UNIVERZA V LJUBLJANI
FAKULTETA ZA DRUŽBENE VEDE

Darko Brvar

DINAMIČNI PRIKAZ ČASOVNIH OMREŽIJ

Magistrsko delo

Mentor: doc. dr. Andrej Mrvar

Ljubljana, 2005

Povzetek

Časovno omrežje je omrežje, ki je podano v poljubno mnogo časovnih trenutkih.
Točke in povezave, ki pripadajo časovnemu omrežju, niso nujno prisotne v vseh časovnih trenutkih. Povezava je prisotna v določenem časovnem trenutku, če sta v tem trenutku prisotni tudi obe njeni krajišči.
Dinamični prikaz omogoča boljši vpogled v značilnosti omrežij tako pri začetnem raziskovanju omrežij kot tudi pri predstavitvi rezultatov raziskav. Osnova za dinamični prikaz je statični prikaz omrežij, to je slikovna predstavitev omrežij.
Programski paket Pajek je namenjen analizi in prikazu velikih omrežij. Analizo izvajamo s pomočjo šestih podatkovnih struktur: omrežje, razbitje, permutacija, skupina, hierarhija in vektor. Paket Pajek omogoča statični prikaz časovnih omrežij in izvoz prikazov v format SVG. SVG je javni označevalni jezik, s katerim je mogoče izdelati spletne strani, ki vključujejo slike z visoko ločljivostjo.

Program PajekToSvgAnim je programski dodatek programskemu paketu Pajek za dinamični prikaz longitudinalnih socialnih in drugih časovnih omrežij. Iz vhodne Pajkove projektne datoteke .PAJ izdela datoteke .SVG, .SVG.GZ in .HTML (datoteka .SVG.GZ je skrčena oblika GZIP datoteke .SVG, ki omogoča hitrejše nalaganje in hitrejši prikaz objektov SVG).
Napisan je v programskem jeziku Python. Python je tolmačeni interaktivni objektno orientirani programski jezik in tako kot programski jezik Java omogoča izdelavo modulov, razredov, izjem in dinamičnih podatkovnih tipov.
Program PajekToSvgAnim ima za različne potrebe uporabnikov vgrajenih kar nekaj izbir, ki se lahko določajo v slikovnem uporabniškem vmesniku. Poleg osnovnih izbir vhodne Pajkove datoteke in imena ter poti izhodne datoteke .SVG omogoča še naslednje izbire:

- dolžino trajanja dinamičnega prikaza prehoda omrežja iz enega časovnega trenutka v drugega;
- dolžino trajanja statičnega prikaza omrežja v časovnem trenutku pred dinamičnim prikazom prehoda v naslednji časovni trenutek;
- številski seznam dinamičnih prehodov med časovnimi trenutki, ki naj se prikažejo;
- možnost izbire med dvema velikostima oblik točk: dejansko velikostjo, ki je

podana v vhodni datoteki, in enotno velikostjo za vse točke;
- velikost pisave oznak točk;
- velikost okna SVG;
- možnost izbire označitve relacij v spletnem sestavku z vgrajeno sliko SVG (ob vsaki naložitvi sestavka) v primeru večkratnega omrežja;
- (ne)prosojnost povezav.

S pomočjo programa PajekToSvgAnim lahko izdelamo zvezne prehode časovnih omrežij med časovnimi trenutki. Pri posameznem prehodu omrežja iz enega časovnega trenutka v drugega se vidijo sledi prejšnjih slik omrežja, torej sledi tistih točk in povezav omrežja, ki v naslednjem časovnem trenutku niso več prisotne. To nam omogoča, da lahko bolj natančno sledimo spremembam omrežja skozi čas kot doslej, ko smo imeli na voljo le statične slike omrežja.

Zahvala

Zahvaljujem se mentorju doc. dr. Andreju Mrvarju za mentorstvo in za vso pomoč pri izdelavi programa PajekToSvgAnim.
Zahvaljujem se prof. dr. Vladimirju Batagelju za veliko idej in nasvetov.
Zahvaljujem se ženi, mami, očetu in bratu za vso spodbudo in potrpljenje pri nastajanju dela.

KAZALO

1 Uvod

Zaradi povečanega zanimanja za raziskovanje longitudinalnih socialnih omrežij in zaradi dejstva, da prikaz omogoča boljši vpogled v značilnosti omrežij tako pri začetnem raziskovanju omrežij kot tudi pri predstavitvi rezultatov raziskav, se je v zadnjem času pojavila potreba po dinamičnem prikazu omrežij (Moody, McFarland, Bender-deMoll, 2003; Mrvar, Batagelj, 2000, str. 137). S tem področjem se ukvarja vse več skupin raziskovalcev, npr.:

- Skye Bender-deMoll in Daniel A. McFarland z Univerze v Stanfordu;
- Stephan Diehl in Carsten Görg z Univerze des Saarlandes v Saarbrücknu;
- več drugih skupin v Nemčiji: D. Wagner z Univerze v kraju Karlsruhe, U. Brandes z Univerze v Konstanzu, F. J. Brandenburg in M. Himsolt z Univerze v Passauu, P. Mutzel s sodelavci z Max-Planck Instituta v Saarbrücknu;
- Tom Sawyer Institute v Berkeleyu;
- skupina Roberta Tamassija z Univerze Brown, ki izdaja tudi omrežno revijo, namenjeno analizi grafov: The Journal of Graph Algorithms and Applications (JGAA);
- Stephen North s sodelavci AT&T Bell Laboratories;
- D. R. White s sodelavci z Univerze v Kaliforniji (Irvine);
- Mitsubishi Electric Research Laboratories (MERL) v Cambridgeu;
- Silicon Graphics v Grasbrunnu;
- Vladimir Batagelj s Fakultete za matematiko in fiziko v Ljubljani in Andrej Mrvar s Fakultete za družbene vede v Ljubljani s sodelavci;
- skupina Tomaža Pisanskega s Fakultete za matematiko in fiziko v Ljubljani.

Osnova za dinamični prikaz je statični prikaz omrežij, to je slikovna predstavitev omrežij. Korenine statičnega prikaza segajo v leto 1931, ko je romunski matematik in filozof Jacob Levi Moreno prvič predstavil "sociogram", to je graf, sestavljen iz točk in povezav, ki predstavlja odnose med osebami (Freeman, 2000; Freeman, Webster, Kirke, 1998, str. 109).

Prvi sociogrami so bili zelo preprosti, narisani ročno, z nekaj relacijami na osebo (Whyte, 1943; Coleman, 1961). Sledil je hiter napredek in razvilo se je več definicij za optimalni prikaz omrežja (Di Battista, Eades, Tamassia, Tollis, 1999; Brandes, Raab, Wagner, 2001).

Že omenjena longitudinalna socialna omrežja se uvrščajo med časovna omrežja. Pri teh omrežjih se zanimanje vrti okrog razumevanja, kako se omrežje razvija in spreminja skozi čas, in okrog iskanja načinov za razvoj modelov socialnih procesov, ki bi pomagali pojasniti opazovane strukture (Doreian et al., 1996, str. 113; Leenders, 1996, str. 165; Nakao, Romney, 1993, str. 109; Snijders, 1998; Suitor et al., 1997, str.1; Weesie, Flap, 1990; Zeggelink et al., 1996, str. 29). Jedro zanimanja je torej dinamika medosebnih odnosov, ki je pomembna za razumevanje socialnega omrežja. Pri tem se pojavi problem, da se te dinamike ne da na zadovoljiv način prikazati s statičnim prikazom (Bearman, Everett, 1993, str. 171).
Rešitev je dinamični prikaz časovnih omrežij, ki raziskovalcem odpira vrata na naslednjih področjih:

- pri odkrivanju značilnosti razvoja omrežja;
- pri spremljanju razvoja izbranih delov omrežja (tirov);
- pri odkrivanju izstopajočih delov omrežja.

V magistrskem delu bodo z dinamičnim prikazom predstavljeni nekateri znani primeri časovnih omrežij. To so:

- omrežje zgodb nadaljevanke Lindenstrasse (Mutzel, 1999; Batagelj, 2005);
- klasično longitudinalno socialno omrežje Sampsonovih menihov (Sampson, 1968; De Nooy, Mrvar, Batagelj, 2005, pogl. 4);
- omrežje novic 11. september 2001 (Corman, Dooley, 2002; Batagelj, Mrvar, 2003);
- omrežje političnih dogodkov KEDS na Balkanu (Schrodt, Davis, Weddle, 2004; Batagelj, 2005, KEDS).

Zgornji primeri časovnih omrežij so že bili predstavljeni s statičnim prikazom, in sicer s programskim paketom Pajek (Batagelj, Mrvar, 2005), ki omogoča prikaz in analiziranje velikih omrežij (Mrvar, 1999; Mrvar, 2001, str. 9). Različica Pajek 1.02 že vsebuje možnost opisa večkratnih omrežij na isti množici točk.

Namen magistrskega dela je dopolniti programski paket Pajek s posebnim programom za dinamični prikaz longitudinalnih socialnih in drugih časovnih omrežij, ki
bi omogočal zvezne prehode in s tem sledenje spremembam med časovnimi trenutki. Pri tem bi se morala ohranjati mentalna slika omrežja, torej bi morale biti spremembe na prikazih omrežja v časovnih trenutkih minimalne.

2 Časovna omrežja

2 . 1 Definicija grafa in omrežja

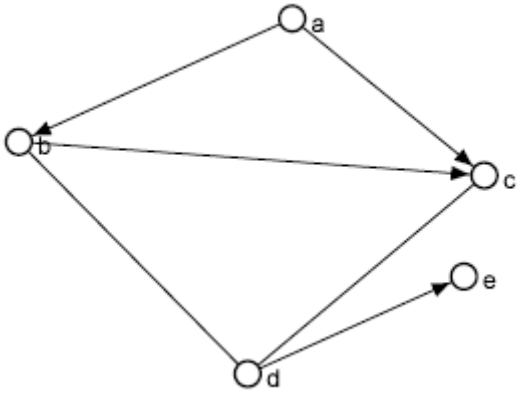
Graf G je definiran kot par $G = (V, L)$, kjer je V množica točk in L množica povezav. Povezave so lahko usmerjene ali neusmerjene.

Graf postane omrežje, če vsebuje dodatne podatke o točkah in/ali povezavah. Omrežje N lahko torej definiramo kot 4-terico množic $N=(V, L, F_V, F_L)$, kjer je V množica točk, L množica povezav, F_V množica lastnosti točk in F_L množica lastnosti povezav.

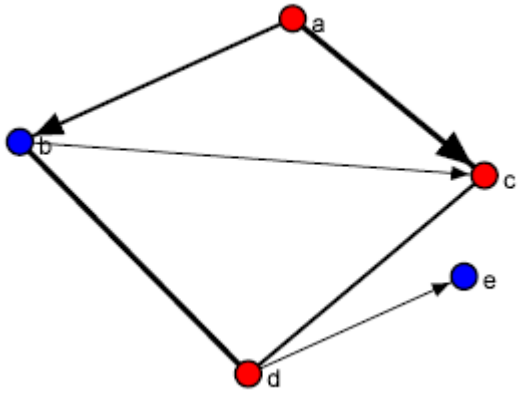
Množica lastnosti točk F_V je množica funkcij $f: V \rightarrow X$, kjer množica X lahko predstavlja množico oznak točk, razbitje točk ali množico številskih lastnosti točk. Na sliki lahko številsko lastnost prikažemo kot velikost točke ali njeno koordinato, imensko lastnost pa kot barvo, obliko lika, ali kot oznako točke.

Množica lastnosti točk F_L pa je množica funkcij $g: L \rightarrow Y$, kjer je Y množica številskih lastnosti povezav. Na sliki jih prikažemo z izpisom vrednosti, debelino črte ali sivino.

Primer: Na Sliki 2.1a je prikazan graf z množico točk {a, b, c, d, e}, množico usmerjenih povezav {(a, b), (a, c), (b, c), (d, e)} in množico neusmerjenih povezav {(b, d),(c, d)}. Na Sliki 2.1b pa je prikazan isti graf z imensko lastnostjo točk (razbitjem), predstavljeno z dvema barvama, ter s številsko lastnostjo povezav, predstavljeno z različnimi debelinami.



Slika 2.1a: primer grafa.



Slika 2.1b: graf z dodatnimi lastnostmi.

2 . 2 Definicija časovnega omrežja

Časovno omrežje je omrežje, ki je podano v poljubno mnogo časovnih trenutkih. Zato ga lahko definiramo na osnovi omrežja na naslednji način:
Časovno omrežje N_T lahko torej definiramo kot 5-terico množic $N_T=(V, L, F_V, F_L, T)$, kjer je V množica točk, L množica povezav, F_V množica lastnosti točk, F_L množica lastnosti povezav in T množica linearno urejenih časovnih trenutkov.
Točke in povezave, ki pripadajo časovnemu omrežju, niso nujno prisotne v vseh časovnih trenutkih. Pri tem velja logična omejitev, da je povezava prisotna v določenem časovnem trenutku, če sta v tem trenutku prisotni tudi obe njeni krajišči.

2 . 3 Predstavitev primerov časovnih omrežij s statičnim prikazom v programskem paketu Pajek

2. 3. 1 Programski paket Pajek

Programski paket Pajek je namenjen analizi in prikazu velikih omrežij (Batagelj, Mrvar, 2005). Analizo izvajamo s pomočjo šestih podatkovnih struktur: omrežje, razbitje, permutacija, skupina, hierarhija in vektor. Vsaka struktura, ki jo preberemo ali dobimo kot rezultat neke analize, ostane shranjena v pomnilniku in na voljo za nadaljnje delo. V programski paket je vgrajenih veliko število učinkovitih algoritmov, prava moč paketa pa se pokaže šele ob kombiniranju večih algoritmov za dosego željenih rezultatov in združevanju zaporedij ukazov za izvedbo določenih opravil v makroje (Mrvar, 1999).
V nadaljevanju bomo uporabili naslednje izbire v glavnem oknu paketa Pajek (Mrvar, 1999):

- File → Network → Read: Preberemo omrežje iz vhodne (.NET) datoteke.
- File → Pajek Project File → Read: Preberemo več omrežij in drugih objektov iz vhodne (.PAJ) datoteke.
- File → Pajek Project File → Save: Shranimo vse naložene objekte na izhodno (.PAJ) datoteko.
- Net → Transform → Generate in Time → All: Iz časovnega omrežja zgradimo zaporedje omrežij v danih časovnih trenutkih. Vnesti moramo prvi čas, zadnji čas in korak.

Uporabili bomo tudi naslednje izbire v slikovnem oknu paketa Pajek:

- Layout → Circular → Original: Avtomatično določanje krožnih prikazov omrežij.
- Layout → Energy → Kamada-Kawai → Free: Avtomatično določanje prikazov omrežij z uporabo energijskega risanja z minimalno skupno energijo v ravnini. V toku minimizacije skupne energije se točke prosto premikajo po ravnini.
- Previous: Nariše se slika prejšnjega omrežja, ki je shranjeno v pomnilniku.
- Next: Nariše se slika naslednjega omrežja, ki je shranjeno v pomnilniku.
- Mark vertices using → Labels: Označimo točke z oznakami.
- Lines → Different Widths: Pri risanju povezav se upošteva njihove debeline.
- Lines → Gray Scale: Pri risanju povezav se upošteva sivina, odvisna od njihovih vrednosti. Tako so povezave z največjo vrednostjo narisane s črno barvo, ostale pa z ustreznim odtenkom sive barve.
- Colors → Edges/Arcs → Relation number: Pri risanju usmerjenih in neusmerjenih povezav večkratnega omrežja zahtevamo uporabo barv, ki so določene za relacije.
- Export → SVG → General: Izpis omrežja v obliko SVG (poglavje 4).
- Export → SVG → Current and all Subsequent: Izpis časovnega omrežja v obliko SVG. Izpišejo se vsa omrežja od izbranega do zadnjega shranjenega omrežja.

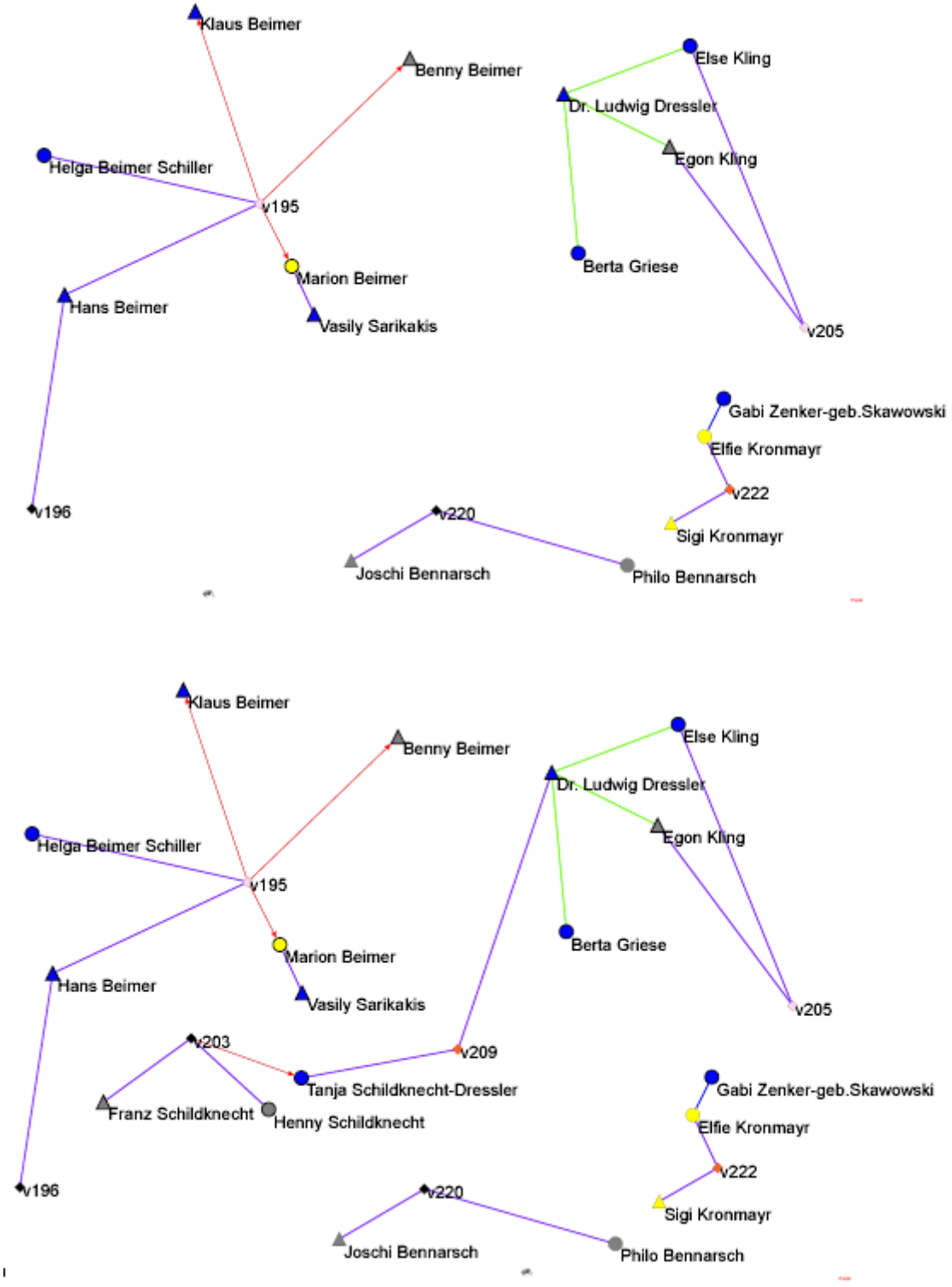
2. 3. 2 Opis omrežij

2. 3. 2. 1 Omrežje zgodb nadaljevanke Lindenstrasse

Podatki omrežja zgodb nadaljevanke Lindenstrasse so dobljeni s tekmovanja v risanju grafov GD'99 (Mutzel, 1999; Batagelj, 2005). Ena od nalog na tem tekmovanju je bila za dani graf izdelati enega od naslednjih prikazov grafa, ki prikazuje igralce in njihove medsebojne relacije:

1. Prikaz celotnega grafa s statičnim prikazom trenutnih in preteklih relacij.
2. Prikaz razvoja grafa skozi čas z dinamičnim prikazom trenutnih in preteklih relacij.

Prikaz celotnega grafa s statičnim prikazom trenutnih in preteklih relacij sta izdelala prof. Mrvar in prof. Batagelj s programskim paketom Pajek. Na Sliki 2.2 sta narisana prikaza Kamada-Kawai za relacije iz prve in druge zgodbe.

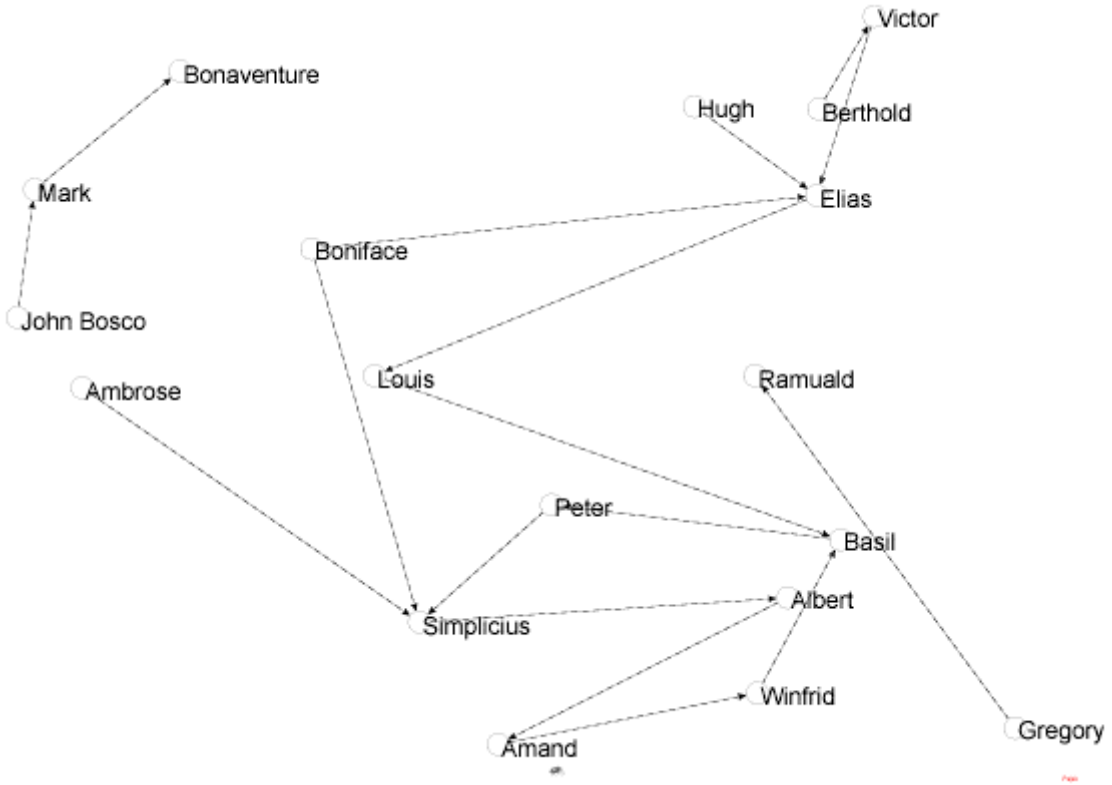


Slika 2.2: Prikaza Kamada-Kawai za relacije omrežja Lindenstrasse v prvi in drugi zgodbi.

Na Sliki 2.2 so povezave pobarvane po pripadnosti relacijam. Tako vijolična barva označuje partnersko relacijo, rdeča družinsko relacijo, zelena poslovno relacijo, modra prijateljsko relacijo in oranžna barva relacijo med osebami, ki se ne marajo.

2. 3. 2. 2 Omrežje Sampsonovih menihov

Podatki omrežja Sampsonovih menihov so dobljeni iz etnografske študije o strukturi socialne skupnosti v enem od angleških samostanov (Sampson, 1968; De Nooy, Mrvar, Batagelj, 2005, pogl. 4). Študija je preučevala socialne relacije med menihi-novinci, ki so se hoteli pridružiti meniškemu redu. Pridobljeni podatki so rezultat ankete, v kateri je moral vsak od novincev našteti tri kolege, s katerimi se razume, in še tri kolege, s katerimi se ne razume. Pri tem jih je bilo treba našteti po vrsti, torej od 1 do 3 (1 – osebi se občasno razumeta, 2 – osebi se večinoma razumeta, 3 - osebi se vedno razumeta) oziroma od -1 do -3 (-1 – osebi se občasno ne razumeta, -2 – osebi se večinoma ne razumeta, -3 – osebi se nikoli ne razumeta). Socialne relacije so bile merjene v petih časovnih trenutkih.
Na Sliki 2.3 je narisan prikaz Kamada-Kawai za negativno relacijo z oznako -1 v drugem časovnem trenutku (prva ponovitev ankete).



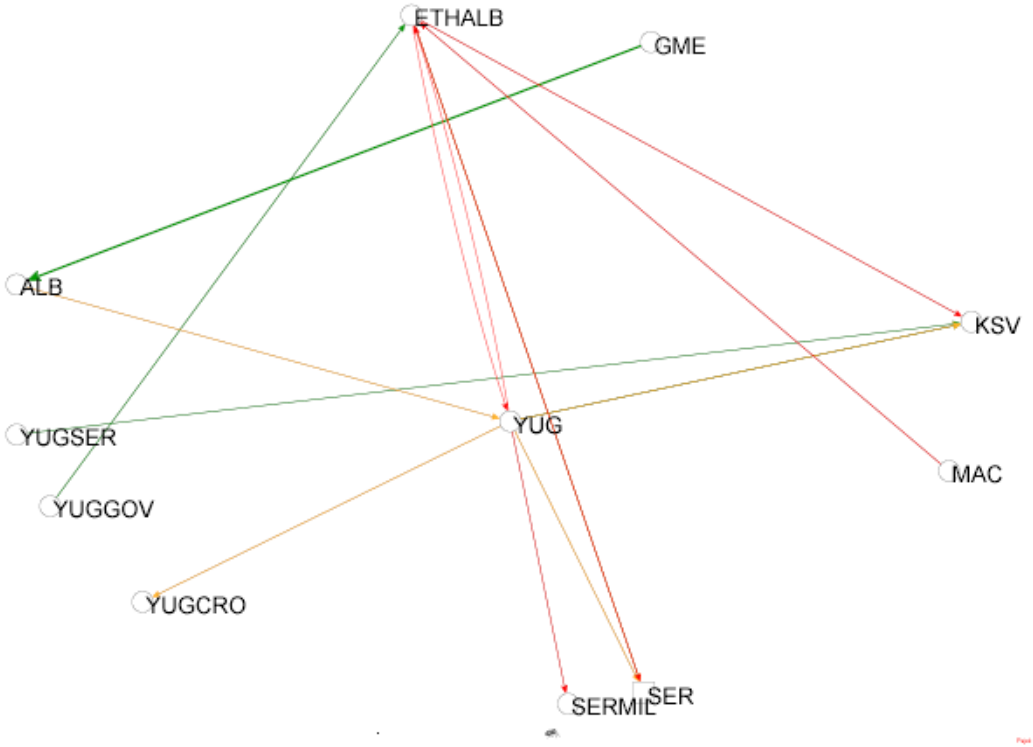
Slika 2.3: Prikaz Kamada-Kawai za negativno relacijo z oznako -1 omrežja Sampsonovih menihov v drugem časovnem trenutku.

Na Sliki 2.3 so prikazane samo povezave, ki pripadajo relaciji z oznako -1. Ta relacija povezuje osebi, ki se občasno ne razumeta.
Vse negativne relacije so v programskem paketu Pajek narisane črtkano.

2. 3. 2. 3 Omrežje političnih dogodkov KEDS na Balkanu

KEDS (Kansas Event Data System) je avtomatiziran sistem za generiranje podatkov o političnih dogodkih na Srednjem Vzhodu (od aprila 1979 do marca 1999), na Balkanu (od aprila 1989 do julija 2003) in v zahodni Afriki (od januarja 1989 do februarja 2002), ki ga je vzpostavil Oddelek za politične vede na Univerzi v Kansasu (Schrodt, Davis, Weddle, 2004; Batagelj, 2005, KEDS). Podatki so se generirali iz objavljenih novic o dogodkih na teh območjih.
Na podlagi pridobljenih podatkov je bilo izdelanih več statističnih modelov za napovedovanje političnih sprememb.

Na Sliki 2.4 je narisan krožni prikaz omrežja s podatki z Balkana v tretjem časovnem trenutku (junij 1989). Točke so razporejene na krogu glede na abecedni vrstni red oznak točk v smeri urinega kazalca. Točka z največ sosedi je ročno prestavljena na sredino.

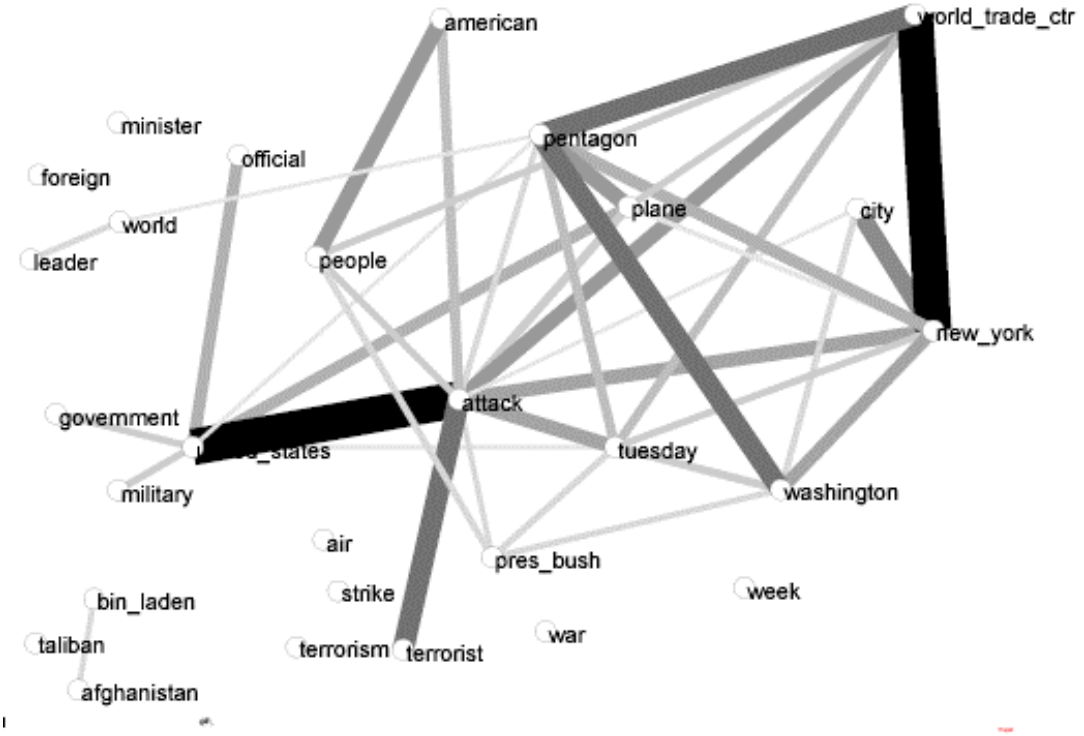


Slika 2.4: Krožni prikaz omrežja s podatki z Balkana v tretjem časovnem trenutku.

Na Sliki 2.4 so nekatere povezave dvostranske, npr. obstaja usmerjena povezava od etnične albanske skupine do bivše Jugoslavije in usmerjena povezava od bivše Jugoslavije do etnične albanske skupine. Te povezave so v Pajku zaradi boljše preglednosti narisane v obliki lokov in ne z ravnimi črtami.
Različne barve povezav zopet označujejo pripadnosti relacijam. Tako zelena barva označuje pozitivno, oranžna nevtralno in rdeča negativno relacijo.

2. 3. 2. 4 Omrežje Reutersovih novic o 11. septembru 2001

Ameriški znanstvenik Steve Corman je s sodelavci z univerze Arizona State University s posebno tehniko za analizo besedil, imenovano CRA (Centering Resonance Analysis), predelal dnevne Reutersove novice o 11. septembru 2001, ko se je zgodil teroristični napad na ZDA, v časovno omrežje sopojavljanja besed (Corman, Dooley, 2002). V analizo so bile vključene vse Reutersove novice, objavljene v 66 dneh, vključno z novicami na dan napada. Na Sliki 2.5 je narisan prikaz Kamada-Kawai za to omrežje na dan 11. september 2001.



Slika 2.5: Prikaz Kamada-Kawai omrežja Reutersovih novic na dan 11. september 2001.

Na Sliki 2.5 imajo povezave različne debeline in različne odtenke sive barve, ker smo v programskem paketu Pajek v slikovnem oknu uporabili izbiri Lines → Different Widths in Lines → Gray Scale. Na ta način lahko iz prikaza razberemo naslednja dejstva:

- na dan napada na ZDA je bilo največ novic o Svetovnem trgovskem centru v New Yorku,
- veliko novic je bilo povezanih s Pentagonom v Washingtonu,
- veliko novic je bilo tudi o tem, da je šlo za teroristični napad,
- malo manj, toda še vedno precej, je bilo novic o tem, da je bil napad usmerjen na Američane.

V nadaljevanju bomo spoznali program SoNIA in spletno storitev GAML, s katerima je že možno izdelati dinamične prikaze časovnih omrežij.

3 Pregled obstoječih programov za dinamični prikaz časovnih omrežij

3 . 1 Program SoNIA

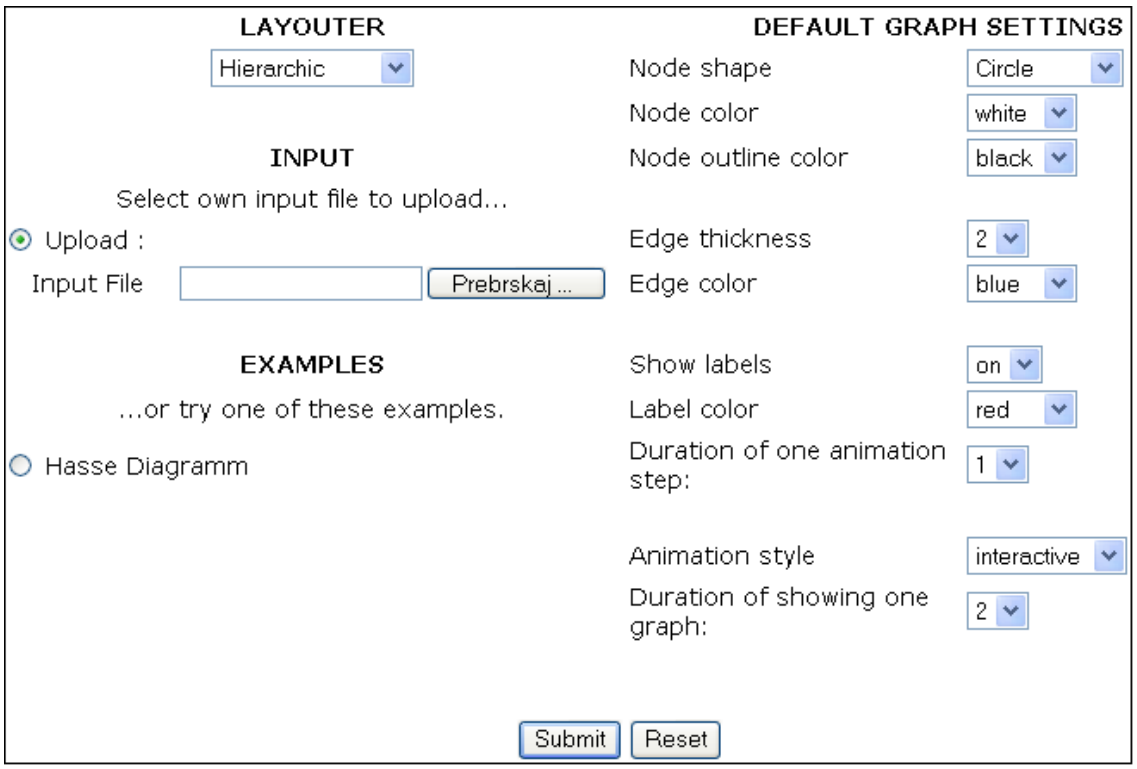
Do določene mere je možno podatke iz programskega paketa Pajek prenesti v program SoNIA (Bender-deMoll, McFarland, 2004), ki je bil izdelan prav za dinamični prikaz časovnih omrežij. SoNIA je zmogljivo orodje, narejeno v programskem jeziku Java, ki upošteva naslednje posebnosti časovnih omrežij:

- zvezno spreminjanje koordinat točk;
- izginjanje točk in povezav in pojavljanje novih točk in povezav glede na to, ali je določena točka oziroma povezava prisotna v določenem časovnem trenutku;
- spreminjanje barv in velikosti točk;
- spreminjanje barv in debelin povezav;
- prikaz večkratnih omrežij na istih točkah.

Program SoNIA omogoča, da se lahko dinamično prikazano časovno omrežje izvozi v datoteko filmskega tipa MOV, ki jo prikaže prosto dostopni prikazovalnik za filme QuickTime.
Pomanjkljivost tega programa je, da ne omogoča izvoza v format SVG. Jezik SVG (Ferraiolo, Jun, Jackson, 2003) je namreč standardni slikovni jezik, s katerim je mogoče izdelati spletne strani, ki vključujejo slike z visoko ločljivostjo (več o tej temi v poglavju 4).

3 . 2 Spletna storitev za dinamični prikaz časovnih omrežij

Na področju dinamičnih prikazov časovnih omrežij je zelo aktivna tudi nemška skupina, ki je izdelala spletno storitev, imenovano GAML (Diehl, Görg et al., 2005). Ta storitev ima vgrajenih več algoritmov za dinamične prikaze časovnih omrežij, ki zadoščajo tudi estetskim zahtevam, kot so zgoščenost, minimalno število križanj povezav in dinamična stabilnost časovnega omrežja (več o tej temi v poglavju 3.2.1).



Slika 3.1: Glavno okno spletne storitve GAML.

Glavno okno spletne storitve GAML (Slika 3.1) ponuja uporabniku poleg izbire vhodne datoteke še naslednje izbire: obliko in barvo točk ter barvo robov točk, debelino in barvo povezav, izbiro prikaza oznak točk in barvo oznak točk, trajanje dinamičnega prikaza omrežja od enega časovnega trenutka do drugega, trajanje statičnega prikaza omrežja v posameznem časovnem trenutku, slog prikaza (interaktiven / neprekinjen) in izbiro avtomatične optimizacije prikaza omrežja (hierarhična / napovedovalna). Napovedovalna optimizacija prikaza omrežja je predstavljena v poglavju 3.2.1.

Vhodna datoteka mora biti posebne oblike, in sicer oblike GraphML (Diehl, Görg et al., 2005), ki je ena od standardnih oblik predstavitve omrežja, osnovana na jeziku XML. Oglejmo si primer vhodne datoteke oblike GraphML.

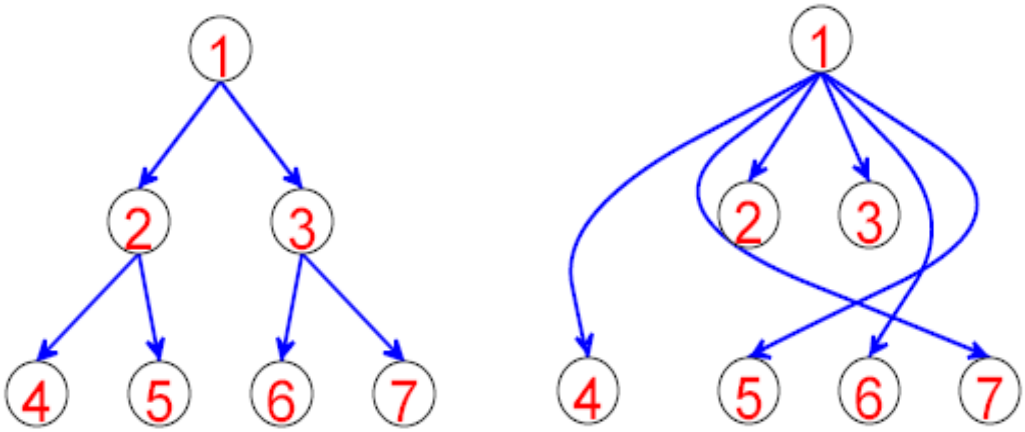
Primer: Vhodna datoteka spletne storitve GAML oblike GraphML.

```
<graphanimation>
  <graph>
    <node id="72" label="1"/>
    <node id="73" label="2"/>
    <node id="74" label="3"/>
```

```
    <node id="75" label="4"/>
    <node id="76" label="5"/>
    <node id="77" label="6"/>
    <node id="78" label="7"/>
    <edge id="79" sourceid="72" targetid="73"/>
    <edge id="80" sourceid="72" targetid="74"/>
    <edge id="81" sourceid="73" targetid="75"/>
    <edge id="82" sourceid="73" targetid="76"/>
    <edge id="83" sourceid="74" targetid="77"/>
    <edge id="84" sourceid="74" targetid="78"/>
  </graph>
  <graph>
    <node id="72" label="1"/>
    <node id="73" label="2"/>
    <node id="74" label="3"/>
    <node id="75" label="4"/>
    <node id="76" label="5"/>
    <node id="77" label="6"/>
    <node id="78" label="7"/>
    <edge id="79" sourceid="72" targetid="73"/>
    <edge id="80" sourceid="72" targetid="74"/>
    <edge id="85" sourceid="72" targetid="75"/>
    <edge id="86" sourceid="72" targetid="76"/>
    <edge id="87" sourceid="72" targetid="77"/>
    <edge id="88" sourceid="72" targetid="78"/>
  </graph>
</graphanimation>.
```

V jeziku GraphML smo opisali usmerjeno omrežje s sedmimi točkami v dveh časovnih trenutkih. Vsaka točka in povezava ima enolično identifikacijsko številko. Identifikacijske številke točk služijo za določanje krajišč povezav.

Spletna storitev GAML na podlagi vhodne datoteke izdela dinamični prikaz omrežja v obliki SVG. Na Sliki 3.2 je hierarhični prikaz danega omrežja v obliki SVG v obeh časovnih trenutkih.



Slika 3.2: Hierarhični prikaz omrežja v obliki SVG v dveh časovnih trenutkih.

3. 2. 1 Ohranjanje mentalne slike časovnega omrežja

V poglavju 2.3.1 smo omenili optimizacijo Kamada-Kawai za statični prikaz časovnih omrežij, ki minimizira energijo sistema točk in povezav v ravnini. V tem poglavju bomo spoznali estetske zahteve za dinamične prikaze časovnih omrežij in primer optimizacije dinamičnih prikazov, ki tem zahtevam zadošča.

Večina znanstvenih krogov, ki se ukvarjajo z dinamičnim prikazom časovnih omrežij, pozna pojem "ohranjanje mentalne slike omrežja" (Misue, Eades, Lai, Sugiyama, 1995). Za tem pojmom se skrivajo razne estetske zahteve, med katerimi sta najbolj pomembni naslednji:

- vrstni red točk mora ostati nespremenjen v vodoravni in navpični smeri,
- za vsako točko mora razdalja do vseh ali 'do skoraj vseh' sosednjih točk ostati (sorazmerno) enaka.

Klasični algoritmi za statični prikaz omrežij v vsakem časovnem trenutku na novo določijo prikaz celotnega omrežja. V večini primerov s tem pristopom ne moremo ohraniti mentalne slike omrežja. Lahko pa to dosežemo z alternativnim pristopom, pri katerem zahtevamo, da so spremembe na prikazih omrežja v časovnih trenutkih minimalne. Na ta način bi bilo laže vidno, kako se točke premaknejo iz enega položaja v drugega.

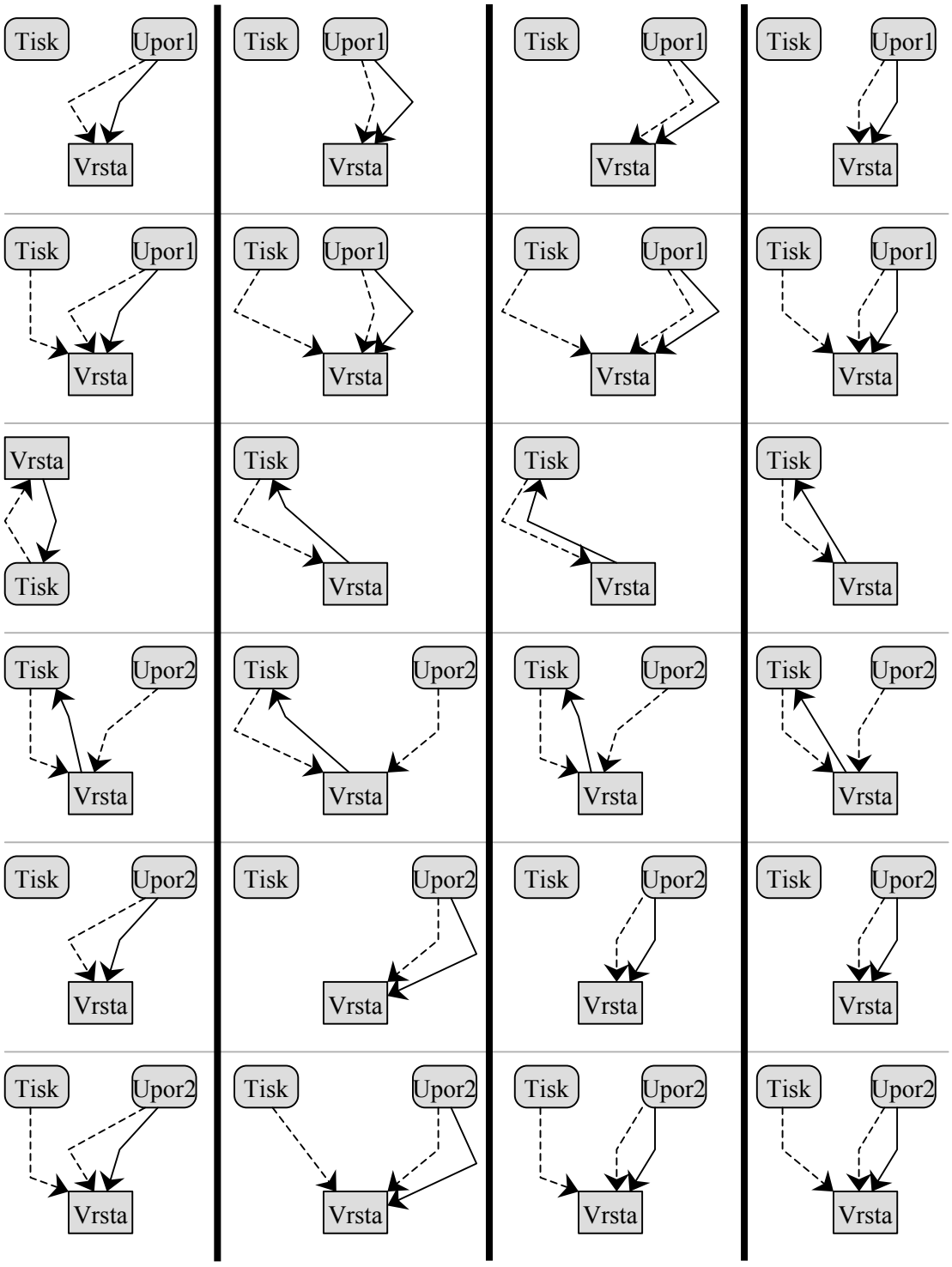
V iskanju boljše rešitve za dinamični prikaz časovnih omrežij, ki bi zadoščali tudi zahtevi za ohranitev mentalne slike omrežja, se pojavi ideja, da bi prikaz celotnega omrežja, dobljen z eno od optimizacij statičnih prikazov, določal prikaze omrežij v vseh časovnih trenutkih. Pri tem pristopu ima vsaka točka omrežja samo en položaj v vseh časovnih trenutkih, v katerih se pojavi. Prav tako se v časovnih trenutkih ne spreminja oblika upogiba povezav, če dopuščamo, da so povezave lahko prikazane v obliki lokov ali lomljenih črt. Na ta način v največji možni meri zadostimo obema zahtevama za ohranitev mentalne slike omrežja, lahko bi rekli, da smo pri tem celo preveč strogi, saj zahtevi dopuščata rahle premike položajev točk.

Iz začetne ideje o estetskih dinamičnih prikazih časovnih omrežij je nastala optimizacija, ki jo bomo poimenovali kar po njenih avtorjih, torej optimizacija Diehl-Görg, ki je znana tudi pod imenom "napovedovalna optimizacija zasnove slike" (angl. "Foresighted Layout").

Optimizacija Diehl-Görg je nastala na podlagi dejstva, da imamo v večini primerov opravka z redkimi omrežji in zato prikazi omrežij v posameznih časovnih trenutkih vsebujejo veliko praznega prostora (Diehl, Görg, Kerren, 2000, 2001). Ta problem rešujejo, da na podlagi intervalov prisotnosti točk množico točk razdelijo na skupine, in sicer dve točki pripadata določeni skupini, če imata ločena intervala prisotnosti. Taki točki sta potem prikazani na istem mestu, s čimer dobimo bolj zgoščen prikaz celotnega omrežija in s tem tudi bolj zgoščene prikaze omrežij v posameznih časovnih trenutkih. Na enak način nato razdelijo še množico povezav, ki so prikazane v obliki lokov ali lomljenih črt.

Oglejmo si primer uporabe optimizacije Diehl-Görg. Gre za primer s področja dodeljevanja virov. Predpostavimo, da imamo omrežni tiskalnik, ki ga uporablja več uporabnikov. V tem primeru imamo čakalno vrsto, ki deluje po načelu »Prvi noter, prvi ven«. Na Sliki 3.3 so prikazani rezultati štirih različnih optimizacij prikaza omrežja na primeru omrežnega tiskalnika. Pri tem usmerjena črtkana povezava pomeni nek zahtevek, na primer usmerjena črtkana povezava od točke Upor1 do točke Vrsta pomeni, da uporabnik Upor1 zahteva, da se sestavek, ki ga želi natisniti, vpiše v čakalno vrsto. Polna usmerjena povezava pa pomeni izključni dostop do nekega vira. Na primer, polna usmerjena povezava od točke Upor1 do točke Vrsta pomeni, da se sestavek, ki ga želi uporabnik Upor1 natisniti, lahko vpiše v čakalno vrsto, in da je v tem času čakalna vrsta zaklenjena za druge zahtevke.

Na Sliki 3.3 lahko opazimo, da se pri klasični optimizaciji za statični prikaz položaja točk Tisk in Vrsta dvakrat zamenjata, kar ne zadošča zahtevi, da mora vrstni red točk ostati nespremenjen v vodoravni in navpični smeri. V drugem stolpcu se položaji točk ne spreminjajo, kar zadošča zahtevi o vrstnem redu točk. Vendar bi se dalo ta prikaz še izboljšati, saj vidimo, da ima točka Upor1 interval prisotnosti [1-2], točka Upor2 pa [4-6]. Intervala prisotnosti sta torej ločena, kar pomeni, da bi lahko ti dve točki prikazali na istem mestu. To je vidno v tretjem stolpcu, kjer je prikazan rezultat optimizacije Diehl-Görg z razbitjem množice točk. Vendar tudi ta prikaz še ni optimalen.



Slika 3.3: Rezultati štirih optimizacij prikaza omrežja na primeru omrežnega tiskalnika v šestih časovnih trenutkih: klasične optimizacije za statični prikaz (prvi stolpec), osnovne optimizacije za dinamični prikaz z uporabo prikaza celotnega omrežja (drugi stolpec),optimizacije Diehl-Görg z razbitjem množice točk (tretji stolpec) in optimizacije Diehl-Görg z dodatnim razbitjem množice povezav (četrti stolpec).

Na primer, če primerjamo usmerjeno črtkano povezavo od točke Upor1 do točke Vrsta v prvem časovnem trenutku in usmerjeno črtkano povezavo od točke Upor2 do točke Vrsta v četrtem časovnem trenutku, hitro opazimo, da sta različni, čeprav bi lahko bili prikazani kot ena povezava in bi s tem pridobili na zgoščenosti prikaza omrežja. Ta zahteva je upoštevana v četrtem stolpcu, kjer je prikazan rezultat optimizacije Diehl-Görg z dodatnim razbitjem množice povezav. Če si dobro ogledamo prikaze omrežja v tem stolpcu, bomo opazili, da je tudi število povezav, potrebno za prikaz omrežja v vseh časovnih trenutkih, najmanjše možno in zato je ta optimizacija od vseh štirih optimizacij najboljša.

V naslednjem poglavju bomo spoznali zmogljivosti jezika za opis vektorskih slik SVG, ki smo ga izbrali za predstavitev dinamičnega prikaza časovnih omrežij.

4 Opis jezika za vektorske slike SVG

4.1 Kaj je jezik SVG?

Jezik SVG je javni označevalni jezik za opis vektorskih slik, s katerim je mogoče izdelati spletne strani, ki vključujejo slike z visoko ločljivostjo. Ima številne dobre lastnosti:

- slike v SVG so vektorske (brez izgube ločljivosti pri transformacijah);
- datoteke .SVG imajo manjšo velikost, kar pomeni hitrejše nalaganje, in lahko jih pregledujemo kar s spletnim brskalnikom, v katerega prej namestimo prosto dostopen dodatek za pregledovanje (Adobe SVG Viewer, 2004);
- v spletnem brskalniku deluje tudi iskanje nizov / besedil v sliki SVG;
- osnova jezika SVG je označevalni jezik XML (Bray et al., 2004), ki ima danes že vodilno vlogo pri izmenjavi najrazličnejših podatkov na spletu in tudi drugje.

Ker je osnova jezika SVG jezik XML, je jezik izredno prožen. Lahko se kombinira z drugimi standardnimi skriptnimi jeziki (Lilley, 2004):

- JavaScript-om, najbolj univerzalnim skriptnim jezikom za brskalnike (Horwat, Ginda, 2003), s katerim je mogoče dostopati do objektov, narejenih v jeziku SVG, in jih tudi nadzirati;
- XHTML-jem, ki povezuje jezika HTML in XML (Axelsson et al., 2002);
- Math ML-jem, ki omogoča pisanje matematičnih formul z jezikom XML (Ion, Miner, 1999);
- Xlink-om, ki omogoča uporabo hiperpovezav v sestavkih XML (DeRose, Maler, Orchard, 2001).

Razvoj jezika SVG podpirajo velika podjetja, kot so Adobe, Corel in W3C. Veliko strokovnjakov na področju razvoja spleta ocenjuje, da je to spletni slikovni jezik prihodnosti (Quint, 2005; Traversa, 2005; Trippe, 2002).

Zaradi vseh naštetih razlogov smo za predstavitev dinamičnega prikaza časovnih omrežij izbrali jezik SVG.

4.2 Zgradba opisa SVG

Opis SVG lahko definiramo kot samostojno datoteko ali kot vgrajeno datoteko v spletni strani.

Spodnji primer predstavlja samostojno datoteko .SVG:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
     width="300" height="300" x="0" y="0">
  ....
  ....
  ....
</svg>.
```

Prva vrstica vsebuje najavo XML glede na to, da je jezik XML osnova jezika SVG. Druga in tretja vrstica vsebujeta povezavo z opisom 'slovnice' jezika SVG (angl. Document Type Definition ali krajše DTD), ki mora biti tako kot najava XML sestavni del vsake datoteke .SVG.

Četrta in peta vrstica vsebujeta značko SVG, ki označuje, da gre za opis SVG. Značka SVG ima naslednje lastnosti:

- `xlmns`: obvezna lastnost, ki definira imenski prostor jezika SVG,
- `height` in `width`: določata velikost okna SVG na zaslonu,
- `x` in `y`: določata položaj okna SVG na zaslonu (jezik SVG ima privzeti koordinatni sistem postavljen tako, da je koordinatno izhodišče v zgornjem levem kotu zaslona).

Opis SVG lahko vključimo tudi kot vgrajeno datoteko v spletno stran:

```
<html>
<body>
<embed src="test.svg" width="500" height="500" type="image/svg+xml" />
</body>
</html>.
```

To naredimo z uporabo značke EMBED, ki ji določimo naslednje lastnosti:

- `src`: ime vgrajene datoteke .SVG (če se nahaja v drugi mapi kot datoteka .HTML, je potrebno navesti celotno pot),
- `height` in `width`: velikost vgrajenega okna SVG na spletni strani,
- `type`: zvrst datoteke oblike MIME (programski standard, ki definira enostaven mehanizem za opisovanje vsebinskih tipov datotek, ki se pošiljajo preko spleta) - za datoteke SVG so predvidene vrednosti image/svg, image/svg-xml in image/svg+xml.

4.3 Osnovni slikovni gradniki v jeziku SVG

Jezik SVG vsebuje naslednje osnovne slikovne gradnike:

- pravokotnik (značka `rect`),
- krog (značka `circle`),
- elipso (značka `ellipse`),
- črto (značka `line`),
- lomljeno črto (značka `polyline`),
- puščico (značka `marker`),
- mnogokotnik (značka `polygon`).

Oglejmo si primer pravokotnika:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300" height="300" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
   <rect x="20" y="20" width="250" height="100"
         style="fill:red;stroke:black;stroke-width:5;opacity:0.5"/>
</svg>.
```

Ta pravokotnik (Slika 4.1) ima zgornje levo oglišče v točki (20, 20). Širok je 250 in visok 100 px. Enota px je okrajšava za enoto piksel ali slikovno točko (angl. pixel), ki jo poznamo kot enoto za ločljivost zaslona (npr. 800 x 600 px), in je privzeta merska enota jezika SVG. Notranjost pravokotnika je rdeče barve, rob pravokotnika pa črne barve in debeline 5 px. Lastnost `opacity` z vrednostjo 0.5 pove, da je pravokotnik polprosojen.

Slika 4.1: primer pravokotnika, narisanega z jezikom SVG.

Drugi osnovni slikovni gradnik jezika SVG je krog. Oglejmo si primer:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300" height="300" version="1.1"
```

```
     xmlns="http://www.w3.org/2000/svg">
   <circle cx="100" cy="50" r="40" stroke="black"
          stroke-width="2" fill="royalblue"/>
</svg>.
```

Iz opisa SVG lahko preberemo, da gre za krog s središčem v točki (100, 50) in polmerom 40 px. Notranjost kroga je kraljevsko modre barve (jezik SVG ima 147 poimenovanih barv), rob kroga pa črne barve in debeline 2 px.

Slika 4.2: primer kroga, narisanega z jezikom SVG.

Krog, ki ga raztegnemo v smeri x ali y, preide v elipso. Zato ima elipsa namesto polmera dve polosi. Če sta polosi enako dolgi, pa je elipsa kar enaka krogu. Oglejmo si primer elipse v jeziku SVG, ki ima veliko polos dolgo 200 px, malo polos pa 80 px:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="500" height="500" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
   <ellipse cx="300" cy="150" rx="200" ry="80"
            style="fill:royalblue;stroke:black;stroke-width:2"/>
</svg>.
```

Če želimo z jezikom SVG narisati črto, uporabimo značko `line`:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300" height="300" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
   <line x1="0" y1="0" x2="300" y2="300"
         style="stroke:black;stroke-width:2"/>
</svg>.
```

Ta črta ima eno krajišče kar v koordinatnem izhodišču, drugo pa v točki (300, 300).

Z značko `line` v jeziku SVG definiramo črto, z značko `polyline` pa lomljeno črto:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="200" height="100" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
   <polyline points="10,10 10,20 20,20 20,40 40,40 40,60 150,60"
             style="fill:white;stroke:darkred;stroke-width:4"/>
</svg>.
```

V našem primeru je lomljena črta temno rdeče barve, debeline 4 px in je določena z naslednjimi točkami: (10, 10), (10, 20), (20, 20), (20, 40), (40, 40), (40, 60), (150, 60).

Če želimo na koncu lomljene črte prikazati puščico (Slika 4.3), moramo uporabiti še značko `marker` znotraj značke `defs`:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="200" height="100" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
   <defs>
      <marker id="Puscica" refX="0" refY="5"
              markerWidth="20" markerHeight="20" fill="darkred">
         <path d="M 0 0 L 10 5 L 0 10 z" />
      </marker>
   </defs>
   <polyline points="10,10 10,20 20,20 20,40 40,40 40,60 150,60"
             style="fill:white;stroke:darkred;stroke-width:4"
             marker-end="url(#Puscica)"/>
</svg>.
```

V zgornjem primeru ima značka `marker` naslednje lastnosti:

- `id`: ime puščice, na katero se sklicujemo v lastnosti `marker-end` značke `polyline`;
- `refX in refY`: relativna odmika puščice od končne točke lomljene črte v smereh x in y, potrebna za centriranje puščice;
- `markerWidth`: širina puščice;
- `markerHeight`: višina puščice;

Značka `marker` vsebuje tudi eno podznačko, to je značko `path`, s katero definiramo obliko puščice s potjo. Pri tem uporabimo naslednje ukaze:

- `M 0 0`: premakni se v točko (0, 0);
- `0 0 L 10 5`: nariši črto od točke (0, 0) do točke (10, 5);
- `10 5 L 0 10`: nariši črto od točke (10, 5) do točke (0, 10);
- `Z`: skleni pot, torej s črto poveži točki (0, 0) in (0, 10).

Slika 4.3: Lomljena črta s puščico, narisana z jezikom SVG.

Oglejmo si še, kako lahko z jezikom SVG definiramo mnogokotnik:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="500" height="500"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
   <polygon fill="red" stroke="blue" stroke-width="10"
            points="350,75  379,161 469,161 397,215
                    423,301 350,250 277,301 303,215
                    231,161 321,161" />
</svg>.
```

Uporabili smo značko `polygon` in ji določili lastnost `points`, ki predstavlja točke mnogokotnika. Na Sliki 4.4 je narisana peterokraka rdeča zvezda z modrim robom debeline 10 px, določena z naslednjimi točkami: (350, 75), (379, 161), (469, 161), (397, 215), (423, 301), (350, 250), (277, 301), (303, 215), (231, 161), (321, 161).

Slika 4.4: Peterokraka zvezda, narisana z jezikom SVG.

4 . 4 Uporabniška interakcija v jeziku SVG

Za uporabniško interakcijo je tako kot v jeziku HTML tudi v jeziku SVG poskrbljeno z značko `script`. Oglejmo si primer kroga, ki se mu spremeni velikost, ko uporabnik klikne nanj (Slika 4.5):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="250" height="250"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
   <script language="Javascript"> <![CDATA[
      function krog_klik(evt) {
         var krog = evt.target;
         var polmer = krog.getAttribute("r");
         if (polmer == 30)
            krog.setAttribute("r", polmer*2);
         else
            krog.setAttribute("r", polmer*0.5);
       }
   ]]> </script>
```

```
   <rect x="2" y="2" width="246" height="246" fill="none"
         stroke="blue" stroke-width="2"/>
   <circle onclick="krog_klik(evt)" cx="100" cy="100" r="30"
           style="fill:red;stroke:black;stroke-width:2"/>
   <g font-family="Verdana" font-size="16" text-anchor="left">
     <text x="15" y="200">
      S klikom na krog se
     </text>
     <text x="15" y="220">
      spremeni njegova velikost.
     </text>
   </g>
</svg>.
```

Ta opis SVG vsebuje funkcijo krog_klik, napisano v jeziku Javascript, ki se izvede ob kliku na krog (odziv na dogodek `onclick` znotraj značke `circle`).

V opisu lahko opazimo tudi znački `g` in `text`, ki ju do sedaj še nismo uporabili.

Značka `g` označuje skupino značk. Lastnosti, ki so podane v tej znački, pripadajo vsem značkam v skupini. Značka `text` pa označuje besedilo. V našem primeru imamo dve levo poravnani besedili s pisavo Verdana in velikostjo črk 16 px.

Slika 4.5: Primer kroga, ki se mu spremeni velikost, ko uporabnik klikne nanj.

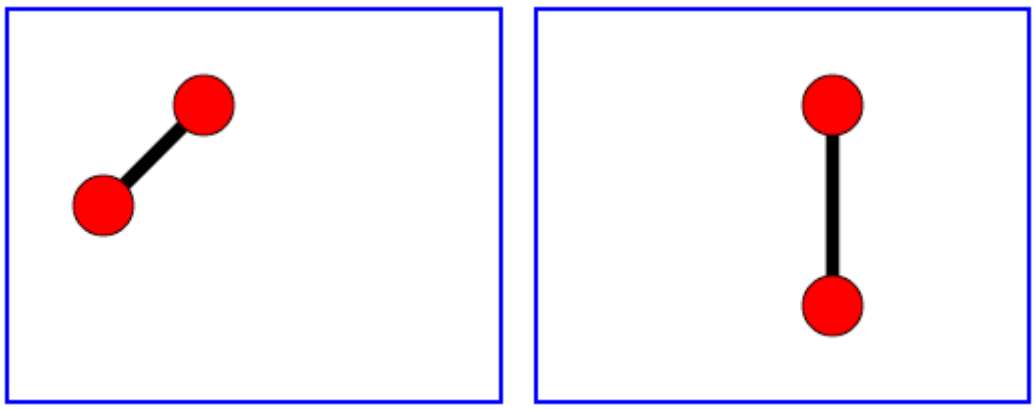
4 . 5 Dinamični prikaz objektov v jeziku SVG

Jezik SVG ima za dinamični prikaz objektov na voljo več značk. Spoznali bomo osnovno značko `animate`, ki se uporablja za dinamični prikaz sprememb vrednosti lastnosti objektov SVG skozi čas. Oglejmo si primer grafa na dveh točkah, povezanih z neusmerjeno povezavo, ki se po določenem času počasi premakne na drug položaj (Slika 4.6):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/2001/PR-SVG-20010719/ DTD/svg10.dtd">
<svg width="250" height="200"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
   <rect x="2" y="2" width="246" height="196"
         fill="none" stroke="blue" stroke-width="2" />
   <line x1="50" y1="100" x2="100" y2="50"
         stroke-width="7" stroke="black">
      <animate attributeName="x1" from="50" to="150"
               begin="10s" dur="10s" fill="freeze"/>
      <animate attributeName="y1" from="100" to="150"
               begin="10s" dur="10s" fill="freeze"/>
      <animate attributeName="x2" from="100" to="150"
               begin="10s" dur="10s" fill="freeze"/>
   </line>
   <circle cx="50" cy="100" r="15" style="fill:red;stroke:black">
      <animate attributeName="cx" from="50" to="150"
               begin="10s" dur="10s" fill="freeze"/>
      <animate attributeName="cy" from="100" to="150"
               begin="10s" dur="10s" fill="freeze"/>
   </circle>
   <circle cx="100" cy="50" r="15" style="fill:red;stroke:black">
      <animate attributeName="cx" from="100" to="150"
               begin="10s" dur="10s" fill="freeze"/>
   </circle>
</svg>.
```

Iz primera lahko vidimo, da ima značka `animate` naslednje lastnosti:

- `attributeName`: ime lastnosti nadrejenega objekta, ki se ji spremeni vrednost;
- `from` in `to`: stara in nova vrednost te lastnosti;
- `begin`: začetek dinamičnega prikaza spremembe vrednosti lastnosti;
- `dur`: trajanje animacije;
- `fill`: izbira, ki vpliva na to, ali nadrejeni objekt po koncu dinamičnega prikaza ohrani novo vrednost lastnosti (`freeze`) ali jo zamenja s staro vrednostjo (`remove`).



Slika 4.6: Dve točki in povezava med njima na začetku in na koncu dinamičnega prikaza.

4.6 Upravljanje objektov SVG v spletnih sestavkih

V poglavju 4.4 smo predstavili način, kako lahko s pomočjo jezika Javascript omogočimo uporabniško interakcijo v jeziku SVG. V tem poglavju pa bomo jezik Javascript uporabili za upravljanje objektov SVG v spletnih sestavkih.

Oglejmo si primer spletnega sestavka, v katerem preko potrditvenih polj s funkcijo, napisano v jeziku Javascript, vplivamo na to, ali se skupina objektov SVG prikaže ali ne:

```
<html>
  <head>
    <title> Svg animation from source Pajek file </title>
    <script language="JavaScript1.2">
      function hilite_elem (checkbox, element_name)
      {
       var svgobj;
       var svgstyle;
       var svgdoc = document.network.getSVGDocument();
       svgobj = svgdoc.getElementById(element_name);
       svgstyle = svgobj.getStyle();
       if (!checkbox.checked)
         { svgstyle.setProperty('display', 'none');
         }
       else
         { svgstyle.setProperty('display', 'inline');
         }
      }
      function hilite_all()
      {
       var svgobj;
       var svgstyle;
       var svgdoc = document.network.getSVGDocument();
       var strel=0;
       for (var i = 1;  i < 3;  i++) {
         svgobj = svgdoc.getElementById(i+'');
         svgstyle = svgobj.getStyle();
         svgstyle.setProperty('display', 'inline');
       }
       while (strel < document.hilite_form.relation.length) {
         document.hilite_form.relation[strel].checked = true;
         strel++
       }
      }
    </script>
  </head>
  <body bgcolor="darkgreen" text="yellow" link="#FFFFFF"
        vlink="#FFFF00" alink="#FFFF00" onload="hilite_all()">
    <center>
      <table>
       <tr>
        <td valign="top">
          <embed  src="linden.svg.gz" name="network" width="784"
                  height="548" type="image/svg-xml" pluginspage=
                  "http://www.adobe.com/svg/viewer/install/"/>
        <td valign="top">
          <form name="hilite_form">
```

```
            <table  border="1" cellpadding="5" cellspacing="1"
                    width="100%" bgcolor="white">
              <tr>
                <td bgcolor="darkgreen">
                  <input  type="checkbox" name="relation" value=""
                          onclick="hilite_elem(this,'1')" />
                </td>
                <td  bgcolor="rgb(194,002,000)">
                  <font color="black">1</font>
                </td>
                <td bgcolor="darkgreen"> Poslovna relacija</td>
              </tr>
              <tr>
                <td bgcolor="darkgreen">
                  <input  type="checkbox" name="relation" value=""
                          onclick="hilite_elem(this,'2')" />
                </td>
                <td bgcolor="rgb(255,127,076)">
                  <font color="black">2</font>
                </td>
                <td bgcolor="darkgreen"> Družinska relacija</td>
              </tr>
            </table>
          </form>
        </td>
      </td>
    </tr>
  </table>
 </center>
</body>
</html>.
```

Ta spletni sestavek vsebuje dve potrditveni polji, ki ob kliku pokličeta funkcijo `hilite_elem` z dvema parametroma, napisano v jeziku Javascript. V prvem parametru navedemo vgrajeni objekt `this`, v katerem so shranjene lastnosti potrditvenega polja, v drugem parametru pa navedemo vrednost lastnosti `id` objekta SVG, ki ga želimo upravljati v spletnem sestavku. V našem primeru želimo upravljati z dvema objektoma SVG. Prvi objekt ima vrednost lastnosti `id` enako 1, drugi objekt pa 2. Oba objekta predstavljata skupino objektov, zato del opisa SVG s tema objektoma izgleda takole:

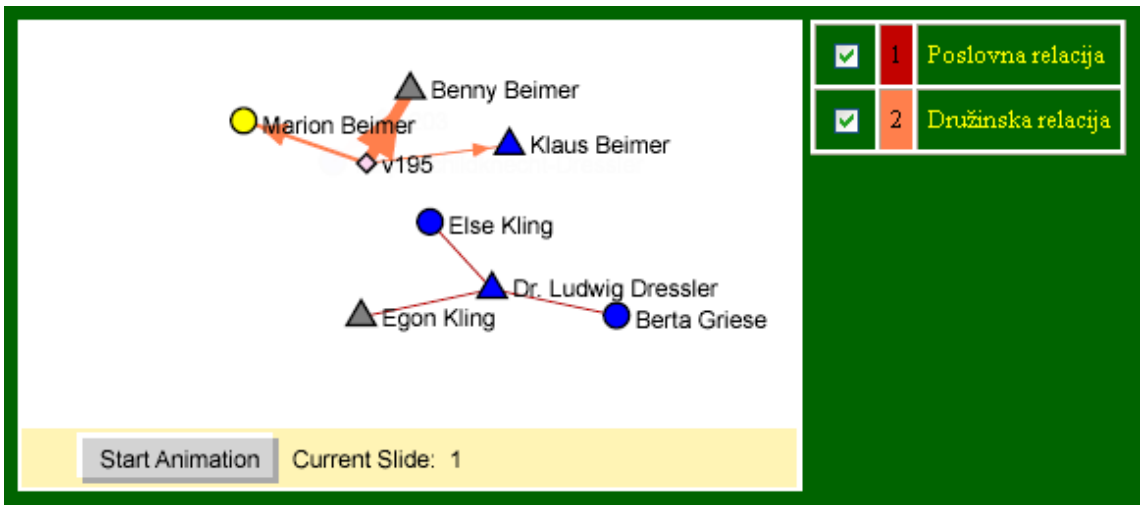
```
<g id="1">
...
</g>
<g id="2">
...
</g>.
```

Funkcija `hilite_elem` preko lastnosti `ID` dostopa do objektov in upravlja z njima s pomočjo naslednjih vgrajenih funkcij:

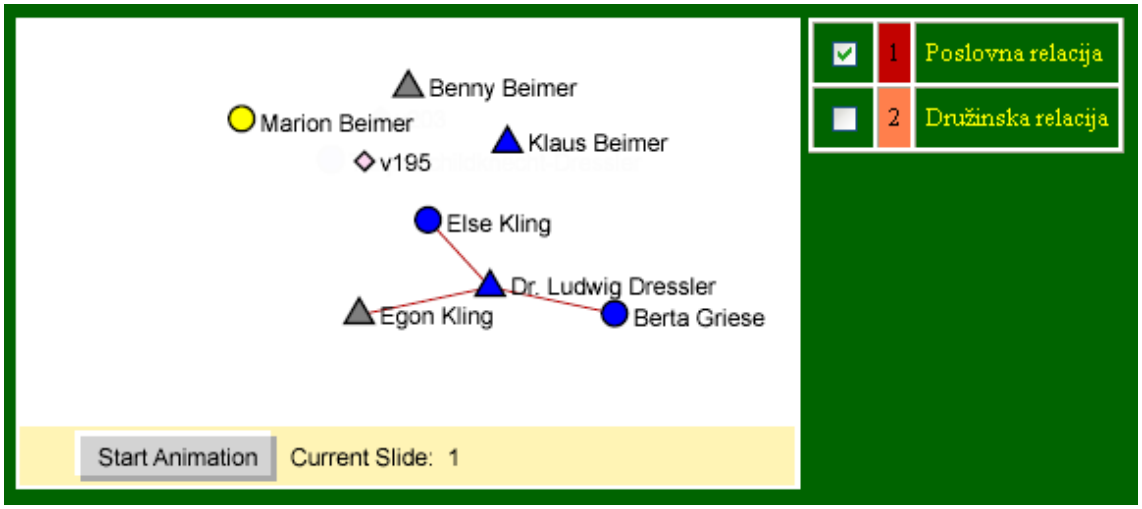
- `getElementById:` funkcija, ki dostopa do objekta SVG,
- `getStyle:` funkcija, ki prebere stilske lastnosti objekta SVG,

- `setProperty`: funkcija, ki spremeni določeno lastnost objekta SVG.

V našem primeru preko potrditvenega polja v spletnem sestavku z vgrajeno sliko SVG s pomočjo funkcije `setProperty` vplivamo na to, ali so povezave, ki pripadajo poslovni relaciji, vidne ali ne. Na enak način vplivamo na prikaz povezav, ki pripadajo družinski relaciji. Na Sliki 4.7a so vidne povezave obeh relacij, na Sliki 4.7b pa samo povezave poslovne relacije.



Slika 4.7a: Spletni sestavek z vgrajeno sliko SVG, v katerem smo izbrali obe potrditveni polji, zato so vidne povezave obeh relacij.



Slika 4.7b: Spletni sestavek z vgrajeno sliko SVG, v katerem smo izbrali samo prvo potrditveno polje, zato so vidne samo povezave poslovne relacije.

V nadaljevanju bomo spoznali programski jezik Python, ki smo ga uporabili za izdelavo slikovnega uporabniškega vmesnika in za izdelavo končnih slikovnih datotek .SVG.

5 Opis programskega jezika Python

5 . 1 Kaj je jezik Python?

V magistrskem delu so metode izdelave programa PajekToSvgAnim vezane predvsem na značilnosti programskega jezika Python (Van Rossum, 2004), ki je bil uporabljen za izdelavo slikovnega uporabniškega vmesnika in za izdelavo končnih slikovnih datotek .SVG. Spodaj so naštete glavne značilnosti jezika Python:

- je tolmačeni interaktivni objektni programski jezik;
- tako kot programski jezik Java omogoča izdelavo modulov, razredov, izjem in dinamičnih podatkovnih tipov;
- nove vgrajene module je mogoče pisati s programskim jezikom C ali C++;
- izvorna koda je javna in je prenosljiva med operacijskimi sistemi UNIX, Windows, OS/2, Mac, Amiga in še mnogimi drugimi;
- uporablja zamikanje za določanje strukture programa;
- njegova standardna knjižnica ima široko paleto uporabnih modulov, od modulov za podporo elektronski pošti, jezikov HTML, XML in drugih označevalnih jezikov, protokola FTP in mnogih drugih internetnih protokolov, do modulov za podporo kriptografiji, arhitekturama COM in CORBA, vmesnikom do večine podatkovnih baz in ogrodjem za izdelavo slikovnih uporabniških vmesnikov.

5 . 2 Podatki v jeziku Python

Ko odpremo delovno okno jezika Python, lahko ukazno vrstico uporabljamo kot 'računalo'. Osnovni operatorji (+, -, *, /) delujejo tako kot v večini programskih jezikov. Oglejmo si nekaj primerov računanja s števili:

```
>>> 2+2
4
>>> # to je vrstično pojasnilo
... 2+2
4
>>> 2+2 # vrstično pojasnilo v isti vrstici skupaj z izrazom
4
>>> (50-5*6)/4
5
>>> # rezultat deljenja dveh celih števil je največje celo število, ki
... # je manjše ali enako dobljenemu realnemu številu:
```

```
... 7/3
2
>>> 7/-3
-3
>>> # če hočemo za zgornji primer dobiti za rezultat število s
... # plavajočo vejico, imamo na razpolago naslednje možnosti:
... 7.0/3
2.3333333333333335
>>> 7/3.0
2.3333333333333335
>>> 7.0/3.0
2.3333333333333335
>>> # ena od možnosti je tudi uporaba vgrajene funkcije za pretvorbo
... # iz celega števila v število s plavajočo vejico:
... float(7)/3
2.3333333333333335
>>> # kot v programskem jeziku C se tudi v jeziku Python znak za
... # enačaj uporablja za to, da se spremenljivki priredi vrednost:
... dolzina = 5
>>> sirina = 4
>>> dolzina * sirina
20
>>> # kompleksna števila so v jeziku Python predstavljena z izrazom
... # real+imagj ali izrazom real+imagJ:
... 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j)/(1+1j)
(1.5+0.5j)
>>> 1j * 1J
(-1+0j)
>>> lahko pa uporabimo tudi vgrajeno funkcijo za definiranje
... kompleksnega števila:
>>> 1j * complex(0,1)
(-1+0j)
>>> # če želimo iz komplesnega števila dobiti realno in imaginarno
... # komponento, uporabimo vgrajeni lastnosti real in imag:
... a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5.
```

Nize lahko v jeziku Python definiramo z enojnima ali dvojnima narekovajema. Poseben pomen ima znak "\", s katerim lahko v niz vključimo posebne znake. Oglejmo si nekaj primerov:

```
>>> ’Zdravo’
’Zdravo’
>>> # če niz definiramo z enojnima narekovajema in je enojni
... # narekovaj tudi del niza, pred tem znakom uporabimo znak "\":
... ’Kako s\’ kaj?’
"Kako s' kaj?"
>>> # če pa niz definiramo z dvojnima narekovajema, nam znaka "\" ni
... # treba uporabiti:
... "Kako s’ kaj?"
"Kako s’ kaj?"
```

```
>>> odgovor1 = 'V redu.\n\tPa ti?'
>>> odgovor1
'V redu.\n\tPa ti?'
>>> # če v nizu uporabimo posebna znaka "\n" in "\t", ju jezik Python
... # pri izpisu razume kot skok v novo vrstico in skok s
... # tabulatorjem:
... print odgovor1
V redu.
       Pa ti?
>>> # če je niz predolg, da bi ga napisali v eni vrstici, uporabimo
... # znak "\" na koncu vrstice za nadaljevanje niza v naslednji
... # vrstici:
... odgovor2 = 'Tudi v redu. Zdaj, ko sem prebolel prehlad, \n\
... grem koncno spet lahko ven na zrak.'
>>> print odgovor2
Tudi v redu. Zdaj, ko sem prebolel prehlad,
grem koncno spet lahko ven na zrak.
>>> mesto = 'Ljubljana'
>>> # do delov niza dostopamo na zelo enostaven način z uporabo
... # indeksov:
... mesto[0]
'L'
>>> mesto[0:4]
'Ljub'
>>> mesto[5:]
'jana'
>>> mesto[-3:] # vrne zadnje tri znake niza
'ana'
>>> 'Ljub'+'ljana' # z operatorjem + lahko dva niza zlepimo skupaj
'Ljubljana'.
```

Jezik Python pozna tudi tri sestavljene podatkovne tipe. To so:

- seznam,
- slovar,
- nabor.

Seznam definiramo z oglatima oklepajema, med katerima navedemo z vejico ločene elemente, ki so lahko različnih podatkovnih tipov. Elemente seznama lahko tudi spreminjamo in brišemo. Spodaj so opisane osnovne možnosti pri delu s seznami:

```
>>> a = [] # prazen seznam
>>> b = [’Ljubljana’, ’Maribor’]
>>> c = [’a’, 1, b]
>>> c
[’a’, 1, [’Ljubljana’, ’Maribor’]] # sezname lahko poljubno gnezdimo
>>> # tako kot pri nizih lahko z uporabo indeksov pridemo do
... # posameznih elementov seznama ali do delov seznama:
... c[0]
’a’
>>> c[1:]
[1, [’Ljubljana’, ’Maribor’]]
>>> len(c) # dolžino seznama izvemo s pomočjo vgrajene funkcije len
3
```

```
>>> # Če želimo izvedeti, ali je nek podatek element seznama,
... # uporabimo operator in. Rezultat je logična vrednost 1 ali 0:
... 'Ljubljana' in b
1
>>> 'Celje' in b
0
>>> c[1] = 2 # drugemu elementu seznama (indeks 1) spremenimo vrednost
>>> c
['a', 2, ['Ljubljana', 'Maribor']]
>>> c.append(3) # na koncu seznama dodamo nov element
>>> c
['a', 2, ['Ljubljana', 'Maribor'], 3]
>>> # V Pythonu je vsak podatkovni tip objektni tip s svojimi
... # lastnostmi in metodami. Seznam ima kot objektni tip več metod.
... # Ena od njih je index, s pomočjo katere izvemo indeks določenega
... # elementa seznama:
... c.index(2)
1
>>> c.reverse() # z metodo reverse lahko seznam obrnemo
>>> c
[3, ['Ljubljana', 'Maribor'], 2, 'a']
>>> # če hočemo iz seznama brisati enega ali več elementov,
... # uporabimo Pythonov ukaz del:
>>> del c[1:2]
>>> c
[3, 'a']
>>> # v jeziku Python lahko definiramo tudi zaporedje:
... range(4)
[0, 1, 2, 3]
>>> range(5,11)
[5, 6, 7, 8, 9, 10]
>>> range(7,20,3) # zaporedje od 7 do 20 s korakom 3
[7, 10, 13, 16, 19].
```

Drugi sestavljen podatkovni tip v jeziku Python je slovar. Definiramo ga z zavitima oklepajema, med katerima navedemo z vejico ločene pare *ključ: vrednost*. Spet si oglejmo nekaj primerov:

```
>>> s = {} # prazen slovar
>>> # v slovar lahko shranimo telefonski imenik:
>>> S = {'Sara': '041/231869', 'Borut': '031/543286'}
>>> S # Python uredi slovar po ključih v abecednem redu
{'Borut': '031/543286', 'Sara': '041/231869'}
>>> S['Sara'] # takole s pomočjo ključa pridemo do vrednosti
'041/231869'
>>> S.keys() # seznam vseh ključev v slovarju
['Borut', 'Sara']
>>> S.values() # seznam vseh vrednosti v slovarju
['031/543286', '041/231869']
>>> len(S) # velikost slovarja
2
>>> # Lahko tudi preverimo, če slovar vsebuje določen ključ ali ne.
... # Pri tem moramo paziti, ker Python loči med malimi in velikimi
... # črkami:
... S.has_key('sara') # naš imenik vsebuje ključ 'Sara'
0.
```

V jeziku Python lahko definiramo tudi nabor. To je sestavljen podatkovni tip, ki je podoben seznamu in se od njega loči po dveh stvareh: določen je z okroglima oklepajema (seznam z oglatima oklepajema) in njegovih elementov ne moremo spreminjati (elemente seznama lahko).

Podatke lahko shranimo v datoteko. Osnovni ukazi za pisanje na datoteko in za branje iz nje so preprosti:

```
datPisi = open('dat1.txt', 'w') # datoteko odpremo za pisanje
datBeri = open('dat2.txt', 'r') # datoteko odpremo za branje
datPisi.write(s)                # na datoteko zapišemo niz s
podat = datBeri.read()          # naenkrat preberemo celo datoteko
podat = datBeri.readline()      # naenkrat preberemo eno vrstico
datPisi.close()                 # zapremo datoteko.
```

5 . 3 Krmilni stavki v jeziku Python

Osnovni krmilni stavki vsakega programskega jezika so zaporedja, vejitveni stavki in zanke. V jeziku Python ima pogojni stavek `if` obliko:

```
if p1 :
   stavki1
elif p2 :
   stavki2
else:
   stavki3.
```

Pri pisanju programa moramo paziti na zamikanje kode, saj jezik Python uporablja zamik za določanje strukture programa. Oglejmo si primer uporabe stavka `if`:

```
>>> # spodaj je primer programčka, ki prebere celo število in nato
... # izpiše, ali je bilo vnešeno negativno ali nenegativno število
... x = int(input("Vnesi celo število: "))
Vnesi celo število: 4
>>> if x < 0:
...    print 'Vnesli ste negativno celo število.'
... elif x >= 0:
...    print 'Vnesli ste nenegativno celo število.'
...
Vnesli ste pozitivno celo število.
>>> # če bi namesto celega števila vnesli število s plavajočo vejico,
... # bi funkcija int izrezala celi del:
... x = int(input("Vnesi celo število: "))
Vnesi celo število: 4.7
>>> x
4.
```

Včasih želimo v programu kake ukaze ponavljati, dokler je izpolnjen dani pogoj. To naredimo z zankami. Jezik Python pozna dve vrsti zank: zanko `while` in zanko `for`. Zapišemo ju na naslednja načina:

```
while pogoj:
  stavki1
else:
  stavki2
```

```
for i in seznam:
  stavki1
else:
  stavki2.
```

Obe zanki lahko vsebujeta stavek `else`, ki se izvrši, ko pogoj ni več izpolnjen (zanka `while`) ali ko se zanka ustavi (zanka `for`).

Če želimo na primer izpisati na zaslon števila od 1 do 100, lahko uporabimo zanko `while` ali zanko `for`:

```
>>> i = 1
>>> while i < 101:
...   print i
...   i = i + 1
... else:
...   print 'Konec.'
```

```
>>> for i in range(1,101):
...   print i
... else:
...   print 'Konec'.
```

Pri programiranju v jeziku Python imamo tudi možnost upravljanja s standardnimi vgrajenimi izjemami. To nam omogoča stavek `try..except`:

```
try:
  stavki1
except izjema:
  stavki2.
```

Oglejmo si primer, ko poskušamo niz pretvoriti v celo število z vgrajeno funkcijo `int`:

```
>>> int('Ljubljana')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ValueError: invalid literal for int(): Ljubljana

>>> try:
...   int('Ljubljana')
... except ValueError:  # uporabimo vgrajeno izjemo ValueError
...   print 'Napaka pri pretvorbi!'
...
Napaka pri pretvorbi!
```

5.4 Funkcije v jeziku Python

V jeziku Python definiramo funkcijo z rezervirano besedo `def` na naslednji način:

```
def ime_funkcije (p1,p2,...,pn):
  stavki
  return vrednost.
```

Oglejmo si primer funkcije, ki za določeno ime barve iz programskega paketa Pajek vrne oznako barve v obliki RGB:

```
>>> def dolociBarvoRGB(p_ime_barve='greenyellow'):
...   barva=''
...   if p_ime_barve=='greenyellow':
...     barva='rgb(217,255,079)'
...   elif p_ime_barve=='yellow':
...     barva='rgb(255,255,000)'
...   elif p_ime_barve=='goldenrod':
...     barva='rgb(255,230,040)'
...   else:
...     barva='rgb(000,000,000)'
...   return barva # poenostavljen primer, ki deluje samo za tri barve
...
>>> dolociBarvoRGB() # parametru funkcije smo določili privzeto vred.
'rgb(217,255,079)'
>>> dolociBarvoRGB('yellow')
'rgb(255,255,000)'.
```

5.5 Moduli v jeziku Python

Ko zapremo Pythonovo delovno okno, izgubimo vse spremenljivke in funkcije, ki smo jih definirali in uporabljali. Če tega nočemo, se moramo pisanja programov lotiti na drugačen način, to je z uporabo modulov, ki jih pozna večina programskih jezikov, med njimi tudi jezik Python.

V splošnem je modul program, shranjen v datoteki s posebnim podaljškom, ki je odvisen od izbranega programskega jezika. V jeziku Python se uporablja podaljšek PY. Modul lahko napišemo v Pythonovem ali kateremkoli drugem urejevalniku besedil in ga nato shranimo v datoteko .PY. Shranjene module lahko uporabimo v novih programih s pomočjo ukaza `import`. Oglejmo si primer uporabe modula.

Primer: Recimo, da imamo v modulu *barve.py* (v glavni mapi jezika Python) shranjeno funkcijo *dolociBarvoRGB*, ki smo jo definirali v poglavju 5.4.

To funkcijo bi sedaj želeli uporabiti v novem modulu, v katerem bi definirali proceduro (funkcija, ki ne vrne nič), ki oznako barve v obliki RGB izpiše na zaslon. Nov modul lahko napišemo na tri načine:

1. način:
```
import barve
def izpisiBarvoRGB(p_ime_barve):
  # pri klicu funkcije moramo navesti tudi modul
  rgb = barve.dolociBarvoRGB(p_ime_barve)
  print rgb
```

2. način:
```
from barve import dolociBarvoRGB
def izpisiBarvoRGB(p_ime_barve):
  # pri klicu funkcije ni treba navesti modula
  rgb = dolociBarvoRGB(p_ime_barve)
  print rgb
```

3. način:
```
from barve import *
def izpisiBarvoRGB(p_ime_barve):
  # pri klicu funkcije spet ni treba navesti modula
  rgb = dolociBarvoRGB(p_ime_barve)
  print rgb.
```

Jezik Python ima tudi veliko vgrajenih modulov. Med njimi so najpomembejši naslednji moduli:

- `__builtin__`: modul z vgrajenimi funkcijami, ki so dostopne v vseh Pythonovih modulih;
- `exceptions`: modul s standardnimi vgrajenimi izjemami;
- `math`: matematični modul;
- `os`: modul, ki služi kot vmesnik do velikega števila funkcij operacijskega sistema;
- `string`: modul za delo z nizi;
- `sys`: Pythonov sistemski modul;
- `time`: modul za delo z datumi in časi.

Oglejmo si primer uporabe nekaterih metod modula `os`:

```
>>> import os   # prikličemo modul os
>>> os.getcwd() # metoda getcwd vrne trenutno delovno mapo
'c:\\'
>>> os.chdir("pajek")   # z metodo chdir spremenimo delovno mapo
>>> os.getcwd()
'c:\\pajek'
>>> os.chdir(os.pardir) # delovna mapa postane nadrejena mapa
>>> os.getcwd()
'c:\\'.
```

V naslednjem poglavju je predstavljen program PajekToSvgAnim za dinamični prikaz časovnih omrežij, ki temelji na jezikih Python in SVG.

6 Predstavitev programa PajekToSvgAnim za dinamični prikaz časovnih omrežij

6 . 1 Opis oblike vhodne Pajkove datoteke

Program PajekToSvgAnim iz vhodne Pajkove projektne datoteke .PAJ izdela datoteke .SVG, .SVG.GZ in .HTML (datoteka .SVG.GZ je stisnjena oblika GZIP datoteke .SVG, ki omogoča hitrejše nalaganje in hitrejši prikaz objektov SVG). Vhodna Pajkova datoteka mora imeti naslednjo strukturo (za primer večkratnega omrežja s številom točk *N*, številom relacij *R* in številom časovnih trenutkov *M*):

***Network** *ime omrežja*
***Vertices** *N*
1 "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
2 "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
…
N "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
***Arcs :1** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Arcs :2** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
…
***Arcs :R** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Edges :1** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Edges :2** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
…
***Edges :R** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Network** *ime omrežja v času* T_1
***Vertices** N_1
1 "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
2 "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
…
$\mathbf{N_1}$ "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
***Arcs :1** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Arcs :2** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
…
***Arcs :R** "*ime relacije*"

tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Edges :1** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Edges :2** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
…
***Edges :R** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Network** *ime omrežja v času* T_2
***Vertices** N_2
1 "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
2 "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
…
$\mathbf{N_2}$ "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
***Arcs :1** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Arcs :2** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
…
***Arcs :R** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Edges :1** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Edges :2** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
…
***Edges :R** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
…
***Network** *ime omrežja v času* T_M
***Vertices** N_M
1 "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
2 "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
…
$\mathbf{N_M}$ "*ozn*" *koordX koordY vred oblika* **x_fact** *raztX* **y_fact** *raztY* **ic** *barva* **[***prisotn***]**
***Arcs :1** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Arcs :2** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
…
***Arcs :R** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Edges :1** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**
…
***Edges :2** "*ime relacije*"
tocka_zač tocka_kon vred **w** *deb* **c** *barva* **[***prisotn***]**

```
   …
…
*Edges  :R  "ime relacije"
   tocka_zač    tocka_kon    vred    w deb    c barva    [prisotn]
   …
*Partition ime_razbitja
*Vertices N
1 / 0
1 / 0   } N
…
1 / 0
```

Na začetku vhodne Pajkove datoteke se nahajajo podatki o celotnem omrežju, torej o vseh točkah in povezavah skupaj z njihovimi lastnostmi. Sledijo podatki o omrežjih v posameznih časovnih trenutkih. Za ta omrežja so podani samo podatki o tistih točkah in povezavah, ki so dejansko prisotne v časovnih trenutkih.
Na koncu vhodne datoteke se nahaja še točkovno razbitje, pri katerem za vsako točko navedemo, ali je v prikazu njena oznaka vidna ali ne (vrednost 1: oznaka je vidna, vrednost 0: oznaka ni vidna).

Lastnosti točk in povezav v posameznih omrežjih so standardne lastnosti, ki so določene v programskem orodju Pajek. Tako lahko točki določimo naslednje lastnosti:

- oznako,
- kooordinati x in y (in z),
- vrednost,
- obliko,
- razteg v smeri x (program zaenkrat upošteva ta razteg tudi za smer y, zato bomo v nadaljevanju uporabljali oznako faktorr kot faktor raztega) ,
- razteg v smeri y (program zaenkrat namesto tega raztega upošteva kar razteg v smeri x),
- barvo,
- interval prisotnosti (npr. [1-3, 7, 12-15]).

Povezavi pa lahko določimo naslednje lastnosti:

- začetno krajišče (obvezen podatek),
- končno krajišče (obvezen podatek),
- vrednost,
- debelino (če ni podana, se privzame kar vrednost povezave),

- barvo,
- interval prisotnosti.

Program zaenkrat omogoča spreminjanje koordinat pri točkah in spreminjanje debeline pri povezavah. Naslednja nadgradnja programa pa bo omogočala tudi spreminjanje velikosti točke, torej spreminjanje faktorja raztega oblike točke.

6.2 Predstavitev metod za izdelavo dinamičnega prikaza časovnih omrežij v jeziku SVG

Izdelava dinamičnega prikaza se začne s prenosom podatkov iz Pajkove vhodne datoteke v Pythonove podatkovne strukture. Tako se za točke izdelajo naslednji seznami:

- Tocke: seznam lastnosti točk po časovnih trenutkih.

  Primer:

```
Tocke[1]=[ {}, {},
           { 'st': 2, 'oznaka': 'Andrej', 'x': 0.1000,
             'y': 0.9000, 'vred': 5, 'oblika': 'box',
             'faktorr': 2, 'barva': 'rgb(0,0,255)'},
           { 'st': 2, 'oznaka': 'Andrej', 'x': 0.6000,
             'y': 0.7000, 'vred': 5, 'oblika': 'box',
             'faktorr': 2, 'barva': 'rgb(0,0,255)'},
           { 'st': 2, 'oznaka': 'Andrej', 'x': 0.6000,
             'y': 0.7000, 'vred': 5, 'oblika': 'box',
             'faktorr': 2, 'barva': 'rgb(0,0,255)'},
           { 'st': 2, 'oznaka': 'Andrej', 'x': 0.1000,
             'y': 0.9000, 'vred': 5, 'oblika': 'box',
             'faktorr': 2, 'barva': 'rgb(0,0,255)'}]
Tocke[2]=[ {}, {}, {},
           { 'st': 2, 'oznaka': 'Borut', 'x': 0.2000,
             'y': 0.3000, 'vred': 7, 'oblika': 'diamond',
             'faktorr': 0.5, 'barva': 'rgb(0,255,0)'},
           { 'st': 2, 'oznaka': 'Borut', 'x': 0.4000,
             'y': 0.1000, 'vred': 7,
             'oblika': 'diamond', 'faktorr': 0.5, 'barva':
             'rgb(0,255,0)'}, {}].
```

V zgornjih dveh seznamih točk vsak slovar (med zavitima oklepajema) predstavlja podatke za točko v enem časovnem trenutku. Če prvi element seznama izpustimo, ker želimo, da se oštevilčenje začne z 1, potem lahko naštejemo 5 časovnih trenutkov.
Točka z indeksom 1 ali krajše točka 1 je vidna od drugega do petega časovnega trenutka, točka 2 pa samo v tretjem in četrtem trenutku.

Točka 1 ima oznako Andrej, je modre barve, ima faktor raztega 2, obliko kvadrata in vrednost 5. Točka 2 pa ima oznako Borut, zeleno barvo, faktor raztega 0.5, obliko romba in vrednost 7. Točki 1 se spremenita koordinati v drugem in petem časovnem trenutku, točki 2 pa v tretjem.

- Tocke_ind: seznam oznak točk, uporabljen tudi kot indeks za iskanje po točkah.

  Primer: `Tocke_ind = ['', 'Andrej', 'Borut', 'Sara', 'Vesna'].`

  Točka z oznako Borut ima indeks 2, točka z oznako Sara pa indeks 3. Do indeksa točke z oznako Borut pridemo s Pythonovim ukazom `Tocke_ind.index('Borut')`. Do oznake pa pridemo z ukazom `Tocke_ind[2]`.

  Za prvi element seznama smo določili kar prazen niz. To smo naredili zato, da se indeks dejansko začne s številko 1.

- Tocke_vred: seznam vrednosti točk.

  Primer: `Tocke_vred = ['', 5, 7, 6, 4].`

  Točka z oznako Borut ima vrednost 7, točka z oznako Sara pa 6.

- Tocke_oblike: seznam oblik tock.

  Primer:
  ```
  Tocke_oblike = [ '', 'box', 'diamond', 'triangle',
                       'ellipse'].
  ```

  Točka z oznako Borut ima obliko romba, točka z oznako Sara pa obliko trikotnika.

- Tocke_faktorr: seznam faktorjev raztegov točk.

  Primer: `Tocke_faktorr = ['', 2, 0.5, 1, 2].`

  Velikost točke z oznako Borut je dvakrat manjša od privzete velikosti, točka z oznako Sara pa ima kar privzeto velikost.

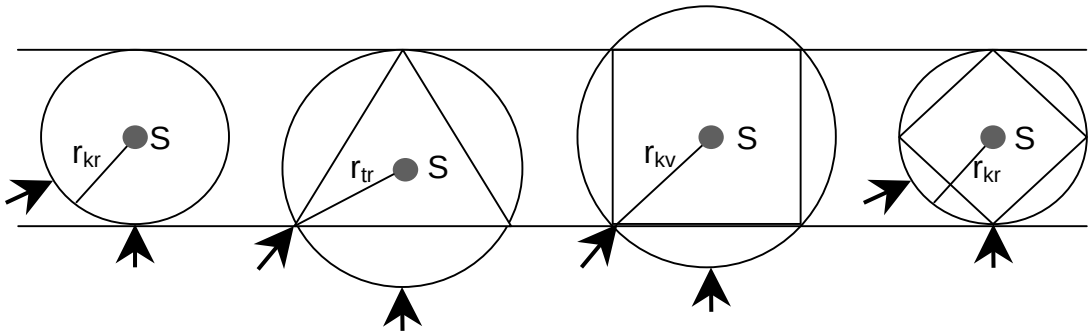
- Tocke_barve: seznam barv tock.

  Primer:
  ```
  Tocke_barve = ['', 'rgb(0,0,255)', 'rgb(0,255,0)',
                     'rgb(255,255,0)', 'rgb(255,0,0)'].
  ```

  Točka z oznako Borut je zelene barve, točka z oznako Sara pa rumene.

- Tocke_krogi: seznam polmerov oblikam točk očrtanih krogov, ki jih potrebujemo zaradi različnih položajev usmerjenih povezav med dinamičnim prikazom.

  Primer: Naj bo privzeta velikost točke 6 enot in naj bo to kar polmer kroga kot osnovne oblike točke, ki je tudi sam sebi očrtan krog (Slika 6.1).



Slika 6.1: Različnim oblikam točk očrtani krogi.

Glede na to, da so vse oblike enake višine, lahko na podlagi te velikosti izračunamo tudi polmere očrtanih krogov ostalih oblik točk. Če ima točka npr. obliko trikotnika, lahko izračunamo

$v = 2r_{kr}$, $r_{tr} = \frac{2}{3}v = \frac{4}{3}r_{kr} = 8$.

Polmer kvadratu očrtanega kroga je enak $r_{kv} = 6\sqrt{2}$, polmer rombu očrtanega kroga pa je enak polmeru osnovnega kroga, ki je enak 6.

Na primeru seznama oblik tock

```
Tocke_oblike = ['',  'box', 'diamond', 'triangle',
                    'ellipse']
```

in primeru seznama faktorjev raztegov točk

```
Tocke_faktorr = ['', 2, 0.5, 1, 2]
```

lahko sedaj zapišemo seznam polmerov oblikam točk očrtanih krogov:

```
Tocke_krogi = ['', 12√2, 3, 8, 12].
```

- Tocke_tren: seznam intervalov prisotnosti točk.

  Primer:
```
Tocke_tren =  [ [[1, 1, 0], [2, 5, 1]],
                [[1, 2, 0], [3, 4, 1], [5, 5, 0]],
                [[1, 5, 1]],
                [[1, 4, 0], [5, 5, 1]] ].
```

Naj bo to seznam intervalov prisotnosti točk za seznam točk Tocke_ind = ['', 'Andrej', 'Borut', 'Sara', 'Vesna']. Prvi dve številki v vsakem notranjem seznamu sta meji intervala, tretja številka pa označuje prisotnost (1) ali odsotnost (0) točke v tem intervalu. Če pogledamo istoležne elemente danih seznamov Tocke_ind in Tocke_tren, vidimo, da se točka z oznako Borut pojavi v tretjem in četrtem časovnem trenutku, točka z oznako Sara pa samo v prvem.

Za izdelavo seznama polmerov oblikam točk očrtanih krogov smo uporabili privzeto velikost točke 6 merskih enot. Pri tem nismo navedli imena merske enote. Zato bi količina 6 enot lahko pomenila 6 cm ali 6 inch ali še kaj tretjega, npr. 6 pikslov ali krajše 6 px. Enota px nam lahko pride prav pri določanju velikosti okna SVG. Namreč če želimo imeti okno SVG veliko kar čez cel zaslon, je najlažje reči, da je veliko toliko, kot je ločljivost zaslona. Zato smo se pri izbiri merske enote za vse objekte SVG odločili za enoto px.

Vrnimo se še na primer seznama lastnosti točke 2 po časovnih trenutkih:

```
Tocke[2] =  [{}, {}, {},
             {'st': 2, 'oznaka': 'Borut', 'x': 0.2000, 'y': 0.3000,
              'vred': 7, 'oblika': 'diamond', 'faktorr': 0.5, 'barva':
              'rgb(0,255,0)'},
             {'st': 2, 'oznaka': 'Borut', 'x': 0.4000, 'y': 0.1000,
              'vred': 7, 'oblika': 'diamond', 'faktorr': 0.5, 'barva':
              'rgb(0,255,0)'}, {}].
```

Ta točka ima v tretjem časovnem trenutku koordinati (0.2, 0.3), v četrtem trenutku pa (0.4, 0.1). Ker ima točka obliko romba, koordinati (0.2, 0.3) v tretjem časovnem trenutku predstavljata središče rombu očrtanega kroga s polmerom 3 (Tocke_krogi[2] = 3). V četrtem časovnem trenutku pa ima ta krog koordinati (0.4, 0.1).
Ostanimo pri tem primeru in se osredotočimo na najmanjšo in največjo vrednost obeh koordinat skupaj. Najmanjša vrednost je 0.1 in največja 0.4. V splošnem sta v programskem paketu Pajek ti dve vrednosti 0 in 1. Glede na to, da smo za objekte SVG izbrali mersko enoto px, je potrebno pretvoriti tudi koordinate danih točk.
Če je na primer velikost okna SVG 800 x 600 px, se koordinati (0.2, 0.3) pretvorita v koordinati (160, 180), koordinati (0.4, 0.1) pa v koordinati (320, 60).

Ko smo prenesli podatke o točkah iz Pajkove vhodne datoteke v Pythonove sezname, naredimo enako še za povezave. Povezava je lahko usmerjena ali neusmerjena, zato ločeno izdelamo sezname za usmerjene in neusmerjene povezave:

- Upovez, Npovez: seznama lastnosti usmerjenih in neusmerjenih povezav po relacijah in po časovnih trenutkih.

  Primer:
```
Upovez[2][1] = [ {}, {}, {},
                 { 'ime': 'A2F1T2', 'od': 1, 'do': 2,
                   'vred': 7, 'debelina': 4, 'barva':
                   'rgb(000,000,000)'},
                 { 'ime': 'A2F1T2', 'od': 1, 'do': 2,
                   'vred': 7, 'debelina': 4, 'barva':
                   'rgb(000,000,000)'}, {}].
```

  Usmerjena povezava z imenom A2F1T2 poteka od točke 1 do točke 2, ima vrednost 7, je črne barve in je prisotna v tretjem in četrtem časovnem trenutku. V obeh trenutkih ima debelino 4.

- Upovez_ind: seznam umetnih oznak usmerjenih povezav po relacijah, uporabljen tudi kot indeks za iskanje po usmerjenih povezavah.

  Primer: `Upovez_ind = [ [], [], ['','A2F1T2','A2F3T1'],[],[],[]].`

  Tokrat potrebujemo sezname znotraj seznama, ker imamo lahko opravka z večrelacijskim omrežjem. V našem primeru imamo pet relacij, od katerih ima le druga dve usmerjeni povezavi, ostale relacije pa nimajo nobene usmerjene povezave. Za prvi element seznama relacij vedno določimo prazen seznam [], ker želimo številčenje relacij začeti z 1. Do oznake prve usmerjene povezave druge relacije pridemo s Pythonovim ukazom `Upovez_ind[2][1]`. Rezultat je niz 'A2F1T2', ki pomeni, da gre za usmerjeno povezavo od točke s številko 1 do točke s številko 2 v drugi relaciji. Pri tem smo uporabili angleške črke, ker oznake povezav kasneje vpišemo v datoteko .SVG, ki je zaradi mednarodne uporabe v angleškem jeziku.

- Npovez_ind: seznam umetno ustvarjenih oznak neusmerjenih povezav po relacijah, uporabljen tudi kot indeks za iskanje po neusmerjenih povezavah.

Primer:
```
Npovez_ind = [ [], ['', 'E1F2T3', 'E1F2T4'], [],
               ['', 'E3F1T4'], [], []].
```

V tem primeru ima prva relacija dve neusmerjeni povezavi, tretja relacija eno, ostale relacije pa nimajo neusmerjenih povezav.

- Upovez_od, Npovez_od : seznama začetnih krajišč usmerjenih in neusmerjenih povezav po relacijah (za neusmerjene povezave je sicer vseeno, katero je začetno in katero končno krajišče, vendar ju bomo uporabljali tudi tu zaradi usklajenosti z imeni povezav).

Primer: Iz primera seznama oznak
`Upovez_ind = [[], [], ['','A2F1T2','A2F3T1'], [], [], []]`
dobimo `Upovez_od = [[], [], ['', 1, 3], [], [], []].`

- Upovez_do, Npovez_do : seznama končnih krajišč usmerjenih in neusmerjenih povezav po relacijah.

Primer: Iz primera seznama oznak usmerjenih povezav
`Upovez_ind = [[], [], ['','A2F1T2','A2F3T1'], [], [], []]`
dobimo `Upovez_do = [[], [], ['', 2, 1], [], [], []].`

- Upovez_vred, Npovez_vred : seznama vrednosti usmerjenih in neusmerjenih povezav po relacijah.

Primer: Seznam `Upovez_vred = [[], [], ['', 7, 10], [], [], []]`
v povezavi s seznamom
`Upovez_ind = [[], [], ['','A2F1T2','A2F3T1'], [], [], []]`
pove, da ima usmerjena povezava z oznako A2F1T2 vrednost 7 in povezava z oznako A2F3T1 vrednost 10.

- Upovez_barve, Npovez_barve : seznama barv usmerjenih in neusmerjenih povezav po relacijah.

Primer:
```
Upovez_barve = [ [], [],
                 ['',  'rgb(000,000,000)',
                       'rgb(000,000,000)'], [], [], []].
```

Ta seznam v povezavi s seznamom Upovez_ind pove, da sta usmerjeni povezavi z oznakama A2F1T2 in A2F3T1 črne barve.

- Upovez_tren, Npovez_tren: seznam intervalov prisotnosti usmerjenih in neusmerjenih povezav po relacijah (odvisna od seznama prisotnosti točk, ker je povezava prisotna samo takrat, ko sta prisotni obe krajišči).

  Primer: Zapišimo seznam največjih možnih intervalov prisotnosti za usmerjene povezave za že omenjeni primer prisotnosti točk:

```
Tocke_tren =  [ [[1, 1, 0], [2, 5, 1]],
                [[1, 2, 0], [3, 4, 1], [5, 5, 0]],
                [[1, 5, 1],
                [[1, 4, 0], [5, 5, 1] ].
```

  Če pri tem upoštevamo seznam usmerjenih povezav po relacijah

```
Upovez_ind = [[], [], ['','A2F1T2','A2F3T1'], [], [], []],
```

  ugotovimo, da ima usmerjena povezava v drugi relaciji od točke 1 do točke 2 največji možni interval prisotnosti [[1, 2, 0], [3, 4, 1], [5, 5, 0]], usmerjena povezava od točke 3 do točke 1 pa ima največji možni interval prisotnosti [[1, 1, 0], [2, 5, 1]], kot presek intervalov prisotnosti obeh točk.

- Upovez_deb, Npovez_deb: seznam intervalov vrednosti debelin za usmerjene in neusmerjene povezave po relacijah.

  Primer: `Upovez_deb[2] = [[], [[3,3,1.5], [4,4,4]], [[2,5,5]]].`

  V tem primeru imamo podatke o debelinah za dve usmerjeni povezavi, ki pripadata drugi relaciji, in sicer prva povezava ima v tretjem časovnem trenutku debelino 1.5 in v četrtem 4, druga povezava pa ima od drugega do petega časovnega trenutka debelino 5.

Ko smo podatke iz vhodne Pajkove datoteke prenesli v Pythonove sezname, lahko na podlagi teh seznamov izdelamo naslednje objekte v jeziku SVG:

- objekt za kontrolo nad cikličnim zaporedjem časovnih trenutkov,
- objekte za oblike točk (elipsa, pravokotnik, trikotnik, romb),
- objekte za povezave, vključno z objekti za puščice pri usmerjenih povezavah,
- objekte za dinamični prikaz točk in povezav,
- objekt za vrstico z gumbom, s katerim lahko uporabnik dinamični prikaz začasno prekine ali ga znova požene od tam, kjer je bil prekinjen.

V poglavju 4.3 smo že spoznali nekaj osnovnih objektov SVG, sedaj pa bomo z osnovnimi objekti zgradili večje objekte, ki bodo omogočili dinamični prikaz. Najprej si oglejmo primer objekta SVG za kontrolo nad cikličnim zaporedjem petih časovnih trenutkov:

```
<rect x="0" y="0" width="0" height="0">
  <animate   id="AnimFRS1" attributeName="x" from="0" to="0"
             begin="1s; AnimFRS4.end+1.0" dur="1.0s" fill="freeze"
             onbegin="firstSlideWait()" onend="moveSlide(2)"/>
  <animate   id="AnimFRS2" attributeName="x" from="0" to="0"
             begin="AnimFRS1.end+1.0" dur="1.0s" fill="freeze"
             onend="moveSlide(3)"/>
  <animate   id="AnimFRS3" attributeName="x" from="0" to="0"
             begin="AnimFRS2.end+1.0" dur="1.0s" fill="freeze"
             onend="moveSlide(4)"/>
  <animate   id="AnimFRS4" attributeName="x" from="0" to="0"
             begin="AnimFRS3.end+1.0" dur="1.0s" fill="freeze"
             onend="moveLastSlide(5)"/>
</rect>.
```

Kot vidimo, gre za namišljen prazen okvir (angl. frame), za katerega se začne dinamični prikaz eno sekundo po začetnem statičnem prikazu. Dinamični prikaz iz enega časovnega trenutka v drugega spet traja eno sekundo, toliko trajajo tudi vmesni statični prikazi. Eno sekundo po koncu petega dinamičnega prikaza se spet začne prvi dinamični prikaz, torej gre za ciklične dinamične prikaze, ki trajajo po eno sekundo, z vmesnimi statičnimi prikazi, ki prav tako trajajo po eno sekundo.

Opisani objekt SVG skrbi za ciklično zaporedje časovnih trenutkov na dva načina:

- Na koncu vsakega dinamičnega prikaza od enega do drugega časovnega trenutka obvesti uporabnika o aktualnem časovnem trenutku tako, da zaporedno številko tega trenutka prikaže v spodnji vrstici okna SVG, kjer se nahaja tudi gumb za začasno prekinitev in nadaljevanje dinamičnega prikaza. Za to nalogo skrbi procedura moveSlide, ki je napisana v jeziku Javascript.
- Pred začetkom prvega in po koncu zadnjega dinamičnega prikaza začasno prekine celotni prikaz in čaka na uporabnika, da jo s pritiskom na gumb požene naprej. To nalogo opravljata proceduri firstSlideWait in moveLastSlide, tudi napisani v jeziku Javascript.

Ko imamo objekt, ki skrbi za ciklično zaporedje časovnih trenutkov, lahko z njegovo pomočjo povežemo objekte oblik točk in objekte za dinamični prikaz točk.
Primer: Predpostavimo, da je ločljivost zaslona 800 x 600 in da je privzeta velikost

točke 6 px. Točka 2 iz prejšnjih primerov ima naslednje lastnosti:

```
Tocke[2]=[ {}, {}, {},
           {'st': 2, 'oznaka': 'Borut', 'x': 0.2000,
            'y': 0.3000, 'vred': 7, 'oblika': 'diamond',
            'faktorr': 0.5, 'barva': 'rgb(0,255,0)'},
           { 'st': 2, 'oznaka': 'Borut', 'x': 0.4000,
            'y': 0.1000, 'vred': 7, 'oblika': 'diamond',
            'faktorr': 0.5, 'barva': 'rgb(0,255,0)'}, {}].
```

Ugotovili smo že, da se v novem koordinatnem sistemu koordinati (0.2, 0.3) pretvorita v koordinati (160, 180) in koordinati (0.4, 0.1) v koordinati (320, 60). Točka je prisotna v tretjem in četrtem časovnem trenutku.
Objekt, ki ustreza opisu točke, ima potem v jeziku SVG naslednji zapis:

```
<!-- Vertex V2: Borut -->
<polygon  points="154,180 160,186 166,180 160,174 154,180"
          style="fill:rgb(0,255,0);stroke:black;
          stroke-width:1.5" opacity="0">
  <animate   id="AnimV2S2" attributeName="points"
             from="154,180 160,186 166,180 160,174 154,180"
             to="154,180 160,186 166,180 160,174 154,180"
             begin="AnimFRS2.begin" dur="1.0s" fill="freeze"/>
  <animate   id="AnimV2S3" attributeName="points"
             from="154,180 160,186 166,180 160,174 154,180"
             to="314,60 320,66 326,60 320,54 314,60"
             begin="AnimFRS3.begin" dur="1.0s" fill="freeze"/>
  <animate   id="AnimV2S2Opa" attributeType="CSS"
             attributeName="opacity" from="0" to="1"
             begin="AnimFRS2.begin" dur="1.0s" fill="freeze"/>
  <animate   id="AnimV2S4Opa" attributeType="CSS"
             attributeName="opacity" from="1" to="0"
             begin="AnimFRS4.begin" dur="1.0s" fill="freeze"/>
</polygon>.
```

Če bi imela točka z oznako Borut obliko kroga, pa bi imel ustrezen objekt SVG naslednji zapis:

```
<!-- Vertex V2: Borut -->
<circle  cx="160" cy="180" r="6" style="fill:rgb(0,255,0);
         stroke:black;stroke-width:1.5" opacity="0">
  <animate id="AnimV2S2" attributeName="cx"
           from="160" to="160" begin="AnimFRS2.begin"
           dur="1.0s" fill="freeze"/>
  <animate id="AnimV2S2" attributeName="cy"
           from="180" to="180" begin="AnimFRS2.begin"
           dur="1.0s" fill="freeze"/>
  <animate id="AnimV2S3" attributeName="cx"
           from="160" to="320" begin="AnimFRS3.begin"
           dur="1.0s" fill="freeze"/>
  <animate id="AnimV2S3" attributeName="cy"
           from="180" to="60" begin="AnimFRS3.begin"
           dur="1.0s" fill="freeze"/>
```

```
  <animate id="AnimV2S2Opa" attributeType="CSS"
           attributeName="opacity" from="0" to="1"
           begin="AnimFRS2.begin" dur="1.0s" fill="freeze"/>
  <animate id="AnimV2S4Opa" attributeType="CSS"
           attributeName="opacity" from="1" to="0"
           begin="AnimFRS4.begin" dur="1.0s" fill="freeze"/>
</circle>.
```

To sta bila dva primera objekta SVG za obliko točke, ki vsebujeta tudi objekt za dinamični prikaz. Sedaj bomo spoznali še malo bolj zahteven primer, ko bomo združili objekt za povezavo, objekt za puščico za usmerjeno povezavo in objekt za dinamični prikaz.

Primer: Spomnimo se že predstavljenega primera usmerjene povezave

```
Upovez[2][1] = [ {}, {}, {},
                 { 'ime': 'A2F1T2', 'od': 1, 'do': 2,
                   'vred': 7, 'debelina': 4, 'barva':
                   'rgb(000,000,000)'},
                 { 'ime': 'A2F1T2', 'od': 1, 'do': 2,
                   'vred': 7, 'debelina': 4, 'barva':
                   'rgb(000,000,000)'},{}].
```

Ta povezava poteka od točke 1 do točke 2, ima vrednost 1, je črne barve in je prisotna v tretjem in četrtem časovnem trenutku. V obeh trenutkih ima debelino 4. Točka 1 in 2 sta točki iz prejšnjih primerov z oznakama Andrej in Borut z naslednjimi lastnostmi:

```
Tocke[1]=[ {}, {},
           { 'st': 2, 'oznaka': 'Andrej', 'x': 0.1000,
             'y': 0.9000, 'vred': 5, 'oblika': 'box',
             'faktorr': 2, 'barva': 'rgb(0,0,255)'},
           { 'st': 2, 'oznaka': 'Andrej', 'x': 0.6000,
             'y': 0.7000, 'vred': 5, 'oblika': 'box',
             'faktorr': 2, 'barva': 'rgb(0,0,255)'},
           { 'st': 2, 'oznaka': ' Andrej ', 'x': 0.6000,
             'y': 0.7000, 'vred': 5, 'oblika': 'box',
             'faktorr': 2, 'barva': 'rgb(0,0,255)'},
           { 'st': 2, 'oznaka': ' Andrej ', 'x': 0.1000,
             'y': 0.9000, 'vred': 5, 'oblika': 'box',
             'faktorr': 2, 'barva': 'rgb(0,0,255)'}]
Tocke[2]=[ {}, {}, {},
           { 'st': 2, 'oznaka': 'Borut', 'x': 0.2000,
             'y': 0.3000, 'vred': 7, 'oblika': 'diamond',
             'faktorr': 0.5, 'barva': 'rgb(0,255,0)'},
           { 'st': 2, 'oznaka': 'Borut', 'x': 0.4000,
             'y': 0.1000, 'vred': 7, 'oblika': 'diamond',
             'faktorr': 0.5, 'barva': 'rgb(0,255,0)'},{}].
```

Na podlagi danih podatkov lahko zgradimo objekt SVG za dinamični prikaz usmerjene povezave od točke 1 do točke 2:

```
<!-- Arc A2F1T2: Andrej->Borut -->
<defs>
<marker id="AA2No1" refX="15" refY="5"
       markerUnits="userSpaceOnUse" fill="rgb(000,000,000)"
       markerWidth="100" markerHeight="100" orient="auto">
  <path d="M 0 0 L 12 5 L 0 10 z"/>
</marker>
</defs>
<g marker-end="url(#AA2No1)" opacity="0">
  <line x1="480" y1="420" x2="160" y2="180"
       style="stroke:rgb(000,000,000);stroke-width:1.5" >
    <animate id="AnimA2No1S2x1" attributeName="x2"
             from="160" to="160" begin="AnimOKS2.begin"
             dur="1.0s" fill="freeze"/>
    <animate id="AnimA2No1S2x1" attributeName="y2"
             from="180" to="180" begin="AnimOKS2.begin"
             dur="1.0s" fill="freeze"/>
    <animate id="AnimA2No1S3x1" attributeName="x2"
             from="160" to="320" begin="AnimOKS3.begin"
             dur="1.0s" fill="freeze"/>
    <animate id="AnimA2No1S3x1" attributeName="y2"
             from="180" to="60" begin="AnimOKS3.begin"
             dur="1.0s" fill="freeze"/>
  </line>
  <animate id="AnimA2No1S2Opa" attributeType="CSS"
           attributeName="opacity" from="0" to="1"
           begin="AnimOKS2.begin" dur="1.0s" fill="freeze"/>
  <animate id="AnimA2No1S4Opa" attributeType="CSS"
           attributeName="opacity" from="1" to="0"
           begin="AnimOKS4.begin" dur="1.0s" fill="freeze"/>
</g>.
```

Oglejmo si še objekt SVG za vrstico z gumbom, s katerim lahko uporabnik dinamični prikaz začasno prekine ali ga požene naprej.

```
<!-- Animation button Start Animation/Stop Animation -->
<g id="button" transform="translate(0 516)" >
   <rect x="2" y="0" width="779" height="30"
         style="fill:#FFEF95; fill-opacity:0.7;" />
   <g id="animateButton" style="font-size:12;"
      onclick="toggleAnimation()"
      onmousedown="toggleButton('animate','down')"
      onmouseup="toggleButton('animate','up')"
      onmouseout="defaultButtonStyle('animate')">
      <rect id="animateDark" x="33" y="5" width="100"
            height="23" style="fill:darkgray;"/>
      <rect id="animateWhite" x="30" y="2" width="100"
            height="23" style="fill:white;"/>
      <rect x="33" y="5" width="97" height="20"
            style="fill:lightgray;"/>
      <text id="toggleAnimText"
            x="42" y="20">Start Animation
      </text>
      <rect x="33" y="5" width="97" height="20"
            style="fill-opacity:0;"/>
   </g>
   <g style="font-size:12;">
      <text x="140" y="20">Current Slide:</text>
      <text id="textCurrentSlide" x="220" y="20"
            text-anchor="begin">1</text>
```

```
    </g>
</g>.
```

Ta objekt poleg osnovnih gradnikov SVG vsebuje tudi prožilnike na naslednje dogodke:

1. Uporabnik z miško pritisne na gumb in miške še ne sprosti: ob tem dogodku se izvede klic procedure, napisane v jeziku Javascript, z imenom toggleButton in parametroma 'animate' in 'down'.
2. Uporabnik sprosti miško s kazalcem na gumbu: ob tem dogodku se izvede klic procedure z imenom toggleButton in parametroma 'animate' in 'up'.
3. Uporabnik sprosti miško s kazalcem izven gumba: ob tem dogodku se izvede klic procedure z imenom defaultButtonStyle in parametrom 'animate'.
4. Kombinacija dogodkov 1 in 2, torej uporabnik z miško pritisne na gumb in nato miško sprosti s kazalcem na gumbu: ob tem dogodku se izvede klic procedure z imenom toggleAnimation.

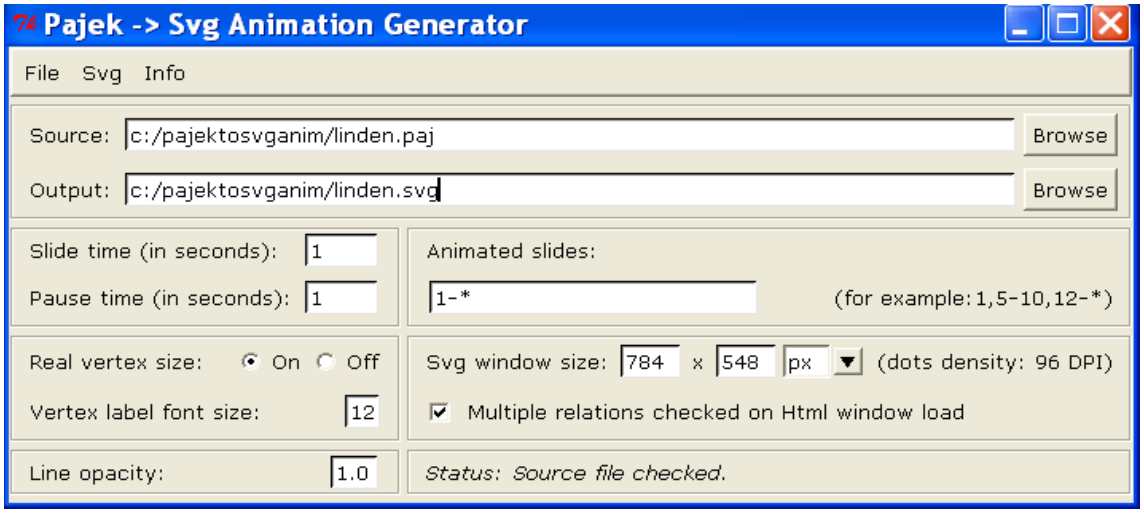
6.3 Predstavitev slikovnega uporabniškega vmesnika

Ob zagonu programa PajekToSvgAnim se odpre slikovni uporabniški vmesnik ali glavno okno programa, preko katerega uporabnik izbere vhodno Pajkovo datoteko in izhodno datoteko .SVG ter požene izdelavo objektov SVG (Slika 6.2). Na voljo ima tudi nekaj dodatnih izbir, preko katerih lahko vpliva na dinamični prikaz omrežja. Te izbire so podrobno opisane v naslednjem poglavju.
Menijska vrstica v glavnem oknu je povezana z naslednjimi ukazi:

- File → Source: izbira vhodne Pajkove datoteke.
- File → Output: izbira izhodne datoteke .SVG.
- File → Exit: izhod iz programa.
- Svg → Generate: izdelava datotek .SVG, .SVG.GZ in .HTML.
- Info → About: informacija o različici programa.

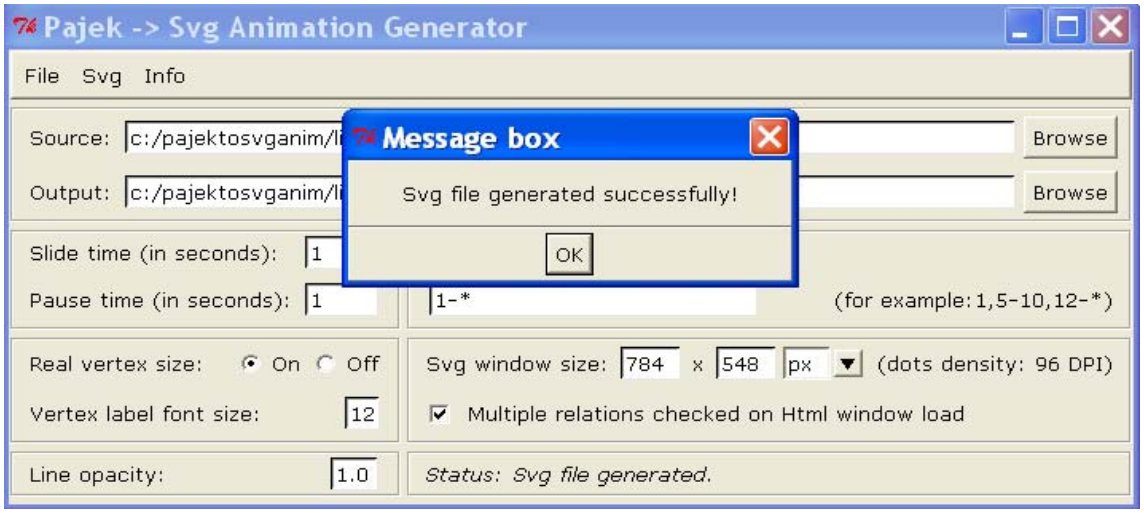
Na Sliki 6.2 je prikazano glavno okno programa PajekToSvgAnim, v katerem je že izbrana vhodna Pajkova datoteka. Ob tem program avtomatično predlaga istoimensko

datoteko .SVG v isti mapi, v spodnji vrstici stanja pa se izpiše sporočilo o slovnični pravilnosti vhodne datoteke. Avtomatično se določi tudi velikost okna SVG glede na to, ali gre za večkratno omrežje ali ne.

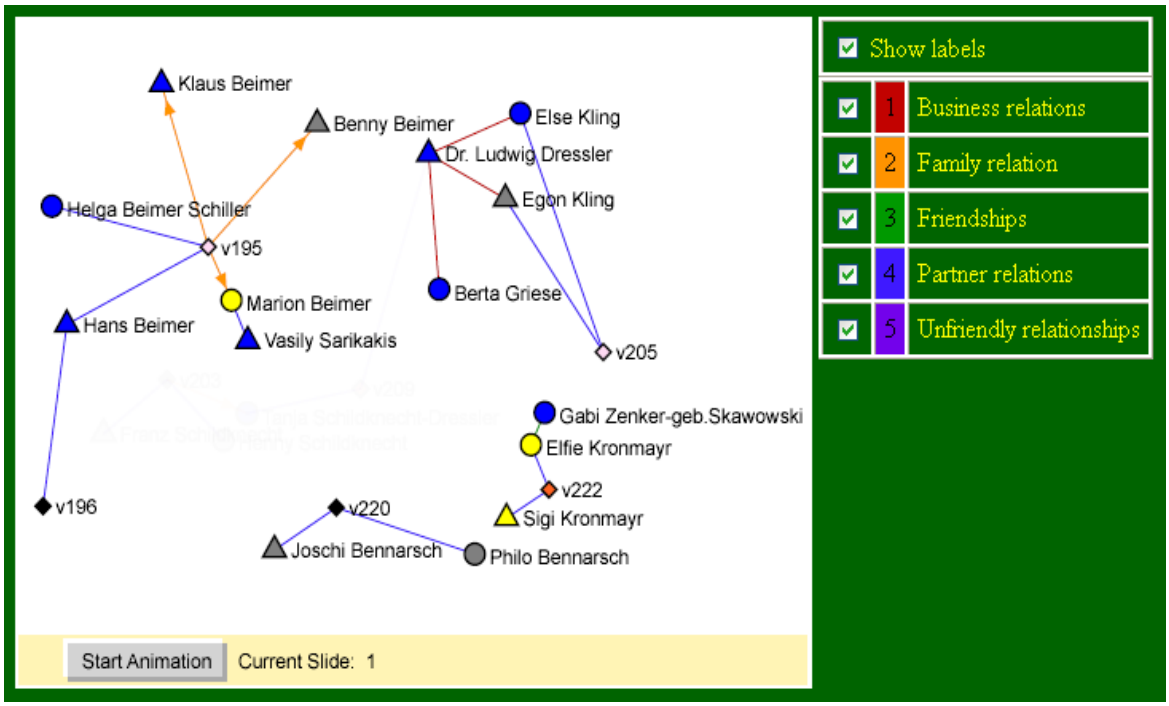


Slika 6.2: Izbira vhodne Pajkove datoteke v programu PajekToSvgAnim.

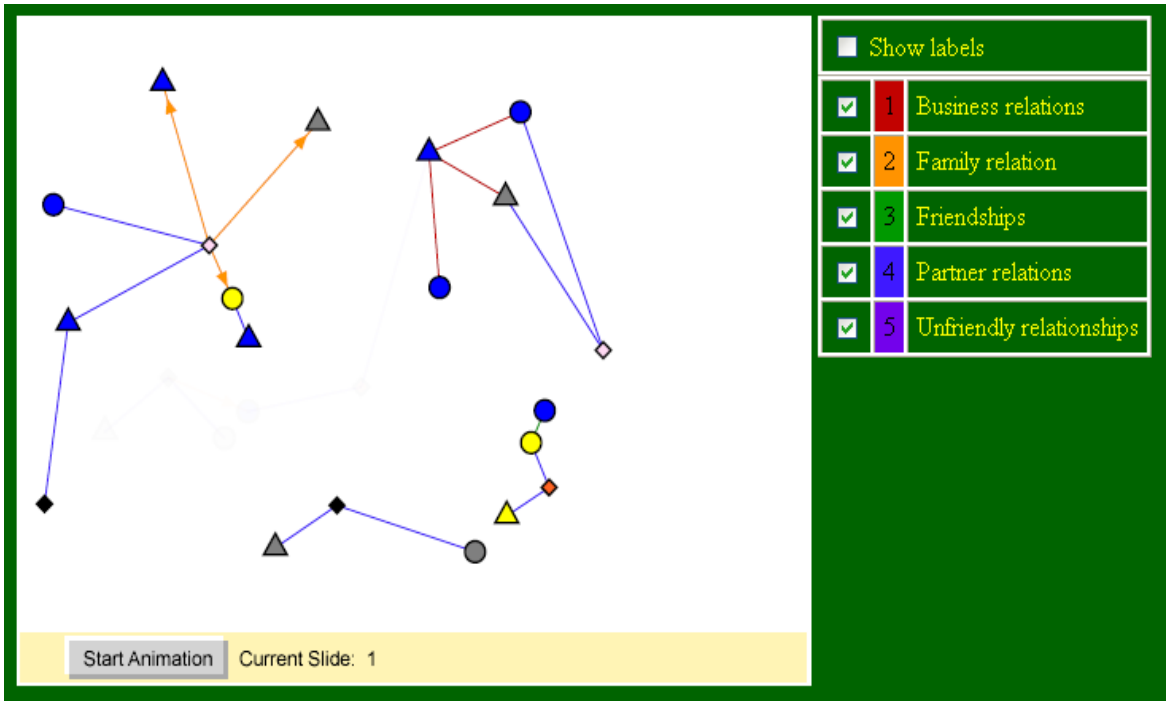
Program najbolj enostavno uporabimo tako, da kar poženemo izdelavo datotek .SVG, .SVG.GZ in .HTML z ukazom Svg → Generate. Če je izdelava datotek uspešna, se prikaže ustrezno sporočilo (Slika 6.3). Dinamični prikaz si ogledamo tako, da odpremo izdelano datoteko .HTML, ki poleg okna SVG vsebuje še izbiro prikaza oznak točk in izbiro prikaza relacij v primeru večkratnega omrežja (Slika 6.4a in Slika 6.4b). Datoteka .HTML je povezana s stisnjeno datoteko .SVG.GZ zaradi hitrejšega prenosa izdelanih objektov v oknu SVG.



Slika 6.3: Sporočilo o uspešni izdelavi objektov SVG v programu PajekToSvgAnim.



Slika 6.4a: Spletni sestavek, v katerem si lahko ogledamo dinamični prikaz izbranega omrežja.



Slika 6.4b: Zgornji spletni sestavek, v katerem ne izberemo prikaza oznak točk.

6.4 Metodologija uporabe programa PajekToSvgAnim za dinamični prikaz časovnih omrežij

Program PajekToSvgAnim ima za različne potrebe uporabnikov vgrajenih kar nekaj izbir, ki se lahko določajo v slikovnem uporabniškem vmesniku. Tako slikovni vmesnik kot tudi vse izhodne datoteke so v angleškem jeziku zaradi mednarodne uporabe programa. Poleg osnovnih izbir vhodne Pajkove datoteke in imena ter poti izhodne datoteke .SVG omogoča še naslednje izbire:

- Dolžino trajanja dinamičnega prikaza prehoda omrežja iz enega časovnega trenutka v drugega (Slide time).
- Dolžino trajanja statičnega prikaza omrežja v časovnem trenutku pred dinamičnim prikazom prehoda v naslednji časovni trenutek (Pause time).
- Številski seznam dinamičnih prehodov med časovnimi trenutki, ki naj se prikažejo (Animated slides).

  Primer: Animated slides: 1, 5-10, 12-*.

  V tem primeru si prikazi omrežja sledijo v naslednjem vrstnem redu:

  1. Statični prikazi od prvega do petega časovnega trenutka. Dinamični prikazi prehodov med petim in desetim časovnim trenutkom (poleg vmesnih statičnih prikazov od šestega do devetega trenutka ter končnega desetega).
  2. Statični prikazi enajstega in dvanajstega časovnega trenutka.
  3. Dinamični prikazi prehodov med dvanajstim in zadnjim časovnim trenutkom (poleg vmesnih statičnih prikazov od trinajstega do predzadnjega trenutka ter čisto zadnjega).
- Možnost izbire med dvema velikostima oblik točk: dejansko velikostjo, ki je podana v vhodni datoteki, in enotno velikostjo za vse točke (Real Vertex Size On/Off).
- Velikost pisave oznak točk (Vertex label font size).
- Velikost okna SVG (Svg window size).
- Možnost izbire označitve relacij v spletnem sestavku z vgrajeno sliko SVG (ob vsaki naložitvi sestavka) v primeru večkratnega omrežja (Multiple relations checked on Html window load).
- Neprosojnost povezav (Line opacity).

V nadaljevanju si oglejmo nekaj primerov, v katerih lahko uporabimo eno ali več od naštetih izbir:

1. Če si želimo bolj podrobno ogledati dinamične prikaze prehodov med časovnimi trenutki, potem povečamo dolžino trajanja dinamičnega prikaza prehoda omrežja iz enega časovnega trenutka v drugega. Privzeta dolžina trajanja je 1 sekunda.

2. Če nas zanimajo statični prikazi omrežja v časovnih trenutkih, potem povečamo dolžino trajanja statičnega prikaza omrežja v časovnih trenutkih. Privzeta dolžina trajanja je 1 sekunda.

3. V primeru, da imamo opravka z omrežjem, ki ima veliko časovnih trenutkov, se večkrat osredotočimo le na nekaj dinamičnih prikazov prehodov med časovnimi trenutki in ne na vse prehode. V tem primeru si lahko pomagamo z možnostjo določitve številskega seznama dinamičnih prikazov prehodov med časovnimi trenutki, ki naj se prikažejo. Privzeti seznam je 1-*, kar pomeni dinamični prikaz vseh prehodov.

4. Možnost izbire med dejansko velikostjo oblik točk, ki je podana v vhodni datoteki, in enotno velikostjo za vse točke je možnost, ki jo poznamo iz programskega paketa Pajek. Uporabimo jo, ko nas dejanske velikosti točk ne zanimajo ali pa ko so točke prevelike za prikaz.

5. Možnost izbire velikosti pisave oznak točk je klasična možnost, ki jo tudi poznamo iz programskega paketa Pajek. Privzeta velikost pisave je 12.

6. Če želimo dinamični prikaz vključiti v svojo spletno stran, potem si lahko pomagamo z možnostjo določitve velikosti okna SVG. Pri tem lahko izbiramo med enotami cm, inch, px. Privzeta velikost okna je velikost celotnega ekrana.

7. Če imamo opravka z večkratnim omrežjem z velikim številom relacij, se lahko zgodi, da bo dinamični prikaz nepregleden. V tem primeru si pomagamo z možnostjo izbire označitve relacij v spletnem sestavku z vgrajeno sliko SVG, s

katero lahko sami določimo, katero relacijo želimo prikazati. Po privzetem se prikažejo vse relacije.

8. Če omrežje vsebuje večkratne povezave, se lahko zgodi, da določene povezave na prikazu ne bodo vidne, ker bodo prekrite z drugo povezavo, ki povezuje isti točki. V programskem paketu Pajek je to rešeno z uporabo ukaza Net → Transform → SortLines → LineValues → Descending, ki uredi povezave po vrednosti padajoče, v programu PajekToSvgAnim pa lahko določimo stopnjo neprosojnosti povezav. Pri tem si lahko pomagamo z naslednjo lestvico:

    1 - povezave so popolnoma neprosojne,
    0.75 - povezave so delno prosojne,
    0.5 - povezave so srednje prosojne,
    0.25 - povezave so zelo prosojne,
    0 - povezave so popolnoma prosojne, torej nevidne.

   Seveda lahko izberemo katerokoli vrednost od 0 do 1, ne samo vrednost iz lestvice. Privzeta vrednost je 1.

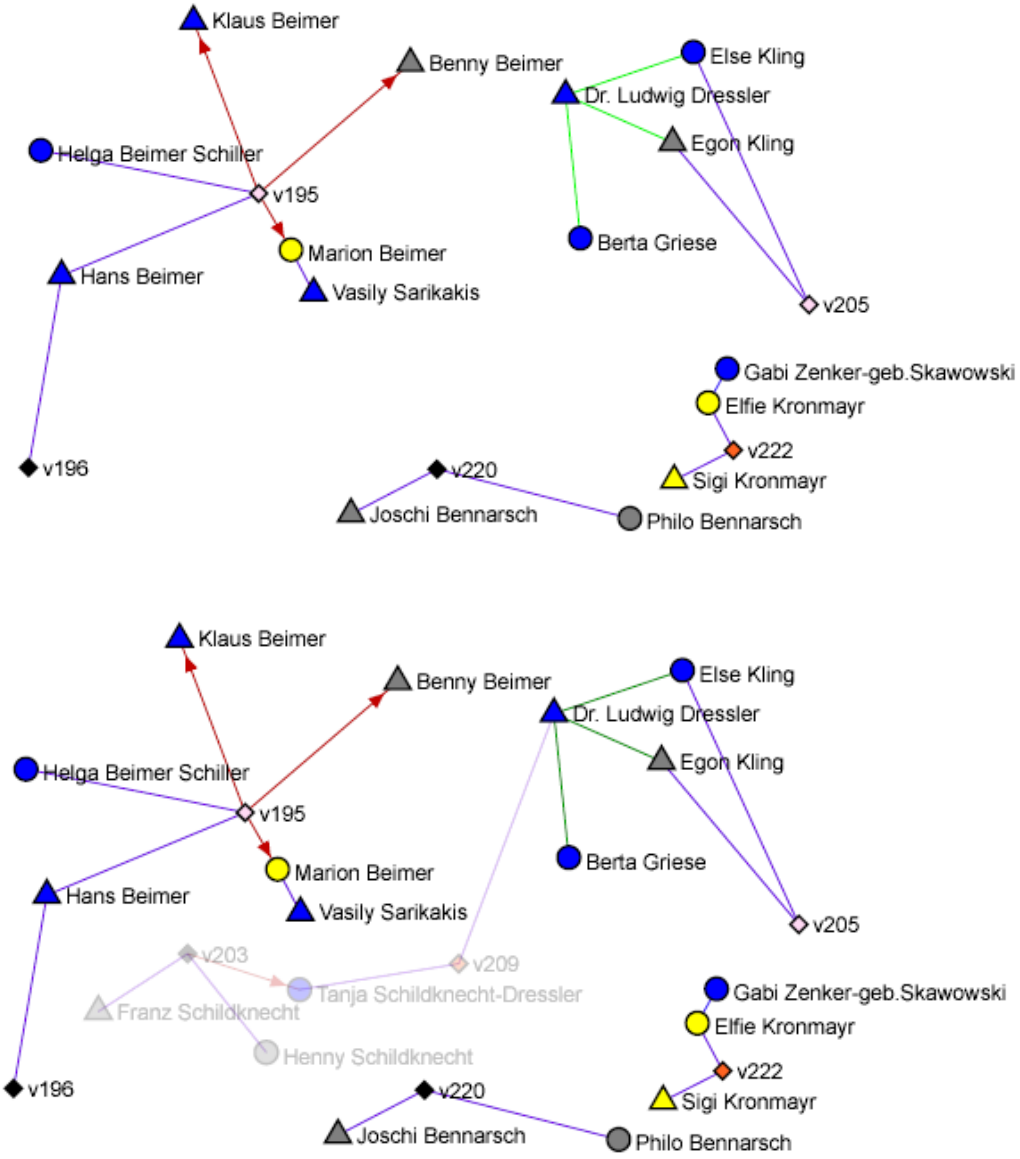
6.5 Predstavitev primerov časovnih omrežij v programu PajekToSvgAnim

6.5.1 Omrežje zgodb nadaljevanke Lindenstrasse

V poglavju 2.3.2 smo predstavili statični prikaz omrežja zgodb nadaljevanke Lindenstrasse s programskim paketom Pajek. V vsakem časovnem trenutku smo za prikaz uporabili metodo Kamada-Kawai, ki je razporedila točke znotraj prikaznega okna z minimiziranjem skupne energije v ravnini. To pa ne zadošča zahtevam za ohranitev mentalne slike omrežja, ki smo jih obravnavali v poglavju 3.2.1. Po teh zahtevah mora vrstni red točk ostati nespremenjen v vodoravni in navpični smeri in za vsako točko mora razdalja do vseh ali do skoraj vseh sosednjih točk ostati (sorazmerno) enaka. Tem zahtevam zadostimo tako, da v paketu Pajek uporabimo metodo Kamada-Kawai na celotnem omrežju in s tem dobimo prikaz, ki določa tudi prikaz omrežja v prvem časovnem trenutku. V naslednjih trenutkih pa točke pustimo na istih mestih ali pa jih malenkostno premaknemo. Pri tem si lahko pomagamo s Pajkovo izbiro v slikovnem oknu Append to Pajek Project file, kjer lahko fiksne položaje ali manjše premike točk

sproti shranjujemo v Pajkovo projektno datoteko in si tako korak za korakom zgradimo dinamično stabilen prikaz omrežja še pred uporabo programa PajekToSvgAnim, ki potem izdela sam dinamični prikaz.

Na sliki 6.5 je narisan prikaz omrežja Lindenstrasse z nepremičnimi točkami, izdelan s programom PajekToSvgAnim, v prvem časovnem trenutku (prva zgodba) in malo zatem, ko se že rahlo opazijo točke in povezave, ki so prisotne šele v drugem časovnem trenutku (druga zgodba). Delno prosojne na novo prihajajoče točke in povezave (npr. točka z oznako v209 in povezava od te točke do točke z oznako Dr. Ludwig Dressler) postajajo vse bolj vidne, čimbolj se bliža drugi časovni trenutek.

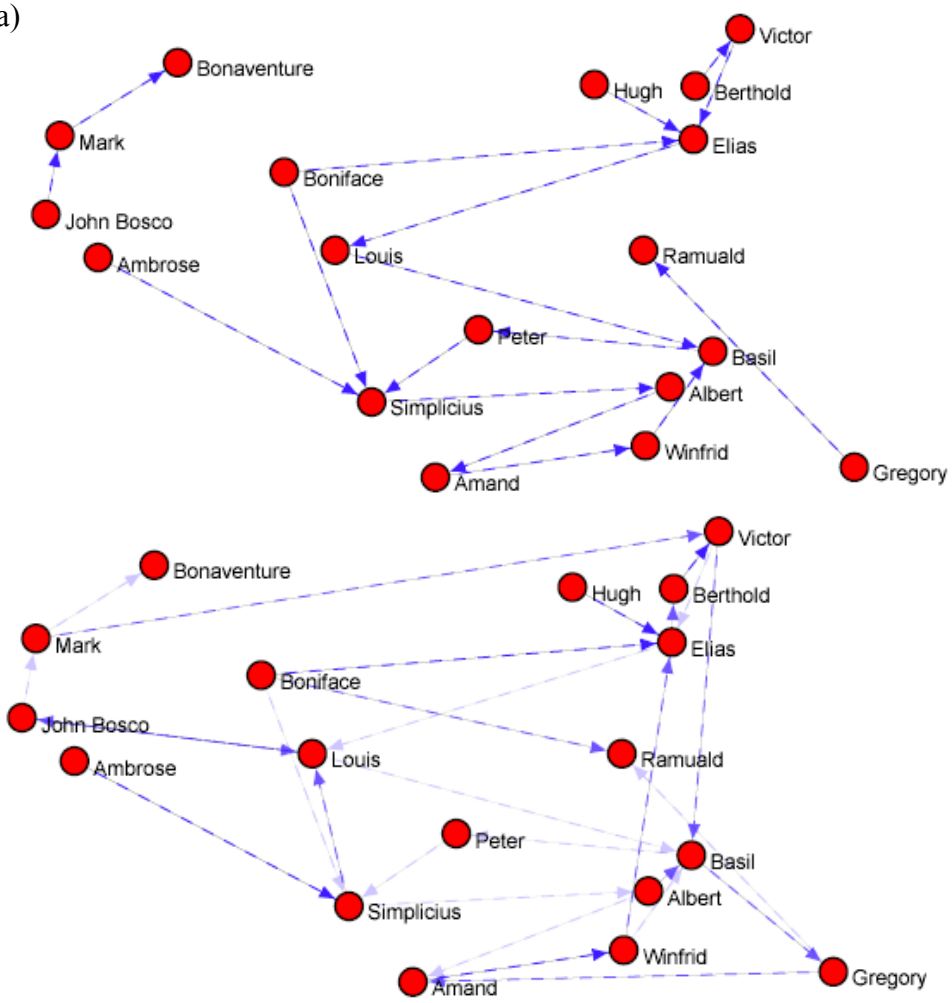


Slika 6.5: Prikaz omrežja Lindenstrasse z nepremičnimi točkami, izdelan s programom PajekToSvgAnim, v prvem časovnem trenutku in malo zatem.

6. 5. 2 Omrežje Sampsonovih menihov

V poglavju 2.3.3 smo predstavili statični prikaz za relacije omrežja Samsonovih menihov s programskim paketom Pajek. Za prikaz je bila uporabljena metoda Kamada-Kawai. Tako kot pri omrežju Lindenstrasse lahko tudi pri tem omrežju korak za korakom pripravimo Pajkovo projektno datoteko, na podlagi katere program PajekToSvgAnim izdela objekte SVG za dinamični prikaz omrežja.

Na sliki 6.6 je narisan prikaz omrežja Sampsonovih menihov z nepremičnimi točkami, izdelan s programom PajekToSvgAnim, v drugem časovnem trenutku (prva ponovitev ankete o medsebojnem razumevanju) in malo pred tretjim časovnim trenutkom (druga ponovitev ankete). Na spodnjem prikazu so vidne tako na novo prihajajoče povezave (npr. povezava od Marka do Victorja) kot tudi povezave, ki v tretjem časovnem trenutku niso več prisotne in zato počasi izginjajo (npr. povezava od Johna Bosco do Marka)

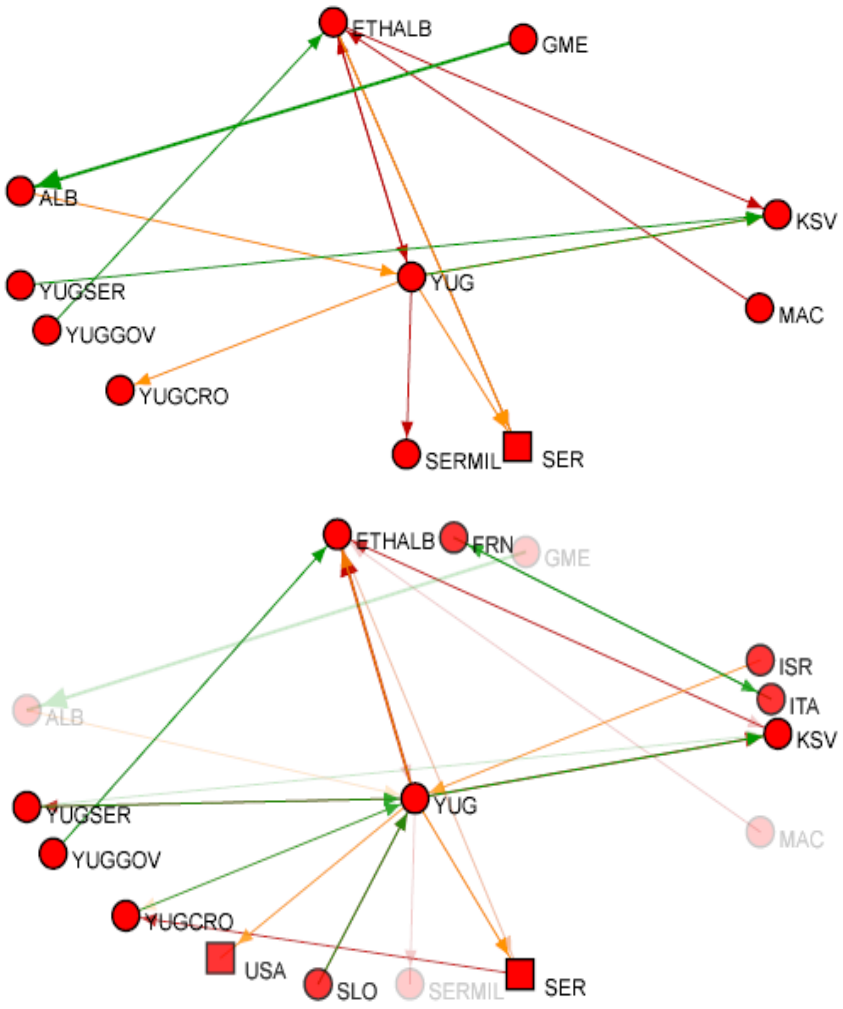


Slika 6.6: Prikaz omrežja Sampsonovih menihov z nepremičnimi točkami, izdelan s programom PajekToSvgAnim, v drugem in malo pred tretjim časovnim trenutkom.

6. 5. 3 Omrežje političnih dogodkov KEDS na Balkanu

V poglavju 2.3.4 smo predstavili statični prikaz omrežja političnih dogodkov KEDS na Balkanu s programskim paketom Pajek. Za prikaz je bila uporabljena krožna metoda, ker večina točk tega omrežja nima veliko sosednjih točk. To metodo uporabimo tudi za dinamični prikaz, ki ga izdela program PajekToSvgAnim.

Na sliki 6.7 je narisan krožni prikaz omrežja političnih dogodkov KEDS na Balkanu, izdelan s programom PajekToSvgAnim, v tretjem časovnem trenutku (junij 1989) in malo pred četrtim časovnim trenutkom (julij 1989). Na spodnjem prikazu so vidne tako na novo prihajajoče točke in povezave (npr. točka z oznako SLO, ki predstavlja Slovenijo, in povezava od Slovenije do bivše Jugoslavije) kot tudi točke in povezave, ki v sedmem časovnem trenutku niso več prisotne in zato počasi izginjajo (npr. točka z oznako ALB, ki predstavlja Albanijo, in povezava od Albanije do bivše Jugoslavije).

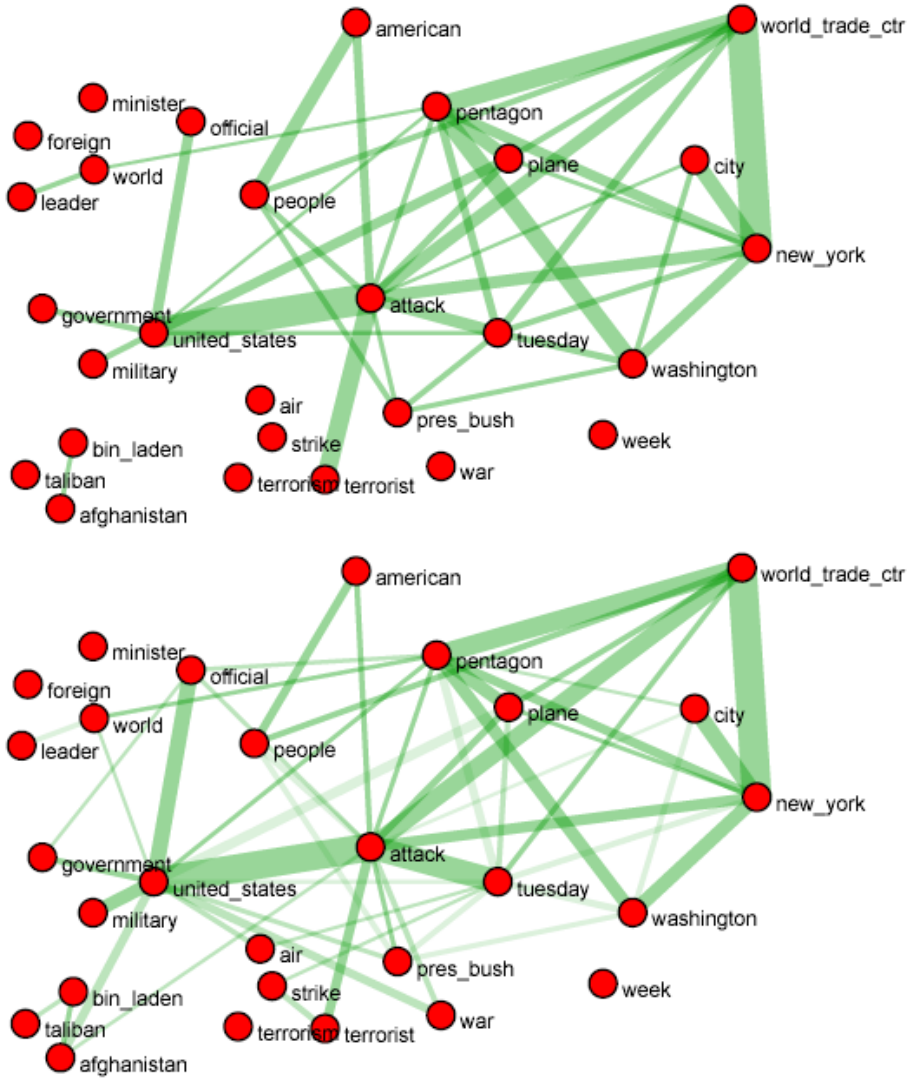


Slika 6.7: Krožni prikaz omrežja KEDS na Balkanu, izdelan s programom PajekToSvgAnim, v tretjem in malo pred četrtim časovnim trenutkom.

6. 5. 4 Omrežje Reutersovih novic o 11. septembru 2001

V poglavju 2.3.5 smo predstavili statični prikaz omrežja Reutersovih novic o 11. septembru 2001 s programskim paketom Pajek. Tako kot pri omrežju Lindenstrasse korak za korakom pripravimo Pajkovo projektno datoteko, na podlagi katere program PajekToSvgAnim izdela objekte SVG za dinamični prikaz omrežja.

Na sliki 6.8 je narisan prikaz omrežja Reutersovih novic o 11. septembru 2001 z nepremičnimi točkami, izdelan s programom PajekToSvgAnim, v prvem časovnem trenutku (11.9.2001) in malo pred drugim časovnim trenutkom (12.9.2001). Na spodnjem prikazu so vidne tako na novo prihajajoče povezave (npr. povezava med napadom in Afganistanom), kot tudi povezave, ki v drugem časovnem trenutku niso več prisotne in zato počasi izginjajo (npr. povezava med predsednikom Bushem in ljudstvom).



Slika 6.8: Prikaz omrežja Reutersovih novic o 11. septembru 2001 z nepremičnimi točkami, izdelan s programom PajekToSvgAnim, v prvem časovnem trenutku in malo pred drugim časovnim trenutkom.

7 Zaključek

V magistrskem delu smo želeli opisati in izdelati dinamični prikaz za časovna omrežja. Za osnovo smo vzeli različne oblike statičnih prikazov v programskem paketu Pajek.
Na začetku smo iskali slikovni jezik, ki poleg slikovne predstavitve točk in povezav z geometrijskimi liki in črtami omogoča tudi naslednje možnosti:

- zvezno spreminjanje koordinat točk;
- izginjanje točk in povezav in pojavljanje novih točk in povezav glede na to, ali je določena točka oziroma povezava prisotna v določenem časovnem trenutku;
- spreminjanje velikosti točk in debelin povezav brez izgube ločljivosti;
- prikaz večkratnih omrežij na istih točkah;
- pregled datotek s spletnim brskalnikom.

Izbrali smo označevalni jezik SVG, ki je izpolnjeval vse naštete zahteve.
Naslednji korak je bila izbira programskega jezika za izdelavo slikovnega uporabniškega vmesnika in za izdelavo končnih slikovnih datotek .SVG. Izbran je bil programski jezik Python predvsem zaradi naslednjih zmogljivosti:

- vgrajenih ima veliko funkcij za delo z nizi, seznami in slovarji;
- tako kot programski jezik Java omogoča izdelavo modulov, razredov in izjem;
- uporablja zamikanje za določanje strukture programa;
- omogoča prevod programov v strojno kodo.

Po izbiri slikovnega jezika in programskega jezika smo izdelali program PajekToSvgAnim kot dodatek programskemu paketu Pajek za dinamični prikaz časovnih omrežij. Metode izdelave programa so bile vezane predvsem na značilnosti jezikov SVG in Python, zato je po strukturi precej preprost in ga je mogoče na enostaven način dopolnjevati.
Delovanje programa PajekToSvgAnim je bilo preverjeno na več primerih, med drugimi tudi na nekaterih znanih primerih časovnih omrežij, kot so:

- omrežje zgodb nadaljevanke Lindenstrasse;
- klasično longitudinalno socialno omrežje Sampsonovih menihov;
- omrežje novic 11. september 2001;
- omrežje političnih dogodkov KEDS na Balkanu.

Vsebina zgoščenke

Na priloženi zgoščenki se nahajajo naslednje mape:

- PajekToSvgAnim: vsebuje program PajekToSvgAnim.exe in še nekaj dodatnih podmap in datotek, potrebnih za delovanje programa.
- Linden: vsebuje datoteke Linden.paj, Linden.svg, Linden.svg.gz in Linden.htm za predstavitev omrežja zgodb nadaljevanke Lindenstrasse.
- Sampson: vsebuje datoteke Sampson.paj, Sampson.svg, Sampson.svg.gz in Sampson.htm za predstavitev omrežja Sampsonovih menihov.
- Reuters: vsebuje datoteke Reuters.paj, Reuters.svg, Reuters.svg.gz in Reuters.htm za predstavitev omrežja Reutersovih novic o 11. septembru 2001.
- Balkan: vsebuje datoteke Balkan.paj, Balkan.svg, Balkan.svg.gz in Balkan.htm za predstavitev omrežja političnih dogodkov KEDS na Balkanu.
- Python234_namestitev: vsebuje datoteko Python-2.3.4.exe za namestitev programskega jezika Python in naslednji podmapi:
  - Site_packages: vsebuje module, ki so potrebni za delovanje programa PajekToSvgAnim. Vsebino te mape je potrebno prenesti v mapo C:\Python234\Lib\Site_packages po namestitvi jezika Python (če je nameščen v mapo C:\Python234).
  - Pmw: vsebuje datoteke, ki so potrebne za izdelavo datoteke PajekToSvgAnim.exe. Vsebino te mape je potrebno prenesti v glavno mapo, kjer je nameščen jezik Python (npr. v mapo C:\Python234, če je jezik Python nameščen v mapo C:\Python234).
- Python_koda: vsebuje tri datoteke, ki jih je potrebno prenesti v glavno mapo jezika Python.
  - PajekToSvgAnim.py: programska koda programa PajekToSvgAnim.
  - Setup.py: namestitvena datoteka, potrebna pri izdelavi datoteke PajekToSvgAnim.exe.
  - PajekToSvgAnim.ico: sličica na ikoni programa PajekToSvgAnim.
- Adobe_SVGViewer: vsebuje programski dodatek spletnemu brskalniku za pregledovanje datotek .SVG.

Za prevod datoteke PajekToSvgAnim.py v datoteko PajekToSvgAnim.exe je potrebno izvesti naslednji ukaz v ukaznem oknu:
C:\Python234>python setup.py py2exe (če je jezik Python nameščen v mapo C:\Python234). Pri tem se izdela mapa C:\Python234\Dist s programom PajekToSvgAnim.exe.

Literatura

1. Adobe SVG Viewer. Adobe Systems Incorporated.
   [URL: http://www.adobe.com/svg/viewer/install/], 29.11.2004.
2. Axelsson J., Epperson B., Ishikawa M., McCarron S., Navarro A., Pemberton S.: XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). W3C Recommendation.
   [URL: http://www.w3.org/TR/2002/REC-xhtml1-20020801/], 1.8.2002.
3. Batagelj V.: Pajek Datasets.
   [URL: http://vlado.fmf.uni-lj.si/pub/networks/data/], 16.1.2005.
4. Batagelj V., Mrvar A.: Risanje grafov in omrežij. Zbornik posvetovanja Dnevi slovenske informatike 1995. Ljubljana, Slovensko društvo informatika, 1995.
5. Batagelj V., Mrvar A.: Density based approaches to Reuters terror news network analysis.
   [URL: http://www-2.cs.cmu.edu/~dunja/LinkKDD2003/papers/Batagelj.pdf], 27.8.2003.
6. Batagelj V., Mrvar A.: PAJEK 1.02, Program for Large Network Analysis.
   [URL: http://vlado.fmf.uni-lj.si/pub/networks/pajek/], 16.1.2005.
7. Bearman P., Everett K.: The Structure of Social Protest. Social Networks 15, 1993.
8. Bender-deMoll S., McFarland D. A.: SoNIA (Social Network Image Animator).
   [URL: http://www.stanford.edu/~skyebend/], 18.9.2004.
9. Brandes U., Raab J., Wagner D.: Exploratory Network Visualization: Simultaneous Display of Actor Status and Connections. Journal of Social Structure 2(4), 2001.
10. Bray T., Paoli J., Sperberg-McQueen C. M., Maler E., Yergeau F.: Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation.
    [URL: http://www.w3.org/TR/2004/REC-xml-20040204/], 4.2.2004.
11. Coleman J. S.: The Adolescent Society. New York: Free Press, 1961.
12. Corman S., Dooley K.: CRA Analyses of News Stories on the Terrorist Attack. Arizona State University: LOCKS.
    [URL: http://locks.asu.edu/terror/], 27.2.2002.

13. De Nooy W., Mrvar A., Batagelj V.: Exploratory Social Network Analysis with Pajek. CUP, January 2005.
14. DeRose S., Maler E., Orchard D.: XML Linking Language (XLink) Version 1.0. W3C Recommendation.
[URL: http://www.w3.org/TR/2001/REC-xlink-20010627/], 27.6.2001.
15. Di Battista G., Eades P., Tamassia R., Tollis I. G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, 1999.
16. Diehl S., Görg C., Kerren A.: Foresighted Graphlayout. Technical Report A/02/2000, Informatik, University of Saarland, December 2000.
[URL: http://rw4.cs.uni-sb.de/~diehl/ganimation/], 13.7.2005.
17. Diehl S., Görg C., Kerren A.: Preserving the Mental Map using Foresighted Layout. Proceedings of Joint Eurographics – IEEE TCVG Symposium on Visualization VisSym'01. Springer Verlag, 2001.
18. Diehl S., Görg C., Pohl M., Birke P., Zimmer S.: GraphAnimation Web Service. Informatik, University of Saarland, 2005.
[URL: http://www2.informatik.ku-eichstaett.de:8080/GA2SVG/], 13.1.2005.
19. Doreian P., Batagelj V., Ferligoj A.: Generalized Blockmodeling. CUP, January 2005.
20. Doreian P., Kapuscinski R., Krackhardt D., Szczypula J.: A Brief History of Balance Through Time. Journal of Mathematical Sociology 21(1-2), 1996.
21. Ferraiolo J., Jun F., Jackson D.: Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation.
[URL: http://www.w3.org/TR/2003/REC-SVG11-20030114/], 14.1.2003.
22. Freeman L. C.: Visualizing Social Groups. Proceedings of the Section on Statistical Graphics American Statistical Association, 2000.
[URL: http://moreno.ss.uci.edu/groups.pdf], 18.1.2005.
23. Freeman L. C., Webster C. M., Kirke D. M.: Exploring Social Structure Using Dynamic Three-Dimensional Color Images. Social Networks 20, 1998.
24. Garg A., Tamassia R.: Advances in Graph Drawing. Proceedings CIAC' 94, LNCS 778, 1994.
25. Gloor P. A., Zhao Y.: TeCFlow - A Temporal Communication Flow Visualizer for Social Network Analysis. Collaborative Knowledge Networks.
[URL: http://www.ickn.org/html/ckn_tools.htm], 25.10.2004.

26. Horwat W., Ginda R.: JavaScript Language Resources.
[URL: http://www.mozilla.org/js/language/], 24.1.2003.
27. Ion P., Miner R.: Mathematical Markup Language (MathML™) 1.01 Specification. W3C Recommendation.
[URL: http://www.w3.org/1999/07/REC-MathML-19990707/], 7.7.1999.
28. Leenders R. Th. A. J.: Longitudinal Behavior of Network Structure and Actor Attributes: Modeling Interdependence of Contagion and Selection. Evolution of Social Networks. New York: Gordon and Breach, 1996.
29. Lilley C.: Graphics Activity Statement.
[URL: http://www.w3.org/Graphics/Activity], 25.10.2004.
30. Misue K., Eades P., Lai W., Sugiyama K.: Layout adjustment and the mental map. Journal of Visual Languages and Computing 6(2), 1995.
31. Moody J., McFarland D., Bender-deMoll S.: Dynamic Network Visualization: Methods for Meaning with Longitudinal Network Movies, 2003.
[URL: http://www.sociology.ohio-state.edu/jwm/NetMovies/], 18.1.2005.
32. Mrvar A.: Izbrani algoritmi analize družboslovnih omrežij. Magistrska naloga. Ljubljana, Fakulteta za elektrotehniko in računalništvo, 1995.
33. Mrvar A.: Analiza in prikaz velikih omrežij. Doktorska disertacija. Ljubljana, 1999.
34. Mrvar A.: Analiza velikih socialnih omrežij. Družboslovne razprave 17(36), 2001.
[URL: http://dk.fdv.uni-lj.si/dr/dr36mrvar.pdf], 18.1.2005.
35. Mrvar A., Batagelj V.: Dinamični prikazi omrežij. Zbornik posvetovanja DSI 2000. Ljubljana: Slovensko društvo Informatika, 2000.
36. Mutzel P.: GD99 contest: Lindenstrasse network, 1999.
[URL: http://kam.mff.cuni.cz/conferences/GD99/contest/graphs/A.html], 18.1.2005.
37. Nakao K., Romney A. K.: Longitudinal Approach to Subgroup Formation: A Re-Analysis of Newcomb's Fraternity Data. Social Networks 15, 1993.
38. Newcomb T. M.: The Acquaintance Process. New York: Holt, Rinehart and Winston, 1961.
39. Quint A.: SVG: Where Are We Now?
[URL: http://xml.com/pub/a/2001/11/21/svgtools.html], 18.1.2005.

40. Schrodt P. A., Davis S. G., Weddle J. L.: KEDS (The Kansas Event Data System). University of Kansas.
[URL: http://www.ku.edu/~keds/index.html], 3.6.2004.
41. Scott J.: Social Network Analysis. Sage Publications Ltd, 1991.
42. Snijders T. A. B.: Models for Longitudinal Social Network Data. Manuscript. University of Groningen, 1998.
43. Suitor J. J., Wellman B., Morgan D. L.: It's About Time: How, Why and When Networks Change. Social Networks 19, 1997.
44. Traversa E.: Scalable Vector Graphics: The Art is in the Code.
[URL: http://www.webreference.com/authoring/languages/svg/intro/index.html], 18.1.2005.
45. Trippe B.: The Next Wave for Graphics.
[URL:http://www.transformmag.com/db_area/archs/2002/04/tfm0204xm.shtml], april 2002.
46. Van Rossum G.: Python.
[URL: http://www.python.org/2.4/], 30.11.2004.
47. Weesie J., Flap H.: Social Networks Through Time. Utrecht, Netherlannds: ISOR, 1990.
48. Whyte W. F.: Street Corner Society. University of Chicago Press, 1943.
49. Zeggelink E. P. H., Stokman F. N.,Van De Bunt G. G.: The Emergence of Groups in the Evolution of Friendship Networks. Journal of Mathematical Sociology 21, 1996.