SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Frane Jakelić

INTERAKTIVNA VIZUALIZACIJA DRUŠTVENIH MREŽA

ZAVRŠNI RAD

Varaždin, 2011.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Frane Jakelić
Redoviti student
Broj indeksa: 35970/07-R
Smjer: Informacijski sustavi
Preddiplomski studij

INTERAKTIVNA VIZUALIZACIJA DRUŠTVENIH MREŽA

ZAVRŠNI RAD

Mentor:
Dr. sc. Markus Schatten, Viši asistent

Varaždin, rujan 2011.

Sadržaj

Popis slika

1. Uvod

Analiza i vizualizacija društvenih mreža su u današnje vrijeme doživjele veliki porast posebno pojavom raznih društvenih mreža na Internetu koje omogućavaju socijalnu interakciju između korisnika. Pojave tih mreža uvelike su olakšale istraživačima pribavljanje prije nezamislivih količina podataka o socijalnoj interakciji između osoba .Željom da se istraže podaci o interakciji između tih sudionika popularizirala se znanost koja ih kvantificira i izvlači relevantne informacije o strukturi te mreže i o sudionici koji se nalaze unutar nje.

Kroz ovaj rad pozabaviti ćemo se vizualizacijom društvenih mreža. Kao uvod u temu rada započinjemo objašnjavanjem osnovnih pojmova vezanih uz teoriju grafova koji će biti korišteni u izradi aplikacije, koja dolazi kao sastavni dio ovog završnog rada. U kratkom opisu najvažnijih pojmova dotaknuti ćemo se definicije grafa, bridova i čvorova te planarnosti samog grafa. Ti osnovni pojmovi će se pokazati veoma relevantnim za vizualizaciju grafova, jer planarnost grafa je jedna od glavnih karakteristika grafa koja se uzima prilikom vizualiziranja grafova, jer grafovi sa što manjim presijecanjem bridova generalno daju ljudskom oku ljepšu sliku.

Nakon kratkog uvoda u teoriju grafova nastavljamo s poviješću nastanka društvenih mreža i razvoju metoda koje su sada glavni alati u analizi i vizualizaciji društvenih mreža, naposljetku su obrađena tri algoritma za vizualizaciju društvenih mreža i grafova općenito. Uzeta su tri algoritma kako bi se moglo prikazati dva različita pristupa istoj vrsti vizualizacije i treći algoritam kao alternativni način prikaza grafova.

Kao dodatak radu opisuje se ukratko implementacija i izrada aplikacije i dani su primjeri aplikacije u upotrebi. Aplikacija je izrađena uz pomoć programskog jezika Javascript i HTML5 <canvas> elementa.

2. Teorija grafova

Definicija 1 *Graf je uređeni par G = (V , E) koji se sastoji od skupa čvorova* (eng. Vertex ili Node) $V = \{v_1, v_2, v_3...v_n\}$ *i skupa bridova* (eng. Edge ili Line) $E = \{e_1, e_2, e_3...e_n\}$.

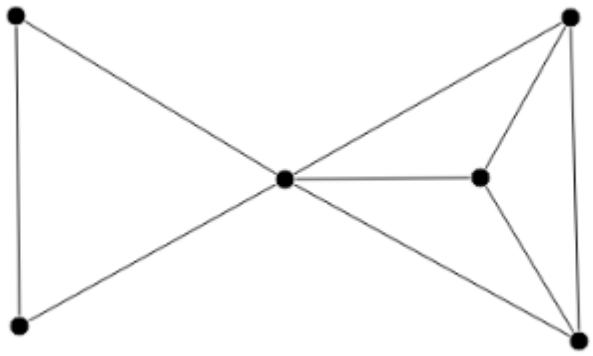
Za koje vrijedi[1]

$$E \subseteq V^2$$

$$E \cap V = 0$$

$$V, E \neq \{\emptyset\}^2$$

Svaki brid se nadalje sastoji od dvoelementnog podskupa (para) od V, koji opisuje vezu između dvaju čvorova $e_n = (v_i, v_j)$.



Slika 2.1: Primjer grafa

2.1. Susjedni čvorovi i bridovi

Definicija 2 *Za čvorove* (v_i, v_j) *kažemo da su susjedni*(eng. adjacent) *ako se između njih nalazi brid koji ih povezuje, a susjedni bridovi su oni koji imaju zajedničku krajnju točku (zajednički čvor).*

Na temelju informacije o susjednim čvorovima i bridovima izrađujemo matricu susjedstva.

[1]Diestel R., Graph Theory 3ed, Hamburg, Springer, 2005. str. 84-86
[2]Za potrebe rada na socijalnim mrežama ne bi imalo smisla raditi s grafovima bez čvorova ili veza među njima

2.2. Matrica susjedstva(pridruženosti)

Matrica susjedstva *(eng. Adjacency matrix)* je jedan od najjednostavnijih načina za prikazivanje odnosa između čvorova, matrica informaciju prenosi na način da su redni brojevi čvorova ispisani na horizontali i vertikali i prelazeći kroz polja u matrici zapisujemo 1 ako je došlo do podudaranja, a nulu ako nema podudaranja između čvorova.[3]

$$A_{i,j} = \begin{cases} 1 & \text{ako postoji brid između čvorova i i j} \\ 0 & \text{ako nema brida} \end{cases}$$

Matrica susjedstva za primjer sliku 2.1:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Matrica susjedstva se koristi kao objekt u kojem se spremaju informacije o grafu u računalnom programu. Prilikom izrade aplikacije koja je dio ovog završnog rada korištena je prilagođena matrica koje prenosi istu informaciju, sparse matrica koja u sebi sadrži samo polja koja nose jedinstvene informacije. Tj. samo polja u kojima je nastalo podudaranje.

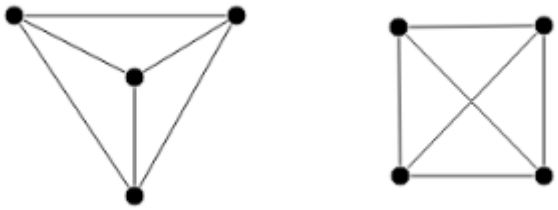
2.3. Planarnost grafa

Definicija 3 *Planarni graf je graf koji može biti prikazan na površini bez presjeka bridova.*[4].

Planarnost je jedna od najbitnijih karakteristika vizualizacije grafova jer se prilikom *"raspetljavanja"* grafova mogu razotkriti informacije o strukturi mreže koje su do tada bile skrivene.

Oba primjera u slici 2.1 su planarna jer planarnost po definiciji zahtjeva da se graf može nacrtati planarno ali i ne određuje da mora biti iscrtan na taj način. Planarnost je svojstvo grafa i on ju zadržava nebitno o njegovom prikazu. Dokazivanje planarnosti grafa u najjednostavnije obliku je crtanje grafa i naknadno razmještavanje čvorova kako bi se dobio graf koji je planaran.

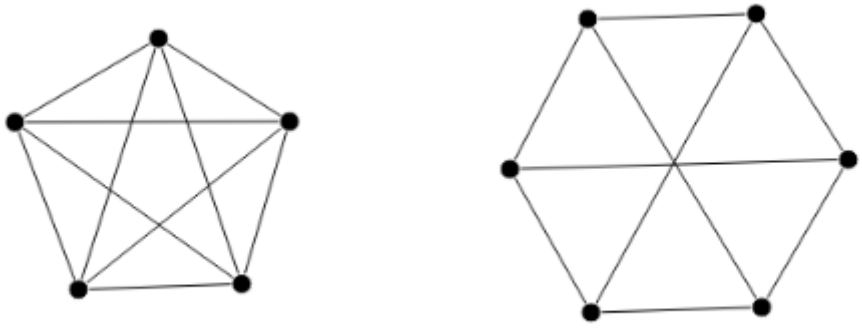
[3]Newman M.E.J., Networks an Introduction, New York, Oxford university press, 2010, str. 110-112
[4]Newman M.E.J., Networks an Introduction, New York, Oxford university press, 2010, str. 129-133



Slika 2.2: Primjer planarnih grafova

Gornja metoda je u najmanju ruku nepraktična, stoga je razvijena metoda za dokazivanje planarnosti većih grafova, koja se temelji na uzimanju manjeg grafa koji je dokazano ne planaran i tražeći njega u grafu čiju planarnost ispitujemo. Dođe li do podudaranja jednoznačno možemo zaključiti da graf nije planaran. Jedan takav primjer je K5 ili UG graf[5]. Na temelju K5 grafa je formuliran Kuratowskijev teorem:

Definicija 4 *Svaki ne planarni graf se sastoji od barem jednog podgrafa koji je ekspanzija K5 ili UG grafa.*



Slika 2.3: K5(lijevo) i UG(desno)

Ali također planarnost grafova u analizi i vizualizaciji društvenih mreža treba uzeti s zadrškom jer u realnom svijetu niti jedna mreža nije u potpunosti planarna, nego njezin veliki dio je planaran stoga to mrežu ne treba odbaciti kao ne planarnu. Stoga se praksi često uzima približna planarnost ovisno o željama istraživača.

[5]U teoriji grafova K_n označava potpuni graf s *n* čvorova. UG predstavlja "Utility graph" koji je potpuni bipartitni graf.

2.4. Tipovi grafova

U slučaju jednostavnog (*eng. simple*) grafa[6] vrijedi:

1. Svaka instanca brida je jedinstvena
2. Svi bridovi su jednaki zbog neuređenosti skupa E

   $e_j = (v_i, v_j) = (v_n, v_k)$
3. Izuzimaju se refleksivne veze(petlje).

Dok kod složenih grafova se mogu pojaviti grafovi s višestrukim bridovima tj. multigrafovi. Ostale vrste grafova koje se također koriste u analizi i vizualizaciji društvenih mreža su:

1. Jednostavni graf
2. Multigraf
3. Bipartitni graf
4. Stabla
5. Hipergrafovi
6. i još mnogi drugi

[6]U modernoj teoriji grafova ako nije posebno naznačeno pod pojmom graf se podrazumijeva jednostavni neusmjereni bezteţinski graf.

3. Kratka povijest društvenih mreža

Definicija 5 *Društvene mreže su mreže u kojima su čvorovi ljudi, ili grupe ljudi, a bridovi neka vrsta socijalne interakcije između ljudi, kao što je prijateljstvo među njima.*[1]

Povijest društvenih mreža seže do davnih vremena, ne kao definirana znanost nego kao jedno od područja zanimacije sociologa. Kroz sljedeće poglavlje biti će spomenuti relevantni znanstvenici koji su postavili temelje znanosti društvenih mreža, također i ljudi koji su pomogli je oblikovati u oblik kakav imamo u sadašnje vrijeme.

3.1. Karinthy

Prvim tekstom koji se bavi znanošću društvenih mreža se smatra kratka priča pod nazivom Chains; Chains se nalazi u Karinthyjevom djelu Everything is Different izdanom 1929 godine. On se u svom djelu bavi pojavom "smanjenja" svijeta, pod smanjenjem ne smatra tehnološki napredak koji omogućuje lakše prelaženje velikih udaljenosti, nego on promatra socijalni aspekt.

On u svom djelu daleko ispred svog vremena predstavlja problem sve veće socijalne povezanosti između ljudi na svijetu i na temelju svog djela definira pojam pet poznanika do bilo koga na svijetu. Pod time misli da se može između bilo koje dvije osobe na svijetu stvoriti veza koja maksimalno prolazi kroz pet ljudi.

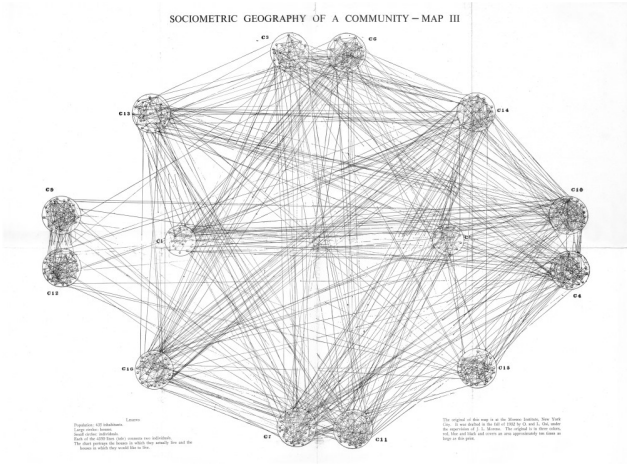
Kako bi poduprio svoju tvrdnju on u djelu opisuje kako je moguće povezati njega i bilo kojeg Nobelovog dobitnika vezom od maksimalno pet poznanika, također kako bi izbjegao kritike na račun odabira testnih subjekata on je također uspjeo povezati sebe preko pet osoba do radnika u Fordovoj tvornici[2].
Tek dvadesetak godina kasnije Milgram se bavio eksperiment malog svijeta i pet stupnjeva međusobne udaljenost,i problemima koji su nastali kao posljedica Karinthyjevog djela Chains.

[1] Newman M.E.J., Networks an Introduction, New York, Oxford university press, 2010, str. 36-46
[2] Newman M.E.J,The structure and dynamics of Networks, New Jersey, Princeton university press, 2006, str. 21-27

3.2. Moreno

Jacob Moreno se smatra kao glavnom osobom koja je postavila temelje za modernu znanost o društvenim mrežama, ta zasluga mu je pripala nakon što je 1933. godine izdao članak koji se bavio dinamikom socijalnih interakcija između grupa ljudi.

Članak je po prvi put objavljen na znanstvenoj konferenciji u New Yorku 1933 godine i to se smatra prvijencom na području i analize društvenih mreža. U svom radu promatrao je interakcije između učenika i učenica unutar razreda, promatrani su razredi od vrtića do osmog razreda. Svakom djetetu je bilo zadano da između svojih kolega izabere osobu s kojom bi najradije želio sjediti i nastavit biti u istom razredu.



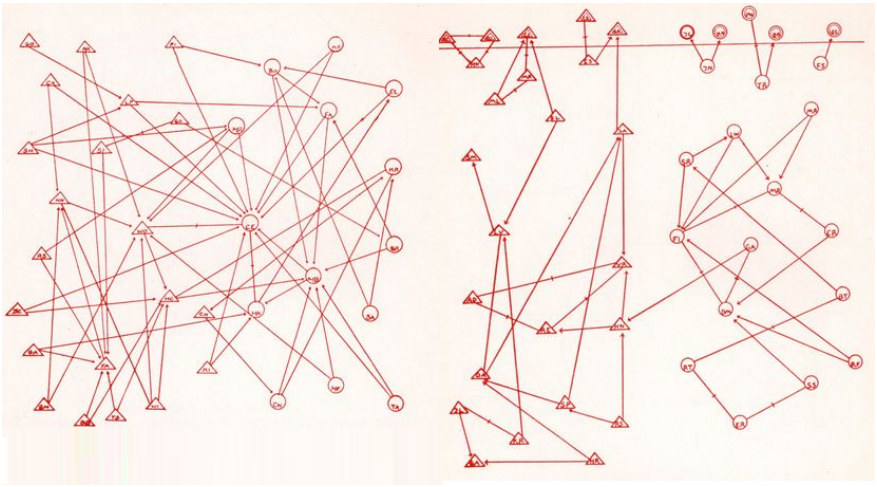
Slika 3.1: Primjer sociograma[3]**.**

Na temelju kvantitativne analize izvršene na dobivenim rezultatima otkrivena je kompleksna struktura unutar razreda, koja se uvelike razlikovala od onoga što je oku bilo vidljivo. Moreno je tada uočio pojave različitih oblika interakcije između učenika, neki učenici su bili izolirani, dok su drugi formirali dvosmjerne veze (parove), trokute ili lance interakcija, dok su neki privukli toliku pozornost na sebe da su se našli u sredini pozornosti kao zvijezde. Jedan od izrađenih dijagrama je prikazan na Slici 3.2[4] i prilikom izrade ovih dijagrama Moreno je skovao pojam sociometrija koji je nadalje proširio u svojoj knjizi koja je izdana 1934 pod imenom *Who shall survive?*.

Sociometrija je matematička studija psiholoških kvaliteta populacija, eksperimentalna tehnika koja je rezultat primjene kvantitativnih metoda. Poduhvat koji obuhvaća metode koje zadiru u

[3]Moreno J., Who shall Survive,New York, Beacon House, 1934., Mapa III
[4]Moreno J., Who shall Survive,New York, Beacon House, 1934., str. 160

evoluciju i organizaciju grupe i samih pozicija koje individualci zauzimaju u njima.[5]

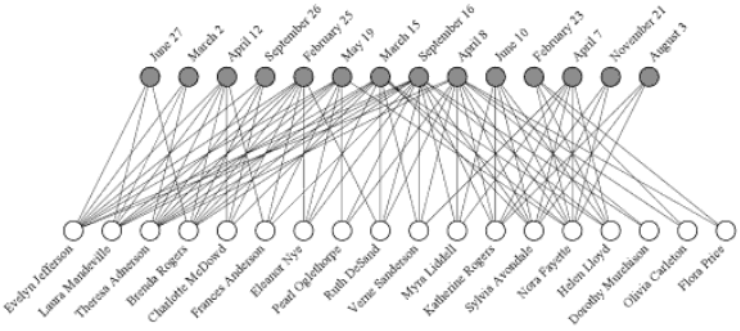


Slika 3.2: Struktura vrtića(lijevo) i petog(desno) razreda prilikom Morenovog istraživanja

3.3. Souther Women Study

Također prilikom predstavljanja povijesti mreža važno je spomenuti članak Davis, Gardner, Gardner, 1941, *"Souther Women Study"* u kojem je tema istraživanja bilo 14 žena i njihovo prisustvovanje javnim događanjima. Temelj istraživanja je bilo da se zaključi koje žene su najbolje povezane tj. koje žene pohađaju najveći broj istih događanja.

Njihovo prikazivanje navika ovih 14 žena je uvjetovalo razvoju socijalnih mreža i metoda koja izvače relevantne informacije iz njih.



Slika 3.3: Southern Women Study[6]

[5]Moreno J., Who shall Survive,New York, Beacon House, 1934., str. 51
[6]Davis, A., Gardner, B. B. and M. R. Gardner, Deep South, Chicago, The University of Chicago Press, 1941.

3.4. Solomonoff i Rapoport

Solomonoff i Rapoport u svom članku iz 1951 po prvi put sistematski opisuju nasumični graf i također na temelju njega definiraju mnoge karakteristike modela.
Definiraju pojavu prelaska grafa iz skupine nepovezanih čvorova u povezano stanje koje se u modernoj teoriji grafova naziva divovska komponenta, taj prelazak se događa nakon povećanja omjera između broja bridova i čvorova i pojavu slabe veze između čvorova koja definira broj čvorova prema kojima postoji put od nasumično odabranog čvora.

Mnoga od njihovih otkrića su postavila temelje za modernu teoriju grafova i analizu mreža.

3.5. Pool i Kochen

Poolov i Kochenov članak postavlja temelje za takozvani eksperiment malog svijeta, njihov članak je uvelike utjecao na Milgrama koji je kasnije proveo eksperiment koji se bavi upravo tematikom opisanom u ovom članku.
Pool i Kochen svoj članak počinju predstavljanjem skupa pitanja koji se bavi stupnjem osobe u mreži. Količinom njegovih poznanika,prosječnom vrijednosti broja poznanstava osoba koje se nalaze u mreži i naposljetku koja je struktura mreže poznanstava.

Ali najbitniji segment u njihovom djelu je istraživanje koje se bavi interakcijom između dvaju ljudi u mreži poznanstava. Oni postavljaju pitanja koja je vjerojatnost da se dvoje nasumično odabranih osoba u mreži poznaju, u slučaju da se ne poznaju koja je vjerojatnost da imaju zajedničkog poznanika i naposljetku koji je najmanji lanac poznanika koji ih povezuje. Njihovo istraživanje se danas u modernoj teoriji mreža opisuje kao eksperiment malog svijeta.
U daljnjem tekstu koji opsuje Milgramova istraživanja ćemo detaljno moći uvidjeti važnost ovog eksperimenta i mreže poznanstava koji on opisuje.

3.6. Travers i Milgram

Milgram je najpoznatiji po svojim istraživanjima na temu eksperimenta malog svijeta. Jedno od najvažnijih članaka u kojima po prvi put spominje eksperiment malog svijeta je izdan u 1967 u časopisu Psychology today. Milgram je provodi veliki broj eksperimenata ali oni najrelevantniji su nastali nakon što je započeo suradnju s Traversom, koji je provodio opsežnu kvantitativnu analizu

nad podacima koja je bila manjkava u dotadašnjim Milgraovim eksperimentima.

Eksperiment se provodio na sljedeći način, formular je poslan svim početnim članovima istraživanja i od njih je bilo zatraženo da pokušaju preko svojih poznanika dostaviti taj formular osobi definiranoj na formularu. Znači bilo je više startnih točaka ali svima je odredište bilo isto. I prilikom prosljeđivanja formulara sljedećoj osobi, od te osobe je bilo zatraženo da ponovi posao prethodnika tj. da ispuni potrebne podatke i da preko nekog od svojih poznanika pokuša taj formular dostaviti odredišnoj osobi.

Pri završetku eksperimenta 64 od 294 lanca su dostigla do odredišne osobe, što je rezultiralo s 29% uspješnosti. Na temelju izračuna duljina lanaca je varirala od 1 do 11 poznanika, dok je medijan provedenog istraživanja iznosio 5.2 i na temelju tog rezultat skovan je pojam šest stupnjeva međusobne udaljenosti, na temelju kojeg je1990 izvedena predstava na Broadway-u[7] koja se upravo bavila predstavljanje eksperimenta malog svijeta.

[7]M.E.J Newman,The structure and dynamics of Networks, New Jersey, Princeton university press, 2006, str. 130-149

4. Vizualizacija

Vizualizacija je grafičko prikazivanje podataka o mreži u prostoru. U sljedećem poglavlju će biti prikazana 3 algoritma za vizualizaciju grafova. Dva algoritma iz grane force-based layout algoritama, Fruchterman-Reingold algoritam koji je implementiran u aplikaciji koja je sastavni dio ovog završnog rada, također i Kamada-Kawai algoritam koji spada pod istu granu algoritama za vizualizaciju ali uzima malo drukčiji pristup, sadržava drugačiji pogleda na isti problem. I naposljetku k-cores dekompozicija koja će služiti kao primjer alternativnog načina vizualizacije grafova jer ne pripada pod granu force-based algoritama.

4.1. Force-based Layout

Force-based layout je skup algoritama koji za crtanje grafova preuzimaju analogiju iz stvarnog svijeta,analogiju u kojoj su čvorovi čelični krugovi a bridovi opruge među njima i naspram toj privlačnoj sili modeliraju i odbojnu silu generiranu međusobnim djelovanjem nabijenih čestica u ovom slučaju čvorova. U sljedeća dva paragrafa biti će definirani zakoni na kojima se temelji ovaj način vizualizacije grafova kako bi se lakše objasnila analogija u pozadini ovih algoritama.

Hookeov zakon

Definicija 6 *Hookeov zakon definira da produljenje opruge je direktno proporcionalno s količinom tereta koji se nalazi na njoj.*[1]

$$F = -kx$$

Hookeov zakon opisuje ponašanje opruga pod naporom, što se više tereta stavlja na oprugu ona se sve više razduljuje, ali u tom trenutku dolazi do izražaja Hookeov zakon jer svaka opruga sukladno gore navedenoj formuli stvara inverznu silu sili koja ju rasteže. Stoga što se više tereta stavlja na oprugu to Hookeove sile djeluju na oprugu i pokušavaju je vratiti u prvotno stanje, u stanje ekvilibrija. Svaka opruga ima svoje stanje ekvilibrija kada su sile koje djeluju na nju minimalne, prilikom dodavanja tereta na oprugu opruga sukladno izračunu dobivenim primjenom ovih sila dolazi do novog stanja ekvilibrija.

[1] Tega J., Hooke's Law, `http://www.universetoday.com/55027/hookes-law/`.

Coulombov zakon

Definicija 7 *Coulombov zakon opisuje među djelovanje električki nabijenih čestica*[2].

Zakon opisuje među djelovanje čestica ako su dvije čestice pozitivno(ili negativno) nabijene one stvaraju privlačnu silu dok čestice različitih polova se privlače.

$$F = k_e \frac{q_1 q_2}{r^2}$$

Što se veći naboji q_1, q_2 to su sila koja nastaju među njima jače. U slučaju jednakih polova što se tijela više udaljavaju to odbojne sile slabe i dolazi do ekvilibrija gdje su sile koja djeluju između čestica smanjene na minimum.

Prilikom implementacije ovih dvaju zakona u algoritam vizualizacije grafa često se uzimaju modificirane verzije ovih zakona jer potpuna sličnost sa stvarnim svijetom nije potrebna nego su potrebne sile koje nastaju primjenom ovih zakona.

Da nastavimo s analogijom Hookeov zakon se uzima kao privlačna sila da se čvorovi ne bi previše razdaljili pod utjecajem Coulombovog zakona. Coulombov zakon razdaljuje čvorove kako ne bi došlo do preklapanja i kako bi se mogao dostići što čišći,planarniji i estetski bolji graf a opruge sprječavaju preveliko razdaljivanje uzrokovano tim odbojnim silama.
Duljine opruga su predefinirane, te se Hookeov zakon koristi samo kada se pod utjecajem odbojnih sila ili interakcije korisnika promjeni duljina. Kada duljina opruge premaši predefiniranu duljinu svi čvorovi povezani s dotičnim čvorom počinju proizvoditi privlačnu silu koja "vuče" čvor nazad na predefiniranu udaljenost. Ekvilibrij unutar ovih algoritama se dostiže kada dođe do mirovanja sustava, kada su svi čvorovi na dostatnoj udaljenosti od ostalih čvorova i kada opruge nisu razdužene dulje od njihove predefinirane udaljenosti.
Relevantni algoritmi u ovom području su:

- Fruchterman-Reingold (Zasebno poglavlje)
- Kamada-Kawai (Opisan u daljnjem tekstu)
- Eadesov algoritam

[2]Wagon J., Coulomb's Law - The law of force, `http://regentsprep.org/Regents/physics/phys03/acoulomb/default.htm`

4.1.1. Kamada-Kawai

Kamada-Kawai također u svom algoritmu koristi takozvani model opruga. Povezuju sve čvorove "oprugama" i smatraju da će balansirani sustav opruga rezultirati balansiranim i estetski lijepim crtežom grafa. Formulu za trenutnu količinu energije u sustavu su formulirali kao potpunu energiju svih opruga koje međusobno vezuju čvorove.

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2}(|p_i - p_j| - l_{ij})^2$$

Duljina opruge l_{ij} je definirana kao duljina između dvaju čvorova p_i, p_j, ali najkraćeg puta između njih i željene duljine opruge L.

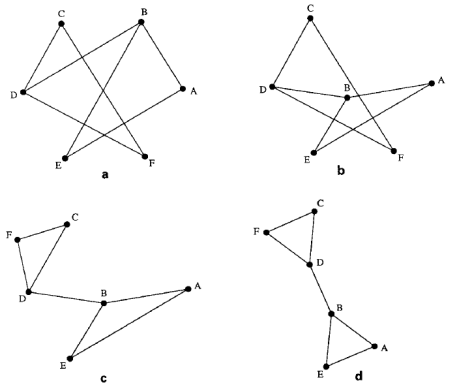
$$l_{ij} = L \times l_{ij}$$

Prema autorima estetski ljepši izgled grafa je direktno vezan s ukupnom energijom svih opruga što se energija smanjuje to je ljepši graf.

Pseudokod za algoritam definira sve potrebne korake za implementacija algoritma također nam prikazuje metodu odabira čvora ili čestice nad kojom će se izvršavati izračun. U pseudokodu je također navedeno da se prilikom izračuna svake promjene čvora tj pomicanja na novo stanje koristi Newton-Raphsonova metoda za izračun lokalnog minimuma. Kako je dokaz podosta velik on je izostavljen iz ovog rada.

```
izračunaj d_ij za 1 ≤ i ≠ j ≤ n;
izračunaj l_ij za 1 ≤ i ≠ j ≤ n;
izračunaj k_ij za 1 ≤ i ≠ j ≤ n;
inicijaliziraj p_1, p_2, ..., p_n;
dok je (max_i Δ_i > ϵ){
p_m čestica koja zadovoljava Δ_m = max_i Δ_i ;
dok je (Δ_m > ϵ){
x_m := x_m + δ_x;
y_m := y_m + δ_y;
}
}
```

[3] Kamada T., Kawai S., An algorithm for drawing general undirected graphs, University of Tokyo, Tokyo, 1989. str. 11



Slika 4.1: Primjer Kamada-Kawai algoritma u upotrebi[3]

4.2. Fruchterman-Reingold

Fruchterman-Reingold algoritam služi za vizualiziranje neusmjerenih grafova, temeljen je na Eadesovom algoritmu, koji je nastao od VLSI tehnike zvane force-directed *placement*.Prema Fruchtermanu i Reingoldu algoritam se temelji na riječima Eades:
Osnovna ideja je sljedeća. Za prikazivanje grafa čvorovi se zamjenjuju čeličnim krugovima a bridovi oprugama kako bi graf činio mehanički sustav. Čvorovi se postavljaju u nasumičnom rasporedu i "otpuštaju" se opruge između čvorova i sustav se pod utjecajem sila opruga polako postavlja u stanje minimalne energije .

Fruchterman-Reingold algoritam radi na dva principa:

1. Čvorovi spojeni bridom bi se trebali crtati blizu jedan drugome
2. Čvorovi se ne bi smjeli crtati preblizu jedan drugome

Što se tiče drugog principa, on uvelike ovisi o raspoloživom prostoru na kojem se crta graf i također o broju čvorova. Neki grafovi zbog svoje veličine su prekomplicirani za "lijepo" prikazivanje, stog jedno od najvažnijih pravila bi trebao biti da se čvorovi ne poklapaju tj da ne zauzimaju isto područje prilikom izrade izgleda grafa.

Prema uputstvima autora (uputstva su izražena pomoću pravila u fizici čestica):
Na udaljenost od 1fm stvara se snažna privlačna nuklearna sila snage 10 puta veće od sage između dvaju protona. Snaga eksponencijalno pada što se udaljenost povećava, sve dok dolazi do svog približnog minimuma na udaljenosti oko 15fm. Ali kada su dvije čestice na udaljenosti od 0.4fm stvara se snažna odbojna sila koja sprječava čestice da se uruše jedna u drugu.

U Fruchterman-Reingold algoritmu čvorovi se ponašaju kao čestice ili nebeska tijela, kao takvi oni proizvode privlačne i odbojne sile koje svojom interakcijom dovode do kretanja čvora i ovom slučaju do što planarnijeg izgleda grafa. Ovakav tip vizualizacije ili simulacije planetarnog,molekularnog sustava se zove problem n tijela.

Ali kako cilj ovog algoritma nije dosljedno reproduciranje problema n-tijela, autori su odlučili primijeniti samo sile koje su potrebne za ostvarenje dvaju gornjih principa i koje će dati zadovoljavajući izgled grafa. Stoga oni kao Eades privlačne sile primjenjuju samo na susjedne čvorove čvora koji se promatra, dok odbojne sile i dalje primjenjuju na sve čvorove. U svakom vremenskom intervalu izračunava se količina pomaka nastala primjenom ovih sila. Prilikom promatranja algoritma možemo zaključiti da ove sile ne proizvode akceleraciju jer bi finalni rezultat bio dinamički ekvilibrij, dok se u finalnom rezultatu traži statični, stoga ove sile izražavaju brzinu tj. količinu pomaka čvora u koordinatnom sustavu.

```
area := W * L; {duljina i širina okvira}
G := (V, E); {čvorovima se dodjeljuju nasumični početni položaji }
k :=√(area/|V|)
function f_a(z) := počni return z^2/k završi;
function f_r(z) := počni return k^2/z završi;
dok je i := 1 to broj_iteracija čini počni
{izračunaj odbojne sile}
dok je v iz V čini počni
{svaki čvor ima dva vektora .pos i .disp }
v.disp := 0;
dok je u iz V čini
ako je (u # v)onda počni
{D predstavlja razliku između dvaju vektora}
D := v.pos − u.pos;
v.disp := v.disp + ( D/|D|) * f_r(|D|)
završi
završi
{izračunaj privlačne sile }
dok je e iz E čini počni
{ svaki brid je uređeni par čvorova .v i .u }
D := e.v.pos − e.u.pos
e.v.disp := e.v.disp − (D/|D|) *f_a(|D|);
e.u.disp := e.u.disp + (D/|D|) * f_a(|D|)
završi
{ Ograniči maksimalni pomak na visinu temperature sustava}
{Spriječi izlazak izvan okvira grafa}
dok je v iz V čini počni
v.pos := v.pos + (v. disp/|v.disp|) * min (v.disp,t);
v.pos.x := min(W/2, max(−W/2, v.pos.x));
v.pos.y := min(L/2, max(−L/2, v.pos.y))
završi
{ Smanjuj temperaturu prilikom svake iteracije }
t := cool(t)
završi
```

U gornjem okviru je prikazan pseudo kod za Fruchterman-Reingold algoritam, izvorni pseudokod koji se nalazi u članku.

4.2.1. Optimalna duljina među povezanim čvorovima

Konstanta k služi za određivanje optimalne udaljenosti između čvorova, ona na temelju površine prostora za crtanje i količine čvorova koje treba iscrtati daje "optimalnu" duljinu brida između čvorova. Što se čvorovi više približavaju željenoj udaljenosti smanjuje se energija grafa i graf polagano dolazi do statičnog ekvilibrija.

$$k = \sqrt{\frac{površina}{|V|}}$$

4.2.2. Privlačna i odbojna sila unutar grafa

Funkcije f_a,f_r predstavljaju privlačnu i odbojnu sila koja se odvija u grafu među čvorovima.

$$f_a(z) = \frac{z^2}{k}$$

$$f_r(z) = \frac{k^2}{z}$$

Funkcije f_a,f_r su definirane kroz ekstezivno testiranje i kao takve najbolje opisuju jačine sila potrebnih za sprovođenje algoritma. Funkcija f_a je privlačna sila i ona je definirana kao kvadrat vektora pomaka kroz konstantu k, dok je funkcija f_r odbojna sila i ona je definirana kao kvadrat konstante k kroz vektor pomaka.

4.2.3. Iteracije i temperatura sustava

Algoritam se odvija predodređen broj iteracija, broj iteracija također direktno utječe na temperaturu sustava u kojoj se graf nalazi. Temperatura sustava radi na principu da izražava maksimalni pomak čestice uzrokovan silama koje djeluju na nju. Kretanje temperature utječe na jačinu promjena na grafu, što iteracije prolaze to se temperatura više smanjuje i na grafu nastaju finije promjene. Najosnovniji model postavlja temperaturu sustava na desetinu širine i linearnom funkcijom je smanjuje u svakoj iteraciji.

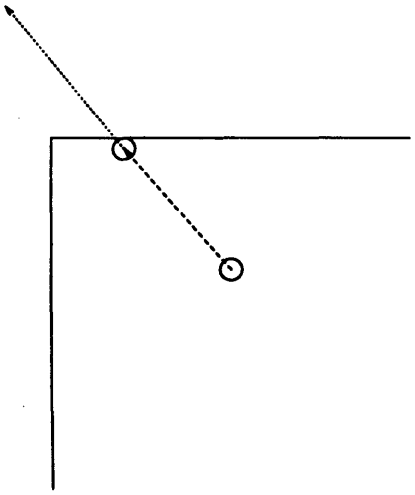
$$temperatura = temperatura * (\frac{1 - trenutna\_iteracija}{maximalna\_iteracija})$$

4.2.4. Okvir

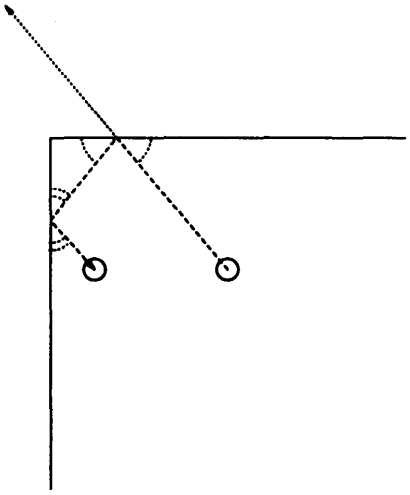
Kako je prostor u kojem se graf crta ograničen, nastao je problem što učiniti s čvorovima koje sila dovede do ruba okvira. Razmotrene su razne mogućnosti a neke od njih su da se nad čvorom primjeni neelastični sudar koji bi rezultirao povratnom putanjom čvora, tj. nad čvorom bi se

stvorile negativne sile koje bi ga vratile unazad.

Drugo ponuđeno rješenje je bilo da se čvor prilikom sudara s zidom ili rubom scene odbije

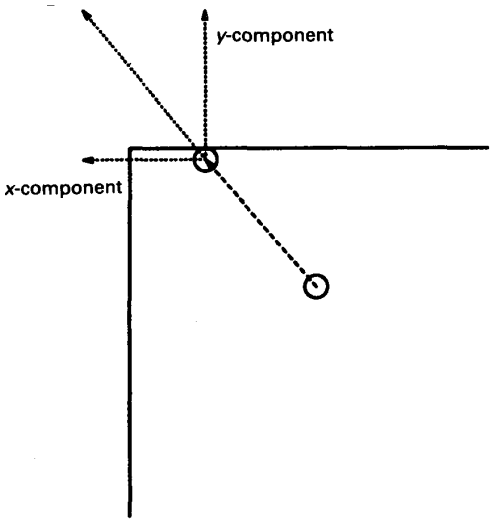


Slika 4.2: Neelastčni sudar[4]



Slika 4.3: Elastični sudar [5]

pod kutom odbijanja i nastavi svoju putanju, tj da njegovo ponašanje odgovara elastičnom sudaru. Dok je na kraju za rješenje odabran treći način koji opisuje takozvane ljepljive čvorove. Čvor prilikom sudara s okvirom se fiksira na mjesto udara i tu ostaje nepomično. Taj čvor prestaje biti pod utjecajem sila ali on je i dalje faktor u jačini sila prema njegovim susjedima.



Slika 4.4: Statični čvor[6]

[4]Fruchterman T. M. J., Reingold E. M., Graph Drawing by Force-directed Placement, Urbana, University of Illinois at Urbana-Champaig, 1991. str. 7

[5]Fruchterman T. M. J., Reingold E. M., Graph Drawing by Force-directed Placement, Urbana, University of Illinois at Urbana-Champaig, 1991. str. 7

[6]Fruchterman T. M. J., Reingold E. M., Graph Drawing by Force-directed Placement, Urbana, University of Illinois at Urbana-Champaig, 1991. str. 8

4.2.5. Vremenska kompleksnost

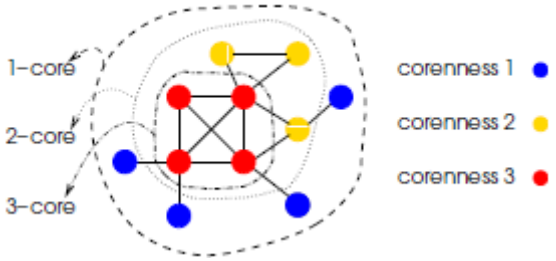
Kompleksnost algoritma je $\theta(|V|^2 + |E|)$. Također što se tiče kompleksnosti algoritma posebnu pozornost treba pridodati broju iteracija. Kao što se može vidjeti iz opisa algoritma i njegovog pseudokoda da broj iteracija nije definiran. Jer prilikom vizualiziranja grafa algoritam nikako ne može znati koji je potreban broj iteracija, pa stoga broj iteracija uvelike koristi o korisniku i njegovim preferencama. Neki grafovi manje složenosti mogu doći u stabilno stanje i u manje od 10 iteracija dok nekima treba mnogostruko više. Ustaljeni broj iteracija koji se koristi u praksi je 100, jer većina grafova nakon 100 iteracija dosegne svoje stabilno stanje i maksimalnu "atraktivnost".

Kompleksnost algoritma je smanjena odlukom tvoraca da se privlačne sile odražavaju samo na susjede a ne na sve čvorove dok se odbojne sile i dalje sprovode nad svim elementima. Također ako graf nije dosegao zadovoljavajući izgled u prvih 100 iteracija korisnik je uvijek u mogućnosti ponovno učitati graf i provesti daljnjih 100 iteracija kako bi imao više zadovoljavajući graf.

4.3. LaNetVi k-cores

Prema riječima autora[7]:
Koristimo k-core dekompoziciju kako bi prikazali kompleksne mreže velikih dimenzija u dvodimenzijalnom prostoru. Dekompozicija se temelji na rekurzivnom izuzimanju slabije vezanih čvorova kako vi uspjeli otpetljati hijerarhijsku strukturu koja se nalazi u mreži fokusirajući se na njezine jezgre.
Algoritam se temelji na određivanju jezgrovitosti određenih čvorova kreće se od najpovezanijih čvorova, čvorova koji imaju najveći red. Čvorovi s najvećom povezanosti se stavljaju u centar kao centralna jezgra i nakon toga na sljedeću razinu se postavljaju čvorovi s sljedećom razinom povezanosti i tako sve dok se ne postave svi čvorovi iz početnog grafa.

Metoda je najvidljivija ako pogledamo primjer u slici 4.1. U središnjoj jezgri imamo čvorove



Slika 4.5: Primjer raspodjele čvorova za k-cores algoritam[8]

jezgrovitosti 3, što znači da je svaki od tih čvorova povezan s tri ostala čvora, dok na sljedećoj razini imamo čvorove jezgrovitosti 2 itd. Na ovom primjeru možemo vidjeti osnovnu ideju ovog algoritma, neovisno o broju čvorova uvijek možemo rastaviti graf na razine ovisno o razini njihove jezgrovitosti. Gledajući na ovaj primjer odmah možemo uvidjeti koji su čvorovi veće važnosti za mrežu, koji su bolje povezani i imaju bolju interakciju s ostalim čvorovima zavisno o problematici koju istražujemo.
Promatrajući pseudokod algoritma možemo postepeno proći kroz sve korake i ukratko objasniti važnost svakog koraka.

```
ulaz: vektori jezgrovitosti C, klaster Q i T, indeksirani čvorom i
za svaki čvor i čini
ako je c_i == c_max onda
postavi ρ_1 i α_1 na temelju uniformne distribucije
u disku radijusa u (u predstavlja veličinu jezgre)
u suprotnom
postavi ρ_1 i α_1 prema jednadžbama definiranima u tekstu
vrati ρ i α vektore
```

[7] Alvarez-Hamelina I., Dall'Astaa L., Barrata A., Vespignani A., k-core decomposition: a tool for the visualization of large scale networks, Bloomington, 2005.
[8] Alvarez-Hamelina I., Dall'Astaa L., Barrata A., Vespignani A., k-core decomposition: a tool for the visualization of large scale networks, Bloomington, 2005. str. 3

k-core dekompozicija

Jezgrovitost svakog čvora je izračunata i spremljena je u skup C u njoj se nalaze sve jezgre i također na temelju među veza između jezgara i njezinih elemenata stvoren je skup Q_m^c koji obuhvaća klastere u kojima se nalaze čvorovi.

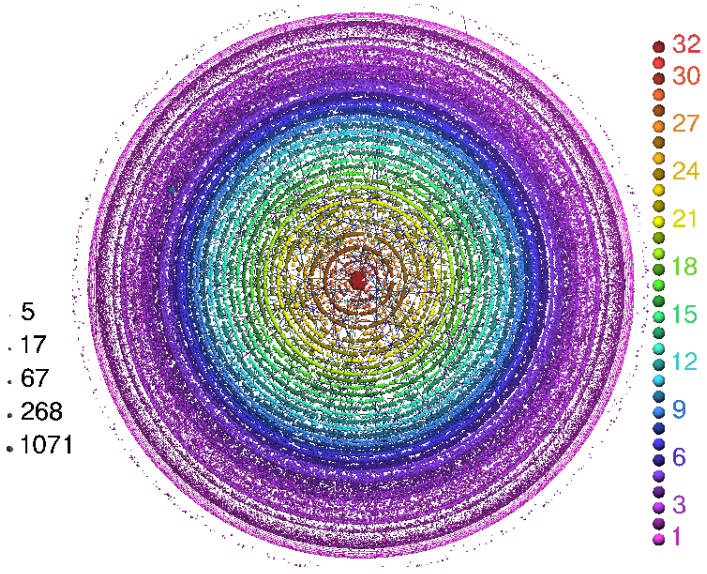
Dvodimenzionalni prikaz grafa

Čvorovi na grafu se crtaju pomoću polarnih koordinata (ρ_i, α_i). Polarne koordinate su odabrane za određivanje lokacija čvorova jer radijus ρ_i je funkcija jezgrovitosti čvora i i njegovih susjeda, dok kut α_i ovisi o rednom broju klastera u kojem se čvor nalazi.

$$\rho_i = (q - \epsilon)(c_{max} - c_i) + \frac{\epsilon}{|V_{cj \geq c1}(i)|} \sum_{jE|V_{cj \geq c1}(i)} (c_{max} - c_i)$$

$$\alpha_i = \sum_{1 \leq m \leq q_i} \frac{|Q_m|}{|Q_{c_i}|} + N(\frac{|Q_{q_i}|}{|Q_{c_i}|}, \pi * \frac{|Q_{q_i}|}{|Q_{c_i}|})$$

Što se tiče k-cores algoritma veoma je važno napomenuti da se pomoću kuta iz polarnih koordinata kreiraju klasteri koji označavaju jaku povezanost između čvorova na istoj razini ili povezanosti između klastera na različitim razinama. Ovaj algoritam predstavlja veoma brz i efikasan načina za vizualizaciju velikih i kompleksnih grafova u kratkom vremenu. Također informacije dobivene ovom vizualizacijom su hijerarhijski poredak čvorova i također popis klastera koje formiraju čvorovi.



Slika 4.6: Primjer za k-cores algoritam[9]

[9] Alvarez-Hamelina I., Dall'Astaa L., Barrata A., Vespignani A., k-core decomposition: a tool for the visualization of large scale networks, Bloomington, 2005. str. 11

5. Aplikacija

Prilikom odabira tehnologija za izradu aplikacije uzeto je u obzir brzina izvođenja algoritma za vizualizaciju i prenosivost aplikacije na druge platforme. Naposljetku uzeti su HTML5 <canvas> element koji je pogonjen Javascript programskim jezikom u pozadini.

Odabir ovih tehnologija je ponajviše uvjetovan na činjenici da svaki operacijski sustav sadržava Internet preglednik koji je u mogućnosti pogoniti ove dvije tehnologije i također izbor tih tehnologija postaje standardom u današnje vrijeme, također prenošenjem ove aplikacije na Internet se postiže puno veća publika za korištenje aplikacije za razliku od desktop aplikacije. O ispravnosti odabira ovih tehnologija će biti puno više u zaključku kada će biti govora o performansama ovog načina vizualizacije grafova.

Javascript programski jezik je od svog nastanka 1995 postao dominantni jezik za programiranje unutar Internet preglednika, on se temelji na prototipnoj paradigmi i nastao je na temelju programskih jezika Scheme i Self. Dok je HTML5 novijeg datuma, njegova popularnost je porasla kroz zadnje dvije godine i sada se trenutno koristi u velikom broju aplikacija iako sami standard za njega još nije definiran. HTML5 <canvas> element je statičan bez upotrebe Javascripta.

5.1. Struktura aplikacije

Kako Javascript ne podržava klasični namespace kao u ostalim programskim jezicima, izrađena su dva objekta koji predstavljaju odvojene namespace-ove.

```
1 var graph = {
2 //Programski kod
3 }
4 var algorithm = {
5 //Programski kod
6 }
7 graph.init(edges,typeof algorithm !== "undefined"? algorithm.algoritam : function(){});
```

U primjeru programskog koda možemo vidjeti način na koji se definira namespace pomoću objekta. Ova metoda je jedna od najčešće korištenih metoda za definiranje namespace-ova, također kao malo pojašnjenje potrebno je samo spomenuti da se svaki Javascript kod koji se izvršava u pregledniku se nalazi u window objektu koji je sastavni dio svakog Javascript engine-a. U prethodnom kodu nije prikazani pomoćni objekt koji sadržava sve operacije s vektorima koje olakšavaju primjenu algoritma i sve potrebne izračune unutar njega.

I naposljetku graph modul dopušta veliku razinu modularnosti tako da dozvoljava dodavanje pro-

zvoljnog dataseta u graf i također mogućnost prikazivanja grafa bez algoritma za vizualizaciju ili uvođenje prozvoljnog algoritma definiranog od strane korisnika.

5.2. Graph objekt

```
var graph={
init : function(edges,algorithm){ ... },
addNodesFromEdgesList : function(EdgesList){ ... },
addEdgesFromEdgesList : function(EdgesList){ ... },
node : function(node){ ... },
addNode : function(node){ ... },
removeNode : function(id){ ... },
getNumberOfNodes : function(){ ... },
getNumberOfEdges : function(){ ... },
edge : function(edge){ ... },
addEdge : function(edge){ ... },
removeEdgeByEdgeId : function(id){ ... },
removeEdgeByNodeId : function(id){ ... },
clearGraph : function(){ ... },
mapNodes : function(funkcija,obj){ ... },
mapEdges : function(funkcija){ ... },
dragNode : function(e){ ... },
addEdgeAnim : function(e){ ... },
nodeClickEvent : function(e){ ... },
nodeDropEvent : function(e){ ... },
getNodeFromXY : function(_x,_y){ ... },
draw : function(){ ... },
dblclick : function(e){ ... },
info : function(){ ... },
hookes : function(v1,v2,id){ ... },
oDuljina : function(v1,v2,id){ ... }
}
```

Unutar objekta Graph definirane su sve metode potrebne za upravljanje izgledom grafa, dodavanjem i izuzimanjem čvorova i bridova, definirane su metode za pomicanje čvorova, detekciju odabira čvora. Također radi vizualno ljepšeg izgleda dodana je slaba privlačna sila prema Hookeovom zakonu, koja omogućava povlačenje čvorova s što većom realnošću.Pod utjecajem Hookeovog zakona pomicanjem jednog čvora ostali prate smjer kretanja tog čvora i stvaraju izgled međusobne povezanosti između čvorova.

Izrađen je kompletan API kako bi se u budućnosti mogao koristiti i za daljnji razvoj aplikacije i dodavanje novih mogućnosti.U kodu koji je dostupan kao dodatak ovom radu definirane su i mnoge metode koje olakšavaju upravljanjem grafa.

U Graph objektu kao najvažniji dio se može izdvojiti dva konstruktora node i edge koji omogućavaju kreiranje objekata s proizvoljnom količinom atributa i samim time ekspanzija za druge vrste grafova je uvelike olakšana.

5.3. Algorithm objekt

```
var algorithm={
init :function(){...},
privlacneSile : function ( z ){...},
odbojneSile : function ( z ){...},
repulsion: function(v1){...},
attraction : function (v1,v2){...},
zavrsetak : function(v1){...},
algoritam : function (){...},
restart : function (e){...}
}
```

Algorithm objekt sadržava sve metode potrebne za izvođenje definiranog algoritma, on uvelike ovisi o graph objektu i njegovim metodama. Aplikacija je izrađena na način da algorithm objekt ovisi o graph objektu ali povratna veza ne postoji, stoga u teoriji korisnik mora samo napisati algoritam i povezati ga s graph objektom i sve bi trebalo besprijekorno trebalo raditi.

Što se tiće implementiranog algoritma, on se u nekim sastavnim dijelovima razlikuje od algoritma definiranog u izvornom članku. Većinom se to radi o sitnim korekcijama parametar kako bi se dobio vizualno ljepši izgled algoritma. Ali glavna korekcija koju treba spomenuti je izračun optimalne duljine bridova.

Izvorna vrijednost:

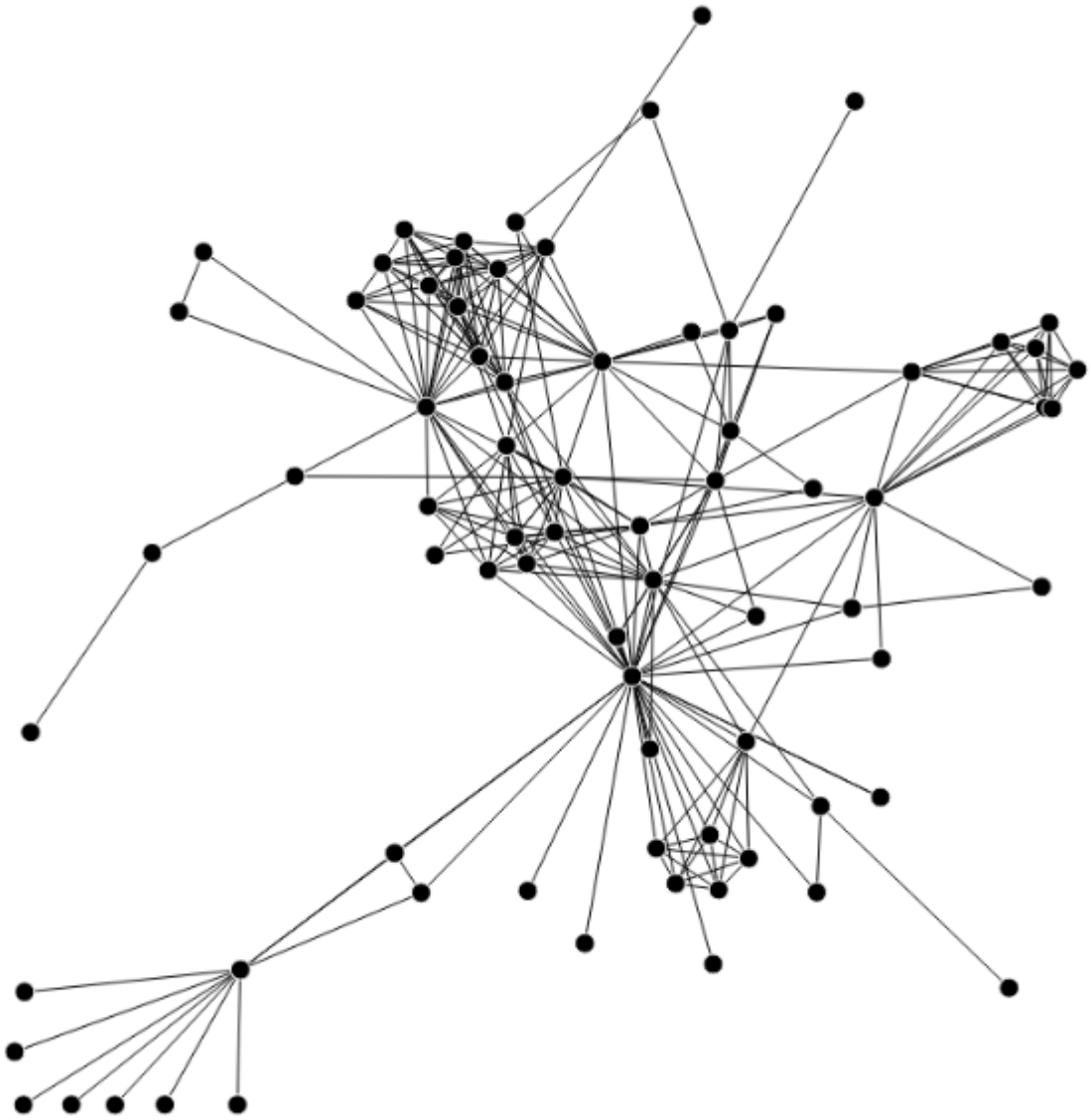
$$k = \sqrt{\frac{površina}{|V|}}$$

Korekcija u algoritmu:

$$k = \sqrt{\frac{\frac{površina}{2}}{|V|}}$$

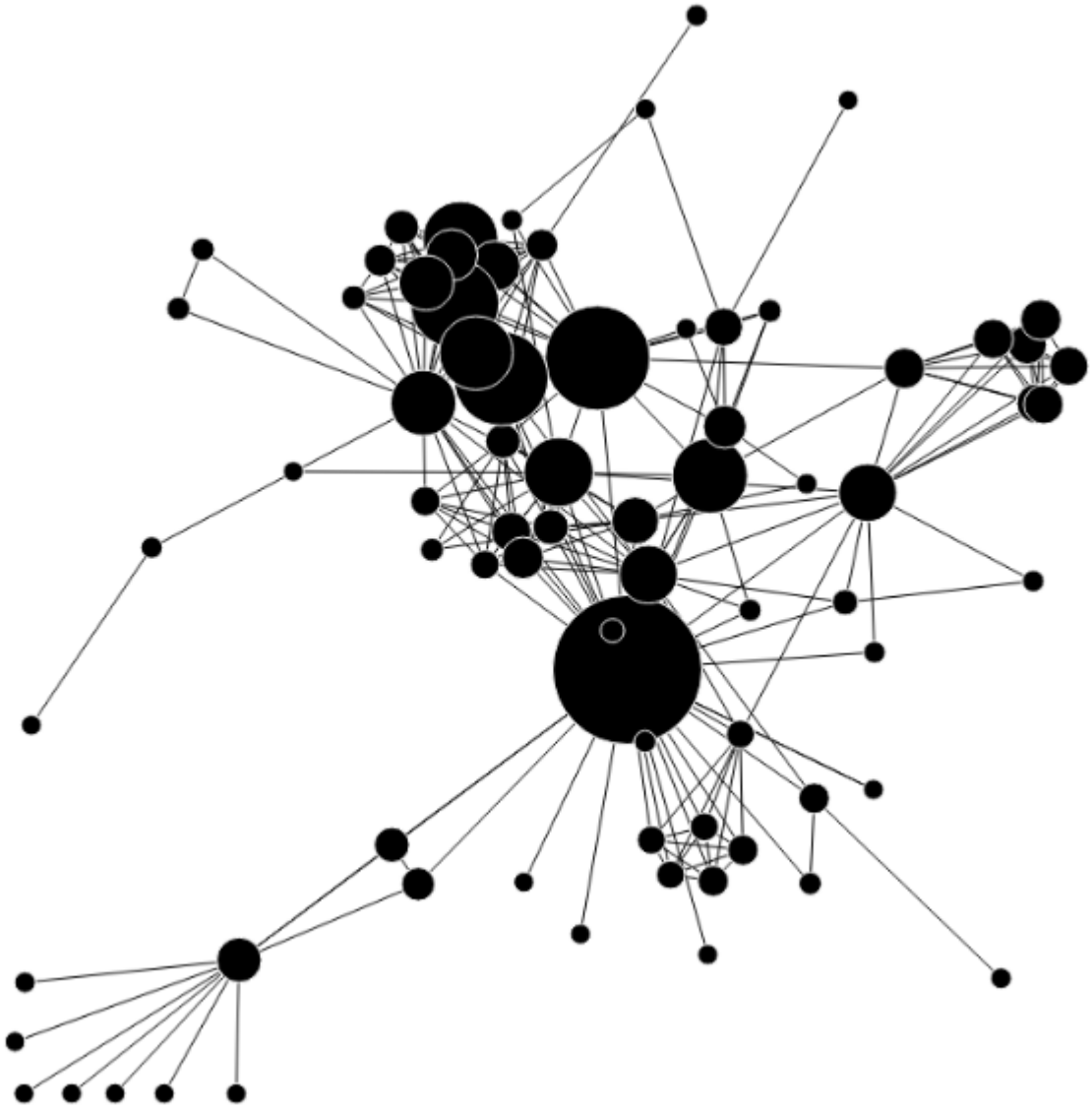
Potreba za ovom promjenom je uvjetovana time da ovaj izračun je davao prevelike vrijednosti duljine bridova i time otežavao postizanje stabilnog stanja grafa i atraktivnog izgleda.

5.4. Primjeri upotrebe



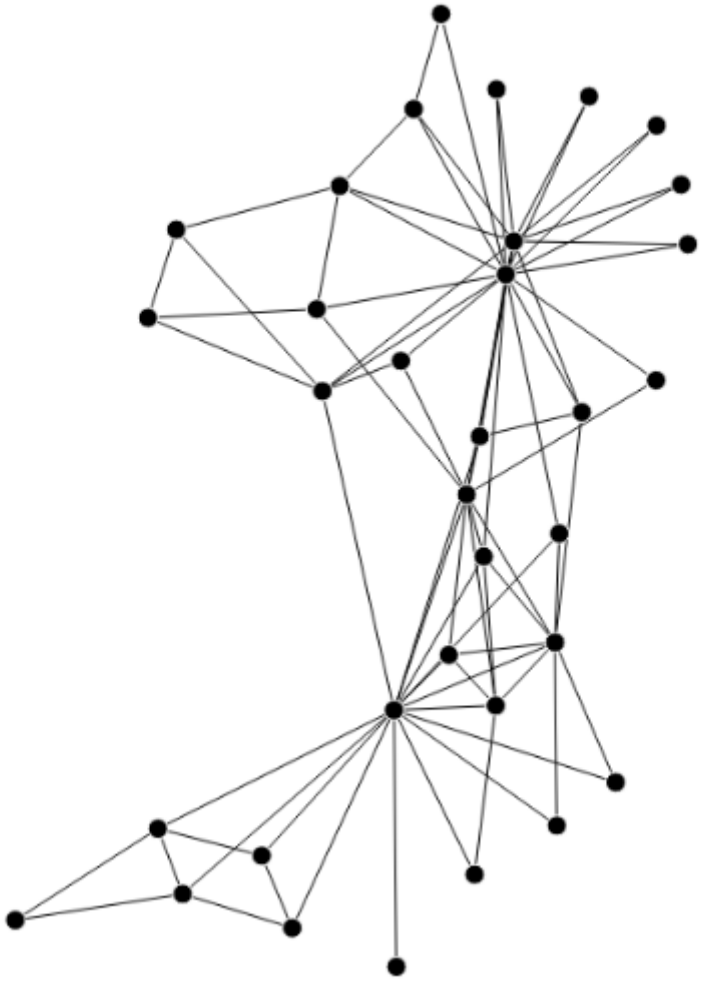
Slika 5.1: Vizualizacija interakcije likova iz Victor Hugo-ovog romana, Les Miserables, bezteˇzinski

Graf na slici 5.1 je generiran uz pomoć izrađene aplikacije, primjenom Fruchterman-Reingold algoritma. Dataset korišten u ovom primjeru je preuzet iz djela D. E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, Addison-Wesley,Reading, MA (1993).



Slika 5.2: Vizualizacija interakcije likova iz Victor Hugo-ovog romana, Les Miserables, težinski

Graf na slici 5.2 je generiran na isti način kao graf sa slike 5.1 ali u ovom slučaju su čvorovima dodane njihove težine te stoga ovaj primjer prikazuje da su modifikacija nad aplikacijom veoma jednostavne.



Slika 5.3: Vizualizacija socijalne mreže izmedu 34 sudionika karate kluba na američkom sveučilištu 1970 godine

Graf na slici 5.3 prikazuje jedan od poznatijih grafova koji se vezuju za socijalne mreže. Zachary's karate club koji potječe iz članka W. W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33, 452-473 1977.

6. Zaključak

U ovom radu posebna je pozornost podarena Fruchterman-Reingold algoritmu. Fruchterman-Reingold algoritam se pokazao nedovoljno dobrim, često grafovi koji su izrađeni u njemu ne dobijaju estetski zadovoljavajući izgled koji je jedan od najvažnijih atributa prilikom vizualizacije grafova.

Jedan od najvećih problema u algoritmu su ne definirani razni ključni segmenti, autori članka nisu formulirali broj iteracija koje su potrebne za izradu estetski lijepog grafa i njihov prijedlog broj iteracija je često nedostatan i za jednostavnije grafove, ali najveća zamjerka im je nedefinirani postupak hlađenja sustava, jer trenutni model koji je implementiran u radu je ne efikasan i često neprikladan za korištenje, te su stoga u implementiranoj aplikaciji dodane male promjene nad tim segmentom algoritma kako bi se poboljšao njegov rad.

Formula za optimalnu duljinu bridova između čvorova je u najmanju ruku manjkava, ona je presimplistički izrađena i kao takva ona se izražava kao površina prostora na kojem se graf iscrtava i broj čvorova koji su predviđeni za iscrtavanje. Prilikom korištenja izvorne formule grafovi su često bili preveliki i nisu bili u mogućnosti doći u stanje mirovanja. Graf sa svojom vremenskom kompleksnošću od $\theta(|V|^2 + |E|)$ je veoma spor i sama njegova implementacija bez preinka je veoma spora.

Ali kako se ne bi samo okomili na Fruchterman-Reingold algoritam također je došlo do velikih problema zbog odluke da se koristi Javascripta i HTML5 <canvas> element. Taj izbor se pokazao dobrim samo na manjim grafovima, ali samim povečanjem broja čvorova te dvije tehnologije su se eksponencijalno usporavale sve do trenutka dok nisu potpuno stale s izvođenjem. Glavni problem je proces iscrtavanja, jer se korištenjem HTML5 <canvas> elementa u svakoj iteraciji slika ponovno iscrtava i u slučaju ako nije došlo do promjene na slici, također <canvas> nema podršku za drag&drop, stoga je i taj dio bilo potrebno implementirati.

Ali taj problem također predstavlja veliki napor na aplikaciju jer se čvoru ne može pristupiti na temelju njegovog reference nego se treba izvršiti provjera za svaki pojedinačni čvor. Provjera se vrši na način da se registriraju XY koordinate gdje je korisnik pritisnuo mišem i na temelju toga

provjeriti da li se taj pritisak nalazi unutar nekog od čvorova.

Naposljetku gledajući odabrane tehnologije i Fruchterman Reingold algoritam mogu samo zaključiti da su postojali mnogo bolji izbori za tehnologije i algoritam. Jer priliko izrađivanja ovog rada obradio sam i druge vrste algoritama za vizualizaciju i uvidio da ima puno boljih algoritama koji su bolje definirani i daju bolje rezultate. A što se tiče tehnologija, misao vodilja je bila da se aplikacija može koristiti na bilo kojoj platformi bez posebnih dodatka, u tome je aplikacija uspješna i po mogućnosti će ona biti prepravljena tako da daje bolje performanse i biva manje zahtjeva prilikom izračuna.

7. Literatura

1. Alvarez-Hamelina I., Dall'Astaa L., Barrata A., Vespignani A., k-core decomposition: a tool for the visualization of large scale networks, Bloomington, 2005.
2. Davis, A., Gardner, B. B. and M. R. Gardner, Deep South, Chicago, The University of Chicago Press, 1941.
3. Diestel R., Graph Theory 3ed, Hamburg, Springer, 2005.
4. Freeman L. C. , Finding Social Groups: A Meta-Analysis of the Southern Women Data, Irvine, University of California, 2003.
5. Fruchterman T. M. J., Reingold E. M., Graph Drawing by Force-directed Placement, Urbana, University of Illinois at Urbana-Champaig, 1991.
6. Kamada T., Kawai S., An algorithm for drawing general undirected graphs, University of Tokyo, Tokyo, 1989.
7. Kaufmann M., Wagner D., Drawing Graphs, New York, Springer, 2001.
8. Moreno J., Who shall Survive,New York, Beacon House, 1934.
9. Newman M. E. J., Networks an Introduction, New York, Oxford university press, 2010.
10. Newman M. E. J., Albert-Laszlo Barbasi, Duncan J. Watts,The structure and dynamics of Networks, New Jersey, Princeton university press, 2006.
11. Wasserman S.,Fauest K., Social Network Analysis, New York, Oxford university press, 1994.
12. Tega J., Hooke's Law,`http://www.universetoday.com/55027/hookes-law/`, Učitano 9.9.2011.
13. Wagon J., Coulomb's Law - The law of force, `http://regentsprep.org/Regents/physics/phys03/acoulomb/default.htm`, Učitano 9.9.2011.

8. Prilog 1 - Programski kod aplikacije

```
var graph = {
    init: function (edges,algorithm) {
        graph.vertices = {};
        graph.edges = {};
        graph.canvas = document.getElementById('platno');
        graph.width = graph.canvas.width;
        graph.height = graph.canvas.height;
    graph.hookes_test = true;
        graph.ctx = graph.canvas.getContext('2d');
    graph.algorithm = algorithm;
    graph.dummypos = new vektor(0,0);
    graph.vertices.eraseNode = new graph.node({id:'eraseNode',x:0,y:0,size:6,ostalo:'eraseNode'})
        ;
    graph.vertices.addEdge = new graph.node({id:'addEdge',x:0,y:0,size:6,ostalo:'addEdge'});
        document.addEventListener("mousedown", graph.nodeClickEvent, false);
        document.addEventListener("mouseup", graph.nodeDropEvent, false);
    document.addEventListener("dblclick",graph.dblclick, false);
        graph.addNodesFromEdgesList(edges);
        graph.addEdgesFromEdgesList(edges);
        setInterval(graph.draw, 1024 / 24);
    },
    addNodesFromEdgesList: function (EdgesList) {
        for (var r1 = 0; r1 < EdgesList.length; r1++) {
            for (var r2 = 0; r2 < EdgesList[r1].length - 1; r2++) {

                if ((typeof graph.vertices[EdgesList[r1][r2]]) === "undefined") {
                    graph.addNode({
                        id: EdgesList[r1][r2],
                        x: Math.floor(graph.width / 2 + 100 * Math.cos(Math.PI * (EdgesList[r1][
                            r2] * 2) / 11)),
                        y: Math.floor(graph.height / 2 + 100 * Math.sin(Math.PI * (EdgesList[r1][
                            r2] * 2 / 11))),
                        size: 6

                    });
                }
            }
        }
    },
    addEdgesFromEdgesList: function (EdgesList) {
        for (var a = 0; a < EdgesList.length; a++) {
            graph.addEdge({
                from: EdgesList[a][0],
                to: EdgesList[a][1],
                id: a
            });
        }
    },
    node: function (node) {
        this.id = node.id;
        this.pos = new vektor(node.x, node.y);
        this.size = node.size;
        this._size = node.size;
        this.expanded = false;
        this.disp = new vektor(0,0);
        this.ostalo = node.ostalo || null;
    },
```

```
55      addNode: function (node) {
56          (typeof graph.vertices[node.id]) === "undefined" ? graph.vertices[node.id] = new graph.
                node(node) : console.log("Duplikat cvora! Id:" + node.id);
57      },
58      removeNode: function(id) {
59          if (typeof graph.vertices[id] !== "undefined") {
60              graph.removeEdgeByNodeId(graph.vertices[id].id);
61              if(typeof graph.info_node !== "undefined" && graph.info_node.id == id){graph.
                    info_node = false;}
62              delete graph.vertices[id];
63          } else {
64              console.log("Ne postoji node! Id:" + id);
65          }
66      },
67      getNumberOfNodes : function(){
68        var res = 0, id;
69          for (id in graph.vertices) {
70              if (graph.vertices.hasOwnProperty(id)) {
71                  res++;
72              }
73          }
74          return res;
75      },
76      getNumberOfEdges : function(){
77        var res = 0, id;
78          for (id in graph.edges) {
79              if (graph.edges.hasOwnProperty(id)) {
80                  res++;
81              }
82          }
83          return res;
84      },
85      edge: function (edge) {
86          this.id = edge.id;
87          this.from = graph.vertices[edge.from];
88          this.to = graph.vertices[edge.to];
89      },
90      addEdge: function (edge) {
91          (typeof graph.edges[edge.id]) === "undefined" ? graph.edges[edge.id] = new graph.edge(
                edge) : console.log("Duplikat brida! Id:" + edge.id);
92      },
93      removeEdgeByEdgeId: function (id) {
94          (typeof graph.edges[id]) !== "undefined" ? delete graph.edges[id] : console.log("Ne
                postoji brid! Id:" + id);
95      },
96      removeEdgeByNodeId: function (id) {
97          if (typeof graph.vertices[id] !== "undefined") {
98              for (var edge in graph.edges) {
99                  if (graph.edges.hasOwnProperty(edge) && (graph.edges[edge].from.id == id || graph
                        .edges[edge].to.id == id)) {
100                     delete graph.edges[edge];
101                 }
102             }
103         } else {
104             console.log("Ne postoji cvor! Id:" + id);
105         }
106     },
107     clearGraph: function () {
108         for (var id in graph.vertices) {
109             if (graph.vertices.hasOwnProperty(id)) {
```

```
                graph.removeNode(id)
            }
        }
    },
    mapNodes: function (funkcija, obj) {
        var res = [],
            tmp, id;
        for (id in graph.vertices) {
            if (graph.vertices.hasOwnProperty(id)) {
                tmp = funkcija.apply(graph, [graph.vertices[id], obj || {}]);
                if (tmp) res.push(tmp);
            }
        }
        return res;
    },
    mapEdges: function (funkcija) {
        for (var id in graph.edges) {
            if (graph.edges.hasOwnProperty(id)) {
                funkcija.apply(graph, [graph.edges[id].from, graph.edges[id].to,graph.edges[id].
                    id]);
            }
        }
    },
    dragNode: function (e) {

        if (graph.selectedNode) {
            graph.selectedNode.pos.x = (e.pageX - graph.canvas.offsetLeft);
            graph.selectedNode.pos.y = (e.pageY - graph.canvas.offsetTop);
            if (graph.selectedNode.pos.x > graph.width - 6) graph.selectedNode.pos.x = graph.
                width - 6;
            else if (graph.selectedNode.pos.x < 6) graph.selectedNode.pos.x = 6;
            else if (graph.selectedNode.pos.y > graph.height - 6) graph.selectedNode.pos.y =
                graph.height - 6;
            else if (graph.selectedNode.pos.y < 6) graph.selectedNode.pos.y = 6;
        }
    },
  addEdgeAnim :function(e){
    if(typeof graph.dummy === "undefined" || graph.dummy === null )return false;
    graph.ctx.beginPath();
        graph.ctx.strokeStyle = 'rga(129,143,154,0.1)';
        graph.ctx.moveTo(graph.dummy.pos.x, graph.dummy.pos.y);

        if(typeof e !== "undefined"){
          graph.ctx.lineTo(e.pageX - graph.canvas.offsetLeft, e.pageY - graph.canvas.offsetTop);
          graph.dummypos.x= e.pageX - graph.canvas.offsetLeft;
          graph.dummypos.y= e.pageY - graph.canvas.offsetTop;
          }else{
            graph.ctx.lineTo( graph.dummypos.x, graph.dummypos.y);
          };
        graph.ctx.stroke();
  },
    nodeClickEvent: function (e) {

        graph.selectedNode = graph.getNodeFromXY(e.pageX - graph.canvas.offsetLeft, e.pageY -
            graph.canvas.offsetTop)[0];
        var brisanje = graph.getNodeFromXY(e.pageX - graph.canvas.offsetLeft, e.pageY - graph.
            canvas.offsetTop);
        for(var br=0;br<brisanje.length;br++){
        if(brisanje[br].ostalo == "eraseNode" )graph.removeNode(brisanje[br].id);
        else if(brisanje[br].ostalo == "addEdge" ){
```

```
            graph.dummy = brisanje[br];
            document.addEventListener("mousemove", graph.addEdgeAnim, false);
        }
        }
        document.addEventListener("mousemove", graph.dragNode, false);
    },
    nodeDropEvent: function (e) {
        graph.selectedNode = false;
        document.removeEventListener("mousemove", graph.dragNode, false);
        if(graph.dummy!=null && graph.getNodeFromXY(e.pageX - graph.canvas.offsetLeft, e.pageY -
            graph.canvas.offsetTop).length>0 && graph.dummy.id!==graph.getNodeFromXY(e.pageX -
            graph.canvas.offsetLeft, e.pageY - graph.canvas.offsetTop)[0].id){
          graph.hookes_test = true;
          graph.addEdge({
            id:graph.getNumberOfEdges()+1,
            from:graph.dummy.id,
            to:graph.getNodeFromXY(e.pageX - graph.canvas.offsetLeft, e.pageY - graph.canvas.
                offsetTop)[0].id,
            duljina:graph.dummy.pos.oduzmi(graph.getNodeFromXY(e.pageX - graph.canvas.offsetLeft,
                 e.pageY - graph.canvas.offsetTop)[0].pos).duljina()
          });
      graph.mapEdges(graph.oDuljina);
      graph.hookes_test=false;
          graph.dummy = null;
        }
        graph.dummy = null;
        document.removeEventListener("mousemove", graph.addEdgeAnim, false);
    },
    getNodeFromXY: function (_x, _y) {
        return graph.mapNodes(function (node, obj) {
            if ((obj.x > node.pos.x - node.size) && (obj.x < node.pos.x + node.size) && (obj.y >
                node.pos.y - node.size) && (obj.y < node.pos.y + node.size)) {
                return node;
            } else {
                return false
            };
        }, {
            x: _x,
            y: _y
        });
    },
    draw: function () {
        graph.ctx.clearRect(0, 0, graph.width, graph.height);
        //if(algorithm.temperatura==graph.width/10)graph.ctx.scale(0.8,0.8)
        background();
        graph.algorithm();
        graph.mapEdges(crtaj_v);
        if(graph.dummy!==null && typeof graph.dummy !== "undefined")graph.addEdgeAnim();
        graph.mapNodes(crtaj_n);
        graph.info();
        if(!graph.hookes_test)graph.mapEdges(graph.hookes);
        function background() {
            var grd = graph.ctx.createRadialGradient(graph.width / 2, graph.height / 2, 30, graph
                .width / 2, graph.height / 2, graph.height);
            grd.addColorStop(0, "#42586d");
            grd.addColorStop(0.5, "#36495a");
            grd.addColorStop(1, "#26323e");
            graph.ctx.fillStyle = grd;
            graph.ctx.fillRect(0, 0, graph.width, graph.height);
        }
```

```
        function crtaj_n(v) {
          if(v.ostalo=="eraseNode" || v.ostalo=="addEdge")return false;
            graph.ctx.fillStyle = 'rgba(0,0,0,0.4)';
            graph.ctx.beginPath();
            graph.ctx.arc(v.pos.x, v.pos.y, v.size, 0, Math.PI * 2, true);
            graph.ctx.fill();
            graph.ctx.strokeStyle = '#818f9a'
            graph.ctx.arc(v.pos.x, v.pos.y, v.size, 0, Math.PI * 2, true);
            graph.ctx.stroke();
            return false;
        }
        function crtaj_v(v1, v2) {
            graph.ctx.beginPath();
            graph.ctx.strokeStyle = 'rga(129,143,154,0.1)';
            var duljina = [v1.pos.udaljenost(v2.pos) - v1.size, v1.pos.udaljenost(v2.pos) - v2.
                size];
            var kut = Math.atan2(v2.pos.y - v1.pos.y, v2.pos.x - v1.pos.x);
            graph.ctx.moveTo(v2.pos.x - (duljina[0] * Math.cos(kut)), v2.pos.y - (duljina[0] *
                Math.sin(kut)));
            graph.ctx.lineTo(v1.pos.x + (duljina[1] * Math.cos(kut)), v1.pos.y + (duljina[1] *
                Math.sin(kut)));
            graph.ctx.stroke();
        } },

    dblclick: function (e) {
        var dbl = graph.getNodeFromXY((e.pageX - platno.offsetLeft), (e.pageY - platno.offsetTop)
            )[0] || false;
        if (dbl.expanded) {
            dbl.size = dbl._size;
            dbl.expanded = false;
            graph.info_node = false;
        } else if (dbl) {
            graph.mapNodes(function (v1) {
                if (v1.expanded) {
                    v1.size = v1._size;
                    v1.expanded = false;
                    graph.info_node = false;
                }
            })
            dbl.size = 30;
            dbl.expanded = true;
            graph.info_node = dbl;
        }else if(!dbl)graph.addNode({
                        id: graph.getNumberOfNodes()+1,
                        x: e.pageX - platno.offsetLeft,
                        y: e.pageY - platno.offsetTop,
                        size: 6
                    });
    },
    info: function () {
        graph.ctx.font = "10px Verdana";
            graph.ctx.textAlign = "center";
            graph.ctx.fillStyle = '#ffffff';
            graph.ctx.fillText("Temperatura: " + algorithm.temperatura, graph.canvas.width-200,
                10,200);
        if (graph.info_node) {
            graph.ctx.font = "10px Verdana";
            graph.ctx.textAlign = "center";
            graph.ctx.fillStyle = '#ffffff';
```

```
            graph.ctx.fillText("Node: " + graph.info_node.id, graph.info_node.pos.x, graph.
                info_node.pos.y + 3, 30);

            graph.ctx.fillStyle = 'rgba(255,0,0,1)';
            graph.ctx.beginPath();
            graph.ctx.arc(graph.info_node.pos.x+graph.info_node.size, graph.info_node.pos.y+graph
                .info_node.size, 6, 0, Math.PI * 2, true);
            graph.ctx.fill();
            graph.vertices.eraseNode.pos.x=graph.info_node.pos.x+graph.info_node.size;
            graph.vertices.eraseNode.pos.y=graph.info_node.pos.y+graph.info_node.size;
            graph.vertices.eraseNode.id = graph.info_node.id;

            graph.ctx.fillStyle = 'rgba(0,0,255,1)';
            graph.ctx.beginPath();
            graph.ctx.arc(graph.info_node.pos.x-graph.info_node.size, graph.info_node.pos.y+graph
                .info_node.size, 6, 0, Math.PI * 2, true);
            graph.ctx.fill();
            graph.vertices.addEdge.pos.x=graph.info_node.pos.x-graph.info_node.size;
            graph.vertices.addEdge.pos.y=graph.info_node.pos.y+graph.info_node.size;
            graph.vertices.addEdge.id = graph.info_node.id;
        }

    },
    hookes : function(v1,v2,id){
      var length = v1.pos.oduzmi(v2.pos),
      udaljenost = length.duljina()-(graph.edges[id].duljina),
      HL = 20 * (udaljenost/length.duljina()),
      kut = Math.atan2(v2.pos.y-v1.pos.y,v2.pos.x-v1.pos.x),
      dis;
      if((graph.selectedNode&& (graph.selectedNode.id != v1.id)) || !graph.selectedNode){

          dis =new vektor(HL*Math.cos(kut),HL*Math.sin(kut))
          if(v1.pos.x+dis.x>graph.width-v1.size || v1.pos.x+dis.x<0+v1.size){
          v1.pos.x += dis.x*(-1);
          v1.pos.y += dis.y;
        }else if(v1.pos.y+dis.y>graph.height-v1.size || v1.pos.y+dis.y<0+v1.size){
          v1.pos.x += dis.x;
          v1.pos.y += dis.y*(-1);
        }else{
          v1.pos = v1.pos.zbroji(dis)
        }
      }


      if((graph.selectedNode&& (graph.selectedNode.id != v2.id)) || !graph.selectedNode){
          dis =new vektor(HL*Math.cos(kut),HL*Math.sin(kut))
          if(v2.pos.x+dis.x>graph.width-v2.size || v2.pos.x+dis.x<0+v2.size){
          v2.pos.x -= dis.x*(-1);
          v2.pos.y -= dis.y;
        }else if(v2.pos.y+dis.y>graph.height-v2.size || v2.pos.y+dis.y<0+v2.size){
          v2.pos.x -= dis.x;
          v2.pos.y -= dis.y*(-1);
        }else{
          v2.pos = v2.pos.oduzmi(dis)
        }
        }

    },
oDuljina : function(v1,v2,id){

```

```
      graph.edges[id].duljina = v1.pos.oduzmi(v2.pos).duljina();
  }


}

var algorithm = {
    init :function(){
    algorithm.eps = 2;
    algorithm.broj_iteracija = 0;
    algorithm.max_iteracija = 100;
    algorithm.temperatura = (function(){return graph.width/10})();
    algorithm.FP = 1;
    algorithm.FO = 1;
    algorithm.scale = 0.85;
    document.addEventListener("keydown",algorithm.restart, false);
    },
    privlacneSile : function ( z ){
      var k =  Math.sqrt( ((graph.width*graph.height/8)) / graph.getNumberOfNodes());
      return algorithm.FP*(Math.pow( z , 2 ) / k);
    },

    odbojneSile : function ( z ){
      var k = Math.sqrt( ((graph.width*graph.height/8)) / graph.getNumberOfNodes());
      return  algorithm.FO*(Math.pow( k , 2 ) / z);
    },

    repulsion: function(v1){
      var duljina = graph.getNumberOfNodes();
      var res = [],
            v2, id;
          for (id in graph.vertices) {
              if (graph.vertices.hasOwnProperty(id)) {
                v2 = graph.vertices[id]
                   if(v1.id!==v2.id){
            var delta = v1.pos.oduzmi(v2.pos)
            var duljina_delte = delta.duljina()
            var sila = algorithm.odbojneSile(duljina_delte);
            v1.disp = v1.disp.zbroji((delta.division(duljina_delte)).multiply(sila))
          }
              }
          }


    },

    attraction : function (v1,v2){

      var delta = v1.pos.oduzmi(v2.pos);
      var duljina_delte = delta.duljina();
      var sila = algorithm.privlacneSile(duljina_delte);

      v1.disp = v1.disp.oduzmi((delta.division(duljina_delte)).multiply(sila));
      v2.disp = v2.disp.zbroji((delta.division(duljina_delte)).multiply(sila));

    },

    zavrsetak : function(v1){
      var duljina = Math.max(algorithm.eps,v1.disp.duljina());
      var dis = v1.disp.division(duljina).multiply(Math.min(duljina,algorithm.temperatura));
```

```
390
391       if(v1.pos.x+dis.x>graph.width-v1.size || v1.pos.x+dis.x<0+v1.size){
392         v1.pos.x += dis.x*(-1)
393         v1.pos.y += dis.y
394       }else if(v1.pos.y+dis.y>graph.height-v1.size || v1.pos.y+dis.y<0+v1.size){
395         v1.pos.x += dis.x
396         v1.pos.y += dis.y*(-1)
397       }else{
398         v1.pos = v1.pos.zbroji(dis)
399       }
400       v1.disp=new vektor(0,0);
401
402     },
403
404     algoritam : function (){
405
406         if(algorithm.temperatura>0.1){
407         graph.mapNodes(algorithm.repulsion)
408         graph.mapEdges(algorithm.attraction)
409         graph.mapNodes(algorithm.zavrsetak)
410         if(algorithm.temperatura<0.2 && graph.hookes_test == true){
411           graph.mapEdges(graph.oDuljina);
412                 graph.hookes_test=false;
413               }
414         algorithm.temperatura *= (1 - algorithm.broj_iteracija / algorithm.max_iteracija);
415         algorithm.broj_iteracija++;
416
417     }
418
419     },
420     restart : function (e){
421       if(e.keyCode==82){
422       algorithm.temperatura=graph.width/10;
423       algorithm.broj_iteracija=0;
424       graph.hookes_test=true;
425       }
426     }
427
428
429
430 }
431
432 graph.init(edges,typeof algorithm !== "undefined"? algorithm.algoritam : function(){});
433 algorithm.init();
```