KLUSTEROINTI HARVOISSA GRAAFEISSA

Havu Miikonen

Pro gradu -tutkielma
Heinäkuu 2022

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

MIIKONEN, HAVU: Klusterointi harvoissa graafeissa
Pro gradu -tutkielma, 39 s.
Matematiikka
Heinäkuu 2022

Tutkielmassa esitellään graafin solmujen klusterointiongelma, tarkastellaan sitä erityisesti harvoissa graafeissa ja ratkaistaan se hajautetun laskennan mallissa.

Lähdetään ensin liikkeelle määrittelemällä graafin metsäisyys ja mitä harvalla graafilla tarkoitetaan. Ensimmäisessä luvussa myös tutustutaan riippumattomiin joukkoihin. Lopuksi esitellään tiukka yläraja maksimaalisen riippumattoman joukon laskevan ahneen algoritmin suoritusajalle, kun algoritmia ajetaan rinnakkaisesti.

Toisessa luvussa määritellään hajautetun laskennan mallit LOCAL ja MPC (*Massively Parallel Computation*), joista jälkimmäistä on tutkittu laajalti 2010-luvulla. MPC-mallissa on käytössä tekniikat graafin eksponentiaatio ja kierrosten tiivistäminen, joiden avulla voidaan simuloida LOCAL-algoritmeja tehokkaasti.

Luvussa 3 perehdytään korrelaatioklusterointiin. Esitellään ongelman ratkaiseva yksinkertainen algoritmi ja todistetaan, että se on 3-approksimaatio. Sitten pohditaan, miten algoritmi voidaan toteuttaa rinnakkaistetusti. Lopuksi osoitetaan muutamia ominaisuuksia, joita pienen metsäisyyden graafeilla on mitä korrelaatioklusterointiin tulee ja hyödynnetään niitä algoritmien suunnittelussa.

Neljännessä luvussa esitellään tutkielman päätulos, MPC-algoritmi korrelaatioklusteroinnille, joka on eksponentiaalisesti nopeampi kuin aiempi nopein tunnettu algoritmi. Sen toteutuksessa hyödynnetään luvun 2 tekniikoita ja luvun 1 tulosta maksimaalisista riippumattomista joukoista.

Asiasanat: graafiteoria, hajautettu laskenta, klusterointi, metsäisyys.

Sisältö

Johdanto

Graafi on tietorakenne, jossa on solmuja ja kaaria. Niillä voi kuvata objektien keskinäisiä suhteita. Kuvitellaan jokin sosiaalisen median verkosto, jolla on miljardi käyttäjää. Verkostoa voidaan kuvata graafilla, jossa solmut ovat käyttäjiä ja kaaret ovat käyttäjien väliset kaveruussuhteet.

Harvassa graafissa on vain vähän kaaria suhteessa solmujen määrään. Oletetaan, että kuvittelemamme somen käyttäjällä voi olla korkeintaan 5000 kaveria ja että kavereiden määrän keskiarvo on 300. Graafi on erittäin harva suhteessa solmujen määrään ja graafin maksimiasteeseen.

Klusterointi on datapisteiden ryhmittelyä jossain mielessä keskenään samanlaisiin joukkoihin. Se on hyvin tyypillinen ohjaamattoman oppimisen menetelmä. On olemassa lukuisia ongelmia, joiden ratkaisuun kuuluu datan ryhmittely. Klusterointi edellä kuvatussa sosiaalisen median kaveruusgraafissa olisi esimerkiksi kaveriporukoiden, yhteisöjen ja sukujen tunnistamista.

Pitkään tietokoneiden suorituskyky kehittyi siten, että mikrosirut pienenivät ja tulivat edullisemmaksi valmistaa, mutta jossain vaiheessa kehitys on tullut siitä, että koneissa on useita ytimiä, jotka käyttävät yhteistä tallennustilaa. Nykyään yhteen ongelmaan liittyvää dataa on niin paljon, että se ei välttämättä mahdu yhden koneen tallennustilaan laskennan tilavaativuudesta puhumattakaan. Laskennassa tarvitaan siis useita koneita rinnakkain.

Kommunikointi koneiden välillä on hidasta tietokoneen laskennan nopeuden mittakaavassa. Moderni prosessori voi suorittaa yhden alkeisoperaation mikrosekunneissa, mutta toiselle koneelle samassa verkossa yhden bitin välittäminen vie joitain millisekunteja. Laskennassa pullonkaula on siis koneiden keskinäisen kommunikaation määrässä. Avuksi otetaan hajautettu laskenta, jossa ongelma pyritään ratkaisemaan niin, että mahdollisimman suuri osa laskennasta tapahtuu yksittäisillä koneilla ja koneiden tarvitsee kommunikoida keskenään vain vähän.

Graafin solmujen ryhmittelyä, korrelaatioklusterointia, ja `Pivot`-algoritmia on käsitelty aiemminkin rinnakkaistetuissa ja hajautetuissa laskennan malleissa, muun muassa artikkeleissa [1] ja [2]. Tässä tutkielmassa tutkitaan klusterointia erityisesti sublineaarisen muistin MPC-mallissa ja kun kohteena ovat harvat graafit.

1 Graafiteoriaa

Esitellään ensin kautta tutkielman käytetyt merkinnät ja käsitteet.

Peräkkäisten kokonaislukujen joukolle käytetään merkintää $[m, n] = \{m, \ldots, n\}$ ja joukon A koolle merkintää $\#A$.

Olkoon $G = (V, E)$ yksinkertainen suuntaamaton graafi. Graafi $H = (V_H, E_H)$ on graafin G aligraafi, jos $V_H \subseteq V$ ja $E_H \subseteq E$, ja merkitään $H \subseteq G$. Joukon $V' \subseteq V$ *indusoima* aligraafi on $G[V'] = (V', E')$, missä $E' = \{uv \in E : u, v \in V'\}$. Vastaavasti kaarijoukon $E' \subseteq E$ indusoimassa aligraafissa $G[E'] = (V', E')$ solmujoukko on $V' = \bigcup_{e \in E'} e$.

Kahden solmun u ja v välinen etäisyys $d(u, v)$ on lyhimmän solmujen u ja v välisen polun pituus. Jos solmujen välillä ei ole polkua, merkitään $d(u, v) = \infty$. Merkitään graafin läpimittaa $diam(G) = \max_{u,v \in V} d(u, v)$. Graafi G on yhtenäinen, kun $diam(G) < \infty$. Yhtenäinen komponentti on graafin maksimaalinen yhtenäinen aligraafi. Klikiksi kutsutaan aligraafia, joka on täydellinen, siis jossa kaikkien solmuparien välillä on kaari. Täydellistä graafia, jossa on n solmua, merkitään K_n.

Solmun $v \in V$ naapurusto on joukko $N(v) = \{u \in V : uv \in E\}$ ja solmun aste on $d(v) = \#N(v)$. Merkitään graafin G maksimiastetta $\Delta = \max_{v \in V} d(v)$. Merkitään solmun naapurustoa, astetta ja maksimiastetta jossain aligraafissa alaindeksillä, esimerkiksi aligraafin H maksimiaste on Δ_H. Lisäksi asteelle solmujen osajoukon $V' \subseteq V$ indusoimassa aligraafissa käytetään lyhennysmerkintää $d_{G[V']}(v) = d_{V'}(v)$. Solmun k-naapurustoksi kutsutaan joukkoa $\{u \in V : 1 \leq d(v, u) \leq k\}$.

Graafi $G = (V, E)$ on puu, jos se on yhtenäinen ja $\#E = \#V - 1$. Toisin luonnehdittuna puut ovat minimaalisia yhtenäisiä graafeja. Graafi on metsä, jos sen yhtenäiset komponentit ovat puita. Puun astetta 1 olevia solmuja kutsutaan lehtisolmuiksi. Tähtigraafi on puu, jossa kaikki solmut yhtä lukuun ottamatta ovat lehtiä.

Esitellään varsin hyödyllinen graafiteorian klassikkotulos.

Lemma 1.1 (Kättelylemma)**.** *Olkoon $G = (V, E)$ graafi. Silloin*

$$\sum_{v \in V} d_G(v) = 2 \cdot \#E.$$

1.1 Metsäisyys

Tässä tutkielmassa käsitellään niin kutsuttuja harvoja graafeja. Luontevasti ajateltuna graafi on harva, jos siinä on vähän kaaria suhteessa solmujen määrään ja kaaria ei ole missään aligraafissa tiheästi. Määritellään tämä ajatus formaalisti.

Määritelmä 1.2 (Metsäisyys)**.** Graafin $G = (V, E)$ metsäisyys λ_G on

$$\lambda_G = \max_{H \subseteq G} \left\lceil \frac{\#E_H}{\#V_H - 1} \right\rceil,$$

missä aligraafeilla $H = (V_H, E_H)$ on $\#V_H \geq 2$.

Huomataan, että graafi G on metsä, jos ja vain jos $\lambda_G = 1$. Havaitaan myös, että jos H on graafin G aligraafi, niin $\lambda_H \leq \lambda_G$, koska kaikki graafin H aligraafit ovat myös graafin G aligraafeja.

Metsäisyys on eräänlainen graafin lokaalin tiheyden mittari. Se luonnehtii graafia rajoittamatta sen rakennetta tarkasti. Toisin on esimerkiksi tasograafien laita, jotka voidaan karakterisoida graafeiksi, joissa ei ole alirakenteena täydellisiä graafeja K_5 ja $K_{3,3}$. Näytetään seuraavaksi, että λ-metsäisessä graafissa ei voi olla liian suurta täydellistä graafia aligraafina.

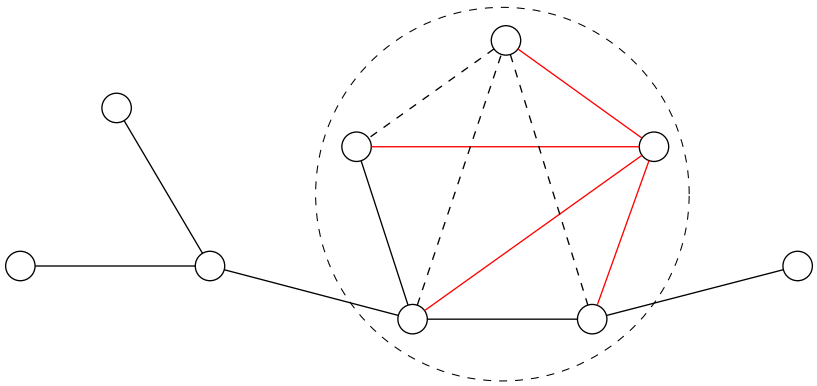
Lemma 1.3. *Täydellisen graafin K_n metsäisyys on $\lambda_{K_n} = \left\lceil \frac{n}{2} \right\rceil$.*

Todistus. Graafin K_n kaikki indusoidut aligraafit on täydellisiä graafeja K_k, joissa on k solmua ja $\binom{k}{2}$ kaarta. Siis

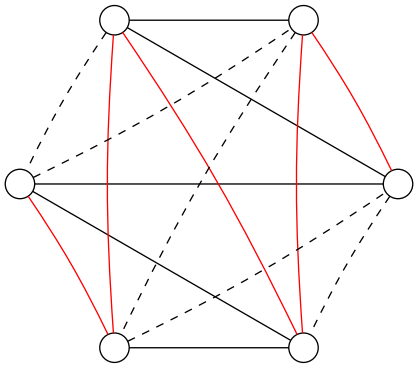
$$\lambda_{K_n} = \max_{k \in [1,n]} \left\lceil \frac{\binom{k}{2}}{k-1} \right\rceil = \left\lceil \frac{n}{2} \right\rceil.$$

□

Edellisestä lemmasta seuraa, että λ-metsäisessä graafissa ei voi olla aligraafina klikkiä $K_{2\lambda+1}$. Tämä on välttämätön mutta ei riittävä ehto graafin λ-metsäisyydelle. Kuvassa 1 on 3-metsäinen graafi, jonka suurin klikki on K_4.



Kuva 1: Esimerkki 3-metsäisen graafin kaarien partitiosta kolmeen metsään. Katkoviivalla ympyröity aligraafi antaa maksimin metsäisyyden määritelmässä.



Kuva 2: Kuuden solmun täydellinen graafi partitioituna kolmeen puuhun.

Esitetään seuraava luonnehdinta metsäisyydelle ilman todistusta. Sen esitti brittiläinen matemaatikko Crispin Nash-Williams vuonna 1961 ([3]) ja ekvivalentin muodon esitti brittiläis-kanadalainen koodinmurtaja William Tutte ([4]) aiemmin samana vuonna. Eräs todistus lauseelle on artikkelissa [5].

Lause 1.4 (Nash-Williamsin lause)**.** *Graafin $G = (V, E)$ (oletus että $\#V \geq 2$) kaaret voidaan partitioida τ metsäksi, jos ja vain jos kaikille graafin G aligraafeille H pätee*

$$\#E_H \leq \tau\,(\#V_H - 1)\,. \tag{1}$$

Huomautettakoon, että λ_G on pienin luku τ, jolle ehto (1) pätee. Siis λ-metsäisen graafin kaaret voidaan partitioida λ metsäksi.

Aiemmin osoitettiin, että täydellisen graafin K_n metsäisyys on $\lceil \frac{n}{2} \rceil$. Edellisestä lauseesta seuraa siis, että täydellinen graafi K_n voidaan partitioida $\lceil \frac{n}{2} \rceil$ puuhun. Parillisilla n partitio saadaan helposti Hamiltonin poluista, kuten kuvassa 2. Parittomilla n partition puut ovat $\frac{n}{2}$ polkua aligraafista K_{n-1} ja yksi tähtigraafi, jossa viimeisestä solmusta on kaari jokaiseen aligraafin K_{n-1} solmuun.

Tarkastellaan seuraavaksi, miten metsäisyys liittyy graafin solmujen asteisiin.

Lemma 1.5. *Graafin G keskimääräinen aste on pienempi kuin $2 \cdot \lambda_G$.*

Todistus. Merkitään $\#V = n$. Lasketaan graafin solmujen asteet käyttämällä ensin kättelylemmaa 1.1, sitten määritelmää 1.2:

$$\sum_{v \in V} d(v) = 2 \cdot \#E \leq 2\lambda(n-1).$$

Jakamalla puolittain luvulla n saadaan

$$\frac{1}{n} \cdot \sum_{v \in V} d(v) \leq 2\lambda \cdot \frac{(n-1)}{n} < 2\lambda.$$

□

Käyttämällä arvioita $\#E \leq \frac{1}{2}\Delta \cdot \#V$ ja $\Delta \leq \#V - 1$ metsäisyyden määritelmästä saadaan, että

$$\lambda_G = \max_{H \subseteq G} \left\lceil \frac{\#E_H}{\#V_H - 1} \right\rceil \leq \max_{H \subseteq G} \left\lceil \frac{\Delta_H \cdot \#V_H}{2(\#V_H - 1)} \right\rceil \leq \max_{H \subseteq G} \left\lceil \frac{(\Delta_H + 1) \cdot (\#V_H - 1)}{2(\#V_H - 1)} \right\rceil = \left\lceil \frac{\Delta_G + 1}{2} \right\rceil.$$

Yläraja on tiukka, sillä yhtäsuuruus on voimassa esimerkiksi täydellisissä graafeissa ja sykleissä. Graafin maksimiaste voi myös olla metsäisyyttä huomattavasti suurempi. Esimerkiksi n solmun tähtigraafissa maksimiaste on jopa $n - 1$, mutta graafin metsäisyys on vain 1. Aiemmin mainittujen tasograafien maksimiaste voi niin ikään olla mielivaltaisen suuri, mutta metsäisyys on korkeintaan 3, koska tasograafissa $\#E \leq 3 \cdot \#V - 6$, kun $\#V \geq 3$. Tämä raja myös saavutetaan, sillä kuvan 1 graafi on tasograafi, jonka metsäisyys on 3.

1.2 Riippumaton joukko

Määritelmä 1.6 (Riippumaton joukko)**.** Olkoon $G = (V, E)$ graafi. Joukko $R \subseteq V$ on *riippumaton joukko*, jos kaikilla u ja $v \in R$ on voimassa $uv \notin E$. Lisäksi R on *maksimaalinen* riippumaton joukko, jos kaikille solmuille $v \in V$ on voimassa toinen ehdoista

1. $v \in R$
2. $N(v) \cap R \neq \emptyset$.

Maksimaalisessa riippumattomassa joukossa solmu on itse riippumattomassa joukossa tai jokin sen naapureista on. Toisin sanoen maksimaaliseen riippumattomaan joukkoon ei voi lisätä yhtään solmua. Tässä kontekstissa ei olla kiinnostuneita mahdollisimman suurista riippumattomista joukoista.

Muodostetaan riippumaton joukko R *ahneesti* seuraavalla tavalla. Valitaan jokin satunnainen solmu $v \in V$, lisätään se riippumattomaan joukkoon ja poistetaan graafista solmu v ja sen naapurusto. Tätä toistetaan, kunnes graafi on tyhjä. Satunnaista valintaa on helpompi käsitellä, jos kiinnitämme solmuille järjestyksen ennen algoritmin suorittamista ja sovitaan, että valitsemme jäljellä olevista solmuista sen, jolla on pienin järjestysluku.

Perustellaan lyhyesti, miksi algoritmi 1 tuottaa maksimaalisen riippumattoman joukon. Algoritmin tulostama joukko R on riippumaton, koska solmun $v \in R$ naapurit poistetaan graafista, kun solmu v lisätään joukkoon R. Algoritmin suorituksen

Algoritmi 1: Ahne riippumaton joukko

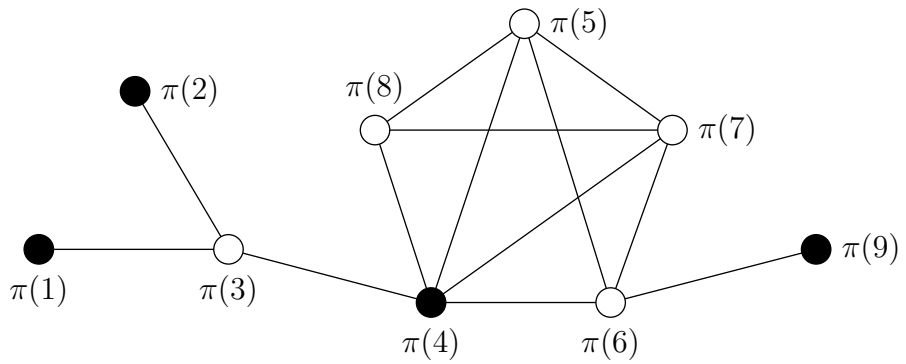
Syöte: Graafi $G = (V, E)$, missä $\#V = n$.
Solmujen satunnainen järjestys $\pi\colon [1, n] \to V$.
Tuloste: Maksimaalinen riippumaton joukko $R \subseteq V$.

$R \leftarrow \emptyset$
kunnes $V = \emptyset$ **toista**
  $v \leftarrow \arg\min_{u \in V} \pi^{-1}(u)$
  $R \leftarrow R \cup \{v\}$
  $V \leftarrow V \setminus (\{v\} \cup N(v))$
palauta R

lopuksi graafi on tyhjä. Solmu $v \in V$ voi poistua graafista kahdella tavalla. Joko se on itse tullut lisätyksi joukkoon R, jolloin maksimaalisuuden ehto 1 on voimassa, tai jokin sen naapureista $u \in N(v)$ on lisätty riippumattomaan joukkoon, ja silloin ehto 2 on voimassa.

Jos solmuja valitaan yksi kerrallaan, niin huonoiten algoritmi suoriutuu graafissa, jossa ei ole lainkaan kaaria, sillä silmukka joudutaan toistamaan n kertaa. Jos vaadimme, että syötegraafin on oltava yhtenäinen, niin silti esimerkiksi polkugraafissa silmukka täytyy toistaa vähintään $\frac{n}{3}$ kertaa, sillä kullakin solmulla on korkeintaan 2 naapuria.

Algoritmi rinnakkaistuu luonnollisella tavalla. Riippumattomaan joukkoon voidaan lisätä useita solmuja yhtä aikaa, kunhan solmut eivät ole naapureita keskenään. Tämä onnistuu siten, että valitaan riippumattomaan joukkoon kerrallaan kaikki *lokaalit minimit* järjestyksen π suhteen, siis solmut v, joilla $\pi^{-1}(v) < \min_{u \in N(v)} \pi^{-1}(u)$. Rinnakkaistettu versio tulostaa saman riippumattoman joukon kuin peräkkäinen algoritmi. Tarkalleen jokainen solmu, joka valitaan algoritmissa 2 joukkoon R', on



Kuva 3: Mustat solmut muodostavat ahneen riippumattoman joukon. Solmut $\pi(1), \pi(3), \pi(4), \pi(6)$ ja $\pi(9)$ muodostavat viiden solmun riippuvuuspolun.

Algoritmi 2: Ahne riippumaton joukko rinnakkaisesti

Syöte: Graafi $G = (V, E)$, missä $\#V = n$.
Solmujen satunnainen järjestys $\pi\colon [1, n] \to V$.
Tuloste: Maksimaalinen riippumaton joukko $R \subseteq V$.

$R \leftarrow \emptyset$
kunnes $V = \emptyset$ **toista**
  $R' \leftarrow \{v \in V : \pi^{-1}(v) < \min_{u \in N(v)} \pi^{-1}(u)\}$
  $R \leftarrow R \cup R'$
  $V \leftarrow V \setminus (R' \cup \bigcup_{v \in R'} N(v))$
palauta R

pienin graafissa jäljellä oleva solmu v jossain vaiheessa algoritmin 1 suoritusta.

Huonoimmassa tapauksessa polkugraafissa solmut ovat kasvavassa järjestyksessä polun yhdestä päästä toiseen ja kerrallaan vain yksi solmu on lokaali minimi. Tällöin algoritmin suoritus myös rinnakkaisesti vie $\mathcal{O}(n)$ kierrosta. Käy kuitenkin ilmi, että suurella todennäköisyydellä[1] edellä kuvatun kaltaiset epäonniset järjestysketjut ovat lyhyitä.

Formalisoidaan seuraavaksi *riippuvuuspolun* käsite. Olkoon $G = (V, E)$ graafi ja kiinnitetään jokin solmujen satunnainen järjestys $\pi\colon [1, n] \to V$. Olkoon $R \subseteq V$ ahne riippumaton joukko, joka saadaan suorittamalla algoritmi 2 solmujen järjestyksellä π. Merkitään $inhib(u) = \arg\min\{\pi^{-1}(v) : v \in N(u) \cap R\}$ sitä riippumattomaan joukkoon kuuluvaa solmun $u \notin R$ naapuria, jolla on pienin järjestysluku järjestyksessä π ja kutsutaan sitä solmun u *inhibiittoriksi*. Riippuvuuspolussa joka toinen solmu kuuluu lopulta ahneeseen riippumattomaan joukkoon. Solmujen järjestysluvut kasvavat polun päästä toiseen. Lisäksi solmut, jotka kuuluvat riippumattomaan joukkoon, ovat inhibiittoreita niille solmuille, jotka eivät kuulu riippumattomaan joukkoon.

Määritelmä 1.7 (Riippuvuuspolku)**.** Indeksijono $k_1, k_2, \ldots, k_{2l+1}$, missä $k_i \in [1, n]$ ja $k_i < k_j$ kaikilla $i < j \in [1, 2l + 1]$, on riippuvuuspolku, jos

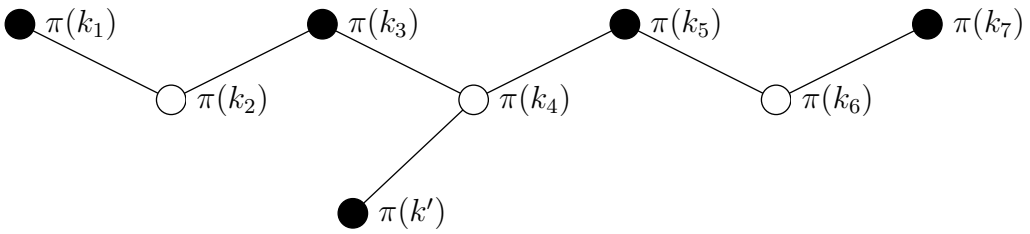
(i) $(\pi(k_1), \ldots, \pi(k_{2l+1}))$ on polku graafissa G,

(ii) $\pi(k_i) \in R$ kaikilla parittomilla i,

(iii) $\pi(k_i) \notin R$ kaikilla parillisilla i,

(iv) $\pi(k_{i-1}) = inhib(\pi(k_i))$ kaikilla parillisilla i.

[1] "Suurella todennäköisyydellä" tarkoittaa, että sopivilla vakioiden valinnoilla suotuisan tapahtuman todennäköisyys on $1 - \frac{1}{n^c}$ millä tahansa halutulla $c > 0$.



Kuva 4: Solmut $\pi(k_1), \pi(k_2), \ldots, \pi(k_7)$ muodostavat riippuvuuspolun.

Kuvassa 4 solmut $\pi(k_1), \ldots, \pi(k_6)$ ja $\pi(k_7)$ muodostavat riippuvuuspolun. Solmulla $\pi(k_4)$ on naapurit $\pi(k_3)$ ja $\pi(k')$, joista tiedetään että $k_3 < k'$ riippuvuuspolun määritelmän mukaan. Oletetaan lisäksi, että $k' < k_4$. Solmu $\pi(k')$ on lokaali minimi ensimmäisellä kierroksella. Se tulee lisätyksi riippumattomaan joukkoon ja sen naapuri $\pi(k_4)$ poistuu graafista. Solmu $\pi(k_4)$ olisi poistunut graafista viimeistään toisella kierroksella, kun sen inhibiittori $\pi(k_3)$ lisätään riippumattomaan joukkoon.

Kun algoritmia ajetaan rinnakkaisesti, niin riippuvuuspolussa seuraavan solmun tulee odottaa korkeintaan niin kauan, että edellinen on poistunut graafista. Graafin pisimmän riippuvuuspolun pituus antaa siis ylärajan tarvittavien kierrosten määrälle.

Lemma 1.8. *Olkoon $G = (V, E)$ graafi ja olkoon $\pi\colon [1, n] \to V$ solmujen satunnainen järjestys. Jos pisin riippuvuuspolku graafissa G on $2l + 1$ solmua pitkä, niin algoritmin 2 suoritus kestää korkeintaan $l + 1$ kierrosta.*

Todistus. Tarkastellaan algoritmin 2 sellaista versiota, jossa solmu $u \notin R$ poistuu graafista vasta, kun $\mathit{inhib}(u) \in R$ tulee käsittelyyn. Tavallisen rinnakkaistetun algoritmin suoritus kestää korkeintaan yhtä kauan kuin tämän version.

Osoitetaan induktiolla, että kierroksella i joukkoon R lisättävä solmu on viimeinen solmu riippuvuuspolussa, jonka pituus on $2(i - 1) + 1$. Kierroksella 1 lisätyn solmun täytyy olla riippuvuuspolun ensimmäinen solmu. Jos solmu v tulee lisätyksi riippumattomaan joukkoon kierroksella $i + 1$, niin kierroksella i sillä oli vielä naapuri u, jolla $\pi^{-1}(u) < \pi^{-1}(v)$. Solmu u poistui graafista kierroksella i, koska solmu $\mathit{inhib}(u)$ lisättiin riippumattomaan joukkoon. Induktio-oletuksen mukaan solmu $\mathit{inhib}(u)$ on $2(i-1)+1$ solmun pituisen riippuvuuspolun viimeinen solmu. Kun sitä polkua jatketaan solmuilla u ja v, niin saadaan $2i+1$ solmua pitkä riippuvuuspolku, jonka viimeinen solmu on solmu v.

Todetaan, että graafin jokainen solmu v tai sen inhibiittori kuuluu johonkin riippuvuuspolkuun. Jos graafin pisin riippuvuuspolku on $2l + 1$ solmua pitkä, niin sen viimeinen solmu poistuu graafista viimeistään kierroksella $l + 1$ ja silloin graafi on tyhjä. □

Esitetään seuraava lause ilman todistusta. Todistus on kokonaisuudessaan artikkelissa Tight Analysis of Parallel Randomized Greedy MIS ([6]).

Lause 1.9. *Olkoon $G = (V, E)$ graafi, jolla $\#V = n$ ja olkoon $\pi\colon [1, n] \to V$ solmujen satunnainen järjestys. Graafin G pisin riippuvuuspolku järjestyksen π suhteen on pituudeltaan luokkaa $\mathcal{O}(\log n)$ suurella todennäköisyydellä.*

Lemman 1.8 kanssa edellisestä lauseesta seuraa, että algoritmi 2 päättyy $\mathcal{O}(\log n)$ kierroksen jälkeen suurella todennäköisyydellä.

Lause 1.9 on merkittävä tulos. Aiemmin oli osoitettu, että satunnaisissa Erdős–Rényi-graafeissa keskimääräinen suoritusaika on $\mathcal{O}(\log n)$ ([7]) ja että yleisissä graafeissa riippuvuuspolun pituus on $\mathcal{O}(\log^2 n)$ suurella todennäköisyydellä ([8]). Lisäksi artikkelissa [6] osoitetaan, että algoritmin 2 suoritusaika $\mathcal{O}(\log n)$ on tiukka. On nimittäin olemassa graafi, jonka pisin riippuvuuspolku on $\Omega(\log n)$ solmua pitkä suurella todennäköisyydellä.

2 Hajautettu laskenta

On olemassa erilaisia hajautetun laskennan malleja eri abstraktiotasoilla. Tässä luvussa esitellään kaksi hyvin erilaista mallia. Yhteistä niille on, että tarkastelu ei riipu tietokoneiden arkkitehtuurista, käyttöjärjestelmästä tai ohjelmointikielestä. Silloin päästään käsiksi ongelmien luonteeseen, siis siihen, minkälaiset ongelmat ovat ylipäätään tehokkaasti ratkaistavissa hajautetussa mallissa.

Tässä yhteydessä ei välitetä käytännön ongelmista, kuten koneiden keskinäisestä synkronoinnista tai siitä miten toimitaan silloin, jos yhdessä koneessa tapahtuu jokin virhe tai jokin koneiden välinen yhteys katkeaa. Oletetaan meillä olevan paitsi ideaalikoneita, myös ideaaliverkko.

2.1 LOCAL-malli

LOCAL-mallissa nimensä mukaisesti ratkaistaan graafiongelmia paikallisesti. Malli on saanut alkunsa Nathan Linialin vuoden 1992 artikkelista [9].

LOCAL-mallissa syötegraafin jokainen solmu on itsenäinen laskukone. Solmuilla on suuruusluokkaa $\mathcal{O}(\log n)$ olevat yksilölliset tunnisteet. Laskenta etenee synkronoiduissa kierroksissa. Solmut voivat joka kierroksella tehdä ensin rajattomasti laskelmia, sitten vaihtaa viestejä *naapuriensa* kanssa ja lähettää *rajattomasti* tietoa niille. Algoritmin suorituskykyä mitataan kierrosten määrässä.

LOCAL-mallissa solmu voi kerätä k kierroksessa tiedon korkeintaan sen k-naapurustosta. Yhden solmun laskentaa ja solmujen välittämien viestien kokoa ei ole rajoitettu, joten kaikki ongelmat voidaan triviaalisti ratkaista $diam(G)$ kierroksessa. Silloin jokainen solmu tuntee koko graafin ja voi suorittaa algoritmin paikallisesti. Jotkin ongelmat luonteeltaan globaaleja, toisin sanoen yhtään pienempi määrä kierroksia ei riitä ongelman ratkaisemiseen. Esimerkki tällaisesta ongelmasta on polkugraafin solmujen värittäminen kahdella värillä niin, että vierekkäiset solmut eivät ole saman värisiä.

Edellisen luvun ahne algoritmi riippumattomalle joukolle voidaan toteuttaa LOCAL-mallissa helposti. Solmut keräävät naapuriensa järjestysluvut ensimmäisellä kierroksella ja tietävät sen perusteella, ovatko ne itse lokaaleja minimejä. Lokaalit minimit merkitsevät itsensä riippumattomaan joukkoon kuuluviksi ja lähettävät toisella kierroksella tiedon tilastaan naapureilleen, jotka taas merkitsevät itsensä poistetuksi graafista. Näitä kahta kierrosta toistetaan, kunnes kaikki solmut tietävät lopullisen tilansa. Kahdessa LOCAL-kierroksessa voidaan siis suorittaa yksi kierros algoritmia 2. Lemman 1.8 ja lauseen 1.9 perusteella LOCAL-algoritmin suorituksessa kestää $\mathcal{O}(\log n)$ kierrosta suurella todennäköisyydellä. Siinä ajassa solmut ovat

ehtineet kerätä $\mathcal{O}(\log n)$-naapurustonsa, mistä seuraa että solmun lopullisen tilan voi määrittää sen $\mathcal{O}(\log n)$-naapuruston perusteella suurella todennäköisyydellä.

2.2 MPC-malli

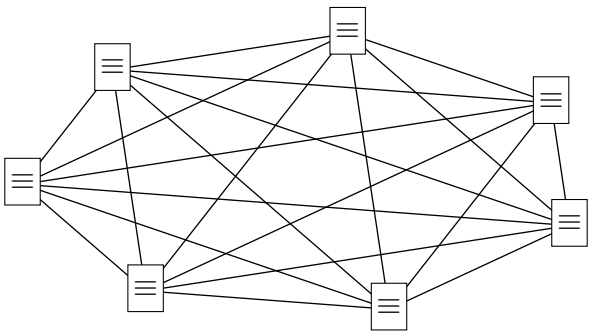
MPC on lyhenne sanoista *massively parallel computation*. Se on abstraktio eräistä laajalti käytössä olevista hajautetun laskennan työkaluista, kuten Googlen MapReducesta ja sitä vastaavasta avoimen lähdekoodin Hadoopista. Mallin katsotaan kehittyneen 2010-luvun aikana artikkeleiden [10, 11, 12, 13, 14] ja [15] myötä.

Kuvitellaan koneita, jotka voivat yhdessä kierroksessa tehdä mielivaltaisia laskelmia sillä datalla, mitä niiden muistissa on, sitten kommunikoida *kaikkien* muiden koneiden kanssa ja lähettää ja vastaanottaa *rajallisesti* tietoa. Algoritmin suorituskykyä mitataan tarvittavien kierrosten määrässä kuten LOCAL-mallissakin. Yleensä havitellaan korkeintaan polylogaritmista suoritusaikaa eli $\mathcal{O}(\log^k n)$ jollain $k \in \mathbb{N}$. MPC-mallissa ratkotaan usein graafiongelmia.

Määritelmä 2.1 (Sublineaarinen MPC-malli)**.** Olkoon G syötegraafi, jossa on n solmua ja N kaarta. Olkoot koneiden määrä M ja konekohtainen muistin määrä $S \in \mathcal{O}(n^\delta)$, missä $0 < \delta < 1$. Koneita on tarpeeksi, jotta koko graafi mahtuu muistiin, siis $M \cdot S \geq N$. Yhdellä kierroksella koneen lähetettämien ja vastaanottamien viestien koko saa olla korkeintaan $\mathcal{O}(S)$.

Lisäksi tässä tutkielmassa oletetaan, että yhden solmun koko naapurusto mahtuu yhdelle koneelle, toisin sanoen $\Delta_G \leq S$.

MPC-mallissa on olemassa erilaisia muistimalleja: $S \in \Omega(n^{1+\delta})$ jollain $\delta > 0$ on superlineaarinen, $S \in \Theta(n)$ lineaarinen ja $S \in \mathcal{O}(n^\delta)$, missä $0 < \delta < 1$, sublineaarinen. Muistin suuruusluokka vaikuttaa oleellisesti algoritmin suunnitteluun. Luonnollisesti nopeiden algoritmien suunnittelu on haastavinta silloin, kun muistia on käytössä vähän suhteessa syötteen kokoon. Superlineaarisessa ja lineaarisessa



Kuva 5: MPC-mallissa kaikki koneet ovat yhteydessä toisiinsa ja syötegraafi on jaettu mielivaltaisesti niiden kesken.

muistimallissa voidaan esimerkiksi käyttää filtering-tekniikkaa ([16]), jossa graafin kaaria karsitaan, kunnes se mahtuu yhdelle koneelle ja ongelma ratkaistaan paikallisesti. Jos graafi on valmiiksi harva, niin ratkaisu on epätyydyttävän triviaali. Lisäksi tietyn muistimallin sisällä algoritmin suoritusaika on sitä pidempi, mitä pienempi muistieksponentti δ on. Suoritusajan analyysissä ei merkitä sitä eksplisiittisesti.

Tässä tutkielmassa tutkitaan algoritmeja ainoastaan sublineaarisessa muistimallissa. Tätä valintaa motivoidaan sillä, että silloin ei tarvitse olettaa, että koneiden muisti skaalautuu lineaarisesti syötteen kokoon nähden. Sublineaarista muistia voi luonnehtia siten, että yksi kone ei voi nähdä vakiokokoista murto-osaa koko graafista.

Yleensä halutaan, että myös globaali muisti on pieni, siis $M \cdot S \in \mathcal{O}(N)$. Luvussa 4 algoritmin 6 yksinkertaistetussa toteutuksessa tarvitaan kuitenkin n konetta, missä n on syötegraafin solmujen määrä. Silloin globaalin muistin koko on $\mathcal{O}(n^{1+\delta})$.

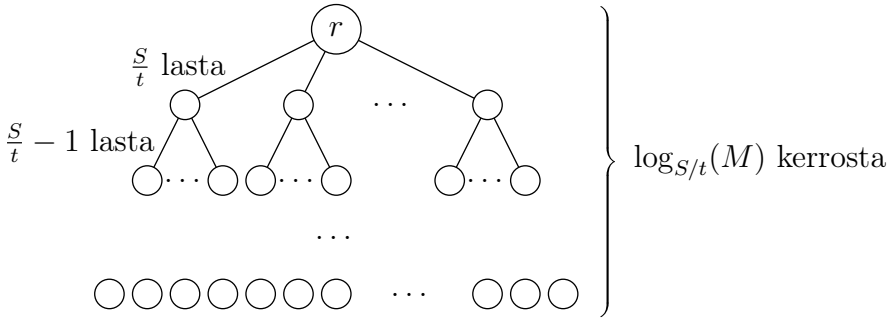
Seuraavaksi esitellään muutamia MPC-mallin tekniikoita, joiden avulla voidaan työskennellä tehokkaasti muistirajoitusten puitteissa.

2.2.1 Lähetyspuu

Usein halutaan lähettää viestejä koneilta toisille, mutta yhden koneen muistin koko S rajoittaa yhdessä kierroksessa lähetettävien viestien määrää. Tähän voidaan käyttään koneiden virtuaalista *lähetyspuuta*. Sen on ensimmäisenä esitelty artikkelissa [12].

Olkoon t viestin maksimikoko. Koneet järjestäytyvät puun muotoisesti virtuaaliseen viestintägraafiin. Lähetyspuussa yksi koneista on juurisolmu ja sillä on $\frac{S}{t}$ lapsisolmua. Kaikilla muilla koneilla on $\frac{S}{t} - 1$ lapsisolmua, lukuunottamatta lehtisolmuja. Koneet kommunikoivat ainoastaan koneille, jotka ovat niiden naapureita puussa. Samalla voidaan laskea jotain kommutatiivisia ja assosiatiivisia funktioita, kuten summa, maksimi tai minimi niistä luvuista joita koneilla oli.

Esimerkiksi graafin kaarien määrä voidaan laskea ja saattaa kaikkien koneiden tietoon $\mathcal{O}(1)$ kierroksessa. Kaarien määrä on luokkaa $\mathcal{O}(n^2)$ ja niiden lukumäärän esitys binäärilukuna vie $\mathcal{O}(\log(n^2)) = \mathcal{O}(\log n)$ bittiä. Silloin juurikoneella on $\mathcal{O}\left(\frac{n^\delta}{\log n}\right)$ lasta ja lähetyspuuhun tulee $\mathcal{O}(\frac{1}{\delta})$ kerrosta, mitä voidaan pitää vakiona, kun δ on kiinnitetty. Jokainen kone alhaalta käsin kerää kaarien määrät lapsiltaan, laskee ne yhteen ja lähettää summan ylöspäin. Juurisolmu lähettää summan kaikille koneille puussa alaspäin.

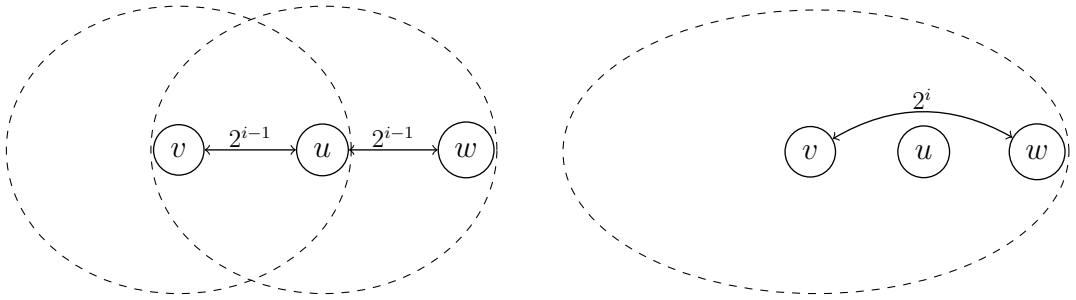


Kuva 6: Lähetyspuun rakenne.

2.2.2 Graafin eksponentiaatio ja kierrosten tiivistäminen

Kierrosten tiivistäminen on tekniikka, jossa k kierrosta LOCAL-algoritmia simuloidaan yhdessä MPC-kierroksessa, kun tunnetaan solmujen k-naapurustot ([17]).

Hyödyntämällä MPC-mallin vapaata kommunikaatiota voidaan kerätä naapurustoja nopeammin kuin LOCAL-mallissa, jossa solmut voivat viestiä vain naapureilleen ja kerätä k kierroksessa k-naapurustonsa. Tätä kutsutaan graafin eksponentiaatioksi ja idea on esitelty artikkelissa [18]. Näytetään induktiolla, että solmu voi kerätä tiedon k-naapurustonsa rakenteesta $\log_2(k) + 1$ MPC-kierroksessa. Ensimmäisen kierroksen jälkeen kukin solmu tuntee oman naapurustonsa. Oletetaan, että i kierroksen jälkeen jokainen solmu tuntee oman 2^{i-1}-naapurustonsa. Kierroksella $i+1$ solmu jakaa tiedon 2^{i-1}-naapurustostaan kaikkien niiden solmujen kanssa, joiden etäisyys solmusta on 2^{i-1}. Nyt solmu tuntee kaikki solmut itsestään etäisyydellä 2^i. Naapuruston keräämisen yhtä askelta on havainnollistettu kuvassa 7.



(a) Ennen kierrosta $i+1$ solmut v ja u tuntevat 2^{i-1}-naapurustonsa, jotka on merkitty katkoviivoin.

(b) Kierroksella $i+1$ solmu u jakaa tiedon 2^{i-1}-naapurustostaan solmulle v, joka tuntee kierroksen jälkeen 2^i-naapurustonsa.

Kuva 7: Graafin eksponentiaatioaskel.

Halutaan suorittaa k kierrosta tarvitseva LOCAL-algoritmi. Koneille kuitenkin mahtuu vain l-naapurustot, missä $l < k$. Voimme typistää tarvittavien kierrosten määrää seuraavalla tavalla:

1. Kerätään solmujen l-naapurustot $\log_2(l) + 1$ MPC-kierroksessa.
2. Simuloidaan LOCAL-algoritmin l ensimmäistä kierrosta yhdessä MPC-kierroksessa.
3. Kaikki solmut päivittävät l-naapurustolleen tiedon omasta tilastaan.
4. Toistetaan vaiheita 2 ja 3 k/l kertaa.

Näin tieto laskennan tilasta kulkeutuu nopeammin graafin laidalta toiselle ja graafin läpimitta on käytännössä kutistunut. MPC-mallin muistirajoituksen vuoksi täytyy varmistaa, että solmujen tilat laskennan aikana ovat pieniä. Esimerkiksi riippumattoman joukon laskemisessa solmun tila on jokin kolmesta: ”kuuluu riippumattomaan joukkoon”, ”poistunut graafista” ja ”lopullinen tila ei vielä tiedossa”.

3 Korrelaatioklusterointi

Kuvitellaan, että halutaan järjestää juhlat ja suunnitella juhlien istumajärjestys. On olemassa lista juhlavieraista ja tieto siitä, ketkä heistä tuntevat toisensa. Oletetaan, että pöytien kokoa tai määrää ei ole ennalta päätetty. Vieraat haluavat istua samassa pöydässä tuttujensa kanssa ja toisaalta eivät halua tuntemattomia ihmisiä pöytäänsä. Onnistuneiden juhlien takaamiseksi halutaan, että vieraat olisivat mahdollisimman tyytyväisiä pöytäseurueeseensa. Pyritään siis suunnittelemaan istumajärjestys niin, että kussakin pöydässä istuisi keskenään mahdollisimman tuttua väkeä ja että toisensa tuntevat vieraat eivät joutuisi istumaan eri pöydissä.

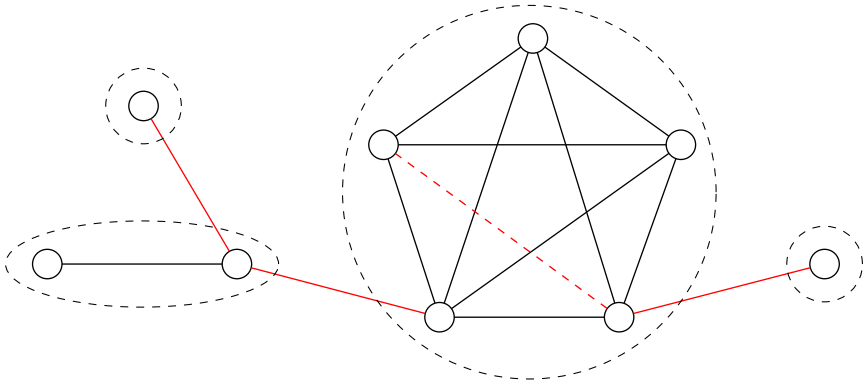
Määritellään ongelma formaalisti. Ongelma on esitelty ensimmäisen kerran artikkelissa [19]. Olkoon $G = (V, E^+ \cup E^-)$ täydellinen graafi, jonka kaaret on merkitty positiivisiksi (E^+) ja negatiivisiksi (E^-). Merkitään solmun v positiivista naapurustoa $N^+(v) = \{u \in V : uv \in E^+\}$ ja negatiivista naapurustoa vastaavasti $N^-(v) = \{u \in V : uv \in E^-\}$. Ajatellaan, että solmut u ja v ovat keskenään samanlaisia, jos $uv \in E^+$, ja erilaisia, jos $uv \in E^-$. Merkinnällä Δ tarkoitetaan graafin positiivista maksimiastetta.

Klusterointi $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ on solmujen joukon V partitio parittain erillisiin, epätyhjiin joukkoihin C_i. Merkitään, että $C(v) = C_i$, kun $v \in C_i$. Sanotaan, että positiivinen kaari $uv \in E^+$ on klusteroitu oikein, jos $C(u) = C(v)$, ja negatiivinen kaari $uv \in E^-$ on vastaavasti klusteroitu oikein, jos $C(u) \neq C(v)$. Muussa tapauksessa kaari on klusteroitu väärin. Klusteroinnissa on siis virhe, jos kaksi erilaista solmua ovat samassa klusterissa tai kaksi samanlaista solmua ovat eri klustereissa. Merkitään klusteroinnin $\mathcal{C}$ virheiden määrää

$$\begin{aligned} cost(\mathcal{C}) = \quad & \#\{uv \in E^- : C(u) = C(v)\} \\ & +\#\{uv \in E^+ : C(u) \neq C(v)\}. \end{aligned}$$

Näistä ensimmäisiä kutsutaan *negatiivisiksi virheiksi* ja jälkimmäisiä *positiivisiksi virheiksi*.

Graafin solmut halutaan ryhmitellä keskenään samanlaisten solmujen klustereihin mahdollisimman hyvin. Tähän on kaksi lähestymistapaa: yhtäältä voidaan maksimoida oikein klusteroitujen kaarien määrää tai toisaalta voidaan minimoida väärin klusteroitujen kaarien määrää. Optimaaliset klusteroinnit ovat samat ongelman molemmissa muotoiluissa, mutta ratkaisun approksimointi on kuitenkin luonteeltaan erilaista. Maksimointiongelmaan on esimerkiksi triviaali 2-approksimaatio. Graafin $G = (V, E)$ klusteroinnissa on enimmillään $\binom{\#V}{2}$ oikein klusteroitua kaarta. Jos $\#E^+ \geq \#E^-$, niin muodostetaan triviaali klusterointi, jossa kaikki solmut ovat yh-



Kuva 8: Kuva klusteroinnista, jossa väärin klusteroidut positiiviset kaaret on merkitty punaisella viivalla ja negatiiviset kaaret vastaavasti punaisella katkoviivalla. Oikein klusteroidut negatiiviset kaaret on jätetty piirtämättä. Klusterien rajat on merkitty katkoviivoilla. Noudatetaan tätä merkintätapaa jatkossa.

dessä klusterissa. Silloin oikein klusteroituja kaaria on $\#E^+$. Jos taas $\#E^+ < \#E^-$, niin laitetaan jokainen solmu omaan yhden solmun klusteriin. Silloin kaikki negatiiviset kaaret on klusteroitu oikein. Molemmissa tapauksissa oikein klusteroituja kaaria on vähintään $\binom{\#V}{2}/2$.

Optimaalisen klusteroinnin löytäminen on NP-täydellinen ongelma, eli ongelmalle ei ole olemassa polynomiaikaista algoritmia, mikäli osoittautuu, että $P \neq NP$. Voidaan nimittäin osoittaa ([19]), että partitio kolmioihin -ongelma[2] voidaan redusoida optimaalisen korrelaatioklusteroinnin löytämiseen tietyssä graafissa.

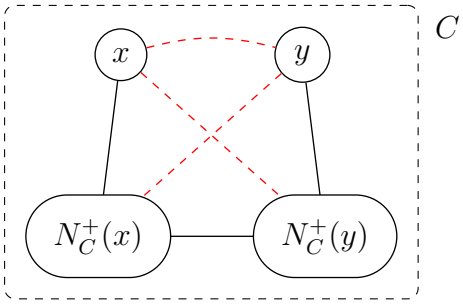
Huomautettakoon, että korrelaatioklusterointi eroaa eräistä muista klusterointiongelmista (esimerkiksi k-means-klusteroinnista) siten, että klustereiden määrää ei ole ennalta päätetty. Lisäksi klusteroitavien objektien ei tarvitse olla peräisin vektoriavaruudesta tai edes metrisestä avaruudesta. Riittää tietää objektien parittaiset suhteet.

Osoitetaan seuraavaksi, että optimaalisen klusteroinnin klusterit eivät ole läpimitaltaan leveitä.

Lemma 3.1. *Olkoon $\mathcal{C}$ optimaalinen klusterointi. Klusterin $C \in \mathcal{C}$ positiivisten kaarten indusoiman graafin läpimitta on korkeintaan 2.*

Todistus. Olkoon $C \in \mathcal{C}$ klusteri, jossa on $k+2 \geq 4$ solmua. Tehdään vastaoletus, että on olemassa solmut x ja $y \in C$, joiden etäisyys on vähintään 3. Solmuilla x ja y ei voi olla yhteisiä naapureita ja ne eivät ole toistensa naapureita, joten niiden yhteenlaskettu positiivinen aste $d_C^+(x)+d_C^+(y)$ klusterissa C on korkeintaan k. Täten niiden yhteenlaskettu negatiivinen aste on vähintään $d_C^+(x)+d_C^+(y) \geq 2(k+1)-k =$

[2]Olkoon $G = (V, E)$ graafi, jossa on $3k$ solmua. Päätösongelmassa kysytään, voidaanko graafin G solmut partitioida kolmen solmun osajoukkoihin, joiden indusoidut aligraafit ovat kolmioita K_3.

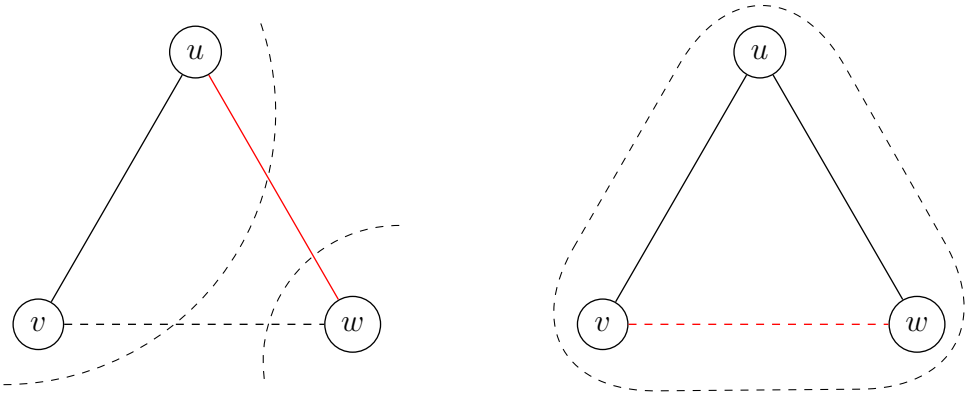


Kuva 9: Solmuilla x ja $y \in C$ ei ole yhteisiä naapureita.

$k + 2$. Toisella solmuista on siis oltava suurempi negatiivinen aste kuin positiivinen aste ja poistamalla se klusterista C omaksi klusterikseen virheiden määrä vähenee aidosti. Optimaalisessa klusteroinnissa ei siis ole yhtään klusteria, jonka läpimitta on suurempi kuin 2. □

Selkeä seuraus lemmasta on, että klusterin positiivisten kaarten indusoima aligraafi on aina yhtenäinen.

Graafille $G = (V, E^+ \cup E^-)$ on olemassa klusterointi $OPT(G)$, jolla $cost(OPT(G)) = 0$, silloin ja vain silloin kun graafin (V, E^+) yhtenäiset komponentit ovat täydellisiä graafeja. Muussa tapauksessa graafissa on vähintään yksi *huono kolmio*.



(a) Huono kolmio, jossa solmu w on eri klusterissa kuin solmut u ja v. Silloin kaari $uw \in E^+$ aiheuttaa positiivisen virheen.

(b) Huono kolmio, jossa kaikki solmut u, v ja w ovat samassa klusterissa. Silloin kaari $vw \in E^-$ aiheuttaa negatiivisen virheen.

Kuva 10: Huonoja kolmioita.

Määritelmä 3.2 (Huono kolmio)**.** Olkoon $G = (V, E^+ \cup E^-)$ täydellinen merkitty graafi. Kolmen solmun joukko $\{u, v, w\} \subseteq V$ on huono kolmio, jos uv ja $uw \in E^+$ mutta $vw \notin E^+$.

Jokin huonon kolmion kaarista aiheuttaa virheen missä tahansa klusteroinnissa. Kuvassa 10 havainnollistetaan eri tapoja klusteroida huonon kolmion solmut. Kaariltaan erillisten huonojen kolmioiden määrästä saadaan alaraja optimaalisen klusteroinnin virheiden määrälle. Huonoja kolmioita käytetään myös seuraavassa alaluvussa esitettävän algoritmin approksimaation analysointiin.

3.1 `Pivot`

Eräs yksinkertainen algoritmi korrelaatioklusteroinnille täydellisessä merkityssä graafissa esitellään artikkelissa [20].

Kuvitteellisten juhliemme istumajärjestys laaditaan seuraavalla tavalla. Valitaan satunnainen istumapaikkaa vailla oleva juhlavieras. Hän istuu pöytään, johon tulevat myös kaikki hänen tuttavansa, joilla ei vielä ole istumapaikkaa. Toistetaan tätä lopuille vieraille, kunnes kaikki istuvat pöydissä.

Algoritmi 3: `Pivot`

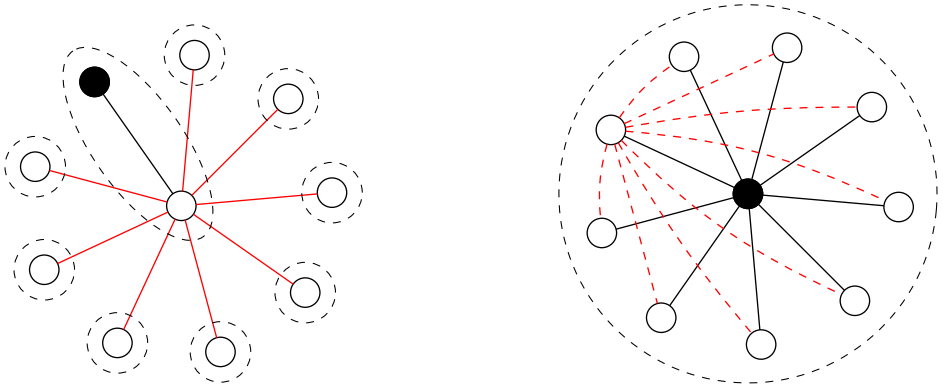
Syöte: Graafi $G = (V, E^+ \cup E^-)$, missä $\#V = n$.
Solmujen satunnainen järjestys $\pi\colon [1, n] \to V$.
Tuloste: Klusterointi $\mathcal{C}$.

$\mathcal{C} \leftarrow \emptyset$
kunnes $V = \emptyset$ **toista**
  $v \leftarrow \arg\min_{u \in V} \pi^{-1}(u)$
  $C \leftarrow \{v\} \cup N^+_{G[V]}(v)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$
  $V \leftarrow V \setminus C$
palauta $\mathcal{C}$

Klusterin muodostavia solmuja v kutsutaan *pivot-solmuiksi*. Pivoteiksi valitut solmut muodostavat riippumattoman joukon. Itse asiassa pivot-solmujen joukko on täsmälleen sama kuin ahne riippumaton joukko, jonka algoritmi 1 tuottaa, kun käytetään samaa solmujen järjestystä π.

Kaikessa yksinkertaisuudessaan algoritmi vaikuttaa lupaavalta, koska ainakin kaikissa klustereissa läpimitta on korkeintaan 2 ja lisäksi ei tehdä positiivisia virheitä pivot-solmun positiivisista naapureista eikä negatiivisia virheitä sen negatiivisista naapureista. Lisäksi algoritmi ei klusteroi huonoihin kolmioihin kuulumattomia kaaria väärin.

Esimerkki 3.3. Kuvan 8 klusterointi saadaan, kun pivot-solmut valitaan kuvassa 3 esitetyn järjestyksen mukaan.



(a) Tähtigraafin eräs optimaalinen klusterointi, jossa tehdään $n-2$ virhettä. Yksi lehtisolmuista on valittu ensimmäiseksi pivot-solmuksi.

(b) Huonoin mahdollinen klusterointi. Keskimmäinen solmu on valittu pivot-solmuksi. Kaikkia virheitä aiheuttavia negatiivisia kaaria ei ole piirretty.

Kuva 11: Tähtigraafin klusterointi `Pivot`-algoritmilla.

Algoritmi ei ole deterministinen. Joillakin pivot-solmujen valinnalla algoritmi voi tuottaa erittäin huonon klusteroinnin, mutta se suoriutuu silti *odotusarvoisesti* hyvin.

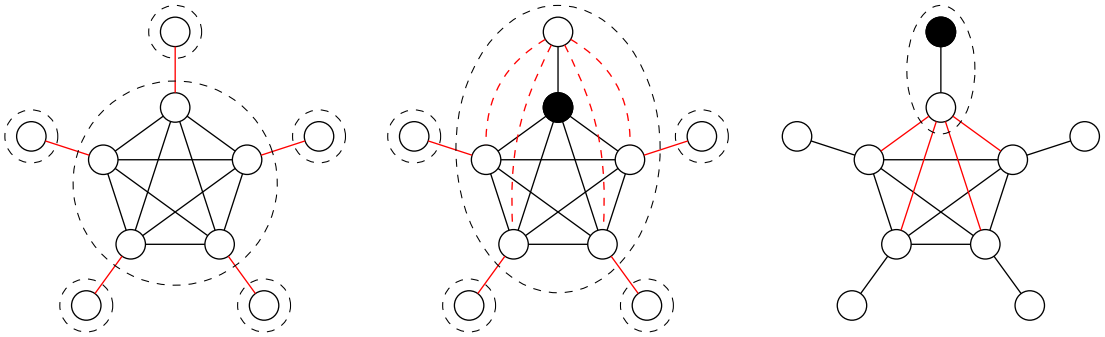
Esimerkki 3.4. Tarkastellaan esimerkkinä n solmun tähtigraafia. Kuvassa 11 oikealla on huonoin mahdollinen tapaus, jossa graafin keskimmäinen solmu on valittu pivotiksi. Tällöin negatiivisia virheitä tulee $\binom{n-1}{2} \in \mathcal{O}(n^2)$. Suurimmassa osassa suorituksista (todennäköisyydellä $1 - \frac{1}{n}$) algoritmi valitsee ensimmäiseksi pivotiksi lehtisolmun ja tuottaa erään optimaalisen klusteroinnin, jossa tehdään vain $n-2$ positiivista virhettä. Odotusarvo virheiden määrälle on siis

$$\mathbb{E}\left[cost(\mathcal{A}(G))\right] = \frac{1}{n} \cdot \binom{n-1}{2} + \frac{n-1}{n} \cdot (n-2) = \frac{3(n-1)(n-2)}{2n} \approx \frac{3}{2} \cdot n.$$

`Pivot`-algoritmin approksimaatiokerroin tähtigraafissa on siis odotusarvoisesti $\frac{3n/2}{n-2} \approx \frac{3}{2}$.

Tähtigraafissa, ja itse asiassa kaikissa puugraafeissa, on mahdollista valita pivot-solmut sellaisella tavalla, että lopputuloksena saadaan optimaalinen klusterointi. Algoritmi siis voidaan suorittaa useita kertoja ja ottaa paras lopputulos. Voitaisiin toivoa, että kaikissa graafeissa jollakin pivot-solmujen järjestyksellä olisi mahdollista tuottaa jokin optimaalinen klusterointi. Näytetään seuraavaksi, että on olemassa eräs graafien perhe, jossa `Pivot`-algoritmilla jäädään aina vakiokertoimen päähän optimaalisesta klusteroinnista.

Esimerkki 3.5. Määritellään graafi $G_n = (V_n, E_n)$ seuraavasti. Olkoon $V_n = \{v_1, \ldots, v_n, u_1, \ldots, u_n\}$ ja $E_n = \{v_i v_j : i < j \in [1, n]\} \cup \{u_i v_i : i \in [1, n]\}$. Kut-



(a) Graafin G_5 optimaalinen klusterointi, jossa tehdään 5 virhettä.

(b) Algoritmin 3 antama klusterointi, jossa tehdään 8 virhettä. Musta solmu valittiin ensimmäiseksi pivotiksi.

(c) Musta solmu on valittu ensimmäiseksi pivotiksi. Jäljelle jäävä graafi on G_4.

Kuva 12

sutaan solmuja v_i sisäsolmuiksi ja solmuja u_i reunasolmuiksi. Toisin sanoen graafissa G_n sisäsolmut muodostavat täydellisen graafin ja kustakin sisäsolmusta v_i lähtee yksi kaari sitä vastaavaan reunasolmuun u_i.

Olkoon $\mathcal{C} = \{\{v_1, \ldots, v_n\}, \{u_1\}, \ldots, \{u_n\}\}$ graafin G_n klusterointi, jossa sisäsolmut muodostavat keskenään yhden klusterin ja reunasolmut muodostavat yhden solmun klustereita. Klusterointi $\mathcal{C}$ on esitetty kuvassa 12(a). Perustellaan lyhyesti, että $\mathcal{C}$ on optimaalinen klusterointi. Kolmikot $\{u_1, v_1, v_n\}$ ja $\{u_i, v_i, v_{i-1}\}$, missä $i \in [2, n]$, ovat kaariltaan erillisiä huonoja kolmioita ja niitä on n kappaletta. Siis missä tahansa klusteroinnissa tehdään vähintään n virhettä. Toisaalta klusteroinnissa $\mathcal{C}$ tehdään n virhettä, yksi kustakin kaaresta v_iu_i. Tästä seuraa, että $cost(\mathcal{C}) = n = cost(OPT(G_n))$.

Käytetään nyt `Pivot`-algoritmia graafiin G_n. Merkitään algoritmin antamaa klusterointia $\mathcal{A}(G_n)$. Graafin symmetrisyyden ansiosta pivot-solmun valintaa voidaan tarkastella kahtena eri tapauksena. Jos ensimmäiseksi pivotiksi valitaan jokin sisäsolmuista v_i, niin se muodostaa naapureistaan klusterin $\{v_1, \ldots, v_n, u_i\}$. Tämä tapaus on esitetty kuvassa 12(b). Jäljelle jää vain triviaali graafi, joka muodostuu solmuista u_j, missä $j \neq i$. Kukin jäljelle jääneistä reunasolmuista tulee vuorollaan valituksi pivotiksi ja muodostaa yksin yhden solmun klusterin. Tällä tavoin muodostetussa klusteroinnissa tehdään $n-1$ positiivista virhettä kaarista v_ju_j kaikilla $j \neq i$ ja samoin $n-1$ negatiivista virhettä jokaisesta kaaresta u_iv_j, missä $j \neq i$. Yhteensä virheitä tulee $2n-2$, mikä on lähes kaksinkertainen määrä optimiin verrattuna. Pian nähdään, että tämä on paras mahdollinen klusterointi, jonka `Pivot`-algoritmi voi antaa graafin G_n solmuille.

Pivotiksi voidaan valita myös yksi reunasolmuista. Tämä tapaus on esitetty kuvassa 12(c). Reunasolmu u_i muodostaa klusterin ainoan naapurinsa v_i kanssa. Täl-

löin aiheutuu $n-1$ positiivista virhettä kaarista v_iv_j, missä $j \neq i$. Solmujen u_i ja v_i poistamisen jälkeen graafi $G_n[V \setminus \{u_i, v_i\}]$ on isomorfinen graafin G_{n-1} kanssa. Tiedetään, että $cost(OPT(G_{n-1})) = n-1$, joten jäljelle jääneen graafin klusteroinnista tulee vähintään $n-1$ virhettä riippumatta seuraavan pivot-solmun valinnasta. Tässäkin tapauksessa tehdään yhteensä vähintään $2n-2$ virhettä. Tarkemmin laskemalla huomataan, että reunasolmun valinnan jälkeen tehdään lopulta vähintään $3n-5$ ja enintään $\binom{n}{2}$ virhettä.

Virheiden määrä palautuu pienempään tapaukseen, ja odotusarvo voidaan muotoilla rekursioyhtälöksi (2). Solmuja valitaan pivoteiksi yhtä suurella todennäköisyydellä, joten molemmat edellä esitetyt tapaukset ovat yhtä todennäköisiä. `Pivot`-algoritmi tekee siis odotusarvoisesti

$$\begin{aligned}\mathbb{E}\left[cost(\mathcal{A}(G_n))\right] &= \frac{1}{2}\left(2n-2\right) + \frac{1}{2}\left(n-1+\mathbb{E}\left[cost(\mathcal{A}(G_{n-1}))\right]\right) \\ &= \frac{3}{2}\left(n-1\right) + \frac{1}{2}\,\mathbb{E}\left[cost(\mathcal{A}(G_{n-1}))\right] \qquad (2)\end{aligned}$$

virhettä. Rekursion perustapauksena on kahden solmun graafi G_1, jossa $\mathbb{E}\left[cost(\mathcal{A}(G_1))\right] = 0$. Kun rekursioyhtälö lasketaan auki, saadaan että

$$\mathbb{E}\left[cost(\mathcal{A}(G_n))\right] = \; 3\sum_{i=1}^{n}\frac{n-i}{2^i}.$$

Vertaamalla summaa optimaaliseen virheiden määrään $cost(OPT(G_n)) = n$ saadaan approksimaatiokerroin

$$\begin{aligned}\frac{\mathbb{E}\left[cost(\mathcal{A}(G_n))\right]}{n} &= 3\left(\sum_{i=1}^{n}\frac{1}{2^i} - \frac{1}{n}\sum_{i=1}^{n}\frac{i}{2^i}\right) \\ &\xrightarrow[n\to\infty]{} 3\sum_{i=1}^{\infty}\frac{1}{2^i} \\ &= 3.\end{aligned}$$

Joten kun n kasvaa, niin `Pivot`-algoritmin odotusarvoinen approksimaatiokerroin graafissa G_n lähestyy lukua 3.

Esimerkistä nähdään, että yleisesti graafeissa ei välttämättä löydetä optimiklusterointia edes ajamalla `Pivot`-algoritmia kaikilla mahdollisilla pivot-solmujen valinnoilla.

Käy ilmi, että edellisen esimerkin odotusarvoinen approksimaatiokerroin 3 on huonoin mahdollinen. Osoitetaan nimittäin seuraavaksi, että kaikissa graafeissa

`Pivot`-algoritmin approksimaatiokerroin on odotusarvoisesti *korkeintaan* 3.

3.1.1 Approksimaation analyysi

`Pivot` on osoitettu 3-approksimaatioksi alkuperäisessä artikkelissa [20]. Todistuksessa oleellinen havainto on seuraava: huono kolmio aiheuttaa virhettä missä tahansa klusteroinnissa, mutta toisaalta kukin kaari voi aiheuttaa vain yhden virheen. Lisäksi `Pivot` ei koskaan klusteroi väärin kaaria, jotka eivät kuulu mihinkään huonoon kolmioon.

Olkoon $G = (V, E^+ \cup E^-)$ täydellinen merkitty graafi. Merkitään graafin G huonojen kolmioiden joukkoa $T = \{\{u, v, w\} : uv, vw \in E^+,\ uw \in E^-\}$ ja huonojen kolmioiden sisältämiä kaaria $E_T = \{uv \in E : \{u, v, w\} \in T\}$. Kolmiolle $t = \{u, v, w\}$ merkitään $E_t = \{uv, vw, uw\}$.

Otetaan avuksi seuraava lineaarinen optimointiongelma:

$$\begin{aligned} \min \quad & \sum_{e \in E_T} x_e \\ \text{s.t.} \quad & x_{uv} + x_{uw} + x_{vw} \geq 1, \quad \{u, v, w\} \in T \\ & 0 \leq x_e \leq 1, \qquad\qquad e \in E_T. \end{aligned}$$

Olkoon $\mathcal{C}$ jokin graafin G solmujen klusterointi. Klusteroinnista $\mathcal{C}$ saadaan sallittu ratkaisu yllä olevaan optimointiongelmaan, kun asetetaan, että $x_e = 1$, kun kaari e on klusteroitu virheellisesti, ja muutoin $x_e = 0$. Ongelman kohdefuktion optimiarvo on silloin alaraja klusteroinnin $\mathcal{C}$ virheiden määrälle.

Optimointiongelman duaaliongelmaksi saadaan

$$\begin{aligned} \max \quad & \sum_{t \in T} y_t \\ \text{s.t.} \quad & \sum_{\substack{t \in T: \\ e \in E_t}} y_t \leq 1, \quad e \in E_T \\ & y_t \geq 0, \quad t \in T. \end{aligned}$$

Optimoinnin vahva duaalisuuslause kertoo, että duaaliongelman kohdefunktion optimiarvo on alaraja primääriongelman kohdefunktion optimiarvolle ([21], lause 7.8). Luonnollisesti maksimointiongelman kohdefunktion arvo millä tahansa sallitulla ratkaisulla on edelleen alaraja optimiarvolle.

Lause 3.6. *Olkoon G graafi, jonka huonojen kolmioiden joukko on T. Algoritmin `Pivot` antama klusterointi $\mathcal{A}(G)$ on odotusarvoisesti 3-approksimaatio. Siis*

$\mathbb{E}\left[cost(\mathcal{A}(G))\right] \leq 3 \cdot cost(OPT(G))$.

Todistus. Merkitään tapahtumaa A_t = "jokin kolmion $t \in T$ solmuista valitaan pivotiksi, kun kaikki kolme ovat vielä graafissa jäljellä" ja tapahtuman todennäköisyyttä p_t. Kun kolmion $t = \{u, v, w\}$ jokin solmu, olkoon se v, valitaan pivotiksi, niin tasan yksi kolmion kaarista, tarkalleen uw, klusteroidaan väärin. Siis p_t on todennäköisyys sille, että yksi kolmion t kaarista aiheuttaa virheen. Virheitä tapahtuu ainoastaan silloin, kun huonon kolmion solmu valitaan pivotiksi, joten odotusarvon lineaarisuudesta saadaan, että $\mathbb{E}\left[cost(\mathcal{A}(G))\right] = \sum_{t \in T} p_t$.

Muodostetaan seuraavaksi duaaliongelman ratkaisu hyödyntämällä todennäköisyyksiä p_t. Olkoon B_e tapahtuma, että kaari e aiheuttaa klusteroinnissa virheen. Kun $e \in E_t$ niin $\mathbb{P}\left(B_e \mid A_t\right) = \frac{1}{3}$, koska jokaisella kolmion t solmulla on yhtä suuri todennäköisyys tulla valituksi pivotiksi. Voidaan siis laskea, että

$$\mathbb{P}\left(B_e \wedge A_t\right) = \mathbb{P}\left(B_e \mid A_t\right)\mathbb{P}\left(A_t\right) = \frac{1}{3}p_t.$$

Kun t ja t' ovat eri kolmioita, joilla on yhteinen kaari e, niin tapahtumat $B_e \wedge A_t$ ja $B_e \wedge A_{t'}$ ovat erillisiä. Voidaan siis laskea yhteen todennäköisyyksiä:

$$1 \geq \sum_{\substack{t \in T: \\ e \in E_t}} \mathbb{P}\left(B_e \wedge A_t\right) = \sum_{\substack{t \in T: \\ e \in E_t}} \frac{1}{3}p_t,$$

mikä on voimassa kaikille $e \in E_T$, joten ratkaisu $y_t = \frac{1}{3}p_t$ ei riko duaaliongelman rajoituksia, ja se on siis sallittu ratkaisu. Lopulta saadaan, että

$$cost(OPT(G)) \geq \sum_{\substack{t \in T: \\ e \in E_t}} \frac{1}{3}p_t = \frac{1}{3} \cdot \mathbb{E}\left[cost(\mathcal{A}(G))\right],$$

mikä todistaa väitteen. □

Yhdessä esimerkin 3.5 kanssa edellinen lause osoittaa, että `Pivot`-algoritmin odotusarvoinen approksimaatiokerroin 3 on tiukka.

3.1.2 `Pivot`-algoritmin rinnakkaistaminen

`Pivot`-algoritmista on tehty rinnakkaistettuja ja hajautettuja versioita, joissa sama analyysi approksimaatiosta pätee. Tarkastellaan niistä tässä kahta: Bulk Synchronous Parallel -mallissa toimivaa algoritmia `C4` (*Correlation Clustering with Concurrency Control*, [2]) ja MapReduce-mallissa toteutettua algoritmia `ParallelPivot` ([1]). Algoritmien toimintaa kuvaillaan vain yleisellä tasolla, eikä mennä mallin tai analyysin yksityiskohtiin.

Eräs idea `Pivot`-algoritmin rinnakkaistamiseksi voisi olla, että poimitaan satunnaisia solmuja pivoteiksi toisistaan riippumatta ja ne muodostavat klustereita samanaikaisesti. `ParallelPivot`-algoritmi toimii juuri niin.

On tärkeää pitää huolta siitä, että solmuja valitaan pivoteiksi yhtä suurella todennäköisyydellä niiden asteesta riippumatta. Sitä tietoa käytettiin lauseen 3.6 todistuksessa. Perustellaan seuraavaksi, miksi jotkin muut hyvältä vaikuttavat strategiat pivot-solmujen valinnalle eivät takaa hyvää approksimaatiota. Esimerkit ovat artikkelista [1].

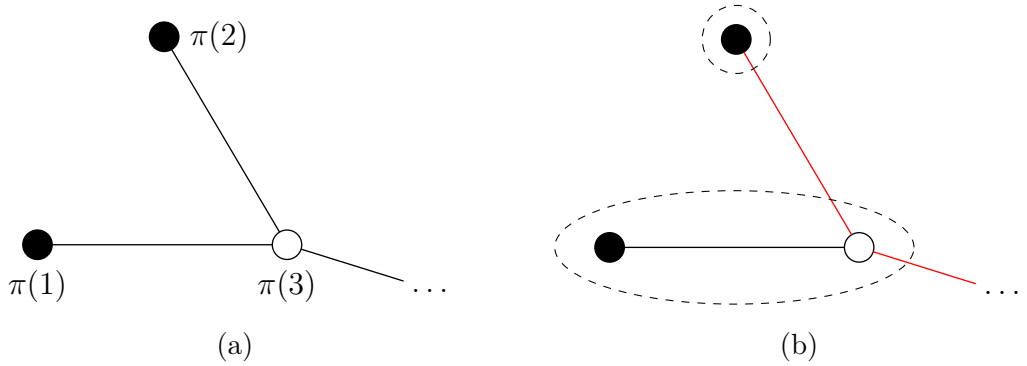
Rinnakkaistetussa algoritmissa voisi olla kannattavaa valita ensin korkea-asteisia solmuja, koska silloin graafi tyhjenee nopeammin. Olkoon solmun v todennäköisyys tulla valituksi pivot-solmuksi $\frac{d^+(v)}{2\#E^+}$. Kuitenkin silloin esimerkin 3.4 tähtigraafissa keskimmäinen solmu valitaan pivot-solmuksi todennäköisyydellä $\frac{1}{2}$. Virheiden määrän odotusarvoksi tulee $\frac{n^2-n-2}{4}$, mikä on suuruusluokaltaan $\mathcal{O}(n)$-kertainen optimiin verrattuna.

Toisaalta pivot-solmujen valinnassa voitaisiin suosia pieniasteisia solmuja.[3] Olkoon solmun v todennäköisyys tulla valituksi pivot-solmuksi nyt $\frac{1/d^+(v)}{\sum_{v\in V} 1/d^+(v)}$. Silloin esimerkin 3.5 graafissa G_n ensimmäiseksi pivot-solmuksi valitaan reunasolmu todennäköisyydellä $\frac{n}{n+1}$ ja sisäsolmu todennäköisyydellä $\frac{1}{n+1}$. Virheiden määrän odotusarvo on noin $\frac{n^2}{3}$, mikä on jälleen $\mathcal{O}(n)$-kertainen verrattuna optimiin.

Voidaan siis todeta, että millä tahansa tavalla valittu ahne riippumaton joukko ei säilytä `Pivot`-algoritmin approksimaatiokerrointa. Oletetaan siksi aina jatkossa, että solmujen järjestys π on valittu permutaatioiden tasajakaumasta. Alaluvussa 3.2 kuitenkin osoitetaan, että pienen metsäisyyden graafeissa voidaan jättää korkea-asteiset solmut `Pivot`-algoritmin ulkopuolelle ilman, että approksimaatiokerroin huononee.

`ParallelPivot`-algoritmin suoritus tapahtuu peräkkäisissä kierroksissa. Jokaisen kierroksen alussa poimitaan solmuja todennäköisyydellä $\frac{\varepsilon}{\Delta^+}$, missä ε on jokin pieni vakio ja Δ^+ on syötegraafin positiivinen maksimiaste kierroksen alussa. Näin saadaan joukko P, jossa on odotusarvoisesti $\frac{\varepsilon n}{\Delta^+}$ solmua. Pivot-solmut eivät saa olla vierekkäisiä, joten joukosta P poistetaan kaikki solmut, joilla on naapureita joukossa P. Lopputuloksena saadaan riippumaton joukko. Joukon P solmut ovat nyt pivot-solmuja ja kukin niistä muodostaa naapurustostaan klusterin. Jos jollakin solmulla $u \notin P$ on useita naapureita joukossa P, niin se liittyy sen pivotin klusteriin, joka on ensimmäinen järjestyksessä π, kuten kuvassa 13. Tähän päättyy yksi kierros. Algoritmin analyysissä osoitetaan, että $\mathcal{O}(\frac{1}{\varepsilon}\log n)$ kierroksen jälkeen graafin mak-

[3] Lubyn kuuluisa algoritmi vuodelta 1986 ([22]) käyttää tätä strategiaa. Siinä riippumaton joukko muodostetaan poimimalla solmuja toisistaan riippumatta todennäköisyydellä $\frac{1}{2d(v)}$. Algoritmi toimii PRAM-mallissa (*parallel random-access machine*) ja sen suoritus vie $\mathcal{O}(\log n)$ kierrosta suurella todennäköisyydellä.



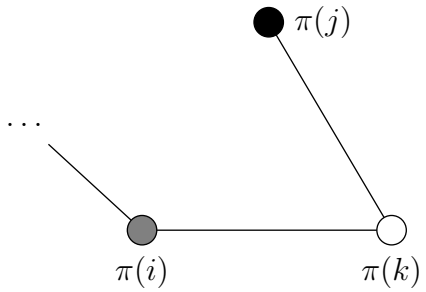
Kuva 13: Samanaikaisista pivot-naapureistaan solmu $\pi(3)$ valitsee sen, jonka järjestysluku on pienin.

simiaste on puolittunut suurella todennäköisyydellä. Kierroksia tarvitaan yhteensä siis $\mathcal{O}(\frac{1}{\varepsilon}\log n \cdot \log \Delta)$, kunnes $\Delta^+ < 1$ ja graafissa on jäljellä enää yksittäisiä solmuja, jotka muodostavat klusterinsa yksin.

`ParallelPivot` ei ole täsmälleen sama algoritmi kuin ihanteellisesti rinnakkaistettu `Pivot`. Suuriasteiset solmut joutuvat poistumaan joukosta P todennäköisemmin kuin muut, joten kaikilla solmuilla ei ole yhtä suuri todennäköisyys tulla pivot-solmuksi. Solmujen järjestystä π taas käytetään ainoastaan tasatilanteiden ratkaisemiseen klusterien muodostamisessa. Artikkelissa [1] osoitetaankin lauseen 3.6 todistusta mukaillen, että `ParallelPivot` on $(3 + \mathcal{O}(\varepsilon))$-approksimaatio.

Toinen lähestymistapa rinnakkaistamiseen on samanlainen kuin algoritmissa 2: voidaan käsitellä useita permutaation π lokaaleja minimejä kerralla. Toisaalta `Pivot`-algoritmissa lokaalien minimien naapurit käyttäytyvät eri hieman eri tavalla. Ne eivät vain poistu graafista, kun *jokin* niiden naapureista valitaan pivotiksi, vaan menevät samaan klusteriin sen pivot-naapurinsa kanssa, joka on ensimmäinen järjestyksessä π. Kuvassa 13 havainnollistetaan tätä tilannetta.

Algoritmissa `C4` pidetään huolta, että pivot-solmuiksi ei valita vierekkäisiä solmuja ja että samanaikaisista pivot-naapureistaan solmu menee järjestyksessä aiemman naapurinsa klusteriin. Algoritmin suoritus on jaettu vaiheisiin. Vaiheen alussa otetaan järjestyksestä π ensimmäisten $\frac{\varepsilon n}{\Delta^+}$ solmun prefiksi, missä Δ^+ on jälleen graafin positiivinen maksimiaste kyseisen vaiheen alussa. Prefiksin solmuja käsitellään järjestyksen alusta alkaen useita rinnakkain. Jos solmu on lokaali minimi, se muodostaa naapuriensa kanssa klusterin. Jos taas käsittelyssä olevalla solmulla on itseään edeltäviä naapureita, se odottaa kunnes kyseisen naapurin tila tiedetään. Artikkelissa [2] ei eksplisiittisesti kerrota, miten kuvan 14 mukaiset tilanteet ratkaistaan, mutta niiden ratkaiseminen ei oleellisesti pidennä yhtä vaihetta. Kyseisten tilanteiden voidaan ajatella olevan harvinaisia, koska lokaaleja minimejä käsitellään suunnilleen kasvavassa järjestyksessä.



Kuva 14: Olkoon $i < j < k$. Solmu $\pi(k)$ ei voi liittyä naapurinsa $\pi(j)$ klusteriin, koska sillä on järjestyksessä aikaisempi naapuri $\pi(i)$, josta ei vielä tiedetä tuleeko siitä pivot-solmu.

`C4` tulostaa saman klusteroinnin kuin `Pivot`, joten se on 3-approksimaatio. `C4`-algoritmin analyysissä osoitetaan, että prefiksin käsittely vie $\mathcal{O}(\frac{1}{\varepsilon}\log n)$ kierrosta BSP-mallissa. Kuten `ParallelPivot`-algoritmissa, yhden vaiheen jälkeen maksimiaste Δ^+ puolittuu suurella todennäköisyydellä, joten prefiksin pituus $\frac{\varepsilon n}{\Delta^+}$ tuplaantuu joka vaiheessa. Koko graafi on täten käsitelty $\mathcal{O}(\log \Delta)$ vaiheen jälkeen. Kokonaisuudessaan algoritmin suoritusajaksi tulee $\mathcal{O}(\frac{1}{\varepsilon}\log n \cdot \log \Delta)$.

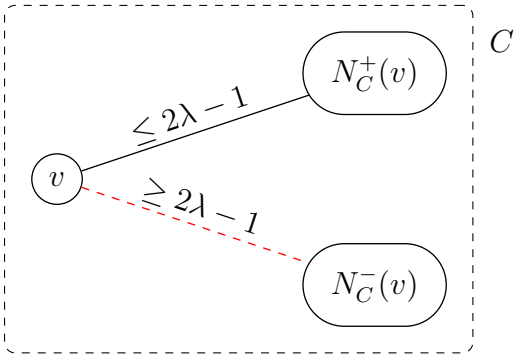
3.2 Pieni metsäisyys

Tässä alaluvussa todistetaan kaksi tulosta, jotka käsittelevät korrelaatioklusterointia pienen metsäisyyden graafeissa. Taustalla oleva havainto on, että kun graafi on harva, niin klustereiden ei kannata olla kovin suuria, koska klusterin indusoimassa aligraafissa on paljon negatiivisia kaaria.

Lemma 3.7. *Olkoon $G = (V, E^+ \cup E^-)$ graafi, jonka positiiviset kaaret E^+ indusoivat λ-metsäisen graafin. Graafille G on olemassa optimaalinen klusterointi, jonka jokaisessa klusterissa on korkeintaan $4\lambda - 2$ solmua.*

Todistus. Olkoon $\mathcal{C}$ graafin G solmujen optimaalinen klusterointi, jossa on vähintään $4\lambda - 1$ solmun klusteri C. Silloin on olemassa solmu $v \in C$, jolla on korkeintaan $2\lambda - 1$ positiivista naapuria klusterissa C, sillä muutoin graafissa $G[C]$ olisi enemmän kuin $\lambda(\#C - 1)$ positiivista kaarta, mikä on ristiriita graafin $G[E^+]$ λ-metsäisyyden kanssa.

Määritellään uusi klusterointi $\mathcal{C}' = \mathcal{C} \setminus \{C\} \cup \{C \setminus \{v\}, \{v\}\}$ erottamalla solmu v klusterista C omaksi klusterikseen. Solmun v poistaminen klusterista C lisää positiivisten virheiden määrää korkeintaan $2\lambda - 1$ ja vähentää negatiivisten virheiden määrää ainakin $\#C - 1 - (2\lambda - 1) \geq 2\lambda - 1$, siis kokonaisuudessaan virheiden määrä ei kasva (kuva 15). Toistamalla tätä argumenttia mistä tahansa optimaalisesta klus-



Kuva 15: Solmulla $v \in C$ on suurempi tai yhtäsuuri negatiivinen naapurusto kuin positiivinen naapurusto.

teroinnista $\mathcal{C}$ saadaan klusterointi, jonka klustereissa on korkeintaan $4\lambda - 2$ solmua ja klusteroinnin optimaalisuus säilyy. □

Hieman muuttamalla argumenttia voidaan todistaa huomionarvoinen väite, että *kaikissa* optimaalisissa klusteroinneissa jokaisessa klusterissa on korkeintaan $4\lambda - 1$ solmua.

Lemmasta 3.7 seuraa, että graafille $T = (V, E^+ \cup E^-)$, jossa positiiviset kaaret indusoivat puun, on olemassa optimaalinen klusterointi, jossa klustereiden koko on korkeintaan 2. Lemman 3.1 seurauksen mukaan optimaalisessa klusteroinnissa klusterit ovat yhtenäisiä, joten kahden solmun klusterissa oleva kaari on positiivinen kaari. Sellaiset klusteroinnit vastaavat pariutuksia positiivisten kaarten indusoimassa puussa. Kahden solmun yhtenäiset klusterit eivät aiheuta negatiivisia virheitä, joten klusteroinnin virheet johtuvat yksinomaan klustereiden välisistä positiivisista kaarista. Vähiten virheitä saadaan, kun "suojellaan" mahdollisimman monta kaarta pariuttamalla mahdollisimman monta solmua klustereihin. Puussa $T[E^+]$ on $\#V_T - 1$ kaarta, joten pienimmillään virheitä tulee $\#V_T - 1 - \#M$, missä M on maksimipariutus. Maksimipariutusta vastaava klusterointi saadaan valitsemalla pivotiksi joka vaiheessa jokin lehtisolmu.

Osoitetaan seuraavaksi, että λ-metsäisessä graafissa "korkea-asteiset" solmut voidaan jättää huomiotta approksimaatiokertoimen lainkaan kärsimättä. Korkea-asteisuus määritellään graafin metsäisyyden suhteen. Lauseen tulkintaa `Pivot`-algoritmin näkökulmasta voisi kuvailla seuraavasti: korkea-asteista solmua ei kannata valita pivotiksi, koska graafin metsäisyyden vuoksi sen naapurustossa on paljon negatiivisia kaaria. Tulos ja todistus ovat artikkelista [23].

Lause 3.8. *Olkoon $G = (V, E^+ \cup E^-)$ täydellinen merkitty graafi, jonka positiiviset kaaret E^+ indusoivat λ-metsäisen graafin. Olkoon $\mathcal{A}$ korrelaatioklusterointialgoritmi,*

jonka approksimaatiokerroin on $\alpha > 1$. Olkoon

$$K = \left\{ v \in V : d_G^+(v) > \lambda \cdot \frac{8\alpha}{\alpha - 1} \right\} \subseteq V$$

korkea-asteisten solmujen joukko. Merkitään matala-asteisia solmuja $M = V \setminus K$. Silloin

$$cost(\mathcal{A}(G[M]) \cup \{\{v\} : v \in K\}) \leq \alpha \cdot cost(OPT(G)).$$

Todistus. Merkitään korkea-asteisten solmujen viereisiä kaaria $\{uv \in E^+ : v \in K\} = E_K^+$ ja $\{uv \in E^- : v \in K\} = E_K^-$. Loput, matala-asteisten solmujen M väliset kaaret olkoon $E_M = E \setminus (E_K^+ \cup E_K^-)$.

Tavallisen kättelylemman sijaan on voimassa

$$\#E_K^+ \leq \sum_{v \in K} d^+(v) \leq 2 \cdot \#E_K^+, \tag{3}$$

sillä summassa kaari $uv \in E_K^+$ lasketaan kahdesti, jos sekä u ja v ovat korkea-asteisia, ja vain kerran, jos ainoastaan toinen solmuista u ja v on korkea-asteinen.

Olkoon $OPT(G)$ jokin optimaalinen klusterointi, jonka klustereissa on korkeintaan $4\lambda - 2$ solmua. Lemman 3.7 mukaan sellainen on olemassa. Merkitään $v(F)$ niiden kaarien lukumäärää joukossa $F \subseteq E$, jotka aiheuttavat virheitä klusteroinnissa $OPT(G)$. Todetaan, että

$$cost(OPT(G)) = v(E_K^+) + v(E_K^-) + v(E_M) \geq v(E_K^+) + v(E_M). \tag{4}$$

Olkoon $v \in K$ mikä tahansa korkea-asteinen solmu. Korkeintaan

$$4\lambda - 3 < 4\lambda \leq \frac{\alpha - 1}{2\alpha} \cdot d^+(v)$$

solmun v naapureista kuuluu sen kanssa samaan klusteriin klusteroinnissa $OPT(G)$. Kaaret niihin naapureihin eivät siis aiheuta positiivisia virheitä. Korkea-asteisilla solmuilla on yhteensä tällaisia naapureita korkeintaan

$$\frac{\alpha - 1}{2\alpha} \cdot \sum_{v \in K} d^+(v) \overset{(3)}{\leq} \frac{\alpha - 1}{2\alpha} \cdot 2 \cdot \#E_K^+ = \frac{\alpha - 1}{\alpha} \cdot \#E_K^+.$$

Toisin sanoen virheitä aiheutuu vähintään

$$v(E_K^+) \geq \left(1 - \frac{\alpha - 1}{\alpha}\right) \#E_K^+ = \frac{1}{\alpha} \cdot \#E_K^+. \tag{5}$$

Joukko $\{C \cap M : C \in OPT(G)\}\}$ on eräs klusterointi matala-asteisten solmujen

aligraafille $G[M]$. Graafin $G[M]$ optimaalisen klusteroinnin $OPT(G[M])$ virheiden määrä on siis korkeintaan $v(E_M)$ $(*)$.

Vedetään kaikki yhteen:

$$\begin{aligned} cost(\{\{v\} : v \in K\} \cup \mathcal{A}(G[M])) &= \#E_K^+ + cost(\mathcal{A}(G[M])) \\ &\overset{(\dagger)}{\leq} \#E_K^+ + \alpha \cdot cost(OPT(G[M])) \\ &\overset{(*)}{\leq} \#E_K^+ + \alpha \cdot v(E_M) \\ &\overset{(5)}{\leq} \alpha \cdot v(E_K^+) + \alpha \cdot v(E_M) \\ &\overset{(4)}{\leq} \alpha \cdot cost(OPT(G)). \end{aligned}$$

$(\dagger)$: Algoritmi $\mathcal{A}$ on α-approksimaatio.

□

Jos meillä on algoritmi, jonka suorituskyky riippuu graafin maksimiasteesta, ja syötegraafin metsäisyys tunnetaan, niin graafista voidaan poistaa metsäisyyteen nähden korkea-asteiset solmut ja siten saadaan mahdollisesti merkittävästi nopeampi suoritusaika.

Algoritmi 4: Korrelaatioklusterointi metsäisyyden λ graafissa

Syöte: Graafi $G = (V, E^+ \cup E^-)$, jonka metsäisyys on λ.
Klusterointialgoritmi $\mathcal{A}$, joka on α-approksimaatio.
Tuloste: Klusterointi $\mathcal{C}$, joka on α-approksimaatio.

$\mathcal{C} \leftarrow \emptyset$
$K \leftarrow \left\{v \in V : d^+(v) > \lambda \cdot \frac{8\alpha}{\alpha - 1}\right\}$
$\mathcal{C} \leftarrow \mathcal{A}(G[V \setminus K]) \cup \{\{v\} : v \in K\}$
palauta $\mathcal{C}$

4 Algoritmeja MPC-mallissa

Tässä luvussa esitellään ensin vakioaikainen algoritmi ja sitten tämän tutkielman päätulos, `Pivot`-algoritmiin perustuva hajautettu algoritmi. Tämän luvun päälähteenä on käytetty artikkelia Massively Parallel Correlation Clustering in Bounded Arboricity Graphs ([23]).

Merkintöjen yksinkertaistamiseksi sovitaan, että graafit ovat aina täydellisiä merkittyjä graafeja ja muun muassa merkinnöillä $d(v)$ ja E tarkoitetaan solmun positiivista astetta ja graafin positiivisia kaaria.

4.1 Vakioaikainen algoritmi

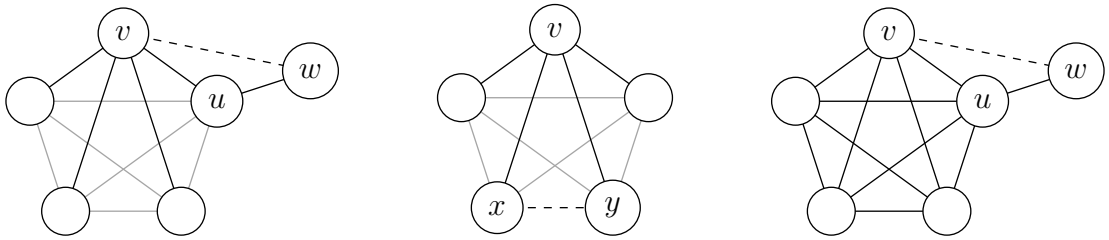
Esitellään algoritmi, joka on $\mathcal{O}(\lambda^2)$-approksimaatio $\mathcal{O}(1)$ MPC-kierroksessa. Siinä muodostetaan ainoastaan täydellisiä klustereita, joista ei aiheudu yhtään negatiivisia virheitä. Graafin jokainen yhtenäinen komponentti, joka on täydellinen graafi, muodostaa klusterin. Kaikki muut solmut ovat klusterissa yksinään. Algoritmi muodostaa siis jokaisessa graafin yhtenäisessä komponentissa triviaaleja klusterointeja.

Lemman 1.3 mukaan λ-metsäisessä graafissa aligraafina voi olla korkeintaan klikki $K_{2\lambda}$, joten solmun v sisältävä yhtenäinen komponentti ei voi olla klikki, jos $d(v) \geq 2\lambda$. Ne solmut, joiden aste on korkeintaan $2\lambda - 1$, voivat tunnistaa olevansa klikissä seuraavalla tavalla. Olkoon π graafin solmujen satunnainen järjestys. Jokainen solmu laskee oman naapurustonsa minimin $min(v) = \min\{\pi^{-1}(x) : x \in N(v) \cup \{v\}\}$. Sanotaan, että solmu v on lokaali minimi, jos $min(v) = \pi^{-1}(v)$. Jokaiselle lokaalille minimille lasketaan luvut

$$\begin{aligned} n(v) &= \#\{u \in N(v) : min(u) = \pi^{-1}(v)\} \\ m_\wedge(v) &= \#\{uw \in E : min(u) = min(w) = \pi^{-1}(v)\} \\ m_\vee(v) &= \#\{uw \in E : min(u) = \pi^{-1}(v) \text{ tai } min(w) = \pi^{-1}(v)\} \end{aligned}$$

ja tarkistetaan seuraavat ehdot:

1. $n(v) = d(v)$. Jos $n(v) < d(v)$, niin jollakin solmun v naapureista on naapuri w, jolla $\pi^{-1}(w) < \pi^{-1}(v)$, ja silloin yhtenäinen komponentti ei ole klikki.
2. $m_\wedge(v) \geq \binom{d(v)+1}{2}$. Jos $m_\wedge(v) < \binom{d(v)+1}{2}$, niin jotkin kaksi solmun v naapuria eivät ole naapureita keskenään.
3. $m_\vee(v) \leq \binom{d(v)+1}{2}$. Edellisen kohdan ehdon ollessa voimassa myös $m_\vee(v) \geq \binom{d(v)+1}{2}$. Jos $m_\vee(v) > \binom{d(v)+1}{2}$, niin jollakin solmun v naapurilla on naapuri w, jolla $\pi^{-1}(w) > \pi^{-1}(v)$, mutta kuitenkin $w \notin N(v)$.



Kuva 16: Solmu v on lokaali minimi. Sen naapurilla u voi olla naapuri w jonka järjestysluku $\pi^{-1}(w)$ on joko (a) suurempi tai (c) pienempi kuin solmun v järjestysluku. Kuvassa (b) solmun v naapurit x ja y eivät ole naapureita keskenään.

Jos jokainen ehto on voimassa, niin solmun v yhtenäinen komponentti on klikki.

Lause 4.1. *Olkoon G λ-metsäinen graafi ja $\mathcal{A}(G)$ algoritmin 5 antama klusterointi graafille G. Silloin $cost(\mathcal{A}(G)) \leq \mathcal{O}(\lambda^2) \cdot cost(OPT(G))$.*

Todistus. Lemman 3.7 mukaan on olemassa optimaalinen klusterointi $OPT(G)$, jossa jokaisessa klusterissa on korkeintaan $4\lambda - 2$ solmua. Lemman 3.1 seurauksena optimaalisessa klusteroinnissa jokainen klusteri on yhtenäinen. Lisäksi algoritmi 5 muodostaa vain yhtenäisiä klustereita. Voidaan siis rajoittua tarkastelemaan graafin G yhtenäistä komponenttia H ja vertailla siinä klusterointeja $OPT(G)$ ja $\mathcal{A}(G)$.

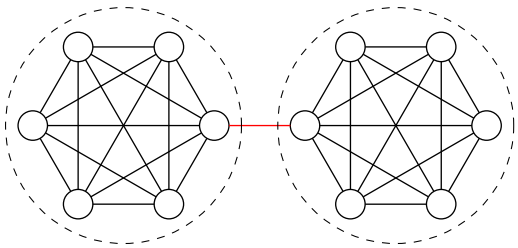
Olkoon siis $H = (V_H, E_H)$ graafin G yhtenäinen komponentti, jossa on n solmua. Graafi H on λ-metsäinen, joten siinä on korkeintaan $\lambda(n-1)$ kaarta. Jos H on täydellinen graafi, niin klusteroinnissa $\mathcal{A}(G)$ joukko V_H on yksi klusteri, kuten klusteroinnissa $OPT(G)$. Muussa tapauksessa algoritmi muodostaa yhden solmun klustereita ja aligraafin H positiiviset kaaret aiheuttavat siten korkeintaan $\lambda(n-1)$ virhettä. Jokaisessa klusteroinnin $OPT(G)$ klusterissa on korkeintaan $4\lambda - 2$ solmua, joten solmut V_H on jaettu vähintään $\frac{n}{4\lambda-2}$ klusteriin. Aligraafi H on yhtenäinen, joten edellä mainittujen klusterien välillä on vähintään $\frac{n}{4\lambda-2} - 1$ kaarta, jotka aiheuttavat positiivisia virheitä klusteroinnissa $OPT(G)$. Siis approksimaatiokerroin on huonoimmillaan korkeintaan $\frac{cost(\mathcal{A}(G))}{cost(OPT(G))} \leq \frac{\lambda(n-1)}{\frac{n}{4\lambda-2}-1} \in \mathcal{O}(\lambda^2)$. □

Approksimaatiokerroin on tiukka vakiokerrointa 4 myöten. Otetaan kaksi täydellistä graafia $K_{2\lambda}$ ja yhdistetään ne toisiinsa yhdellä kaarella kuten kuvassa 17. Optimaalisessa klusteroinnissa kukin $K_{2\lambda}$-aligraafi on klusteri ja ainoastaan niiden välillä oleva kaari on klusteroitu väärin. Algoritmi 5 taas tekee jokaisesta solmusta oman klusterinsa ja virheitä tulee $2\binom{2\lambda}{2} + 1 = 4\lambda^2 - 2\lambda + 1$ kappaletta.

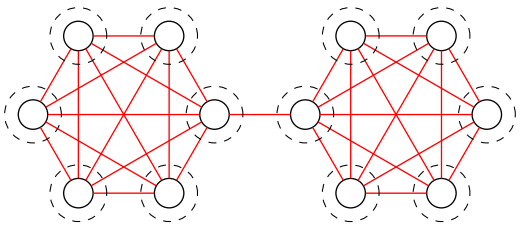
Algoritmi 5: Vakioaikainen korrelaatioklusterointialgoritmi

Syöte: Graafi $G = (V, E)$, jonka metsäisyys on λ.
Solmujen satunnainen järjestys $\pi\colon [1, n] \to V$.
Tuloste: Klusterointi $\mathcal{C}$.

kaikille $u \in V$ **rinnakkain**
  $min(u) = \min\{\pi^{-1}(x) : x \in N(u) \cup \{u\}\}$
$V' \leftarrow \{v \in V : min(v) = \pi^{-1}(v)\}$
kaikille $v \in V'$ **rinnakkain**
  jos $d(v) \geq 2\lambda$ **niin**
    $status(v) \leftarrow ei$
  muutoin
    $n(v) \leftarrow \#\{u \in N(v) : min(u) = \pi^{-1}(v)\}$
    jos $n(v) < d(v)$ **niin**
      $status(v) \leftarrow ei$
    muutoin
      $m_\wedge(v) \leftarrow \#\{uw \in E : min(u) = min(w) = \pi^{-1}(v)\}$
      jos $m_\wedge(v) < \binom{d(v)+1}{2}$ **niin**
        $status(v) \leftarrow ei$
      muutoin
        $m_\vee(v) \leftarrow \#\{uw \in E : min(u) = \pi^{-1}(v)$ tai $min(w) = \pi^{-1}(v)\}$
        jos $m_\vee(v) > \binom{d(v)+1}{2}$ **niin**
          $status(v) \leftarrow ei$
        muutoin
          $status(v) \leftarrow kyllä$
$\mathcal{C} \leftarrow \{\{u\} : u \in V,\ status(min(u)) = ei\} \cup \{\{v\} \cup N(v) : v \in V',\ status(v) = kyllä\}$
palauta $\mathcal{C}$



(a) Optimaalinen klusterointi, jossa tehdään yksi virhe.



(b) Algoritmin 5 antama klusterointi, jossa tehdään $\#E$ virhettä.

Kuva 17

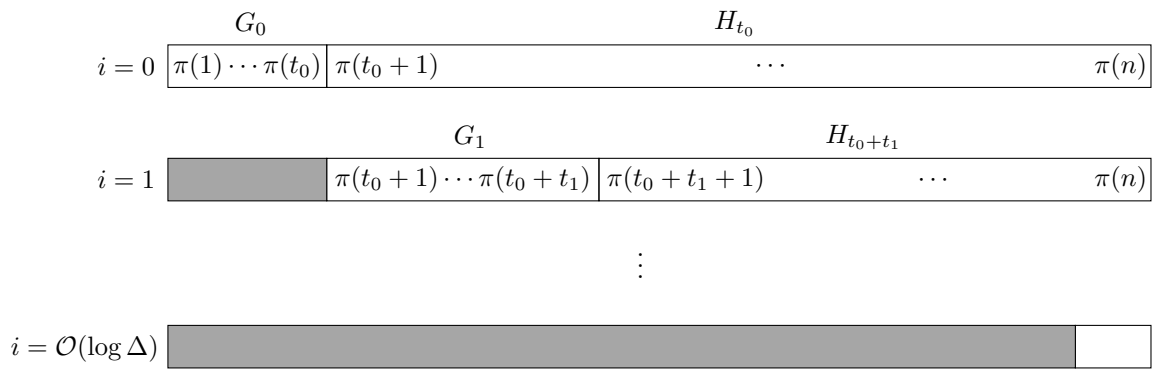
4.2 `Pivot` MPC-mallissa

Tässä esitellään algoritmi, joka tuottaa korrelaatioklusteroinnille 3-approksimaation $\mathcal{O}(\log\log n \cdot \log \Delta)$ kierroksessa sublineaarisen muistin MPC-mallissa. Se on eksponentiaalisesti nopeampi kuin aiemmat tunnetut algoritmit. MPC-mallin kaltaisissa BSP- ja MapReduce-malleissa toteutetut algoritmit `C4` ja `ParallelPivot` käyttävät $\mathcal{O}(\log n \cdot \log \Delta)$ kierrosta omissa malleissaan. Lisäksi MPC-mallissa voidaan simuloida LOCAL-mallin algoritmeja ja siten saadaan $\mathcal{O}(\log n)$ kierrosta tarvitseva algoritmi korrelaatioklusteroinnille. Muodostetaan ensin ahne riippumaton joukko P suoraviivaisesti simuloimalla LOCAL-algoritmia MPC-mallissa $\mathcal{O}(\log n)$ kierroksessa. Kukin solmu $u \notin P$ katsoo naapurustoaan ja lähettää inhibiittorilleen viestin yhdessä kierroksessa. Seuraavalla kierroksella solmut $v \in P$ muodostavat klusterin naapurustonsa niistä solmuista, joiden inhibiittori se itse on.

Tämän luvun algoritmi muistuttaa toteutukseltaan alaluvussa 3.1.2 kuvailtua algoritmia `C4`. Algoritmissa 6 otetaan solmujen satunnainen permutaatio, lohkaistaan siitä sopivan kokoisia prefiksejä ja käsitellään prefiksigraafeja vaiheittain. Lohkon käsittelyssä käytetään graafin eksponentiaatiota ja kierrosten tiivistämistä. Maksimiaste rajoittaa graafin eksponentiaatiota, joten lohkon koko valitaan niin, että prefiksigraafin maksimiaste on suurella todennäköisyydellä riittävän pieni. Kussakin vaiheessa lohkon koko riippuu jäljellä olevan graafin maksimiasteesta. Kun osoitetaan, että maksimiaste puolittuu jokaisen lohkon käsittelyn jälkeen, niin seuraavassa vaiheessa voidaan ottaa kaksi kertaa niin suuri lohko kuin edellisessä.

Esityksen ja analyysin selkeyden vuoksi määritellään lohkon käsittelylle alirutiini. Siinä lohkosta valitaan maksimaalinen ahne riippumaton joukko, jonka solmuja käytetään pivot-solmuina. Alirutiinin valitsemat solmut muodostavat naapurustoistaan klustereita kunnioittaen järjestystä π.

Menemättä tarkemmin teknisiin yksityiskohtiin huomautettakoon, että alirutiini



Kuva 18: Algoritmin 6 toiminta vaiheissa.

Algoritmi 6: `Pivot` sublineaarisen muistin MPC-mallissa

Syöte: Graafi $G = (V, E)$, jonka maksimiaste on Δ.
Solmujen satunnainen järjestys $\pi\colon [1, n] \to V$.
Tuloste: Klusterointi $\mathcal{C}$.

$a \leftarrow 0$
kaikille $i \in [0, \mathcal{O}(\log \Delta)]$ **toista**
  $t_i \leftarrow \mathcal{O}\left(\frac{n \log n}{\Delta / 2^i}\right)$
  $G_i \leftarrow G\left[\{\pi(a+1), \ldots, \pi(a+t_i)\}\right]$
  $\pi_i \leftarrow$ Permutaation π rajoittuma graafin G_i solmuille
  $P \leftarrow$ Algoritmin 7 tuloste syötteillä G_i ja π_i
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\{u \in V : v = inhib(u)\} \cup \{v\} : v \in P\}$
  Klusteroidut solmut merkitään poistetuksi graafista.
  $a \leftarrow a + t_i$
palauta $\mathcal{C}$

Algoritmi 7: Algoritmin 6 alirutiini

Syöte: Graafi $G' = (V', E')$, jonka maksimiaste on Δ'.
Solmujen satunnainen järjestys $\pi'\colon [1, n'] \to V'$.
Tuloste: Maksimaalinen riippumaton joukko P.

Asetetaan jokaiselle solmulle $v \in V'$ oma kone.
Kerätään koneille graafieksponentiaatiolla solmujen $\mathcal{O}(\frac{\log n'}{\log \Delta'})$-naapurustot.
Simuloidaan algoritmia 2 $\mathcal{O}(\log \Delta')$ tiivistetyssä kierroksessa.
palauta *P, algoritmin 2 antama riippumaton joukko*

voidaan toteuttaa myös niin, että globaalia muistia tarvitaan vähemmän. Silloin tarvitaan hienostuneempia tekniikoita ja saadaan vain hieman pidempi suoritusaika. Yksityiskohdat löytyvät artikkelista [23].

Lemma 4.2. *Olkoon $G = (V, E)$ graafi ja π graafin solmujen satunnainen järjestys. Algoritmissa 6 prefiksigraafin G_i maksimisaste Δ' on luokkaa $\mathcal{O}(\log n)$ suurella todennäköisyydellä.*

Todistus. Yksinkertaisuuden vuoksi ja yleisyyden kärsimättä tarkastellaan algoritmin ensimmäistä vaihetta, jossa $i = 0$. Olkoon prefiksigraafissa $\frac{n \log n}{\Delta}$ solmua. Todennäköisyys sille, että jokin tietty solmu $u \in V$ on prefiksigraafissa, on $\left(\frac{n \log n}{\Delta}\right)/n = \frac{\log n}{\Delta}$. Olkoon $v \in V$ jokin solmu, jolla $\pi^{-1}(v) \in [1, t_0]$. Odotusarvo solmun v naapu-

rien määrälle prefiksigraafissa on

$$\mathbb{E}\left[d_{G_0}(v)\right] \leq d_G(v) \cdot \frac{\log n}{\Delta} \leq \log n.$$

Mielivaltaisen solmun asteen odotusarvoa rajoittaa ylhäältä $\log n$, joten prefiksigraafin maksimiaste Δ' on odotusarvoisesti korkeintaan $\log n$.

Näytetään, että Δ' on myös suurella todennäköisyydellä suuruusluokkaa $\mathcal{O}(\log n)$. Chernoffin rajasta binomijakaumalle saadaan, että

$$\mathbb{P}\left(\Delta' \geq c \cdot \log n\right) \leq e^{-\frac{(c-1)^2 \cdot \log n}{c+1}} = \frac{1}{n^{\frac{(c-1)^2}{c+1}}} = \frac{1}{n^{\mathcal{O}(c)}}.$$

Siis on voimassa $\Delta' \in \mathcal{O}(\log n)$ todennäköisyydellä $1 - \frac{1}{n^c}$ halutulla vakiolla c. □

Osoitetaan seuraavaksi, että alirutiinilla voidaan valita lohkosta pivot-solmut $\mathcal{O}(\log\log n + \log\Delta)$ kierroksessa.

Lemma 4.3. *Olkoon $G = (V, E)$ graafi, jossa on n solmua ja jonka maksimiaste on $\Delta \in \mathcal{O}(\log n)$. Olkoon $\pi\colon [1, n] \to V$ satunnainen solmujen järjestys. Silloin algoritmilla 7 voidaan simuloida LOCAL-algoritmi ahneelle riippumattomalle joukolle graafissa G järjestyksen π suhteen $\mathcal{O}(\log\log n + \log\Delta)$ MPC-kierroksessa suurella todennäköisyydellä.*

Todistus. Lauseen 1.9 mukaan suurella todennäköisyydellä jokainen graafin G solmu voi määrittää oman lopullisen tilansa $\mathcal{O}(\log n)$-naapurustonsa järjestyksen perusteella. Solmun $\mathcal{O}(\log n)$-naapurusto ei kuitenkaan välttämättä mahdu yhden koneen muistiin.

Oletetaan, että graafissa G pisin riippuvuuspolku järjestyksen π suhteen on $l \cdot \log n$ solmua pitkä jollain $l > 0$. Näytetään, että sopivalla vakion c valinnalla $\left(c \cdot l \cdot \frac{\log n}{\log\Delta}\right)$-naapurustot mahtuvat koneelle, jonka muisti on $\mathcal{O}(n^\delta)$. Valitaan sellainen $c > 0$, että $c \cdot l < \delta$. Silloin

$$\begin{aligned}
& c \cdot l \cdot \frac{\log n}{\log\Delta} \cdot \log\Delta = c \cdot l \cdot \log n < \delta \cdot \log n \\
\iff & e^{c \cdot l \cdot \frac{\log n}{\log\Delta} \cdot \log\Delta} < e^{\delta \cdot \log n} \\
\iff & \Delta^{c \cdot l \cdot \frac{\log n}{\log\Delta}} < n^\delta \\
\implies & \mathcal{O}\left(\Delta^{\frac{\log n}{\log\Delta}}\right) \in \mathcal{O}\left(n^\delta\right).
\end{aligned}$$

Sopivalla vakion c valinnalla siis solmun $\mathcal{O}\left(\frac{\log n}{\log\Delta}\right)$-naapurusto mahtuu yhden koneen muistiin.

Nimetään jokaiselle solmulle oma kone. Kerätään graafin eksponentiaatiolla solmujen $\mathcal{O}\left(\frac{\log n}{\log \Delta}\right)$-naapurustot kunkin solmun omalle koneelle $\mathcal{O}\left(\log\left(\frac{\log n}{\log \Delta}\right)\right) = \mathcal{O}(\log \log n)$ kierroksessa. Sitten simuloidaan graafissa LOCAL-algoritmi ahneelle riippumattomalle joukolle $\mathcal{O}(\log \Delta)$ tiivistetyssä kierroksessa. Algoritmin suoritukseen menee siis $\mathcal{O}(\log \log n + \log \Delta)$ MPC-kierrosta.

□

Seuraavaksi osoitetaan, että prefiksigraafin käsittelyn jälkeen jäljellä olevan graafin aste on pienentynyt.

Lemma 4.4. *Olkoon $G = (V, E)$ graafi, jossa $\#V = n$ ja olkoon $\pi\colon [1, n] \to V$ solmujen satunnainen järjestys. Olkoon $H_t \subseteq G$ aligraafi, joka saadaan, kun algoritmissa 6 prosessoidaan ensimmäiset t solmua $\pi(1), \ldots, \pi(t)$. Silloin graafin H_t maksimiaste Δ_{H_t} on korkeintaan $\mathcal{O}(\frac{n \log n}{t})$ suurella todennäköisyydellä.*

Todistus. Otetaan käyttöön merkintä $d_{H_k}(v) = d_k(v)$ ja kiinnitetään $c > 0$. Olkoon $t' \in [1, t]$ ja olkoon $v \in V$ jokin solmu. Tarkastellaan solmun v astetta $d_{t'-1}(v)$ ennen solmun $\pi(t')$ poistamista graafista. Jos $d_{t'-1}(v) \leq \frac{c \cdot n \log n}{t}$, niin väite on voimassa, koska solmun v aste ei voi kasvaa, kun käsitellään solmut $\pi(t'), \ldots, \pi(t)$.

Olkoon siis $d_{t'-1}(v) > \frac{c \cdot n \log n}{t}$. Kehitetään seuraavaksi yläraja todennäköisyydelle, että solmu v on yhä graafissa $H_{t'}$ solmun $\pi(t')$ käsittelyn jälkeen. Solmu v voi jäädä graafiin vain, jos se itse ja mikään sen naapureista ei ole $\pi(t')$. Koska π on valittu tasajakaumasta, niin todennäköisyys sille, että jokin solmuista $u \in N_{H_{t'-1}}(v) \cup \{v\}$ on $\pi(t')$ on

$$\frac{1 + d_{t'-1}(v)}{n - (t' - 1)} \geq \frac{d_{t'-1}(v)}{n} > \frac{c \cdot n \log n}{tn} = \frac{c \cdot \log n}{t}.$$

Siis todennäköisyys sille, että yli $((c \cdot n \log n)/t)$-asteinen solmu v ei poistunut graafista ensimmäisten t solmun käsittelyn aikana on korkeintaan

$$\left(1 - \frac{c \cdot \log n}{t}\right)^t \leq e^{-c \cdot \log n} = n^{-c}.$$

Väite seuraa todennäköisyyden subadditiivisuudesta. □

Kun algoritmissa 6 asetetaan $t_i = \mathcal{O}\left(\frac{n \log n}{\Delta_G / 2^i}\right)$ niin lemman 4.4 mukaan vaiheessa j graafin G, josta on poistettu graafien $G_1, \ldots, G_{j-1}$ solmut, maksimiaste on korkeintaan

$$\mathcal{O}\left(\frac{n \log n}{(2^j - 1)\frac{n \log n}{\Delta_G}}\right) = \mathcal{O}\left(\frac{\Delta_G}{2^j}\right).$$

Maksimiaste siis puolittuu joka vaiheessa. Tarvitaan korkeintaan $\mathcal{O}(\log \Delta)$ vaihetta, että jäljellä on viimeistä lohkoa pienempi määrä solmuja. Loput graafista käsitellään

kutsumalla algoritmia 7 vielä kerran.

Nyt voimme viimein todistaa luvun päätuloksen.

Lause 4.5. *Olkoon $G = (V, E)$ graafi, jonka solmujen määrä on n ja maksimiaste Δ, ja olkoon $\pi\colon [1, n] \to V$ solmujen satunnainen järjestys. Graafissa G voidaan suorittaa* `Pivot` *järjestyksen π suhteen $\mathcal{O}(\log\log n \cdot \log \Delta)$ MPC-kierroksessa suurella todennäköisyydellä.*

Todistus. Algoritmin 6 suorituksessa on $\mathcal{O}(\log \Delta)$ vaihetta, joissa kussakin suoritetaan algoritmi 7 kerran. Prefiksigraafissa G_i on joka vaiheessa $\mathcal{O}(\frac{n \log n}{\Delta/2^i}) = n'$ solmua ja lemman 4.2 mukaan sen maksimiaste on $\mathcal{O}(\log n) = \Delta'$. Kun sijoitetaan nämä algoritmin 7 suoritusaikaan $\mathcal{O}(\log\log n' + \log \Delta')$, niin saadaan $\mathcal{O}\left(\log\log\left(\frac{n \log n}{\Delta/2^i}\right) + \log\log n\right) = \mathcal{O}(\log\log n)$. Graafissa on $\mathcal{O}(\log \Delta)$ vaiheen jälkeen jäljellä enää pieni määrä solmuja. Ne voidaan käsitellä $\mathcal{O}(\log\log n)$ MPC-kierroksessa algoritmilla 7. Yhteensä MPC-kierroksia tarvitaan siis $\mathcal{O}(\log\log n \cdot \log \Delta)$.

Epäonnistumisen todennäköisyys kussakin vaiheessa on korkeintaan $\frac{1}{n^c}$ halutulla vakiolla c. Kun todennäköisyydet summataan $\mathcal{O}(\log \Delta)$ vaiheen yli, epäonnistumisen todennäköisyys on yhä pieni. Siis algoritmin suoritusaika on $\mathcal{O}(\log\log n \cdot \log \Delta)$ kierrosta suurella todennäköisyydellä.

□

Huomautettakoon, että algoritmissa 6 klusteroinnin kokoaminen voidaan korvata pelkän riippumattoman joukon keräämisellä. Siten saadaan maksimaaliselle riippumattomalle joukolle MPC-algoritmi, joka suoriutuu samassa ajassa.

Yllä esitetty algoritmi on yhä pohjimmiltaan `Pivot`, joten se on odotusarvoisesti 3-approksimaatio. Olkoon G pienen metsäisyyden λ graafi. Kun lauseessa 3.8 asetetaan $\alpha = 3$, niin voidaan ensin poistaa graafista solmut, joiden aste on yli $\frac{8 \cdot 3}{3-1} \cdot \lambda = 12\lambda$. Lopulle graafille, jossa $\Delta = 12\lambda$, voidaan tehdä tässä luvussa esitetty nopeutettu `Pivot`-muunnelma $\mathcal{O}(\log\log n \cdot \log \lambda)$ MPC-kierroksessa.

Viitteet

[1] F. Chierichetti, N. Dalvi ja R. Kumar, "Correlation Clustering in MapReduce", *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, s. 641–650.

[2] X. Pan, D. Papailiopoulos, S. Oymak, B. Recht, K. Ramchandran ja M. I. Jordan, "Parallel Correlation Clustering on Big Graphs", *Advances in Neural Information Processing Systems*, 2015, s. 82–90.

[3] C. Nash-Williams, "Edge-Disjoint Spanning Trees of Finite Graphs", *Journal of the London Mathematical Society*, vsk. 1, nro 1, s. 445–450, 1961.

[4] W. T. Tutte, "On the Problem of Decomposing a Graph into n Connected Factors", *Journal of the London Mathematical Society*, vsk. 1, nro 1, s. 221–230, 1961.

[5] B. Chen, M. Matsumoto, J. Wang, Z. Zhang ja J. Zhang, "A Short Proof of Nash-Williams' Theorem for the Arboricity of a Graph", *Graphs and Combinatorics*, vsk. 10, nro 1, s. 27–28, 1994.

[6] M. Fischer ja A. Noever, "Tight Analysis of Parallel Randomized Greedy MIS", *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2018, s. 2152–2160.

[7] N. Calkin ja A. Frieze, "Probabilistic Analysis of a Parallel Algorithm for Finding Maximal Independent Sets", *Random Structures & Algorithms*, vsk. 1, nro 1, s. 39–50, 1990.

[8] G. E. Blelloch, J. T. Fineman ja J. Shun, "Greedy Sequential Maximal Independent Set and Matching are Parallel on Average", *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, 2012, s. 308–317.

[9] N. Linial, "Locality in Distributed Graph Algorithms", *SIAM Journal on Computing*, vsk. 21, nro 1, s. 193–201, 1992.

[10] H. Karloff, S. Suri ja S. Vassilvitskii, "A Model of Computation for MapReduce", *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 2010, s. 938–948.

[11] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein ja Z. Svitkina, "On Distributing Symmetric Streaming Computations", *ACM Transactions on Algorithms (TALG)*, vsk. 6, nro 4, s. 1–19, 2010.

[12] M. T. Goodrich, N. Sitchinava ja Q. Zhang, "Sorting, Searching, and Simulation in the MapReduce Framework", *International Symposium on Algorithms and Computation*. Springer, 2011, s. 374–383.

[13] P. Beame, P. Koutris ja D. Suciu, "Communication Steps for Parallel Query Processing", *Journal of the ACM (JACM)*, vsk. 64, nro 6, s. 1–58, 2017.

[14] ——, "Skew in Parallel Query Processing", *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2014, s. 212–223.

[15] A. Andoni, A. Nikolov, K. Onak ja G. Yaroslavtsev, "Parallel Algorithms for Geometric Graph Problems", *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, 2014, s. 574–583.

[16] S. Lattanzi, B. Moseley, S. Suri ja S. Vassilvitskii, "Filtering: A Method for Solving Graph Problems in MapReduce", *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, 2011, s. 85–94.

[17] A. Czumaj, J. Łacki, A. Madry, S. Mitrovic, K. Onak ja P. Sankowski, "Round Compression for Parallel Matching Algorithms", *SIAM Journal on Computing*, vsk. 49, nro 5, s. STOC18-1–STOC18-44, 2020.

[18] C. Lenzen ja R. Wattenhofer, "Brief announcement: Exponential Speed-Up of Local Algorithms Using Non-Local Communication", *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of Distributed Computing (PODC)*, 2010, s. 295–296.

[19] N. Bansal, A. Blum ja S. Chawla, "Correlation Clustering", *Machine Learning*, vsk. 56, nro 1–3, s. 89–113, 2004.

[20] N. Ailon, M. Charikar ja A. Newman, "Aggregating Inconsistent Information: Ranking and Clustering", *Journal of the ACM (JACM)*, vsk. 55, nro 5, s. 1–27, 2008.

[21] M. Mäkelä, *Matemaattinen optimointi I*, 2019.

[22] M. Luby, "A Simple Parallel Algorithm for the Maximal Independent Set Problem", *SIAM Journal on Computing*, vsk. 15, nro 4, s. 1036–1053, 1986.

[23] M. Cambus, D. Choo, H. Miikonen ja J. Uitto, "Massively Parallel Correlation Clustering in Bounded Arboricity Graphs", *35th International Symposium on Distributed Computing*, 2021.