Joonas Muhonen

Suurten graafien visualisointi

Tietotekniikan kandidaatintutkielma

30. huhtikuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joonas Muhonen

Yhteystiedot: joonas.a.muhonen@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: Suurten graafien visualisointi

Title in English: Visualization of large graphs

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 20+0

Tiivistelmä: Graafit ovat yleisesti tietotekniikassa esiintyvä tietorakenne joita voidaan käyttää visualisoimaan useita reaalimaailman ilmiöitä kuten tietoverkkoja ja riippuvuussuhteita ohjelmiston moduulien välillä. Graafin koon kasvaessa sen visualisointi tuottaa vaikeuksia yleisesti käytössä oleville menetelmillä niiden aikavaativuuden vuoksi. Kirjallisuuskatsauksen tarkoituksena on selvittää mitä menetelmiä on olemassa nopeuttaa graafien piirtämistä.

Avainsanat: Graafit, Graafinpiirtoalgoritmit,GPU

Abstract: Graphs are commonly used data structure in information technology which can be used to visualize many real world phenomenons such as computer networks and relations between components in modules of computer programs. As graph size increased visualizing it becomes difficult with commonly used methods due to time complexity of said methods. In this literature review the purpose is to find out what methods exist for speeding up graph drawing.

Keywords: Graphs, Graph drawing algorithms, GPU

Kuviot

Sisältö

1 Johdanto

Graafinpiirtoalgoritmit ovat tärkeä komponentti visualisoitaessa dataa joka noudattaa graafin rakennetta kuten ohjelmiston komponenttien väliset riippuvuudet. Ohjelmistojen rakenne on yleensä riittävän yksinkertainen että hidaskin algoritmi pystyy tuottamaan kuvan riittävän nopeasti mutta visualisoitaessa suurempia tietomääriä kuten sosiograafeja (Fagnan, Zaïane ja Goebel 2012) joissa solmujen määrä voi olla satojatuhansia tai enemmän voi perinteisten algoritmien suoritusaika kasvaa liian hitaaksi jotta käyttäjä ei joutuisi odottamaan graafin visualisaation valmistumista. Tässä tutkimuksessa keskitytään graafeihin joiden solmujen määrä on vähintään 10000 ja mikäli algoritmin suoritusaika 100000 solmulla on tiedossa käytetään sitä vertaamaan algoritmien nopeutta.

Suurten graafien piirtäminen tuottaa perinteisille algoritmeille vaikeuksia niiden laskennallisen vaativuuden vuoksi, esimerkiksi Fruchterman-Reingold algoritmin aikavaativuus on kertaluokkaa $\mathcal{O}\left(n^2\right)$. Algoritmien aikavaativuuden madaltaminen ei vaikuta todennäköiseltä ongelman luonteen vuoksi koska solmujen on hyljittävä toisiaan päällekkäisyyksien välttämiseksi, on kuitenkin olemassa keinoja joiden ansiosta algoritmi voi hylkiä vain lähellä olevia solmuja.

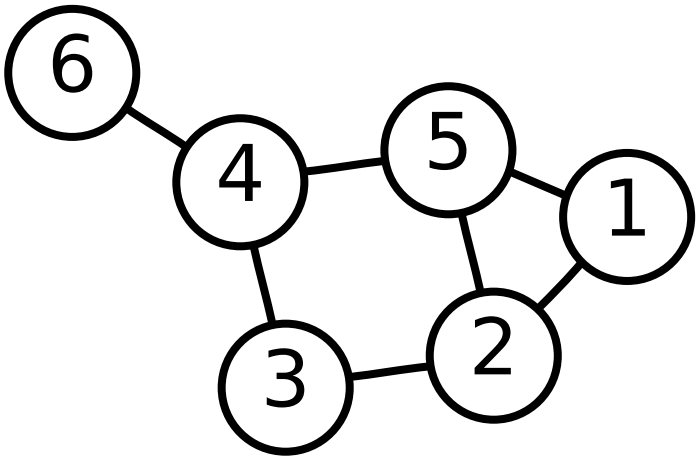
Koska graafin piirtäminen on visuaalinen ongelma on luontevaa pohtia miten algoritmit saadaan parhaiten muunnettua toimimaan näytönohjaimilla joiden yksisäikeinen suorituskyky on erittäin heikko mutta jotka pystyvät suorittamaan suuren määrän laskentaa rinnakkain. Siirtämällä graafinpiirtoalgoritmi näytönohjaimelle myös vältetään graafin solmujen sijaintitiedon siirtäminen prosessorilta näytönohjaimelta mikä osaltaan myös parantaa suorituskykyä. Löydettyjä algoritmeja voidaan myös hyödyntää nopeuttamaan laskentaa moderneilla prosessoreilla joissa on käytettävissä useita ytimiä laskentaa varten.

Kirjallisuuskatsauksessa tarkoituksena on tarkastella mitä algoritmeja on olemassa graafien visuaaliseen esittämiseen erityisesti keskittyen niiden suorituskykyyn. Suurten graafien piirtämisessä suorituskyky on saavutettavissa nykyisillä laitteistoilla algoritmien komponenttien suorittamisella yhtäaikaisesti ja graafin osiointiin perustuvilla ratkaisuilla. Suorituskyvyn kasvattamisella on myös mahdollista saada graafinpiirrosta riittävän nopeaa jotta graafia

ei tarvitse laskea etukäteen ja jopa muutokset graafin rakenteeseen ovat mahdollisia kun graafi on jo katsottavissa (Frishman ja Tal 2008) kuitenkin säilyttäen algoritmin muodostaman graafin visualisaation laatu.

Graafin kuvan piirtämisessä on myös tärkeää varmistaa graafista saatavan kuvan laatu. Kuten aiemmin todettiin päällekkäisiä solmuja on vältettävä mutta on myös tärkeää että solmut sijaitsevat järkevästi siten, että toisiinsa liittyvät solmut ovat kuvassa lähellä toisiaan. On myös mahdollista ottaa huomioon graafin kuvassa olevien toisiaan leikkaavien kaarien määrä (Purchase, Cohen ja James 1995) ja onkin olemassa keinoja antaa graafin laadulle numeerisia arvoja (Purchase 2002). Tässä kirjallisuuskatsauksessa oletetaan että tarkastellut algoritmit pystyvät tuottamaan riittävän laadukkaan kuvan graafista ja keskitytään enemmän algoritmien suorituskykyyn.

2 Taustatiedot



Kuvio 1. Esimerkki graafista.

Koska graafeihin liittyvillä termeillä on myös muita merkityksiä sisällytetään kirjallisuuskatsaukseen lyhyt lista aiheeseen liittyviä termejä joita kirjallisuuskatsauksessa käytetään. Taustatiedot sisältävät myös lyhyen esittelyn tutkittaviin keinoihin nopeuttaa graafien piirtämistä.

2.1 Aiheeseen liittyvät termit

Graafi on tietorakenne jossa on joukko solmuja joita yhdistää joukko kaaria, matemaattisesti $G = (V,E)$ missä V on joukko solmuja ja $E \subset \{(a,b) : a,b \in V\}$ joukko kaaria. On myös mahdollista valita kaarijoukkojoukko E siten, että kahta solmua voi yhdistää useampi kaari mistä voi olla hyötyä yhdistettäessä osagraafeja joiden välillä on useampia kaaria, laskennallisesti tämä voidaan toteuttaa myös antamalla kaarille painoarvo mutta tämä ei aina ole haluttua. Tarkempaa määritelmää varten katso esimerkiksi ‘Handbook of graph theory‘ (Gross ja Yellen 2003).

Graafinpiirtoalgoritmi on algoritmi, joka liittää jokaiseen graafin solmuun sijainnin ja sallii siten graafin visuaalisen esittämisen. Yksikertaisimmillaan valitaan satunnainen sijainti jokaiselle solmulle graafissa, mutta tämä ei ole erityisen hyvä algoritmi graafien piirtämiseen. Yksi ehkä tunnetuimmista graafinpiirtoalgoritmeista on Fruchterman-Reingold algo-

ritmi (Fruchterman ja Reingold 1991), jatkossa lyhenne FR viittaa Fruchterman-Reingold algoritmiin. Fruchterman-Reingold algoritmi toimii pohjana usealle muulle algoritmille kuten Parallel Fruchterman-Reingold algoritmi (Gajdoš ym. 2016) jotka pyrkivät parantamaan algoritmin toimintaa tai muodostuneita visualisaatiota.

Tyypillinen voimasimulaatiopohjainen graafinpiirtoalgoritmi toimii seuraavasti:

```
t = max
DO:
    FOR e in E:
        computeAttractiveForce(e)
    end
    FOR v in V:
        FOR u in V:
            computeRepulsiveForce(v,u)
        end
    end
    FOR v in V:
        updatePosition(v)
    end
    t = newT
WHILE t > limit
```

Lämpötila on graafinpiirtoalgoritmin toiminnasta riippuva keino tarkastella kuinka lähellä graafin kuva on graafin solmujen välisten voimien lokaalia minimiä. Lämpötilan käsitettä käytetään useimmiten menetelmissä joissa graafinpiirtoalgoritmi simuloi jotakin reaalimaailman vastinetta kuten yhteen kytkettyjä jousia tai sähkövarauksia. Lämpötilaksi valitaan usein joko solmujen liikkumien pituuksien summa iteraatiossa tai solmujen välisten voimien summa iteraation loputtua. Jos graafin piirtämiseen kuluvien iteraatioiden määrä ei ole tiedettävissä voi graafin lämpötilan putoamista jonkin tietyn rajan alle pitää tietona siitä että algoritmi ei enää juurikaan muuta muodostuvaa kuvaa vaikka iteraatioita jatkettaisiin. Graafin kuvan lämpötilaa voidaan myös käyttää rajoittamaan suuret muutokset kuvassa varhaisiin iteraatoihin jolloin myöhemmät iteraatiot toimivat hienosäätönä kuvaan kuten Fruchterman

ja Reingold (1991) esittelemä menetelmä toimii.

2.2 Graafin solmujoukon jakaminen osiin

Graafit voidaan jakaa osiin siten, että jokaiseen osioon voidaan soveltaa eri optimisaatioita kuten rinnakkaislaskentaa tai yksinkertaisempia algoritmeja joiden suorituskyky on parempi kuin tarkemmilla graafinpiirtoalgoritmeilla. Osioiden muodostamiseen on olemassa varsin rajattu määrä menetelmiä joten niihin ei ole tarve keskittyä erityisen tarkasti ja useimmat artikkelit ohittavat graafin jakamisen osiin tekniset yksityiskohdat. Osiointialgoritmeja ovat muun muassa kaksikomponenttiosiointi (Tarjan 1972), kaariosiointi (Girvan ja Newman 2002) ja virtaosiointi (Wu ja Huberman 2004). Kun osiot on muodostettu ja jokainen osio on käsitelty osiot voidaan yhdistää esimerkiksi FR-algoritmilla muodostaen kokonaiskuvan graafista.

Graafi voidaan myös jakaa osiin poistaen graafista solmuja, käsittelemällä tämä osagraafi jollakin graafinpiirtoalgoritmilla ja sen jälkeen lisäämällä poistettuja solmuja takaisin graafiin jonka jälkeen graafinpiirtoalgoritmia jatketaan käyttäen aiemmin saatuja sijainteja solmujen alkusijainteina. Nämä niin kutsutut monitasomenetelmät pyrkivät parantamaan graafinpiirtoalgoritmin suorituskykyä vähentämällä tarvittavien iteraatioiden määrää pienemmillä alkugraafeilla ja valmiiksi lähellä matalaa lämpötilaa olevilla osagraafeilla myöhemmissä iteraatioissa.

2.3 Graafinpiirtoalueen jakaminen osiin

Graafin solmut eivät saa sijaita päällekkäin jotta graafista piirretty kuva on helposti tarkasteltavissa. Graafinpiirtoalgoritmit saavuttavat tämän muodostamalla hylkiviä voimia graafien solmujen välillä mutta tämä on aikavaativuudeltaan $\mathcal{O}\left(n^2\right)$. Koska kaukana sijaitsevien solmujen voimat ovat erittäin pieniä ei niitä tarvitse tarkastella, tämän toteuttaminen ei kuitenkaan ole yksinkertaista. Algoritmit eivät tiedä mitkä solmuista sijaitsevat lähellä toisiaan mutta on kuitenkin mahdollista tallentaa solmujen sijainnit tietorakenteeseen joka sallii käsitellä vain osaa solmuista jotka tietorakenteen mukaan sijaitsevat lähellä tarkasteltavaa solmua.

Piirtoalueen jakamisella osiin on saavutettavissa suuriakin suorituskykyparannuksia mutta kuten Fagnan, Zaïane ja Goebel (2012) huomauttavat paras visualisaatio saavutetaan kun graafin annetaan kasvaa tarpeen mukaan. Jos piirtoalueen jako osiin ei salli piirtoalueen kasvamista graafin piirron edetessä voi olla, että graafi täytyy piirtää uudelleen suuremmalla piirtoalueella riittävän laadukkaan kuvan saavuttamiseksi. Inhimillistä virhearviota ei voi ottaa huomioon $\mathcal{O}$ kertaluokassa. Piirtoalueen jakamisen osiin täytyy myös tapahtua riittävän nopeasti jotta se ei ole merkittävä osa graafin piirtoon kuluvasta ajasta tai muuten saavutettavat suorituskykyhyödyt voivat jäädä pieniksi.

2.4 Rinnakkaistus

Mikäli jokin algoritmin osa voidaan suorittaa rinnakkain voidaan tällä saavuttaa joissakin tilanteissa merkittäviä nopeutuksia. FR-algoritmissa solmujen hylkivät voimat voidaan laskea rinnakkain ja Jeowicz ym. (2013) antavat esimerkin saavutettavista tuloksista. Algoritmi jakaa solmujoukon siten, että kukin säie laskee hylkivät voimat vain osalle solmuja. Vaikka tämä ei suoraan vaikuta algoritmin aikavaativuuteen $\mathcal{O}\left(\frac{n^2}{m}\right)$, missä m on säikeiden määrä havaitaan että säikeiden määrän lähestyessä solmujen määrää algoritmin aikavaativuus lähestyy kertaluokkaa $\mathcal{O}(n)$. Käyttämällä hyväksi sekä prosessorin laskentatehoa, että näytönohjainta Jeowicz ym. (2013) saavuttavat nopeutuksia jotka vaihtelevat samasta nopeudesta jopa yli 100 kertaisiin nopeutuksiin verrattuna laskentaan pelkästään prosessorilla. Rinnakkaislaskentaa hyödyntävät keinot nopeuttaa graafin piirtämistä eroavat algoritmisista parannuksista rakenteellisella tavalla ja nopeutus on riippuvainen käytetystä laitteistosta. Rinnakkaislaskenta on kuitenkin tärkeä ottaa huomioon algoritmeja tarkastellessa koska kaikkia algoritmeja ei voida rinnakkaistaa ja siten ne eivät voi hyödyntää kaikkea käytettävissä olevaa laskentatehoa graafin piirtämiseen.

3 Graafin solmujoukon jakaminen osiin

Graafin osiin jakamisella voidaan saavuttaa merkittäviä suoritusaikaparannuksia edellyttäen että osiin jakaminen voidaan toteuttaa ilman että osiointiin kuluva aika on yhtä pitkä tai pidempi kuin graafin piirtämiseen kuluva aika. Seuraavat menetelmät hyödyntävät graafin osiointia graafin piirtämisessä.

3.1 Osiointiin perustuva menetelmä

Fagnan, Zaïane ja Goebel (2012) esittelevät menetelmät COMB (Community Boundary) ja COMC (Community Circles) jotka toimivat osiomalla graafi. Menetelmät muodostavat osiot käyttämällä osiointi algoritmia jonka Clauset, Newman ja Moore (2004) ovat esitelleet. COMB-algoritmi tunnistaa osioista merkitsevät solmut joiden sijainteihin sovelletaan Fagnan ryhmän (Fagnan, Zaïane ja Goebel 2012) esittelemää muokattua FR-algoritmia jossa merkitsevien solmujen kooksi käytetään arviota kuinka suuren tilan osagraafi vaatii. Kun merkitsevät solmut on käsitelty algoritmi sijoittaa loput solmuista satunnaisesti niiden merkitsevän solmun ympärille. Saatu kuva graafista käsitellään uudelleen muokatulla FR-algoritmilla kuitenkin ilman että solmut poistuvat merkitsevän solmun ympäristöstä. COMC-algoritmi sijoittaa osioiden solmut ympyrän kehällä minimoidakseen osion sisäisten kaarien pituudet ja käyttää ympyröiden keskipistettä muokatun FR-algoritmin käyttämän graafin solmuina.

Fagnan, Zaïane ja Goebel (2012) tarkastelevat COMC menetelmän nopeutta ja heidän tuloksiensa mukaan COMC on jopa 34 kertaa nopeampi kuin FR-algoritmi ja COMB noin 4% nopeampi. Tuloksista on havaittavissa että COMB-algoritmi käyttää merkittävästi aikaa graafin viimeistelyyn muokatulla FR-algoritmilla, COMC ohittaa tämän vaiheen kokonaan ja siten saavuttaa merkittävästi lyhyemmän suoritusajan. Fagnan, Zaïane ja Goebel (2012) mainitsevat menetelmien eduksi myös mahdollisuuden käyttää erikokoisia solmuja sallien reaalimaailmassa esiintyvien graafien visualisointia jossa solmun koolla ilmoitetaan jotain arvoa solmusta. Toisena etuna Fagnan, Zaïane ja Goebel (2012) mainitsevat etteivät kyseiset algoritmit rajoita graafin visualisaatiota tiettyihin reunoihin. FR-algoritmi olettaa että graafin

kuvan tarvitsemat reunat ovat ennestään tiedossa mutta poistamalla tämä oletus ja sallimalla graafin kuvan täyttää mielivaltainen tila voidaan saada laadukkaampi visualisaatio graafista.

3.2 Monitasomenetelmät

Monitasomenetelmät luovat karkean graafin alkuperäisestä graafista joka sisältää vain osan solmuista graafista, kun tämä graafi on saatu käsiteltyä graafinpiirtoalgoritmilla lisätään osagraafiin solmuja alkuperäisestä graafista ja käytetään ensimmäisessä iteraatiossa saatua visualisaatiota seuraavan iteraation ensimmäisen askeleen alustukseen satunnaisten sijoitusten sijaan. Matemaattisesti olkoon G_0 annettu graafi, luodaan graafit $G_1, \ldots, G_k$ siten että jokaisessa graafissa on vähemmän solmuja kuin edeltävässä. Pienin graafi tasolla k piirretään halutulla menetelmällä josta saatua kuvaa voidaan hyödyntää graafin G_{k-1} alkusijoituksena.

Hachul ja Jünger (2004) esittelevät FM^3 (Fast Multipole Multilevel Method) menetelmän joka hyödyntää monitasomenetelmien perusideaa, graafin jakamista osiin ja piirtoalueen jakamista osiin nelipuun avulla, menetelmän aikavaativuus on luokkaa $\mathcal{O}(|V|\log|V|+|E|)$. Hachul ja Jünger (2005) vertaavat menetelmää toisiin piirtoalgoritmeihin ja kokeellisissa testeissä FM^3 on jopa 97 kertaa nopeampi kuin vertailtava FR pohjainen GVA menetelmä (Grid Variant Algorithm) joka pohjautuu FR menetelmän lisäksi piirtoalueen jakamiseen ruudukoksi jossa solmujen hylkivät voimat lasketaan vain solmua ympäröivissä ruuduissa olevista solmuista (Fruchterman ja Reingold 1991).

Lipp, Wolff ja Zink (2016) myös viittaavat Gronemannin menetelmään joka pohjautuu FM^3 algoritmiin, heidän testeissään Gronemannin kehittämä FastMultipole (OGDF: Open Graph Drawing Framework, https://ogdf.uos.de/) algoritmi on nopein testatuista algoritmeista ja jopa 50 kertaa nopeampi kuin Lippin ryhmän (Lipp, Wolff ja Zink 2016) esittelemä WSPD menetelmä ja siten lähes 1000 kertaa nopeampi kuin FR menetelmä. FastMultipole on optimoitu hydyntämään prosessorin SIMD eli yksi käsky, monta dataa käskyjä (engl. single instruction, multiple data) ja on siten verrattavissa muiden menetelmien näytönohjainlaskentaa pohjautuviin versioihin.

4 Graafinpiirtoalueen jakaminen osiin

Graafinpiirtoalgoritmeissa suurimman suorituskyvyn vie solmujen sijoittaminen siten, että yksikään solmu ei sijaitse toisen päällä. Seuraavat menetelmät tarjoavat eri keinoja tallentaa tieto lähellä olevista solmuista siten, että solmujen hylkivät voimat tarvitsee laskea vain lähellä olvista solmuista.

4.1 Puurakenteet

Yksinkertaisimmat ja parhaiten tutkitut menetelmät tallentaa sijaintitieto rakenteeseen josta lähellä sijaitsevat kappaleet voidaan hakea nopeasti ovat erilaiset puurakenteet. Puurakenteita käytetään useimmiten parantamaan niin kutsuttujen lähimpien naapurien ongelmien suorituskykyä ja on käytetty jo pitkään laskemaan voimia pistemäisten kappaleiden välillä kuten Barnes ja Hut (1986) esittelemä partikkelisimulaatio jonka esittelemä menetelmä laskea pisteiden välisiä voimia toimii pohjana useimmille seuraavista menetelmistä.

Quigley ja Eades (2000) esittelevät FADE-algoritmin graafien piirtämiseen, algoritmi toimii rakentamalla geometrinen osiointi algoritmin jokaisessa iteraatiossa solmujen sijainneista ja laskemalla voimat rakenteessa lähellä olevista solmuista, jos tietorakenteen solmun leveys jaettuna etäisyydellä tarkasteltavasta solmusta osion keskipisteeseen alittaa annetun rajan jaetaan osio osa-osioihin ja toteutetaan tarkastelu uudestaan, jos etäisyys on enemmän kuin annettu raja lasketaan voima osiosta kokonaisuutena. Algoritmin laskennallinen vaativuus on luokkaa $\mathcal{O}(n \log n)$ ja kokeellisissa tuloksissa algoritmin nopeus on vastaavasti nopeampi kuin samat voimafunktiot ilman geometrista osiointia.

Verratakseen kehittämäänsä FR+WSPD algoritmia Lipp, Wolff ja Zink (2016) ovat toteuttaneet nelipuuhun pohjautuvan FR pohjaisen graafinpiirtoalgoritmin. Lipp, Wolff ja Zink (2016) toteuttavat kokeellisia testejä joissa FR+Quad algoritmi on noin 150 kertaa nopeampi kuin FR-algoritmi kun graafissa on 100000 solmua. Algoritmin nopeusetuna on myös hyvä suorituskyky pienillä graafeilla koska kehittyneemmät algoritmit useimmiten toimivat hitaasti pienillä graafeilla.

ForceAtlas2-algoritmi on Jacomy ym. (2014) kehittämä graafinpiirtoalgoritmi joka on kehitetty Gephi-ohjelmistoa varten. Algoritmi käyttää optimisaatiota jonka Barnes ja Hut (1986) ovat esitelleet solmujen hylkivien voimien laskemiseen ja muokkaa voimafunktiota graafin kuvan konvergenssin parantamiseksi. Algoritmin nopeus on verrattavissa toisiin $\mathcal{O}(n \log n)$ algoritmeihin. Brinkmann, Rietveld ja Takes (2017) ovat toteuttaneet ForceAtlas2-algoritmin hyödyntäen näytönohjainlaskentaa saavuttaen noin 50-kertaisen nopeutuksen verrattuna prosessorilaskentaan ja on siten verrattavissa muiden algoritmien toteutuksiin hyödyntäen rinnakkaislaskentaa.

4.2 Tilantäyttökäyrä

Tilantäyttökäyrät ovat käyriä joiden avulla useampiulotteisiin avaruuden pisteisiin voidaan yhdistää yksiulotteinen indeksi tai päinvastoin yksiulotteiseen indeksiin voidaan yhdistää useampiulotteisen avaruuden piste. Graafin solmut voidaan sijoittaa tilantäyttökäyrälle muodostamaan kuva graafista tai voidaan tilantäyttökäyrää käyttää estimoimaan mitkä graafin solmuista ovat lähellä toisiaan. Muodostamalla tilantäyttökäyrä graafinpiirtoalgoritmin jokaisessa iteraatiossa ja käyttämällä solmujen indeksiä arviomaan mitkä solmuista ovat lähellä tarkasteltavaa solmua voidaan arvioida lähellä olevia solmua. Pisteiden indeksien läheisyys ei välttämättä takaa että pisteet sijaitsevat lähellä toisiaan avaruudessa mutta useimmiten arvio on riittävä ja arviota voidaan parantaa ottamalla enemmän pisteitä jotka sijaitsevat käyrällä lähellä toisiaan huomioon.

Muelder ja Ma (2008) esittävät menetelmän graafien piirtämiseen käyttäen tilantäyttökäyrää solmujen sijoittamiseen graafin kuvassa. Menetelmässä graafi osioidaan käyttäen osiointialgoritmia jonka Huang ja Nguyen (2007) ovat esitelleet, tämän jälkeen solmut sijoitetaan tilantäyttökäyrälle sijoittaen samassa osiossa olevat solmut lähelle toisiaan ja jättämällä osioiden välille tyhjää tilaa. Menetelmä jonka Muelder ja Ma (2008) esittelevät eroaa muista menetelmistä merkittävästi, graafin kuva valmistuu yhdessä iteraatiossa ja on siten jopa 100 kertaa nopeampi kuin FR-algoritmi. Menetelmä ei myöskään simuloi kaarien välisiä voimia ja on riippuvainen osiointialgoritmin kykyyn luoda graafille sopiva osiointi.

Ježowicz ym. (2014) esittelevät FR-algoritmiin pohjautuvan menetelmän käyttäen tilantäyt-

tökäyrää lähellä olevien solmujen arvioimiseen. Menetelmä siirtää graafin sijaintia tilantäyttökäyrällä jokaisessa iteraatiossa satunnaisesti, laskee graafin solmuille jokaisessa iteraatiossa tilantäyttökäyrän indeksin ja järjestää solmujoukon käyttäen QuickSort-algoritmia indekseille, tämän jälkeen solmujen väliset hylkivät voimat lasketaan vain solmuista jotka sijaitsevat lähellä toisiaan tilantäyttökäyrällä. Ježowicz ym. (2014) kokeellisten testien perusteella algoritmi on jopa 195 kertaa nopeampi kuin FR-algoritmi.

Gajdoš ym. (2016) toteuttavat Ježowicz ym. (2014) esittelemän algoritmin pienillä muutoksilla käyttäen näytönohjainlaskentaa hyödyksi. Gajdoš ym. (2016) siirtävät graafin solmuja jokaisen iteraation alussa välttääkseen graafin kuvaan muutoin muodostuvaa ruudukkorakennetta, mutta säilyttävät graafin sijainnin tilantäyttökäyrällä. Gajdoš ym. (2016) esittelemä menetelmä saavuttaa näytönohjaimen rinnakkaislaskentaa hyödyntäen noin 4-kertainen nopeutus prosessorilaskentaan verrattuna ja on siten samassa kertaluokassa kuin Gronemannin esittelemä FM^3 pohjainen algoritmi.

4.3 Hyvin separoituva parien erotus

Hyvin separoituva parien ositus on menetelmä jakaa pistejoukko osiin siten, että jokainen piste kuuluu p-säteiseen ympyrään ympyrään ja ympyröiden välinen etäisyys on vähintään $s \cdot p$, missä s on osituskerroin. Jatkossa hyvin separoituvaan parien ositukseen viitataan lyhenteellä WSPD (engl. well separated pair decomposition) Lipp, Wolff ja Zink (2016) esittelevät FR+WSPD menetelmän joka nopeuttaa FR-algoritmin toimintaa muodostamalla WSPD jaettua puurakennetta hyödyntämällä. Jaettu puurakenne muodostetaan jakamalla pistejoukon pisin sivu kahteen osaan rekursiivisesti kunnes jokaisessa osajoukossa on vain yksi solmu. FR+WSPD käyttää jaetun puurakenteen muodostamiseen solmujen sijainneista algoritmia jonka Callahan ja Kosaraju (1995) ovat esitelleet jolloin WSPD:n muodostaminen on aikavaativuudeltaan $\mathcal{O}(n \log n)$.

Kun WSPD solmujen sijainneista on muodostettu algoritmi laskee solmuun kohdistuvat hylkivät voiman solmuista jotka sijaitsevat samassa WSPD:n ympyrässä kuin solmu ja ottaa huomioon muista WSPD:n ympyröistä ympyrän keskipisteen yhtenä solmuna. Siten algoritmi onnistuu vähentämään huomattavasti solmuja jotka täytyy ottaa huomioon hylkiviä voi-

mia laskettaessa. Lipp, Wolff ja Zink (2016) kokeellisissa testeissä FR+WSPD on noin 2 kertaa nopeampi kuin FM^3 kun graafi sisältää 100000 solmua ja siten noin 195 kertaa nopeampi kuin FR-algoritmi. Lipp, Wolff ja Zink (2016) myös tarkastelevat vertaamiensa algoritmien laatua ja FR+WSPD sijoittuu verrattujen algoritmien keskellä saavuttaen useimpiin muihin algoritmeihin verrattavissa olevan graafin kuvan laadun.

5 Yhteenveto

Kirjallisuuskatsauksessa löydettiin useita menetelmiä jotka toimivat nopeammin kuin FR-algoritmi suurten graafien visualisointiin. Suurin osa algoritmeista ovat aikavaativuudeltaan $\mathcal{O}(n \log n)$ ja menetelmät jotka eivät hyödynnä rinnakkaislaskentaa ovat siten nopeudeltaan hyvin lähellä toisiaan. Vertailluista menetelmistä nopeimpia ovat rinnakkaislaskentaa hyödyntävät nopeutukset muihin algoritmeihin kuten OGDF kirjaston FastMultipole joka pohjautuu FM^3 algoritmiin. Rinnakkaislaskentaa hyödyntävissä menetelmissä suurimmat nopeuserot tulevat mitä luultavammin eroista laskennan rinnakkaistuksen toteutuksissa. Kaikista algoritmeista ei löytynyt rinnakkaislaskentaa hyödyntävää versiota ja jatkotutkimuksessa kannattaa muiden menetelmien löytämisen lisäksi tarkastella onko jo löydettyjä algoritmeja mahdollista nopeuttaa hyödyntäen rinnakkaislaskentaa.

Löydetyistä menetelmistä selvästi poikkeavin on menetelmä jonka Muelder ja Ma (2008) esittelevät tilantäyttökäyrän pohjautuen ja jonka heikkoutena on kuvan laatu johtuen solmujen satunnaisesta sijoituksesta tilantäyttökäyrälle. Algoritmien muodostamien kuvien laatua kannattaa tarkastella enemmän, Lipp, Wolff ja Zink (2016) antavat joitakin arvoja FR, FR+Quad, GVA, FM^3, FR+WSPD ja FastMultipole algoritmeille, mutta lähempi tarkastelu on tarpeen.

Lähteet

Barnes, Josh, ja Piet Hut. 1986. “A hierarchical O (N log N) force-calculation algorithm”. *nature* 324 (6096): 446–449. doi:10.1038/324446a0.

Brinkmann, G. G., K. F. D. Rietveld ja F. W. Takes. 2017. “Exploiting GPUs for Fast Force-Directed Visualization of Large-Scale Networks”. Teoksessa *2017 46th International Conference on Parallel Processing (ICPP),* 382–391. Elokuu. doi:10.1109/ICPP.2017.47.

Callahan, Paul B, ja S Rao Kosaraju. 1995. “A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields”. *Journal of the ACM (JACM)* 42 (1): 67–90. doi:10.1145/200836.200853.

Clauset, Aaron, Mark EJ Newman ja Cristopher Moore. 2004. “Finding community structure in very large networks”. *Physical review E* 70 (6): 066111. doi:10.1103/PhysRevE.70.066111.

Fagnan, J., O. Zaïane ja R. Goebel. 2012. “Visualizing community centric network layouts”. Teoksessa *2012 16th International Conference on Information Visualisation,* 321–330. Heinäkuu. doi:10.1109/IV.2012.61.

Frishman, Y., ja A. Tal. 2008. “Online Dynamic Graph Drawing”. *IEEE Transactions on Visualization and Computer Graphics* 14, numero 4 (heinäkuu): 727–740. ISSN: 2160-9306. doi:10.1109/TVCG.2008.11.

Fruchterman, Thomas M. J., ja Edward M. Reingold. 1991. “Graph drawing by force-directed placement”. *Software: Practice and Experience* 21 (11): 1129–1164. doi:10.1002/spe.4380211102.

Gajdoš, Petr, Tomáš Ježowicz, Vojtěch Uher ja Pavel Dohnálek. 2016. “A parallel Fruchterman–Reingold algorithm optimized for fast visualization of large graphs and swarms of data”. *Swarm and Evolutionary Computation* 26:56–63. ISSN: 2210-6502. doi:https://doi.org/10.1016/j.swevo.2015.07.006.

Girvan, Michelle, ja Mark EJ Newman. 2002. “Community structure in social and biological networks”. *Proceedings of the national academy of sciences* 99 (12): 7821–7826. doi:10.1073/pnas.122653799.

Gross, Jonathan L, ja Jay Yellen. 2003. *Handbook of graph theory.* CRC press.

Hachul, Stefan, ja Michael Jünger. 2004. “Drawing large graphs with a potential-field-based multilevel algorithm”. Teoksessa *International Symposium on Graph Drawing,* 285–295. Springer. doi:10.1007/978-3-540-31843-9_29.

———. 2005. “An experimental comparison of fast algorithms for drawing general large graphs”. Teoksessa *International Symposium on Graph Drawing,* 235–250. Springer. doi:10.1007/11618058_22.

Huang, Mao Lin, ja Quang Vinh Nguyen. 2007. “A fast algorithm for balanced graph clustering”. Teoksessa *2007 11th International Conference Information Visualization (IV'07),* 46–52. IEEE. doi:10.1109/IV.2007.10.

Jacomy, Mathieu, Tommaso Venturini, Sebastien Heymann ja Mathieu Bastian. 2014. “ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software”. *PloS one* 9 (6). doi:10.1371/journal.pone.0098679.

Jeowicz, T., M. Kudelka, J. Plato ja V. Snáel. 2013. “Visualization of Large Graphs Using GPU Computing”. Teoksessa *2013 5th International Conference on Intelligent Networking and Collaborative Systems,* 662–667. Syyskuu. doi:10.1109/INCoS.2013.126.

Ježowicz, Tomáš, Petr Gajdoš, Eliška Ochodková ja Václav Snášel. 2014. “A New Iterative Approach for Finding Nearest Neighbors Using Space-Filling Curves for Fast Graphs Visualization”. Teoksessa *International Joint Conference SOCO'14-CISIS'14-ICEUTE'14,* 11–20. Springer. doi:10.1007/978-3-319-07995-0_2.

Lipp, Fabian, Alexander Wolff ja Johannes Zink. 2016. “Faster Force-Directed Graph Drawing with the Well-Separated Pair Decomposition”. *Algorithms* 9 (3): 53. doi:10.3390/a9030053.

Muelder, Chris, ja Kwan-Liu Ma. 2008. “Rapid graph layout using space filling curves”. *IEEE Transactions on Visualization and Computer Graphics* 14 (6): 1301–1308. doi:10.1109/TVCG.2008.158.

Purchase, Helen C. 2002. “Metrics for graph drawing aesthetics”. *Journal of Visual Languages & Computing* 13 (5): 501–516. doi:10.1006/jvlc.2002.0232.

Purchase, Helen C, Robert F Cohen ja Murray James. 1995. “Validating graph drawing aesthetics”. Teoksessa *International Symposium on Graph Drawing,* 435–446. Springer. doi:10.1007/BFb0021827.

Quigley, Aaron, ja Peter Eades. 2000. “Fade: Graph drawing, clustering, and visual abstraction”. Teoksessa *International Symposium on Graph Drawing,* 197–210. Springer. doi:10.1007/3-540-44541-2_19.

Tarjan, Robert. 1972. “Depth-first search and linear graph algorithms”. *SIAM journal on computing* 1 (2): 146–160. doi:10.1137/0201010.

Wu, Fang, ja Bernardo A Huberman. 2004. “Finding communities in linear time: a physics approach”. *The European Physical Journal B* 38 (2): 331–338. doi:10.1140/epjb/e2004-00125-x.