hyväksymispäivä arvosana

arvostelija

Paikalliset hajautetut verkkoalgoritmit

Mika Laitinen

Helsinki 3.3.2011

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Sisältö

1 Johdanto

Paikalliset hajautetut verkkoalgoritmit ovat algoritmeja, joissa verkon tietomäärää ei käsitellä keskitetysti, vaan jokaisella verkon alueella on käytössään vain rajallinen määrä tietoa ympäristöstään. Paikallinen hajautettu verkkoalgoritmi, *PHV*, on algoritmi, joka ajetaan jokaisessa verkon solmussa. Jokainen verkon solmu sisältää prosessorin. Prosessorit voivat välittää muille prosessoreille lisätietoa, mikäli niiden välinen keskustelu on etukäteen mahdollistettu. Verkossa tämä tarkoittaa sitä, että prosessorisolmut voivat keskustella keskenään, jos ja vain jos näiden kahden prosessorisolmun välillä on kaari.

Paikalliset hajautetut verkkoalgoritmit poikkeavat ohjelmoinnille tyypillisestä ajatusmaailmasta — sen sijaan, että ohjelmoidaan yksittäisiä prosessoreita, ohjelmoidaan verkkoja, joissa tehokkaasti ratkeavat ongelmat poikkeavat yksittäisillä prosessoreilla ratkaistavista ongelmista, tai joihin ei ole olemassa käytännöllistä tehokasta algoritmia. PHV:eillä pyritään tyypillisesti ratkaisemaan ongelmia hyvin suurissa verkoissa, sillä PHV:iden aikavaatimukset kasvavat usein hyvin hitaasti verkon koon funktiona, ja toisinaan ongelmat voidaan jopa ratkaista vakioajassa. Mikäli nopeaa tarkan vastauksen antavaa PHV:tä ei ole mahdollista kehittää, on usein kuitenkin mahdollista kehittää nopea PHV approksimoimaan oikeaa tulosta, ja toisinaan, varsinkin hyvin suurten verkkojen tapauksessa, tällainen tulos voi olla riittävä.

Verkkojen ohjelmointi ajamalla samaa algoritmia jokaisessa verkon solmussa on mielenkiintoinen lähestymistapa myös siksi, että tällaista laskentamallia ei tunneta vielä kovin hyvin, mutta on olemassa kuitenkin 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.

Tässä tutkielmassa tutustutaan siihen, minkälaisia ongelmia paikallisilla hajautetuilla algoritmeilla voidaan ratkaista. Tutkielmassa katetaan sekä positiivisia että negatiivisia tuloksia eri ongelmiin, ja pyritään muodostamaan kokonaiskuva PHV:eiden mahdollisista käyttötarkoituksista.

2 Hajautettu laskentamalli

Kyseessä oleva malli koostuu suuntaamattomasta verkosta $G = (V, E)$, 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 solmun tilaa, 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.

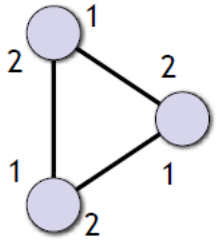
Mikäli prosessorit voidaan erottaa toisistaan esimerkiksi tunnistenumeroilla, voidaan kaikki ongelmat ratkaista ajassa $O(\mathrm{diam}(G))$, jossa $\mathrm{diam}(G)$ on verkon halkaisija. Tässä ajassa kaikille prosessoreille saadaan kommunikoitua koko verkon rakenne. Koska prosessorien käyttämää laskenta-aikaa 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 [1] kutsuu ongelmaa *paikallisesti laskettavaksi*, mikäli ongelma voidaan ratkaista alle $O(\mathrm{diam}(G))$-ajassa.

2.1 Porttinumerointimalli

Suomela [2] 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. Malleista, joissa prosessoreilla ei ole tunnistenumeroita, Suomela mainitsee porttinumerointimallin. Porttinumerointimalli ei ole kovin vahva malli, ja se pystyy ratkaisemaan vain tietynlaisia ongelmia. Porttinumerointimallissa solmu, jonka asteluku on d, voi viitata sen naapureihin kokonaisluvuilla $1, 2, \ldots, d$. Nämä viitenumerot annetaan jokaiselle solmulle ennen algoritmin ajoa.

Huolimatta siitä, että kyseessä on heikko malli, voidaan porttinumerointimallilla ratkaista joitain epätriviaaleja verkko-ongelmia. Suomela [3] osoittaa, että porttinumerointimallilla voidaan luoda 3-approksimointialgoritmi pienimmän solmupeitteen löytämiseen. Porttinumerointimallin suurin ongelma on kuitenkin se, että on olemassa paljon ongelmia, joissa voidaan konstruoida tilanne, jossa porttinumerointimallissa ei ole mahdollista löytää algoritmia, joka kykenisi ratkaisemaan tilanteen.

Kuvassa 1 on verkon väritysongelma, jossa kaikki verkon solmut ovat keskenään symmetrisessa tilanteessa. Tällöin kaikkien solmujen ajaessa samaa algoritmia, niiden pitäisi päätyä samaan lopputulokseen. On selvää, että verkon väritysongelmassa ei ole kuitenkaan suotavaa, että kaikki verkon solmut valitsevat itselleen saman värin, joten verkon väritys ei ole ongelma, jonka voisi kaikissa tapauksissa ratkaista porttinumerointimallilla.



Kuva 1: Kuva kolmisolmuisesta verkosta, jossa kaikki solmut ovat keskenään symmetrisessa tilanteessa, joten tilanne on porttinumerointimallissa ratkaisukelvoton. Kuvan kokonaisluvut kuvaavat solmujen naapureiden porttinumerointeja.

Symmetrian voi rikkoa satunnaisuuden avulla, mutta on olemassa myös deterministinen vaihtoehto symmetrian rikkomiseen. Uniikkeja tunnistenumeroita käyttämällä ei voida päätyä symmetriseen tilanteeseen [1].

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

for all $v \in V$ **do**
  Kerää kaikki tieto verkosta G.
  Ratkaise ongelma oman solmun näkökulmasta.
  Valitse solmun lopputila ja lopeta algoritmin suoritus.
end for

Tässä tutkielmassa keskitytään tunnistenumeromallin mahdollisuuksien analysoimiseen ja esitellään sekä positiivisia että negatiivisia tuloksia siitä, mitä ongelmia voidaan ratkaista hajautetussa laskentamallissa vakioajassa. Lisäksi huomiota kiinnitetään paikallisesti laskettaviin ongelmiin, joihin on olemassa tehokkaita algoritmeja.

3 Tyypilliset ongelmat

Hajautetussa laskentamallissa ongelmien ratkaisumenetelmät ovat huomattavasti erilaisia kuin perinteisempien ongelmien ratkaisussa käytettävät menetelmät. Tyypillinen ongelma hajautetussa laskentamallissa on verkon väritys. Monet ensimmäiset merkittävät tulokset hajautetussa laskentamallissa liittyvätkin juuri verkon värittämiseen. Verkon värittämisessä onkin monia tutkimuskohteita, esimerkiksi verkon kolmivärittäminen tai verkon ominaisuuksista, esimerkiksi 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. Colen-Vishkinin algoritmilla valmista verkon väritystä voidaan vähentää siihen asti, kunnes verkossa on jäljellä enää kuusi väriä. Suunnatussa verkossa, jossa solmuilla on korkeintaan yksi jälkeläinen, on mahdollista löytää 3-väritys. Colen-Vishkinin algoritmin löytämästä 6-värityksestä voidaan päästä 3-väritykseen vakioajassa käyttämällä algoritmeja, jotka vähentävät verkosta yhden värin kommunikaatiokierrosta kohti.

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, ei ongelma olisi edes paikallisesti ratkeava.

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

Algorithm 2 Colen-Vishkinin algoritmi

```
for all v ∈ V do
  if v:llä on jälkeläinen u then
    Kysy jälkeläisen u väriä c(u).
    Vertaa omaa väriä c(v) jälkeläisen väriin c(u).
    Etsi oikeanpuolimmaisin bitti, indeksiltään i, jossa värit poikkeavat.
    Valitse uudeksi väriksi 2 · i + c(v)_i, jossa c(v)_i on nykyisen värin i:nnen bitin
    arvo, eli 0 tai 1.
  else
    Aseta uudeksi väriksi 1.
  end if
end for
```

Lähteet

1 Nathan Linial. Locality in distributed graph algorithms. SIAM Journal on Computing, 21(1):193-201, 1992;

2 Jukka Suomela. Models of distributed computing: port numbering and local algorithms. `http://www.cs.helsinki.fi/u/josuomel/doc/fmt-presentation-2010-02-26.pdf`. Luettu 21.02.2011.

3 Jukka Suomela. `http://www.cs.helsinki.fi/u/josuomel/dda-2010/adobe/lecture-1.pdf`. Luettu 21.02.2011.