SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4574

Automatska vizualizacija grafova

Boris Generalić

Zagreb, lipanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 17. ožujka 2016.

ZAVRŠNI ZADATAK br. 4574

Pristupnik: **Boris Generalić (0036478499)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Automatska vizualizacija grafova**

Opis zadatka:

Obrada različitih vrsta podataka često kao rezultat generira grafove - strukture koje se sastoje od vrhova te bridova kojima su parovi vrhova povezani. Tim bridovima mogu biti pridružene i težine pa govorimo o težinskim grafovima. Kako bi se čovjeku olakšalo razumijevanje i interpretacija takvih rezultata, nastale grafove potrebno je na prikladan način vizualizirati.
U okviru ovog završnog rada potrebno je proučiti načine vizualizacije grafova (jednostavnih i težinskih). Posebnu pažnju potrebno je posvetiti algoritmima koji automatski generiraju razmještaj čvorova u ravnini na temelju simulacije djelovanja sila. Potrebno je napisati osnovnu implementaciju takvog algoritma i prikazati rad na nekoliko primjera grafova, komentirati problem početnog slučajnog odabira položaja vrhova, razmotriti načine rješavanja tog problema te implementirati odgovarajuće rješenje. Radu je potrebno priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 17. lipnja 2016.

Mentor:
Marko Čupić
Doc. dr. sc. Marko Čupić

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

SADRŽAJ

1. Uvod

Internet je u svojoj suštini jedna velika mreža koja povezuje računala. Načini na koje mi ljudi koristimo usluge na Internetu, pogotovo društvene mreže, čini nas međusobno povezanima. Podaci o povezanosti i odnosima, ne samo nas ljudi, nego i ostalih stvari kojima se bavimo, dostupni su na Internetu na različite načine. Vizualizacijom odnosa između stvari često možemo uočiti veze i zaključke koji nam na prvi pogled iz surovih podataka nisu vidljive.

Zadatak ovog rada je istražiti kako grafom vizualizirati kompleksne odnose između objekata na temelju pripremljenih podataka. Da bi u tome uspjeli, proučit ćemo algoritam za raspoređivanje vrhova grafa temeljen na simulaciji djelovanja sila (engl. *Force Directed Algorithm*). Napisat ćemo implementaciju alata za vizualizaciju grafova koji koristi navedeni algoritam te koji će korisniku omogućiti interakciju s grafom prilikom simulacije razvoja rasporeda vrhova grafa.

2. Osnove teorije grafova

Teorija grafova je grana matematike koja se bavi proučavanjem grafova. Na najnižoj razini apstrakcije graf je struktura podataka koja se sastoji od skupa vrhova koji su međusobno proizvoljno povezani skupom bridova. Grafovi nam omogućuju efikasno modeliranje različitih mreža iz stvarnog svijeta, gdje vrhovima grafa predstavljamo elemente mreže (npr. ljudi, računala, gradovi), dok bridovima predstavljamo u kakvoj su vezi povezani elementi (npr. prijatelji, ethernet veza, ceste). Grafovi kao struktura podataka i algoritmi za rad s njima imaju vrlo široku primjenu u računarstvu [7].

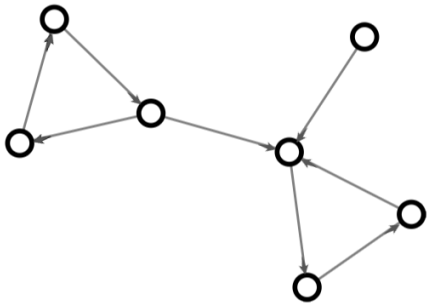
Matematički formalno: ***Graf** je uređena trojka* $G = (V, E, \varphi)$, gdje je $V = V(G)$, gdje je V neprazan skup **vrhova**, E skup **bridova** koji je disjunktan s V, a φ funkcija koja svakom bridu iz E pridružuje dva, ne nužno različita vrha iz V. Graf često zapisujemo kao par $G = (V, E, \varphi)$ ili samo G [10].
Postoji više vrsta grafova, u nastavku ćemo prvo objasniti podjelu grafova s obzirom na usmjerenost bridova koje dijelimo na usmjerene i neusmjerene grafove te nakon toga podjelu grafova s obzirom na težinu bridova koje dijelimo na težinske i bezteže inske grafove.

2.1. Podjela prema usmjerenosti bridova

Za graf $G = (V, E, \varphi)$ kažemo da je orijentirani ili usmjereni [7], ako je relacija E asimetrična, tj. ako vrijedi:

$$(v_i, v_j) \in E \Rightarrow (v_j, v_i) \notin E \tag{2.1}$$

Bridovi kod usmjerenog grafa imaju orijentaciju te oni predstavljaju uređene parove koji povezuju početni i završni vrh. Primjer usmjerenog grafa prikazan je na slici 2.1.

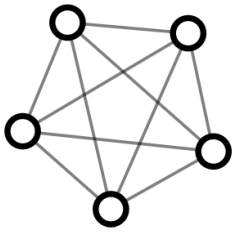


Slika 2.1: Primjer usmjerenog grafa

Za graf $G = (V, E, \varphi)$ kažemo da je neorijentirani ili neusmjereni [7], ako je relacija E simetrična, tj. ako vrijedi:

$$(v_i, v_j) \in E \Rightarrow (v_j, v_i) \in E \tag{2.2}$$

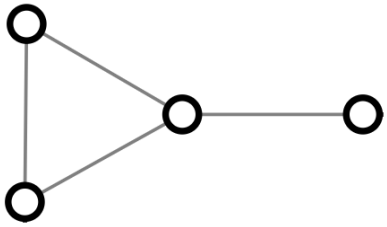
Bridovi kod neusmjerenog grafa nemaju orijentaciju te oni predstavljaju neuređene parove koji povezuju dva vrha grafa. Primjer neusmjerenog grafa prikazan je na slici 2.2.



Slika 2.2: Primjer neusmjerenog grafa

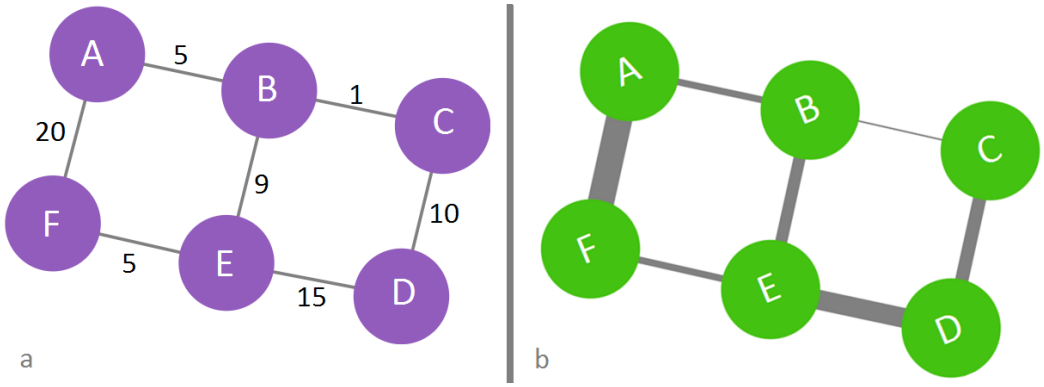
2.2. Podjela prema težini bridova

Bezteˇzinski graf [7] je onaj graf kojem bridovi nemaju pridružene težine. Primjer takvog bezteˇzinskog, ujedno i neusmjerenog, grafa prikazan je na slici 2.3.



Slika 2.3: Primjer bezteˇzinskog neusmjerenog grafa

Težinski graf [7] je onaj graf kojem bridovi imaju pridružene težine. Težine se mogu izraziti kao numeričke vrijednosti iznad bridova (slika 2.4a), ili prikazati debljinom brida koja je proporcionalna njegovoj težini (slika 2.4b).



Slika 2.4: Primjer prikaza težina bridova

3. Algoritam za raspoređivanje vrhova grafa temeljen na simulaciji djelovanja sila

Algoritam za raspoređivanje vrhova vođen silom [8] pripada razredu algoritama za crtanje grafova. Rezultat algoritma je estetski prihvatljiv izgled grafa, s visokom razinom simetrije, uz minimalnu razinu presijecanja bridova i preklapanja vrhova grafa te ravnomjernom distribucijom vrhova grafa u prostoru i ravnomjernom duljinom bridova [3]. Da bi se postigli navedeni rezultati, algoritam koristi dobro poznate zakone fizike da bi modelirao odnos između vrhova i bridova grafa. U nastavku ćemo objasniti moguće probleme ovakvog pristupa.

3.1. Graf kao fizikalni model vrhova i bridova

Vrhove grafa obično predstavljamo kao tijela, kojima je pridjeljen naboj ili masa, dok su bridovi predstavljeni kao elastične opruge koje povezuju vrhove. Između svih vrhova djeluje odbojna sile, ali samo između vrhova povezanih bridom djeluje privlačna sila. Osim navedenih sila, često se koriste sila tromosti te gravitacijska sila iz razloga da imamo bolju kontrolu kod vizualizacije grafa.

Algoritam u svakoj iteraciji računa utjecaj sila na vrhove grafa te nakon toga razmješta vrhove grafa temeljem rezultantne sile na svaki od vrhova. Postupak se ponavlja sve dok neki od uvjeta prekida nisu zadovoljeni. Često korišteni uvjeti prekida [9] su:

- maksimalni rezultanti pomak tijekom jedne iteracije je manji od neke fiksne vrijednosti,
- prijeđen je maksimalni broj iteracija.

3.1.1. Odbojna sila

Odbojna sila [9] djeluje između svih vrhova grafa. Time sprječavamo neželjeno preklapanje vrhova, što pridonosi smanjenu razine presjecanja bridova grafa.

Odlučimo li se da vrhove grafa predstavimo kao točkaste naboje, Coulombov zakon nam kaže da je električna sila između dva naboja možemo računati prema sljedećem izrazu:

$$F_E(r) = k_e \frac{q_1 q_2}{r^2}, \tag{3.1}$$

gdje k_e predstavlja iznos Coulombove konstante, q_1 i q_2 su iznosi naboja, dok je r udaljenost između dva točkasta naboja. Električna sila je odbojna ukoliko su naboji istog polariteta.

3.1.2. Privlačna sila

Privlačna sila [9] djeluje samo između parova vrhova koji su povezani bridom. Razlog zašto uvodimo privlačnu silu je taj što algoritam kod računanja odbojne sile nastoji razmaknuti vrhove što je više moguće, zbog toga dodajemo privlačnu silu da bi zadržali vrhove unutar ekrana te da bi postigli ravnomjernu duljinu bridova.

Označimo li s k_H konstantu rastezljivosti opruge, s x označimo trenutnu duljinu opruge te s x_0 duljinu nerastegnute opruge, Hookeov zakon kaže da ako se tijelo na elastičnoj opruzi pomakne iz ravnotežnog položaja, tj. ako se opruga rastegne ili stisne, djelovat će povratna sila (elastična sila opruge), koja će nastojati tijelo vratiti u ravnotežni položaj. Iznos te sile prema izrazu 3.2 je proporcionalan pomaku tijela iz ravnotežnog položaja.

$$F_H(x) = k_s(x - x_0) \tag{3.2}$$

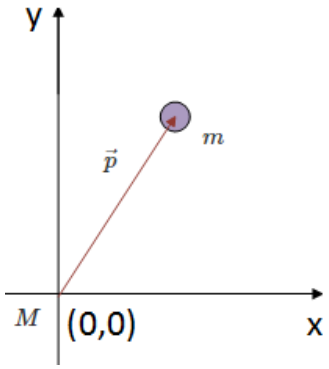
3.1.3. Gravitacijska sila

Da bi zadržali vrhove grafa unutar ekrana koristimo gravitacijsku silu [9]. Način na koji to možemo postići je da postavimo tijelo mase M u ishodište koordinatnog sustava $(0, 0)$, kao što je prikazano na slici 3.1. Tada za svaki vrh grafa računamo gravitacijsku silu prema izrazu:

$$\vec{F_G}(\vec{p}) = -G\frac{mM}{\| \vec{p} - \vec{0} \|^2}\hat{p} = -G\frac{mM}{\| \vec{p} \|^2}\hat{p}, \tag{3.3}$$

gdje p predstavlja vektor položaja vrha te m masu pridruženu vrhu (slika 3.1). Zbog jednostavnosti uzimamo da je $M = 1$, pa time dobivamo:

$$\vec{F_G}(\vec{p}) = -G\frac{m}{\| \vec{p} \|^2}\hat{p}. \tag{3.4}$$



Slika 3.1: Prikaz tijela mase M u ishodištu koordinatnog sustava te vrha grafa s vektorom položaja $\vec{p}$ i masom m

3.1.4. Sila otpora

Umjesto da samo prikažemo konačan izgled grafa kao rezultat algoritma za raspoređivanje vrhova vođen silom, željeli bi simulirati razvoj grafa u stvarnom vremenu. Međutim da bi izgled simulacije bio što realniji, želimo biti u mogućnosti prilagoditi brzinu kretanja vrhova tijekom simulacije te zbog toga uvodimo brzinu kretanja vrha, koju ćemo označavati s v. Na brzinu kretanja vrha možemo utjecati silom otpora [11], slično kao otpor u zraku ili viskoznost kod tekućina. Silu otpora za pojedini vrh grafa izračunavamo prema izrazu:

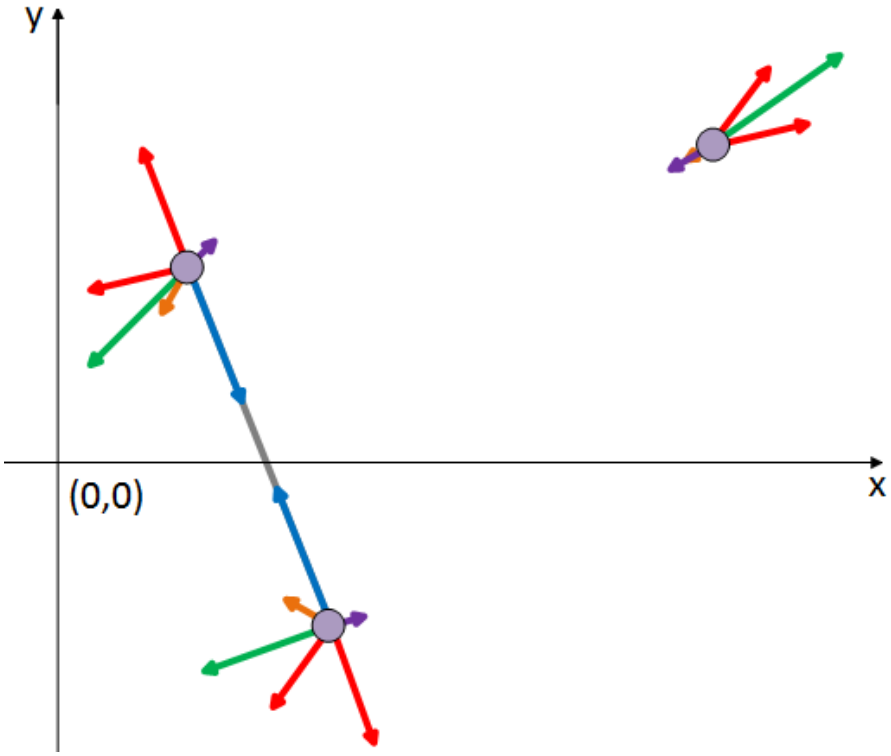
$$\vec{F_D}(\vec{v}) = -b\vec{v}, \tag{3.5}$$

gdje je $\vec{v}$ vektor brzine vrha grafa, dok je b koeficijent otpora.

Na slici 3.2 možemo vidjeti utjecaj navedenih sila na tri vrha grafa. Vektori sila obojani su sljedećim bojama:

- crvena – odbojna sila, djeluje između svih vrhova grafa,
- plava – privlačna sila, djeluje između vrhova koji su povezani bridom,
- narančasta – gravitacijske sila, djeluje s obzirom na ishodište koordinatnog sustava,
- ljubičasta – sile otpora, djeluje suprotno s obzirom na smjer djelovanja ostalih sila te

– zelena – rezultanta sila, vektorski zbroj svih navedenih sila



Slika 3.2: Prikaz vektora svih sila koje djeluju na tijela

3.2. Algoritam Fruchterman-Reingold

Thomas Fruchterman i Edward Reingold su 1991. godine predstavili algoritam u kojem su vrhove grafa modelirali kao nedjeljive čestice ili nebeska tijela koja međusobno djeluju jedni na druge odbojnom silom, dok vrhovi povezani bridom se privlače [3]. Kod crtanja grafa ovim algoritmom teži se sljedećim uvjetima:

– duljine svih bridova trebale bi biti jednake,

– konačan raspored mora osigurati što višu razinu simetrije te

– jednolika distribucija vrhova u prostoru.

Za izračun odbojne i privlačne sile korišteni su sljedeći izrazi [8]:

$$f_r(d) = -k^2 d, \tag{3.6}$$

$$f_a(d) = d^2 k. \tag{3.7}$$

d označava udaljenost između dva vrha, dok k predstavlja optimalnu udaljenost između vrhova koju računamo prema izrazu 3.8.

$$k = \sqrt{\frac{W \times H}{|V|}} \tag{3.8}$$

W predstavlja širinu te H visinu ekrana gdje je prikazan graf, a $|V|$ je ukupan broj vrhova grafa.

Prije nego li pogledamo pseudokod algoritma Fruchterman-Reingold pogledajmo strukturu podataka kojom predstavljamo vrhove grafa prikazanu programskim kôdom 5.1.

Listing 3.1: Razred `Node`

```
public class Node {
    private Point2D position;
    private Point2D delta;
}
```

Članska varijabla `position` predstavlja vektor položaja vrha grafa s komponentama x i y koje predstavljaju koordinate vrha grafa prikazanog na ekranu. Članska varijabla `delta` je također dvodimenzionalni vektor koji je rezultat međudjelovanja odbojnih i privlačnih sila na vrh grafa u jednoj iteraciji algoritma.

Postupak algoritma prikazanog pseudokodom (1) je prilično jednostavan, iterativno ponavljamo sljedeće korake:

1. za svaki vrh računamo utjecaj odbojne silu na vrh kojom djeluju ostali vrhovi,
2. za svaki brid izračunavamo privlačnu silu između dva vrha koje brid spaja te
3. na temelju izračunatih sila dodajemo vektor pomaka vektoru položaja vrha.

Algorithm 1 Algoritam Fruchterman-Reingold [8]

učitaj W i H te izračunaj koeficijent k prema izrazu 3.8
inicijaliziraj temperaturu sustava $t := W/10$
$G := (V, E)$; {slučajnim odabirom odaberi koordinate vrhova grafa}
for $(i := 0; i < iters; inc(i))$ **do**
  {izračunaj odbojne sile između vrhova}
  for $(v : V)$ **do**
    $v.delta := 0$
    for $(u : V)$ **do**
      if $u \neq v$ **then**
        $d := v.position - u.position$
        $v.delta := v.delta + (d/|d|) * f_r(|d|)$
      end if
    end for
  end for
  {izračunaj privlačne sile između vrhova povezanih bridovima}
  for $(e : E)$ **do**
    $d := e.v.position - e.u.position$
    $e.v.delta := e.v.delta - (d/|d|) * f_a(|d|)$
    $e.u.delta := e.u.delta + (d/|d|) * f_a(|d|)$
  end for
  for $(v : V)$ **do**
    {ograniči $v.delta$ temperaturom sustava}
    $v.position := v.position + (v.delta/|v.delta) * min(v.delta, t)$
  end for
  $t := ohladiTemperaturu(i, k)$;
end for

Razlog zašto je ovaj algoritam zanimljiv je zbog toga što uvodi parametar temperature koji utječe na odabir pomaka za koji pridodajemo vektor položaja svakog vrha grafa u jednoj iteraciji algoritma. Postupak je sljedeći [8]: uzmimo da je temperatura na početku jednaka nekoj početnoj vrijednosti (npr desetina širine ekrana) te ju postupno u svakoj iteraciji algoritma smanjujemo prema odabranom planu hlađenja, u želji da na kraju svega postignemo minimalnu energiju sustava. Opisani postupak je zapravo simulirano kaljenje temperature sustava [8].

3.2.1. Simulirano kaljenje

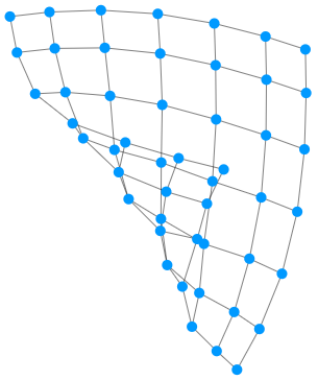
Algoritam simuliranog kaljenja [14] pripada skupini optimizacijskih algoritama, koje nazivamo *metaheuristikama*. Inspiracija proizlazi iz postupka kaljenja metala koji se primjenjuje u metalurgiji, gdje se metal na početku zagrije do svoje temperature taljenja koja se neko vrijeme održava te se temperatura postupno smanjuje sve dok metal ne dođe do stanja minimalne energije, odnosno do stanja gdje su gibanja atoma unutar metala minimalna. Time metal prolazi kroz različita stanja te se odabirom pravilnog plana hlađenja postižu bolja svojstva metala. Odabir plana hlađenja utječe na brzinu algoritma i kvalitetu pronađenog rješenja. Ohladimo li sustav prebrzo algoritam će prije doći do nižih temperatura, no postoji veća vjerojatnost da struktura dobivenog rezultata loša. Kod implementacije algoritma korištene su dva plana hlađenja, opisana u nastavku.

Linearni plan hlađenja

Kod linearnog plana hlađenja temperature se u svakoj iteraciji algoritma crtanja grafa smanjuje linearno prema izrazu:

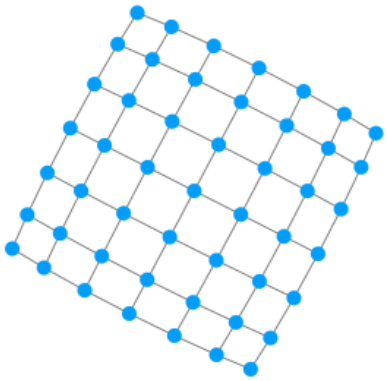
$$T_i = T_0 - i * \beta, \tag{3.9}$$

gdje i označava trenutnu iteraciju, a za β se uzima fiksna vrijednost kojom utječemo na brzinu hlađenja.



Slika 3.3: Rezultat dobiven linearnim planom hlađenja uz veći β

Na slici 3.3 možemo vidjeti rezultat izvođenja grafa uz linearni plan hlađenja. Kao što se vidi iz prikazanog grafa, temperatura sustava se prebrzo ohladila što je rezultiralo upadom algoritma u lokalnom minimumu.



Slika 3.4: Rezultat dobiven linearnim planom hlađenja uz manji β

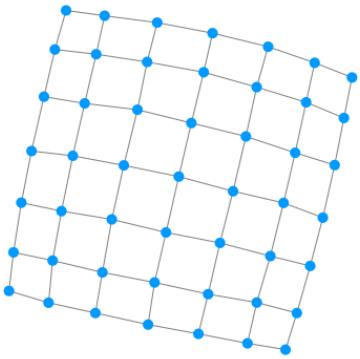
Pažljivim odabirom parametra β, pazeći pritom da ne ohladimo sustav prebrzo, dobivamo prihvaljiv raspored vrhova grafa prikazanog na slici 3.4.

Logaritamski plan hlađenja

Logaritamski plana hlađenja računa temperaturu u svakoj iteraciji algoritma crtanja grafa prema izrazu:

$$T_i = \frac{T_0}{log(i)}. \tag{3.10}$$

Navedeni plan hlađenja je osjetno sporiji u odnosu na linearni plan hlađenja te se on rijetko koristi u praksi [14].



Slika 3.5: Rezultat dobiven logaritamskim planom hlađenja

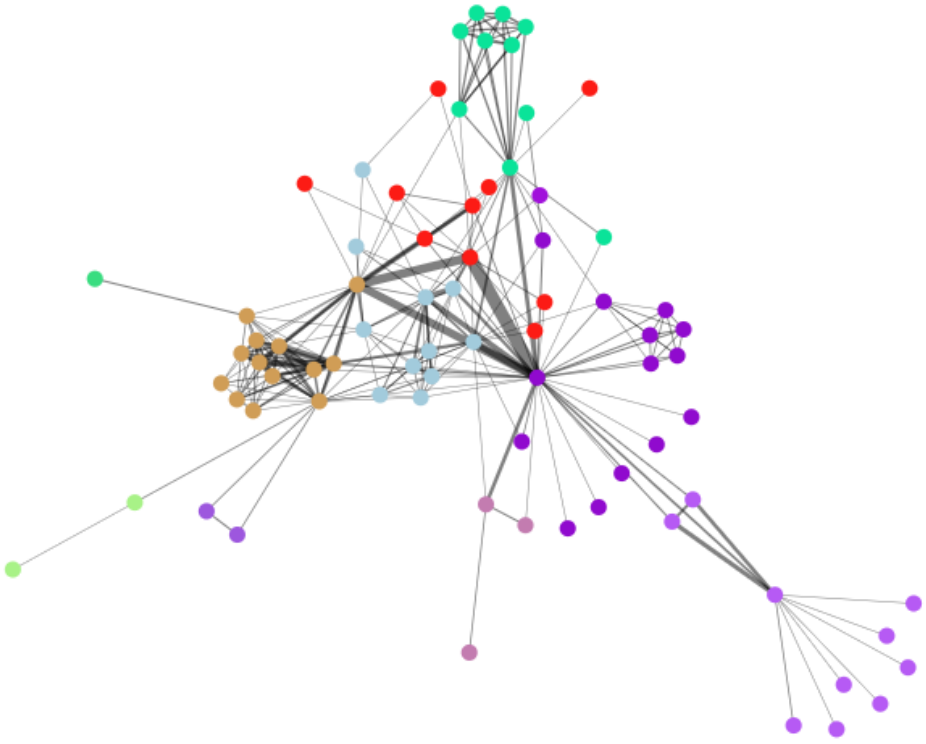
Na slici 3.5 prikazan je rezultat izvođenja grafa uz logaritamski plan hlađenja na istom primjeru grafa kao i kod linearnog plana hlađenja. Prema prikazanom grafu možemo vidjeti da je u ovom slučaju algoritam pronašao optimalni raspored, no u osjetno sporijem vremenu nego kod linearnog plana hlađenja.

3.3. Rezultati

Sada kada smo se upoznali s algoritmom za raspoređivanje vrhova vođenog silom, pogledajmo rezultate dobivene vlastitom implementacijom algoritma Fruchterman-Reingold.

3.3.1. Primjer grafa kompleksne mreže

Na slici 3.6 možemo vidjeti prikaz grafa kompleksne mreže koja predstavlja odnos između likova u drami *Les Misérables*, koju je napisao francuski autor Victor Hugo [2]. Prikazani graf je neusmjereni i težinski, kojem su težine bridova su u ovom slučaju

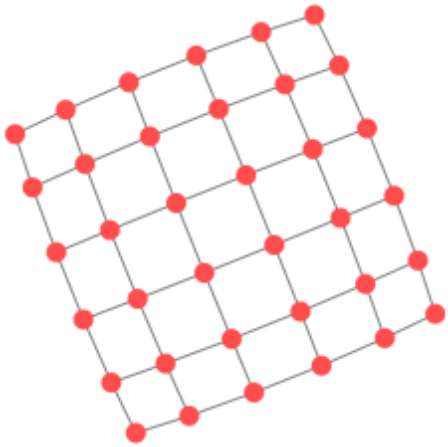


Slika 3.6: Prikaz grafa likova iz drame *Les Misérables*

izražene debljinom bridova grafa te su one deblje između onih likova koji su u češćoj interakciji kroz dramu. Grupe likova ili grozdovi, termin koji se koristi u teoriji kompleksnih mreža, prikazane su zasebnim bojom vrhova. Struktura grafa preuzeta je iz [2].

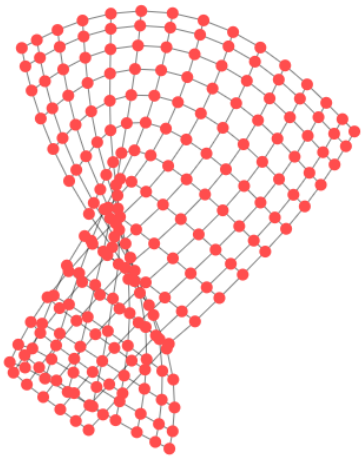
3.3.2. Primjer grafa 2D kvadratne rešetke

Na slici 3.7 možemo vidjeti prikaz grafa 2D kvadratne rešetke reda $n = 6$.



Slika 3.7: Graf 2D kvadratne rešetke reda $n = 6$

Prikazani graf ima $6x6 = 36$ vrhova i kao što vidimo algoritam je postigao zadovoljavajući raspored vrhova grafa, gdje se niti jedan par bridova ne preklapaju.



Slika 3.8: Graf 2D kvadratne rešetke reda $n = 15$

Graf prikazan na slici 3.8 iste je vrste kao i prethodni graf, ali reda $n = 15$. Prikazani graf ima $15x15 = 255$ vrhova te možemo primjetiti da prikazano rješenje nema optimalan raspored vrhova, već da je algoritam ponovo zaglavio u lokalnom optimumu. U nastavku ćemo objasniti razloge ovakvih problema te predložiti moguća poboljšanja algoritma.

3.4. Problemi

Glavni problem algoritma Fruchterman-Reingold je njihova ograničenost da dobro prikazuje grafove s manje od 40 vrhova [8]. Kao što smo mogli vidjeti na prikazanim rezultatima, algoritam daje nezadovoljavajuće rezultate za veće grafove.

Razlog takvog problema jest činjenica da fizikalni model vrhova i bridova kod većih grafova ima puno lokalnih minimuma, stanja s minimalnom energijom, u kojima algoritam često zaglavi. Postupkom simuliranog kaljenja te korištenjem različitih planova hlađenja možemo donekle rješiti taj problem. Međutim taj postupak za određene planove hlađenja je vrlo spor, što nije izbor ako bismo željeli korisniku u stvarnom vremenu prikazati simulaciju razvoja grafa.

Također, usko grlo algoritma Fruchterman-Reingold je računanje odbojnih sila između vrhova, taj se fenomen u fizici naziva problem n tijela [12]. To je najsporiji dio algoritma, čime vremenska složenost algoritma iznosi $O(n^2)$, što nam predstavlja problem kod simulacije većih grafova. U idućem poglavlju opisati ćemo neke od mogućih rješenja za navedene probleme.

4. Veći grafovi

Potaknuti problemima navedenim na kraju prethodnog poglavlja u ovom poglavlju ćemo se osvrnuti na neke od mogućih rješenja, koja ćemo na kraju koristiti kod vlastite implementacije alata za vizualizaciju grafova. Najprije ćemo dati na odgovor kako što bolje izbjeći zastoj algoritma u lokalnom minimumu, dok će u drugom dijelu poglavlja biti objašnjen algoritam Barnes-Hut kojim rješavamo izračun sila kod problema n tijela.

4.1. Problem lokalnih minimuma

Lokalni minimum je stanje gdje graf pronalazi raspored vrhova kod kojeg je razina energije fizikalnog sustava vrhova i bridova minimalna. U takvom moguće su samo sitne promjene položaja vrhova koje ne pridonose boljem rasporedu minimalne energije grafa, iako bolji raspored vrhova zapravo postoji [8].

Zastoj algoritma u lokalnom minimumu se može u velikom broju slučajeva izbjeći pametnim inicijalnim rasporedom vrhova grafa. Što je inicijalni raspored vrhova bolji, to je algoritmu potrebno manje vremena da pronađe zadovoljavajući raspored vrhova [9].

Ideja je sljedeća. Umjesto da inicijalni raspored vrhova prepustimo slučajnom odabiru, odmah na početku primjenimo postupak simuliranog kaljenja koristeći linearni plan hlađenja te ubrzano kroz par iteracija hladimo sustav. Dobiveni raspored nakon toga koristimo kao inicijalni raspored algoritma raspoređivanja vrhova vođenog silom [8].

Također izlaz iz lokalnih minimuma možemo ostvariti tako da korisniku omogućimo interakciju s grafom u stvarnom vremenu [4], gdje korisnik može promjenom parametara (npr. koeficijent tromosti vrhova, odbojna sila, duljina opruge) sila fizikalnog sustava vrhova i bridova utjecati na brzinu postizanja stanja minimalne energije sustava, odnosno povoljnog rasporeda vrhova grafa. Prepreka koja nas sprječava da bi ostvarili interakciju s grafom u stvarnom vremenu je kvadratna složenost izračuna

odbojnih sila između vrhova, da bi taj problem ostvarili u linearno – logaritamskoj složenosti koristi se algoritam Barnes–Hut koji je opisan u nastavku.

4.2. Algoritam Barnes - Hut

Glavna ideja kako da ubrzamo izračun sila u problemu n tijela počiva na tome da tijela koje su dovoljno blizu jedni drugima grupiramo u grupu te ih aproksimiramo kao jedno tijelo čiji je vektor položaja jednak upravo centru mase grupe tih tijela. Koristeći navedenu aproksimaciju postupak izračuna sila između tijela moguće je obaviti u $O(nlog(n))$ vremenskoj složenosti [13].

4.2.1. Centar mase grupe tijela

Centar mase grupe tijela je prosječni položaj tijela unutar grupe, podijeljena masom svih tijela koja čine grupu. Ako grupu čine dva tijela mase m_1 i m_2 koji se nalaze na koordinatama (x_1, y_1) i (x_2, y_2), tada je ukupna masa grupe određena izrazom 4.1, a x i y koordinate centra mase grupe izrazima 4.2 i 4.3.

$$M = m_1 + m_2 \tag{4.1}$$

$$x = \frac{x_1 m_1 + x_2 m_2}{M} \tag{4.2}$$

$$y = \frac{y_1 m_1 + y_2 m_2}{M} \tag{4.3}$$

4.2.2. Struktura podataka

Barnes-Hut algoritam koristi stablastu strukturu za grupiranje tijela koja su dovoljno blizu jedni drugima [13]. Algoritam rekurzivno dijeli tijela u grupe spremajući ih u stablo kod kojeg čvorovi, osim listova, imaju četiri čvora dijeteta. Jedan čvor takvog stabla prikazan je programskim kodom 4.1.

Listing 4.1: Razred `QuadTreeNode`

```
public class QuadTreeNode {
        private double mass;
        private Point2D centerOfMass;
        private Node value;
        private QuadTreeNode[] children;
}
```

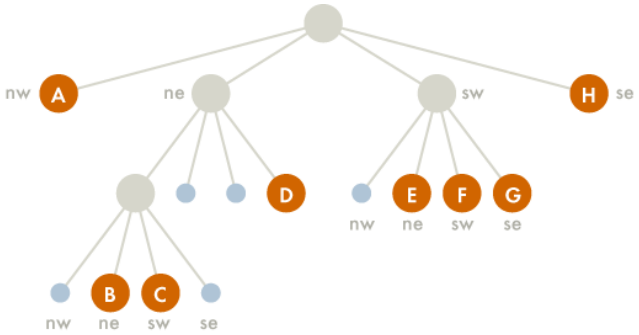
Članska varijabla `mass` je ukupna masa svih čvorova djece, `centerOfMass` je centar mase čvora, članska varijabla `value` je pohranjeni vrh grafa, ili `null` ako čvor ima djecu, `children` je polje koje sadržava 4 čvora djece ili `null` ako čvor nema djecu.

Svaki čvor stabla predstavlja jedan od četiri kvadranta prostora svog roditelja, korijen stabla nema roditelja te on predstavlja čitav prostor ravnine. Kao što možemo vidjeti na slici 4.1, svaki kvadrant se dijeli u četiri kvadranta sve dok svaki kvadrant ne sadrži samo jedno ili nijedno tijelo. Ovakvom politikom podjele prostora smo efektivno do-

Slika 4.1: Podjela prostora na kvadrante.
Preuzeto iz [13]

bili dvije vrste čvorova, a to su unutrašnji čvorovi, odnosno vanjski čvorovi.

Unutrašnji čvor predstavlja dio prostora koji je podijeljen u četiri kvadranta te on sadrži četiri čvora dijeteta. Unutrašnji čvor sadrži grupu tijela te sprema informaciju o centru mase grupe i njezinoj ukupnoj masi. Djeca unutrašnjog čvora mogu opet biti unutrašnji čvorovi, vanjski čvorovi ili prazni čvorovi. Prazni čvorovi su rezultat toga što se na nekim dijelovima prostora ne nalaze tijela. Korijen stabla je upravo unutrašnji čvor. Vanjski čvor je čvor koji sadrži te ujedno predstavlja jedno tijelo.



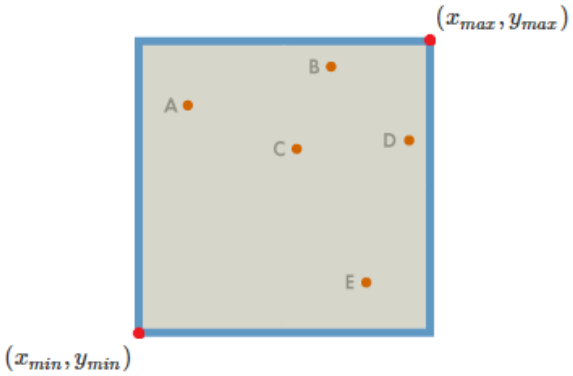
Slika 4.2: Izgled stabla.
Preuzeto iz [13]

Primjer stabla prikazan je na 4.2, gdje je su sivom bojom označeni unutrašnji čvorovi, crvenom vanjski čvorovi te plavom prazni čvorovi. Djecu unutrašnjeg čvora razlikujemo po oznakama NW (engl. *Northwest*), NE (engl. *Northeast*), SW (engl. *Southwest*) i SE (engl. *Southeast*).

4.2.3. Izgradnja stabla

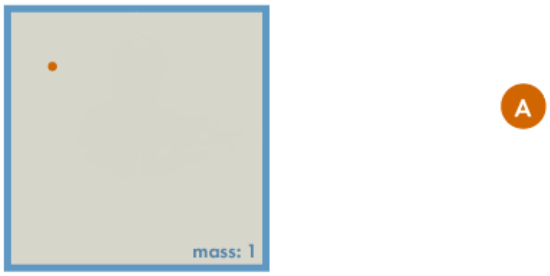
Da bi dodali tijelo t u stablo čiji je korijen čvor r slijedimo sljedeću rekurzivnu proceduru:

1. ako r ne sadrži tijelo, postavi t kao dijete od čvora r,
2. ako je r unutrašnji čvor, ponovno izračunaj centar mase i ukupnu masu čvora r, pa rekurzivno dodaj tijelo t unutar čvora r,
3. ako r je vanjski čvor, koji već sadrži tijelo t_1, onda r postaje unutrašnji čvor, rekurzivno dodaj tijela t_1 i t unutar čvora r te ponovno izračunaj centar mase i ukupnu masu čvora r.



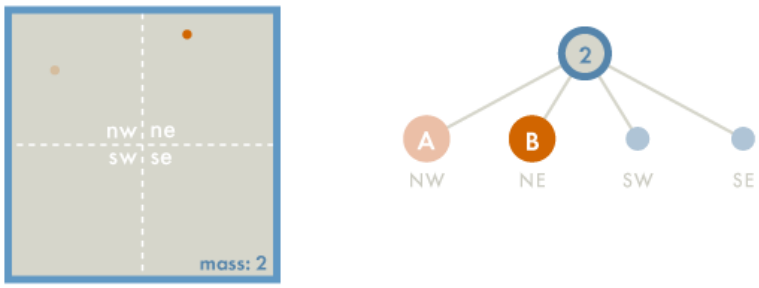
Slika 4.3: Primjer rasporeda tijela u ravnini.
Preuzeto iz [13]

Pogledajmo kako algoritam izgradnje stabla radi na primjeru u nastavku. Najprije se odrede (x_{min}, y_{min}) i (x_{max}, y_{max}) koordinate prostora unutar kojeg se nalaze tijela (slika 4.3). Unutar prikazanog prostora nalaze se četiri tijela koja redom stavljamo u stablo.



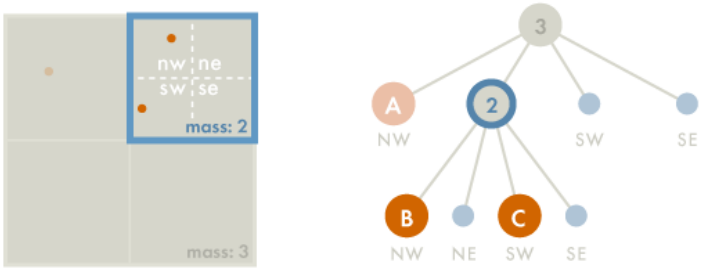
Slika 4.4: Dodavanje tijela A.
Preuzeto iz [13]

Prvo dodamo tijelo A u stablo. Korijen ne sadrži djecu pa dodamo A kao dijete (slika 4.4).



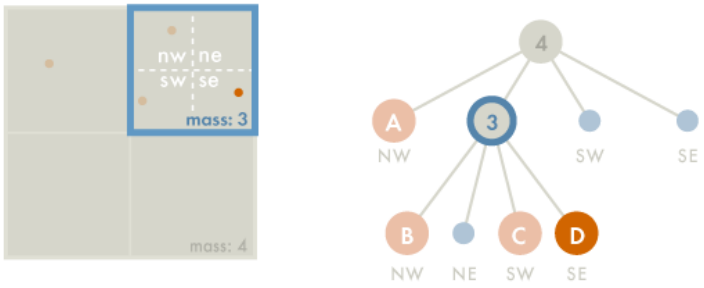
Slika 4.5: Dodavanje tijela B.
Preuzeto iz [13]

Zatim u stablo stavljamo tijelo B. Pošto je korijen vanjski čvor i on već sadrži tijelo A, dijelimo prostor na četiri kvadranta te korijen postaje unutrašnji čvor. Nakon toga rekurzivno stavljamo tijela A i B. Tijelo A će završiti na NW poziciji, dok će tijelo B završiti na ne poziciji unutrašnjeg čvora (slika 4.5).



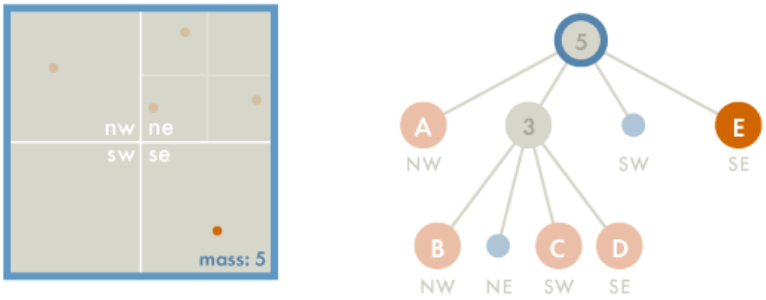
Slika 4.6: Dodavanje tijela C.
Preuzeto iz [13]

Pozicija tijela C odgovara kvadrantu NE koji je ujedno vanjski čvor i sadrži tijelo B. Ponovo dijelimo prostor na četiri kvadranta te vanjski čvor NE sada postaje unutrašnji čvor te dodamo tijela B i C rekurzivno u čvor NE (slika 4.6)



Slika 4.7: Dodavanje tijela D.
Preuzeto iz [13]

Kod dodavanja tijela D činimo 2 koraka. U prvom koraku opet odabiremo kvadrant NE te u drugom koraku dodajemo tijelo D kao dijete unutrašnjeg čvora NE u kvadrant SE (slika 4.7)



Slika 4.8: Dodavanje tijela E.
Preuzeto iz [13]

Na kraju dodamo tijelo E, njegova pozicija odgovara kvadrantu SE korijena stabla (slika 4.8). Na kraju korijen našeg stabla sadrži centar mase i ukupnu masu od svih 5 tijela koja se nalaze u stablu.

4.2.4. Izračun sila

Prilikom izračuna sile na tijelo t algoritam Barnes-Hut koristi sljedeću važnu aproksimaciju. Grupu tijela (unutrašnji čvor) čiji je centar mase dovoljno udaljen od tijela t, predstavimo kao jedno tijelo čiji je vektor položaja jednak centru mase grupe i čija je masa jednaka ukupnoj masi svih tijela unutar te grupe [13].

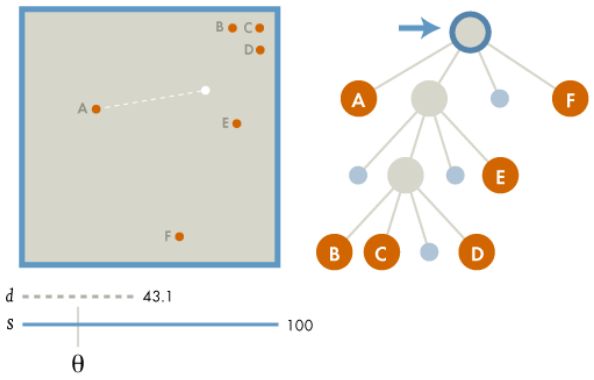
Da bi odredili je li tijelo dovoljno udaljeno od centra mase grupe, računamo kvocijent s/d, gdje je s širina kvadranta grupe, a d je udaljenost između tijela t i centra mase grupe. Kvocijent uspoređujemo s konstantom ϑ, koju ujedno zovemo i Barnes-Hut konstanta. Ako je kvocijent manji od ϑ, tada je tijelo dovoljno daleko od centra mase grupe te grupu aproksimiramo kao jedno pseudo-tijelo. Iznos konstante ϑ utječe na brzinu i preciznost algoritma te uobičajeno uzimamo $\vartheta = 0.5$ [13].
Silu na tijelo t računamo rekurzivno prolazeći čvorovima stabla, počeviši od korijena stabla r.

1. ako je r vanjski čvor i ne sadrži tijelo t, izračunaj silu kojom tijelo sadržano u čvoru r djeluje na tijelo t te dodaj izračunatu silu ukupnoj sili na tijelo t,

2. inače čvor r je unutrašnji čvor, izračunaj omjer s/d. Ako je $s/d < \vartheta$, izračunaj silu kojom čvor r djeluje na tijelo t te dodaj izračunatu silu ukupnoj sili na tijelo t,

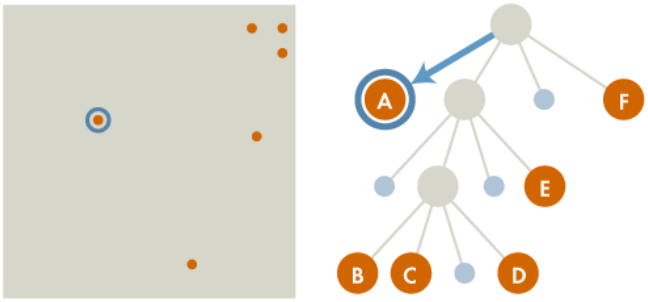
3. inače ako $s/d >= \vartheta$ ponovi postupak rekurzivno za svako dijete čvora r.

Pogledajmo izračun sile na tijelo A na primjeru stabla na slici 4.9. Počinjemo od korijena koji je unutrašnji čvor. Korijen sadrži centar mase šest tijela: $m_A = 1$, $m_B = 2$, $m_C = 3$, $m_D = 4$, $m_E = 5$ i $m_F = 6$.



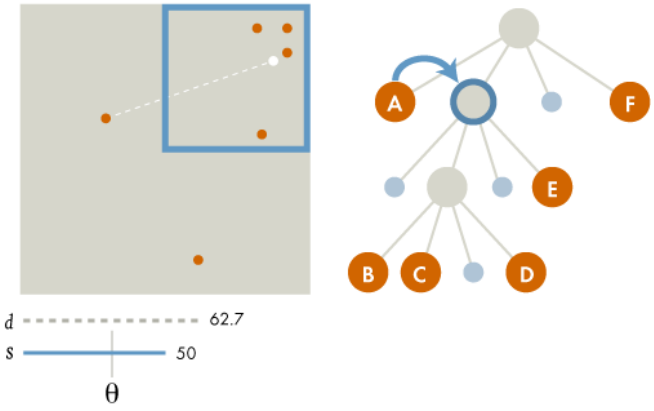
Slika 4.9: 1. korak izračuna sile na tijelo A.
Preuzeto iz [13]

Na početku promatramo korijen stabla. Korijen je unutrašnji čvor, širina kvadranta korijena iznosi $s = 100$ (plava boja), dok udaljenost tijela A od centra mase korijena (bijela točka) iznosi $d = 43.1$. Omjer $s/d = 100/43.1 > \vartheta = 0.5$, pa se spuštamo rekurzivno i ponavljamo postupak za svako dijete korijena.



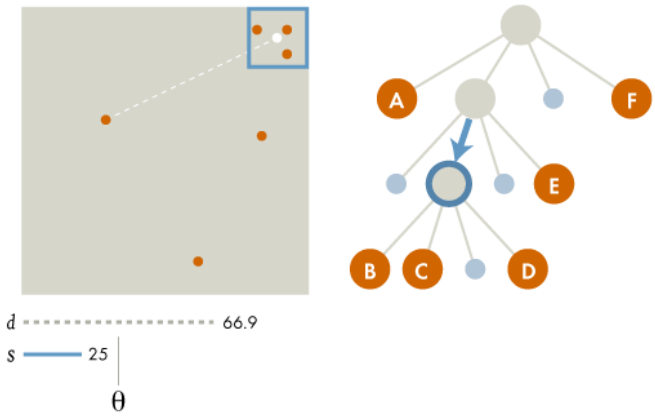
Slika 4.10: 2. korak izračuna sile na tijelo A.
Preuzeto iz [13]

U drugom koraku naišli smo upravo na tijelo A. Tijelo ne djeluje silom samo na sebe pa ovaj korak preskačemo (slika 4.10).



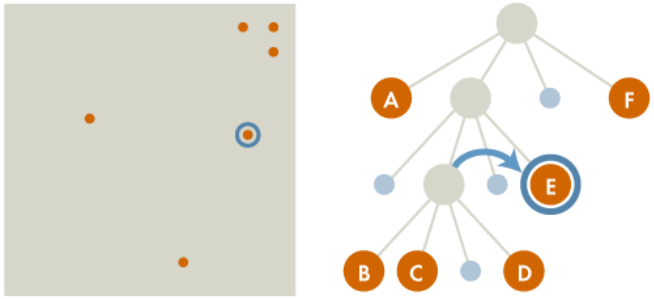
Slika 4.11: 3. korak izračuna sile na tijelo A.
Preuzeto iz [13]

Sljedeći promatrani čvor je unutrašnji koji sadrži centar mase tijela B, C, D i E. Računajući omjer dobivamo: $s/d = 50/62.7 > \vartheta = 0.5$, ponovo se rekurzivno spuštamo te prolazimo kroz djecu trenutno promatranog čvora (slika 4.11).



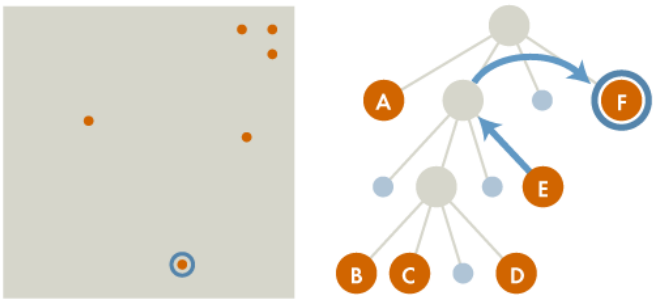
Slika 4.12: 4. korak izračuna sile na tijelo A.
Preuzeto iz [13]

Čvor kojeg promatramo u četvrtom koraku algoritma (slika 4.12) je unutrašnji čvor koji sadrži centar mase tijela B, C i D. Računajući dobivamo $s/d = 25/66.9 < \vartheta$ te ovaj čvor aproksimiramo kao jedno tijelo. Izračunamo silu kojom čvor vrši na tijelo A te pridodamo izračunatu silu ukupnoj sili na tijelo A.



Slika 4.13: 5. korak izračuna sile na tijelo A.
Preuzeto iz [13]

Ovaj puta se ne spuštamo rekurzivno nego prelazimo na vanjski čvor koji sadrži tijelo E. Računamo silu između tijela A i E te dodamo izračunatu silu sili na tijelo A (slika 4.13).



Slika 4.14: 6. korak izračuna sile na tijelo A.
Preuzeto iz [13]

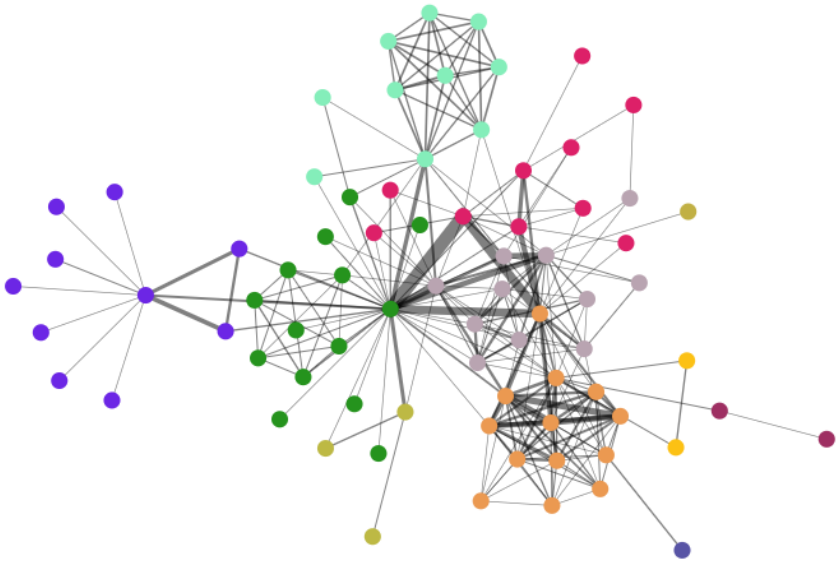
U posljednjem koraku (slika 4.14) vraćamo se razinu iznad te računamo silu između tijela A i F nakon čega dodamo izračunatu silu sili na tijelo A.

4.3. Rezultati

Nakon implementacije prethodno navedenih postupaka u nastavku su prikazani rezultati dobiveni na istim primjerima grafova kao i u prošlom poglavlju.

4.3.1. Primjer grafa kompleksne mreže

Na slici 4.15 vidimo prikaz grafa kompleksne mreže koja predstavlja odnos između likova u drami *Les Misérables*.



Slika 4.15: Prikaz grafa likova iz drame *Les Misérables*

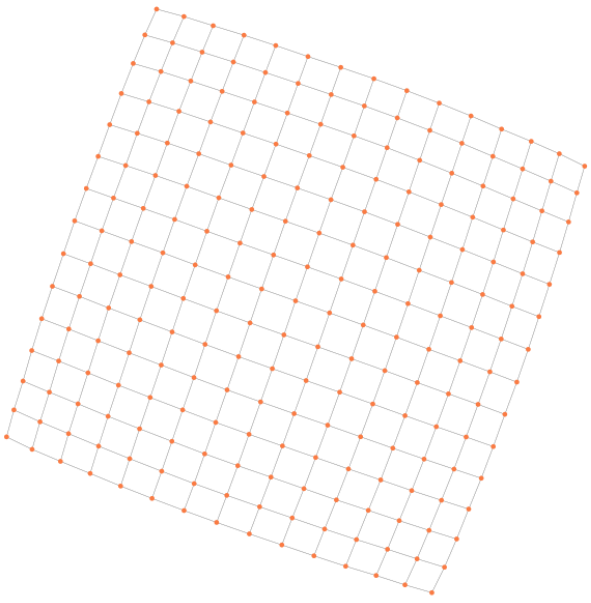
Na slici 4.16 prikazan je izgled grafa u trenutku početka algoritma raspoređivanja vrhova.

Slika 4.16: Početni raspored vrhova grafa

Razlog ovakvog rasporeda vrhova je simulirano kaljenje koje pokrećemo prije početka algoritma da bi dobili što bolji mogući inicijalni raspored vrhova.

4.3.2. Primjer grafa 2D rešetke

U prošlom poglavlju vidjeli smo da je algoritam kod prikaza kvadratne rešetke reda $n = 15$ zaglavio u lokalnom minimumu. Sada dobivamo zadovoljavajući raspored vrhova, prikazan na slici 4.17.



Slika 4.17: Prikaz grafa 2D kvadratne rešetke reda $n = 15$

5. Alat za vizualizaciju grafova

Do sada smo se upoznali s algoritmom za raspoređivanje vrhova grafa vođenog silom te s problemima i tehnikama koje ih rješavaju kod vizualizacije većih grafova. U ovom poglavlju ćemo iskoristiti to znanje na praktičnom primjeru implementacije alata za vizualizaciju grafova. Programska izvedba alata za vizualzaciju grafova ostvarena je u programskom jeziku Java, verzije JDK 1.8, dok je za prikaz grafova te za izradu grafičkog korisničkog sučelja korištena tehnologija JavaFX, verzije 8. Prije nego što krenemo u implementacijske detalje, objasnit ćemo način na koji ćemo realizirati interakciju s grafom u stvarnom vremenu.

5.1. Simulacija sustava n tijela

Simulacija gibanja n tijela poznat je problem u računarstvu koji se rješava postupcima računalne simulacije i numeričke integracije [1]. U našem slučaju ta tijela su vrhovi grafa. Da bi predstavili vrh grafa kao tijelo koristimo strukturu podataka prikazanu programskim kôdom 5.1.

Listing 5.1: Razred `Node`

```
public class Node {
        public double mass;
        public Point2D position;
        public Point2D force;
        public Point2D velocity;
}
```

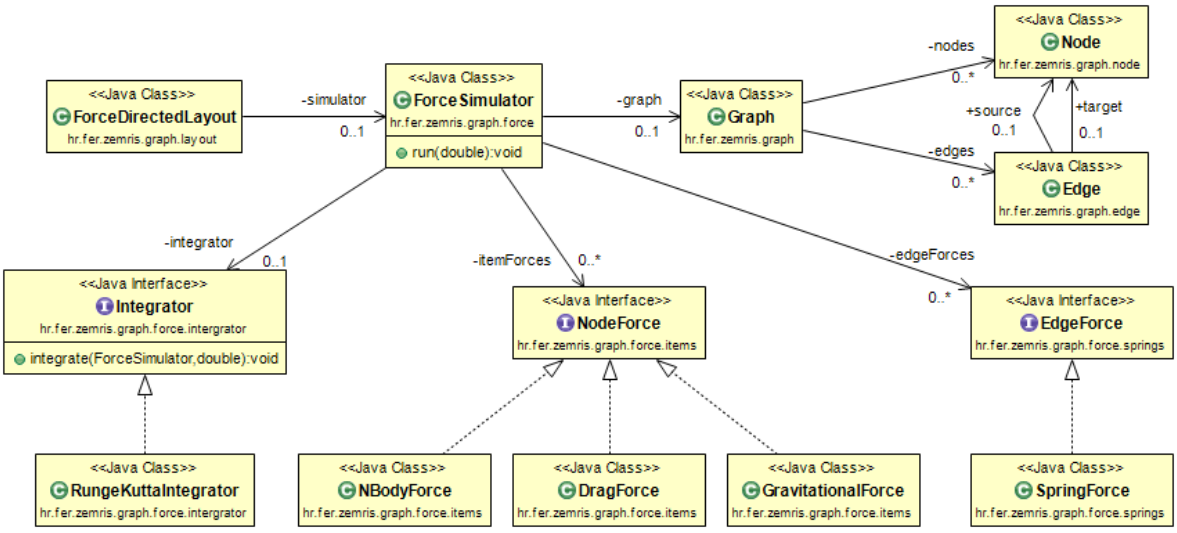
Članskom varijablom `mass` vrhu grafa pridružujemo masu, `position` je vektor položaja, `force` je vektor sile koja djeluje na vrh i `velocity` je vektor brzine gibanja vrha.

Trenutno stanje sustava n tijela je iznos vektora položaja, sile i brzine tijela u trenutnom vremenskom trenutku. Računalo radi u diskretnoj vremenskoj domeni te vrijeme

dijelimo na diskretne vremenske trenutke. Duljina vremenskog intervala između dva trenutka u vremenu nazivamo vremenski korak (engl. *time step*). Računalna simulacija na temelju trenutnog stanja sustava i dobivenog vremenskog koraka računa stanje sustava koje će se dogoditi za vrijeme određeno veličinom dobivenog vremenskog koraka. Na početku se izračuna utjecaj svih sila na svako od tijela te se nakon toga metodom numeričke integracije utjecaj sila integrira kroz vrijeme određeno veličinom vremenskog koraka kako bi se izračunali nove vrijednosti vektora položaja, sile i brzine tijela. Veličina vremenskog koraka utječe na točnost simulacije. Što je veličina manja, rezultat simulacije će biti realniji, no zbog većeg broja izračuna i sporiji. Na preciznost simulacije utječe i izbor metode numeričke integracije [1].

5.2. Implementacija

Pojednostavljeni UML-dijagram prikazan je na slici 5.1. U nastavku ćemo objasniti koja je uloga pojedinog razreda i sučelja u implementaciji alata.



Slika 5.1: UML dijagram razreda

5.2.1. Razred `Graph`

Razredom `Graph` je modelirana konfiguracija grafa kojeg prikazujemo. Razred sadrži referencu na kolekciju objekata razreda `Node` te referencu na kolekciju razreda `Edge`. Razred `Node` je reprezentacija vrha grafa. On enkapsulira vektore položaja, sile i brzine vrha te masu vrha. Brid grafa predstavljen je razredom `Edge`. On sadrži dvije reference na primjerke razreda `Node`, koji odgovaraju vrhovima koje brid spaja.

Kako bi prikazali vrhove i bridove grafa na ekranu, razred `Node` izveden je iz razreda `javafx.scene.shape.Circle` dok je razred `Edge` izveden iz razreda `javafx.scene.shape.Line`.

5.2.2. Sučelja `NodeForce` i `EdgeForce`

Razredi koji implementiraju sučelje `NodeForce` predstavljaju sile koje djeluju na vrhove grafa. Razred `NBodyForce` služi za izračun odbojnih sila između vrhova grafa Izračun sila vrši se prema algoritmu Barnes–Hut. Razred `DragForce` predstavlja silu otpora zraka kojom se utječe na brzinu kretanja vrhova grafa, dok razred `GravitationalForce` služi da bi se vrhove grafa zadržalo u centru ekrana.

Sučelje `EdgeForce` predstavlja sile na bridove grafa. Razred `SpringForce` implementira to sučelje te predstavlja privlačnu silu između dva vrha grafa povezana bridom.

5.2.3. Razred `ForceSimulator`

Razred `ForceSimulator` predstavlja postupak računalne simulacije objašnjene u potpoglavlju 5.1. Postupak računalne simulacije implementiran je u metodi `run` koja kao jedini parametar prima vremenski korak. Metoda koristi referencu na primjerak razreda `Graph` gdje se prvo za svaki vrh grafa prolazi po svim silama iz kolekcije primjeraka razreda `NodeForce` i računa se utjecaj svake od sila na pojedini vrh. Isti se postupak ponavlja za sve bridove, samo što se sada prolazi po svim silama iz kolekcije primjeraka razreda `EdgeForce`. Nakon toga slijedi postupak matematičke integracije utjecaja sila kroz vrijeme koji obavlja konkretna implementaciji sučelja `Integrator`.

5.2.4. Razred ForceDirectedLayout

Razred `ForceDirectedLayout` predstavlja grafičku komponentu, koju stavljamo na scenu JavaFX-aplikacije, koja se jednom kada pokrenemo aplikaciju, prikazuje kao prozor na našem ekranu. Osim što služi za prikaz vrhova i bridova grafa na ekranu, razred sadrži logiku konstantnog pokretanja računalne simulacije. Do sad smo sve ispričali o računalnoj simulaciji, no da bi bili u mogućnosti ostvariti interaktivnu simulaciju u stvarnom vremenu potrebno je ostvariti mehanizam koji će asinkrono pozvati metodu `run` razreda `ForceSimulator`. JavaFX koristi *pulse* sustav koji nam omogućuje

asinkrono izvođenje zadataka [6]. Nama jedino preostaje da stvorimo primjerak anonimnog razreda koji nasljeđuje apstraktni razred `javafx.animation.AnimationTimer` te nadjačamo metodu `handle` u kojoj ćemo pozivati metodu `run` razreda `ForceSimulator`. Primjer programskog kôda 5.2 prikazuje rješenje.

Listing 5.2: Pokretanje simulacije

```
AnimationTimer timer = new AnimationTimer() {
    @Override
    public void handle(long now) {
        simulator.run(timeStep);
    }
};
timer.start();
```

Nakon što pozovemo metodu `start` primjerka razreda `AnimationTimer` dobivamo garanciju da će se prilikom obrade svakog *pulse* događaja izvršiti metoda `handle` koju smo definirali primjerkom anonimnog razreda. JavaFX *pulse* sustav obrađuje *pulse* događaje frekvencijom od 60 sličica po sekundi [6].

5.3. Pohrana strukture grafa

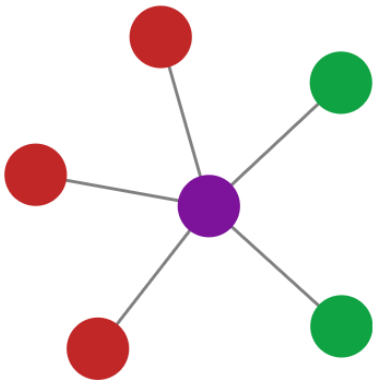
Za pohranu i učitavanje različitih struktura grafova koristi se format *JSON* (engl. *JavaScript Object Notation*), koji nam omogućava da vrhove i bridove grafa reprezentiramo kao objekte u obliku parova atribut-vrijednost. Da bi dobili Java-objekte tipa `Node` i `Edge` iz datoteke formata *JSON* koja sadrži strukturu grafa, koristimo Java biblioteku otvorenog koda *Gson* [5].
U nastavku je prikazan sadržaj datoteke *JSON* (kôd 5.3) grafa prikazanog na slici 5.2, uz popratno objašnjenje.

Listing 5.3: Primjer *JSON* datoteke

```
{
        "nodes":[
                {"name":"0","group":0},
                {"name":"1","group":1},
                {"name":"2","group":1},
                {"name":"3","group":1},
                {"name":"4","group":2},
```

```
            {"name":"5","group":2}
        ],
        "links":[
            {"source":0,"target":1,"value":1},
            {"source":0,"target":2,"value":1},
            {"source":0,"target":3,"value":1},
            {"source":0,"target":4,"value":1},
            {"source":0,"target":5,"value":1}
        ]
}
```

Datoteka sadrži polje `nodes` koje sadrži 6 objekata tipa `Node` i polje `links` koje sadrži 5 objekata tipa `Edge`.



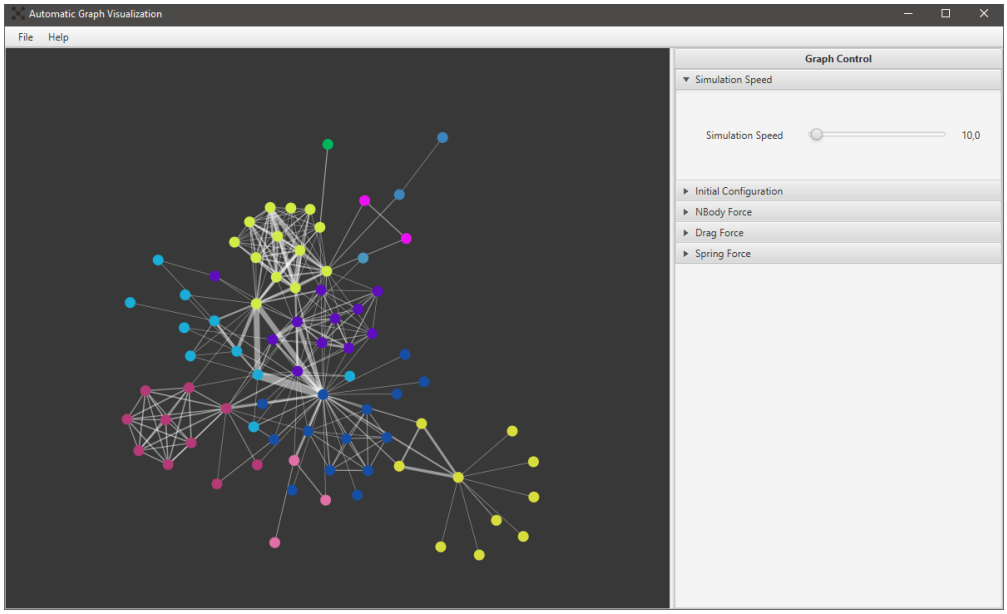
Slika 5.2: Prikaz grafa iz datoteke *JSON*

Objekti tipa `Node` sastoje se od dva atributa. Atribut `name` predstavlja ime vrha i njegova vrijednost je jedinstvena za svaki vrh grafa. Atribut `group` predstavlja grupu kojoj pripada vrh. Kod prikaza vrhovi jedne grupe obojani su istom bojom (slika 4.15).

Objekti tipa `Edge` sastoje se od tri atributa. Atributi `source` i `target` su objekti tipa `Node` koji su spojeni bridom, dok atribut `value` predstavlja težinu koja je dodijeljena bridu, kod neusmjerenog grafa težine svih bridova jednake su 1.

5.4. Grafičko korisničko sučelje

Za izradu grafičkog korisničko sučelje korišten je alat *JavaFX Scene Builder*, koji generira datoteku formata *FXML*. Kod pokretanja aplikacije čita se sadržaj datoteke te dobijemo vršnu grafičku komponentu koja unutar sebe sadrži grafičke komponente koje smo definirali alatom *JavaFX Scene Builder*. Nakon toga u sredinu te komponente stavimo grafičku komponentu s prikazom grafa koju smo izradili programski.



Slika 5.3: Izgled grafičkog korisničkog sučelja

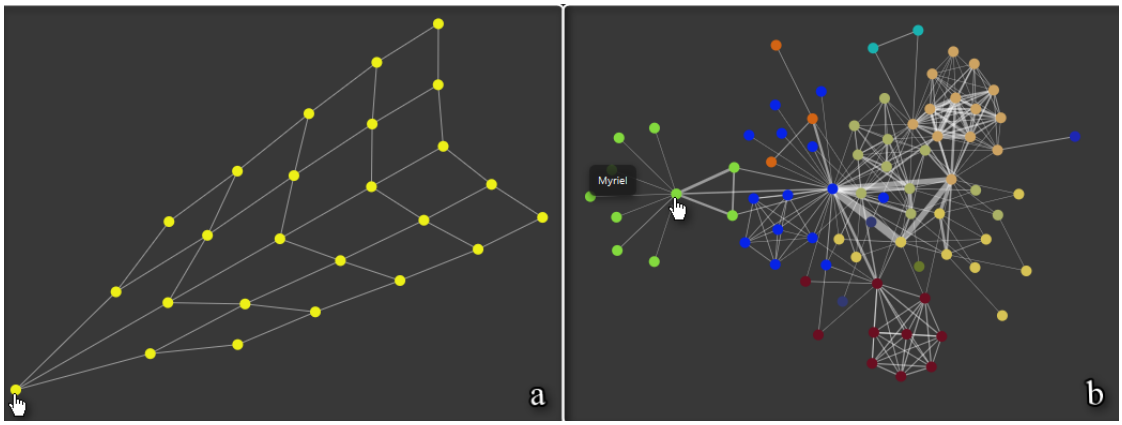
Grafičko korisničko sučelje prikazano je na slici 5.3 te se sastoji od 3 bitna dijela opisana u nastavku.

5.4.1. Izbornička traka

Izbornička traka sastoji se od dva izbornika *File* i *Help*. Izbornik *File* sadrži dvije izborničke stavke *Open* i *Restart*. Pritiskom miša na izborničku stavku *Open* otvara se prozor u kojemu odaberemo *JSON* datoteku s definicijom strukture grafa kojeg želimo prikazati, dok pritiskom na izborničku stavku *Restart* se ponovo pokreće crtanje grafa te se parametri sila postavljaju na početne vrijednosti. Izbornik *Help* nudi jednu izborničku stavku *About*, koja otvara prozor na kojem su navedena imena autora i mentora.

5.4.2. Komponenta prikaza grafa

Grafička komponenta na kojoj se nalaze vrhovi i bridovi grafa implementirana je u razredu `ForceDirectedLayout`. Zbog stalnog izvođenja računalne simulacije razvoja grafa, čvorovi i bridovi su prikazani dinamički. To omogućava korisniku stalnu interakciju s prikazanim grafom. Npr. korisnik može mišom povući neki od vrhova grafa (slika 5.4a) ili pritiskom miša na pojedini vrh se prikazuje ime tog vrha (slika 5.4b).



Slika 5.4: Primjeri korisnikovih interakcija mišom na grafom

Ostale akcije koje su podržane ovom komponentom su:

- uvećavanje / smanjivanje te
- translacija prikaza grafa (lijevo / desno / gore / dolje).

5.4.3. Izbornik Graph Control

Izbornik *Graph Control* sadrži izborničke stavke od kojih u jednom trenutku samo jedna može biti otvorena. Mijenjanjem postavki unutar izborničkih kartica izravno utječemo na način izvođenja računalne simulacije te na sile koje djeluju između vrhova grafa.

Sadržane su sljedeće izborničke stavke:

- *Simulation Speed* – utječe na brzinu izvođenja simulacije,
- *Initial Configuration* – odabiremo vrstu početnog pozicioniranja vrhova grafa,
- *Nbody Force* – postavke parametara sile implementirane razredom `NbodyForce`,
- *Drag Force* – postavke parametara sile implementirane razredom `DragForce`,
- *Spring Force* - postavke parametara sile implementirane razredom `SpringForce`.

6. Zaključak

Vizualizacija grafova često je korištena kod analize različitih vrsta podataka iz razloga da bi se prikazale veze između podataka koje često, nama ljudima, nisu shvatljive iz samih podataka. Graf se sastoji od skupa vrhova koji su međusobno povezani bridovima. Postoje brojni načini na koji možemo prikazati graf, međutim da bi veze između vrhova grafa bile bolje razumljive potreban nam je dobar raspored vrhova grafa.

Postoje različite vrste algoritama koji automatski raspoređuju vrhove grafa na ekranu s ciljem da ostvare dobar raspored vrhova. U ovome radu bavimo se algoritmom koji za raspoređivanje vrhova grafa koristi simulaciju međudjelovanja sila. Osim estetski prihvatljivog rasporeda vrhova grafa koji je rezultat takvog algoritma, ostale prednosti su laka razumljivost i jednostavnost implementacije.

Razmatranjem algoritma Fruchterman-Reingold objasnili smo nedostatak takvog algoritma, koji se javlja kada bi htjeli korisniku omogućiti interakciju sa simulacijom grafa u stvarnom vremenu. Problem se javlja zbog kvadratne složenosti kod izračuna međudjelovanja sila između vrhova, što kod posebno dolazi do izražaja kod prikaza većih grafova.

Kod implementacije alata za vizualizaciju grafova u ovom radu, za problem izračuna sila između vrhova grafa koristili smo algoritam Barnes-Hut, koji taj problem rješava u linearno-logaritamskoj složenosti. Također prije pokretanja simulacije razmještanja vrhova tehnikom simuliranog kaljenja tražili smo povoljniji početni raspored vrhova grafa, da bi ubrzali pronalazak konačnog prihvatljivog raspored.

Algoritam za raspoređivanje vrhova grafa vođen simulacijom sila često je korišten za implementaciju alata za vizualizaciju grafova. Kod analize velikih količina podataka zahtjeva se da takvi alati podrže dinamičan prikaz većih grafova. Time se stalno teži pronalasku novih metoda kojima bi se ubrzao rad tih algoritama, zbog čega je ovo područje i danas aktivno.

LITERATURA

[1] Hrvoje Ban. *Paralelna simulacija gibanja nebeskih tijela*. Fakultet elektrotehnike i računarstva, 2010. URL `https://bib.irb.hr/datoteka/476947.Ban_Hrvoje_zavrsni-rad.pdf`.

[2] Mike Bostock. Force-Directed Graph, 2013. URL `http://bl.ocks.org/mbostock/4062045`.

[3] T. Fruchterman i E. Reingold. *Graph drawing by force-directed placement*. Software Practice and Experience, 1991.

[4] Jeffrey Heer, Stuart K. Card, i James A. Landay. *prefuse: a toolkit for interactive information visualization*. the prefuse visualization toolkit, 2011. URL `http://vis.stanford.edu/files/2005-prefuse-CHI.pdf`.

[5] Google Inc. Gson Java library, 2008. URL `https://github.com/google/gson`.

[6] Wallace Jackson. *Beginning Java 8 Games Development*. Apress Media LLC New York, NY 10013, 2014.

[7] Patrick Jahnichen. *Finding and Analyzing Social Networks in unstructured web log data using probabilistic topic modeling*. University of Leipzig, Germany, 2010. URL `http://amor.cms.hu-berlin.de/~jaehnicp/pdf/master_thesis.pdf`.

[8] S. G. Kobourov. *Force-Directed Drawing Algorithms*. CRC Press, 2013.

[9] Johannes F. Kruiger i Maarten L. Terpstra. *Hooking up forces to produce aesthetically pleasing graph layouts*. Rijksuniversiteit Groningen, 2015. URL `https://www.academia.edu/12423255/Hooking_up_forces_to_produce_aesthetically_pleasing_graph_layouts`.

[10] Pavčević Mario. *Uvod u teoriju grafova*. Knjižnica FER-A, Fakultet elektrotehnike i računarstva, 2006.

[11] Rod Nave. *Air Friction*. Department of Physics and Astronomy, Georgia State University, 2000. URL http://hyperphysics.phy-astr.gsu.edu/hbase/airfri.html.

[12] Ivan Stojić. *Vizualizacija kompleksnih mreža u kontekstu sljedivosti evolucije informacija u razvoju proizvoda*. Prirodoslovno-matematički fakultet, 2012. URL http://www.unizg.hr/rektorova/upload_2012/Ivan%20Stojic%20-%20Vizualizacija%20kompleksnih%20mreza.pdf.

[13] Ventimiglia Tom i Wayne Kevin. *The Barnes-Hut Algorithm*. Department of Computer Science, Princeton University, 2003. URL http://arborjs.org/docs/barnes-hut.

[14] Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike*. ZEMRIS Fakultet elektrotehnike i računarstva, 2013.

Automatska vizualizacija grafova

Sažetak

U ovome radu opisan je algoritam korišten za automatsku vizualizaciju grafova koji vrhove grafa raspoređuje na temelju simulacije međudjelovanja sila. Objašnjen je algoritam Fruchterman-Reingold te tehnika simuliranog kaljenja koju algoritam koristi kako bi postigao minimizirao energiju sustava vrhova grafa, s ciljem da se postigne prihvatljiv raspored vrhova grafa. Na temelju rezultata za različite vrste grafova, s obzirom na broj vrhova, objašnjen je problem kvadratne vremenske složenosti kod izračuna sila između vrhova grafa kod tog algoritma.

Od mogućih rješenja objasnili smo algoritam Barnes-Hut za izračun sila između vrhova grafa te smo koristili tehniku simuliranog kaljenja kako bi ostvarili što povoljniji početni raspored vrhova grafa, da bi ubrzali pronalazak konačnog prihvatljivog raspored. Koristeći navedena rješenja napravljena je vlastita implementacija alata za vizualizaciju grafova. Alat omogućava korisniku interakciju s grafom, tijekom simulacije međudjelovanja sila na vrhove grafa, u stvarnom vremenu.

Ključne riječi: graf, vrhovi, bridovi, sile, algoritam, Fruchterman-Reingold, simulirano, kaljenje, sustav, n-tijela, Barnes-Hut, interakcija

Automatic Graph Visualization

Abstract

In this bachelor's thesis force-directed algorithm is described for automatic graph visualisation. We described Fruchterman-Reingold algorithm and its usage of simulated annealing to minimize the energy of the system and to achive better vertex layout. Based on results for various graphs, considering number of vertices, we discuss the problem of force calculation among vertices which is executed in quadratic time complexity.

We considered Barnes-Hut algorithm for speeding up the force calculation between vertices and we used simulated annealing in order to achieve better inital graph layout, before we actually run the force-directed algorithm. Using these approaches we implemented our own graph visualisation tool which supports real-time interaction with graph during the layout process.

Keywords: graph, vertices, edges, forces, force-directed, algorithm, Fruchterman-Reingold, simulated, annealing, system, n-body, Barnes-Hut, interaction