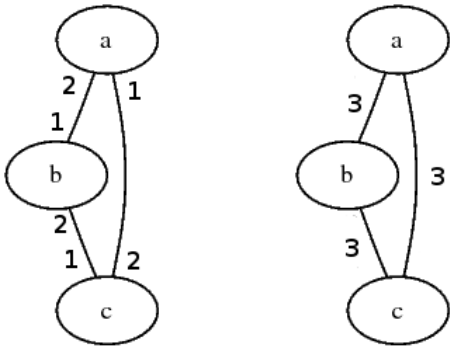
prosessorin oma lopputila. Prosessorin lopputila tarkoittaa verkon kannalta sitä tilaa, mitä ratkaisemaan prosessori on osoitettu. Tämä tila on tyypillisesti esimerkiksi sen solmun tila, johon prosessori on sijoitettu. Esimerkiksi verkon väritysongelmissa prosessorin lopputila on prosessorin omalle solmulleen valitsema väri.

PHV:t toimivat 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 ajassa $O(\mathrm{diam}(G))$, jossa $\mathrm{diam}(G)$ on verkon halkaisija. 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 $O(\mathrm{diam}(G))$-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 1: 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, ja löytää pienin virittävä puu, jos millään kahdella kaarella ei ole samaa painoa [PS09, 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 tunnistenumero-

malliin.

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. Toisin sanoen, mikäli käytetään tarpeeksi aikaa, mikä tahansa ongelma, jolla on ratkaisu, voidaan ratkaista algoritmilla 1. Tämän vuoksi on luontevaa keskittyä tutkimaan ongelmia, jotka ovat paikallisesti laskettavia. Myös approksimaatioalgoritmien löytäminen on kiinnostava tutkimuskohde, varsinkin jos ongelman tarkka ratkaisu ei ole paikallisesti laskettavissa.

Algorithm 1 Algoritmi kaikkien 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.

Seuraavassa kappaleessa esitellään ongelmatyyppejä, joiden lähestyminen on luontevaa tunnistenumeromallissa. Seuraavassa kappaleessa pyritään havainnollistamaan PHV:eiden käyttäytymistä ongelmanratkaisussa, ja analysoidaan tehokasta algoritmia väritysongelmaan.

3 Tyypilliset ongelmat

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 PHV:eillä. 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. Monet ensimmäiset merkittävät tulokset hajautetussa laskentamallissa liittyvätkin juuri solmuvärittämiseen. Solmuvärittämisessä onkin monia tutkimuskohteita, esimerkiksi rajoitettujen verkkojen kolmivärittäminen tai verkon ominaisuuksista, kuten solmujen maksimiasteesta riippuvien nopeiden väritysalgoritmien löytäminen.

Yksi ensimmäisistä tuloksista hajautetussa laskentamallissa on Colen-Vishkinin algoritmi värien vähentämiseen suunnatussa verkossa, jossa jokaisella solmulla on korkeintaan yksi jälkeläinen. Aiemmasta määritelmästä poiketen 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.

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, sillä Colen-Vishkinin algoritmi kuitenkin vähentää värien määrää kierroksen i värien määrää c_i logaritmisesti, eli $\log(c_i) = c_{i+1}$. Kierroksella $i + 2$ värien määrä on vastaavasti luokkaa $\log(\log(c_i))$, eli algoritmi vaatii iteroidun logaritmin verran kommunikaatiokierroksia kuuden värin saavuttamiseen. Täten algoritmin aikavaativuus on $O(\log^* k)$, jossa k on värien määrä alussa.

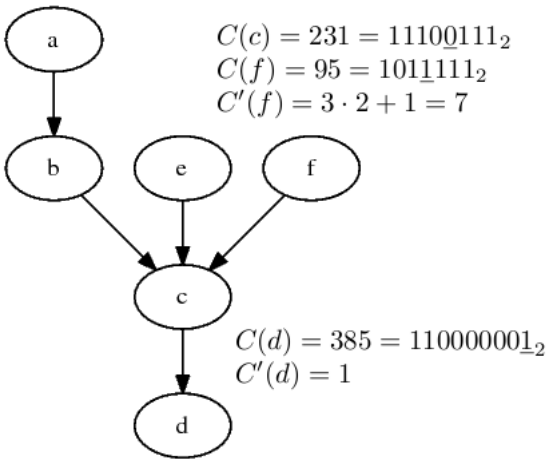
Algorithm 2 Colen-Vishkinin algoritmi

while $c(v) > 5$ **do**
  if v:llä on jälkeläinen u **then**
    Kysy jälkeläisen u väriä $C(u)$.
    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 while

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.

Jotta algoritmi toimisi oikein, täytyy osoittaa, että se säilyttää laillisen värityksen jokaisella kierroksella. Jotta kaksi naapurisolmua a ja b voisivat valita saman värin, täytyy olla $2 \cdot i_a + C(a)_{i_a} = 2 \cdot i_b + C(b)_{i_b} \Leftrightarrow 2(i_a - i_b) = C(b)_{i_b} - C(a)_{i_a}$, jossa i_a ja i_b ovat indeksit, joissa solmujen a ja b värit poikkeavat jälkeläistensä väreistä. Koska yhtälön oikea puoli voi saada arvoja vain joukosta $\{-1, 0, 1\}$, täytyy olla $i = i_a = i_b$, eli $0 = C(b)_i - C(a)_i \Leftrightarrow C(a)_i = C(b)_i$. Näin ei kuitenkaan voi olla, sillä jos a ja b ovat toistensa naapureita, täytyy toisen olla toisen jälkeläinen. Tällöin $C(a)_i \neq C(b)_i$, eli a ja b valitsevat eri värit, ja algoritmi tuottaa jokaisella kierroksella laillisen värityksen.

Colen-Vishkinin algoritmilla ei kuitenkaan voida aina päästä alle kuuden värin, ja algoritmin suorituksen voikin solmussa lopettaa, mikäli valittu väri on välillä $0 - 5$. Esimerkiksi jos solmun jälkeläisen väri on 1, ja oma väri on 5, valitaan uudeksi väriksi jälleen 5.



Kuva 2: Colen-Vishkinin algoritmin yhden askeleen havainnollistamista. Kuvassa solmu f pyrkii ratkaisemaan seuraavan värinsä, ja kysyy ainoan jälkeläisensä c:n väriä. Värit $C(c)$ ja $C(f)$ poikkeavat indeksissä $i = 3$ oikealta laskettuna. Algoritmin 2 mukaisesti solmu f valitsee uudeksi värikseen $C'(f) = 2 \cdot i + C(v)_i$. Muut solmut toimivat samoin, paitsi jos niillä ei ole jälkeläistä. Tällöin solmu valitsee oman värinsä viimeisen bitin, eli solmun d tapauksessa uudeksi väriksi tulee 1.

4 Tutkimuskohteita ja päätelmiä

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

Tässä esseessä 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 al-

goritmeissa on tyypillisesti osoitettu, että lähetettävät viestit ovat kooltaan maltillisia. Lisäksi approksimaatioalgoritmeissa kasvattamalla alussa annetun tiedon määrää voi usein parantaa saavutetun tuloksen approksimaatiotarkkuutta.

Vaikka tutkimus on PHV:eiden alalla 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 paikalliset hajautettujen verkkoalgoritmien tutkimus on vielä nuorta, vaikuttaa siltä, että käytännön sovellusten määrä PHV:eille voi kasvaa tulevaisuudessa paljon. Käsiteltävät tietomäärät kasvavat jatkuvasti, ja tietoa hajautetaan jatkuvasti enemmän. Tällöin PHV:t, jotka kykenevät parhaimmillaan jopa vakioaikaiseen toimintaan, voivat olla huomattavan arvokkaita työkaluja ongelmanratkaisuun. Aivan kaikkiin ongelmiin PHV:eiden 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 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.

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.