Hierarhični prikazi velikih omrežij

Jernej Bodlaj

doktorska disertacija

predana

Fakulteti za računalništvo in informatiko

kot del izpolnjevanja pogojev za pridobitev naziva

doktor znanosti

s področja

računalništva in informatike

Ljubljana, 2015

HRUSKA.SI
Naložba v vašo prihodnost
Operacijo delno financira Evropska unija
Evropski socialni sklad

Hierarhični prikazi velikih omrežij

Jernej Bodlaj

doktorska disertacija
predana
Fakulteti za računalništvo in informatiko
kot del izpolnjevanja pogojev za pridobitev naziva
doktor znanosti
s področja
računalništva in informatike

Ljubljana, 2015

IZJAVA

Izjavljam, da sem avtor dela in da slednje ne vsebuje gradiva, ki bi ga kdorkoli predhodno že objavil ali oddal v obravnavo za pridobitev naziva na univerzi ali na drugem visokošolskem zavodu, razen v primerih, kjer so navedeni viri.

— Jernej Bodlaj —
februar 2015

Oddajo so odobrili

prof. dr. Vladimir Batagelj
redni profesor za diskretno in računalniško matematiko
MENTOR IN ČLAN OCENJEVALNE KOMISIJE
Fakulteta za matematiko in fiziko

prof. dr. Franc Solina
redni profesor za računalništvo in informatiko
SOMENTOR IN ČLAN OCENJEVALNE KOMISIJE

doc. dr. Boštjan Slivnik
docent za področje računalništva in informatike
PREDSEDNIK OCENJEVALNE KOMISIJE

prof. dr. Saša Divjak
redni profesor za računalništvo in informatiko
ČLAN OCENJEVALNE KOMISIJE

prof. dr. Tomaž Pisanski
redni profesor za diskretno in računalniško matematiko
ZUNANJI ČLAN OCENJEVALNE KOMISIJE
Fakulteta za matematiko in fiziko

PREDHODNA OBJAVA

Izjavljam, da so bili rezultati obravnavane raziskave predhodno objavljeni/sprejeti za objavo v recenzirani reviji ali javno predstavljeni v naslednjih primerih:

[1] J. Bodlaj, V. Batagelj, A. Orbanić. *OpenGL®* Graphical User Interface for net.Plexor – interactive large network analysis library. *Proceedings of the 20th International Electrotechnical and Computer Science Conference*, 139–142, Portorož (Slovenija), 2011.

[2] J. Bodlaj, M. Cerinšek, V. Batagelj. Visualization of traffic. *Third conference on the Analysis of Mobile Phone Datasets, NetMob 2013*, 44:480–495, MIT Cambridge (Massachusetts, ZDA), 2013.

[3] M. Cerinšek, J. Bodlaj, V. Batagelj. Symbolic clustering of users and antennae. *Third conference on the Analysis of Mobile Phone Datasets, NetMob 2013*, 20:211–226, MIT Cambridge (Massachusetts, ZDA), 2013.

[4] J. Bodlaj, V. Batagelj. Network Analysis of Publications on Topological Indices from the Web of Science. *Molecular Informatics*, 33(8):514–535, 2014. doi: 10.1002/minf.201400014

[5] J. Bodlaj. Network data file formats. R. Alhajj, J. Rokne (urednika), *Encyclopedia of Social Network Analysis and Mining*, 2:1076–1091, Springer, 2014.

V recenzijo je bil poslan še naslednji članek:

[1] J. Bodlaj, V. Batagelj. Hierarchical link clustering algorithm in networks. *Physical Review E*, 2014.

Potrjujem, da sem pridobil pisna dovoljenja vseh lastnikov avtorskih pravic, ki mi dovoljujejo vključitev zgoraj navedenega materiala v pričujočo disertacijo. Potrjujem, da zgoraj navedeni material opisuje rezultate raziskav, izvedenih v času mojega podiplomskega študija na Univerzi v Ljubljani.

Izjavljam, da so bili rezultati obravnavane raziskave predhodno objavljeni oz. javno predstavljeni še v naslednjih primerih:

Ostale javne objave in prezentacije:

[1] J. Bodlaj, V. Batagelj. Hierarhično razvrščanje v omrežjih, posplošitev na povezave in algoritem. *Sredin seminar za računalniško matematiko*, 2011.

[2] J. Bodlaj, V. Batagelj, A. Orbanić. *OpenGL®* Graphical User Interface for net.Plexor – interactive large network analysis library. *Abstracts of the 7th Slovenian international conference on graph theory*, stran 148, 2011.

[3] J. Bodlaj, V. Batagelj, A. Orbanić. net.Plexor, interaktivna knjižnica za analizo in vizualizacijo velikih omrežij. *Seminar za diskretno matematiko, Plemljev seminar*, 2011.

[4] J. Bodlaj, V. Batagelj. Network Analysis of Publications on Topological Indices from Web of Science. *Abstracts of the Workshop COMPUTERS IN SCIENTIFIC DISCOVERY 6*, Portorož (Slovenija), stran 32, 2012.

[5] J. Bodlaj, M.Cerinšek, V. Batagelj. Analiza podatkov za izziv D4D (Data For Development). *Sredin seminar za računalniško matematiko*, 2013.

[6] J. Bodlaj. Datotečni formati za shranjevanje omrežij. *Sredin seminar za računalniško matematiko*, 2013.

[7] J. Bodlaj, V. Batagelj. Hierarchical link clustering of networks. *1st European Conference on Social Networks, Abstract book*, Barcelona (Španija), stran 29, 2014.

[8] B. Horvat, J. Bodlaj. net.Plexor - zbirka orodij za vizualizacijo in analizo velikih omrežij. *Posvetovanje o informatiki v energetiki*, Portorož (Slovenija), 2014.

Potrjujem, da zgoraj navedeni material opisuje rezultate raziskav, izvedenih v času mojega podiplomskega študija na Univerzi v Ljubljani.

Nekaterim omrežjem resnično zaupam...

Avtor disertacije, *Kamniški vrh*, 2014.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*
Jernej Bodlaj
Hierarhični prikazi velikih omrežij

POVZETEK

V disertaciji predstavljamo nekatere izboljšave postopkov v analizi in vizualizaciji omrežij. Eden od temeljnih pristopov v analizi omrežij je razvrščanje. V množici pristopov h klasičnemu razvrščanju smo zaznali dve možnosti za izboljšavo. To sta vpeljava relacijskih omejitev v algoritme razvrščanja in uvedba mer različnosti, ki poleg strukture omrežja upoštevajo tudi opise oz. lastnosti elementov vhodnih omrežij. Na področju orodij za analizo in prikazovanje omrežij smo opazili, da le redka strukturirajo potek analize. Večinoma analiza poteka v diskretnih korakih, kar je iz časovnih, organizacijskih in drugih razlogov lahko neugodno. Podoben občutek imamo glede prikazovanja v analizi omrežij. V prikazovanje lahko vnesemo več prvin interaktivnosti (sodejnosti) in ga tako približamo uporabniku. Da bi bila uporabniška izkušnja še boljša in v realnem času, moramo vpeljati primerno abstrakcijo. V obliki prikazov s povzetki prometa v mobilnih omrežjih Slonokoščene obale smo pripravili nekaj primerov abstrakcije. Tekom raziskav oz. razvoja novega algoritma za razvrščanje smo naleteli na naslednje vprašanje: katera omrežja uporabiti za testiranje? Tekom iskanja odgovora smo ugotovili, da na področju topoloških indeksov analiza bibliografskih omrežij še ni bila narejena. Vključili smo še pregled omrežnih datotečnih formatov, v okviru katerih podajamo napotke za zbiranje omrežnih podatkov in njihovo shranjevanje ter izpostavimo nekaj orodij za analizo omrežij. Zgodbo disertacije smo s tem zaokrožili v smiselno celoto.

Hierarhično razvrščanje je postopek, s katerim iščemo notranje močno povezane skupine (gruče ali združbe) vozlišč v omrežju. Idejo lahko pretvorimo v razvrščanje povezav v omrežju in v njem iščemo skupine povezav. Pri tem se množice vozlišč, ki pripadajo skupinam povezav, prekrivajo. Najprej predstavimo osnovni hierarhični združevalni algoritem za razvrščanje vozlišč, ki smo ga pretvorili v nov algoritem za razvrščanje povezav v omrežjih. Uvedli smo tudi nove mere različnosti, ki za razliko

od obstoječih, ki upoštevajo le strukturne lastnosti omrežja, upoštevajo tudi lastnosti (opise, atribute) vozlišč in/ali povezav omrežja. Za razliko od nekaterih obstoječih algoritmov smo naš algoritem zasnovali tako, da je na povezavni graf vhodnega omrežja vezan le implicitno, s tem pa se zmanjša njegova zahtevnost. Prostorsko in časovno zahtevnost algoritma smo statistično raziskali na umetnih in realnih omrežjih z znanimi lastnostmi. Za primere enostavnih omrežij smo zahtevnost preučili tudi analitično. Pravilnost delovanja algoritma oz. smisel njegovih rezultatov smo preverili na realnih omrežjih, in sicer s primerjavo rezultatov algoritma z vnaprej znanimi razvrstitvami.

Nabor omrežij za testiranje je med drugimi vključeval bibliografska omrežja iz področja topoloških indeksov. Slednja smo konstruirali iz bibliografskih podatkov o člankih, ki smo jih na temo topoloških indeksov in na njej sorodnih temah pridobili iz spletne storitve *Web of Science*. Osnovna ideja, torej pridobivanje omrežnih podatkov za vrednotenje našega algoritma, je prerasla v izčrpno scientometrično analizo področja topoloških indeksov. Najprej smo naredili splošno analizo zbranih podatkov in se nato poglobili v nekatere bolj podrobne analize avtorjev, člankov, revij, podpodročij topoloških indeksov in ključnih besed. Nenazadnje smo analizirali tudi odnose kombinacij avtorjev in revij, revij in člankov idr. Uporabili smo scientometrične postopke, kot sta najmočnejša in glavna pot citiranja, določanje tem na poteh citiranja glede na ključne besede del, analizo soavtorstev, iskanje skupin sodelujočih avtorjev z otoki v omrežju sodelovanj ter z dvodelnimi sredicami v omrežju avtorjev in del. Raziskali smo tudi naravo citiranja avtorjev, izpostavili pomembne revije in citiranja med deli v njih, poiskali revije, ki jim določeni avtorji dajejo prednost, in določili hierarhijo podobnosti sodelujočih avtorjev glede na ključne besede, ki jih uporabljajo. Na izbrani reviji smo izvedli časovno analizo.

Že v času razvoja algoritma in izvajanja navedenih analiz smo začeli razvijati novo celovito orodje za analizo omrežij – interaktivno grafično okolje *net.Plexor*. To orodje temelji na knjižnici – ogrodju za delo z velikimi omrežji. V orodje smo vgradili zgornji algoritem, razne podporne metode in druge algoritme. Orodje smo zasnovali tako, da je vanj možno dodajati nove funkcionalnosti. V disertaciji izpostavimo le dele, za katerimi stoji avtor sam. To so predvsem logika, ki skrbi za prikazovanje omrežnih podatkov, logika za organizacijo izvajanja delovnega procesa (sheme analize) in logika uporabniškega vmesnika v splošnem, ki predstavlja osnovo za interaktivnost. Gre za nov pristop k strukturi sistema organizacije delovnega procesa s shemami analize ali s t.i. vizualnim programiranjem, ki ga pri drugih orodjih za analizo omrežij nismo zasle-

dili. Uvedli smo tudi koncept spreminjanja parametrov algoritmov v realnem času, ki ga prav tako nismo srečali pri drugih orodjih. V orodje smo vključili tri vrste prikazov omrežij: s prvinami zunanjih pogledov (opazovalec ni del prikaza), s prvinami notranjih pogledov (opazovalec je dejavni del prikaza in po njem lahko potuje) in prikaze hierarhično urejenih omrežij. Na področju analize omrežij prikazi omrežij z notranjim pogledom predstavljajo novost. Gre pravzaprav za prvine navidezne resničnosti.

Za primer abstrakcije smo razvili nekaj prikazov prometa s povzetki v mobilnih telekomunikacijskih omrežjih. Iz podatkov polletnega obdobja mobilnega prometa med več kot tisoč baznimi postajami s Slonokoščene obale smo pripravili združena omrežja prometa. Dobili smo zelo gosta omrežja in v želji, da jih prikažemo, razvili primerne postopke. Promet smo prikazali na več različnih ravninskih, geografsko orientiranih prikazih. Sprva smo omrežja prikazali z običajnimi zgostitvenimi prikazi z daljicami, nato še s posebnimi razpršenimi vzorci, ki prikaze prometa naredijo bolj naravne. Razvili smo še t.i. zvezdne prikaze. Promet smo prikazali tudi z nekaterimi skeletnimi prikazi. Pripravili smo dve animaciji razvoja prometa skozi čas in zgostitvene prikaze nekaterih razvrstitev baznih postaj.

Opisana področja dopolnimo še z opisi začetnih faz v analizi omrežij. Osredotočili smo se na omrežne datotečne formate oz. na shranjevanje omrežnih podatkov na pomnilne medije v obliki datotek. V manjšem obsegu se osredotočimo tudi na postopke, kako do omrežnih podatkov priti, kako jih zbrati in kako jih pretvarjati. Lastnosti omrežnih datotečnih formatov se razlikujejo glede na potrebe, kot so: berljivost, prenosljivost, splošnost, prostorske zahteve, hitrost kodiranja in dekodiranja, branja, pisanja in druge. Omrežne datotečne formate smo razvrstili glede na njihove osnovne lastnosti in izpostavili pomembne vidike, na katere moramo biti pozorni pri delu z njimi. Izpostavljene vidike nekaj bolj poznanih omrežnih datotečnih formatov smo med seboj primerjali, določene pa tudi natančneje opisali. Izpostavili smo problematiko predstavitve t.i. slogov omrežij. V zvezi z omrežnimi datotečnimi formati smo raziskali obstoječa orodja za analizo omrežij, vključujoč programe za pridobivanje in pretvarjanje omrežnih podatkov. Rezultate smo podali v tabelah in prikazih. Posebej smo pripravili nekaj splošnih nasvetov, ki jih je pri pretvarjanju datotek z omrežnimi podatki smotrno upoštevati. Raziskava je nastala za potrebe prispevka v enciklopediji o analizi družbenih omrežij in pridobivanju omrežnih podatkov. Tako podrobnega pregleda omrežnih datotečnih formatov drugod nismo zasledili.

V prihodnosti se bomo na raziskovalnem področju osredotočili na paralelizacijo na-

šega algoritma za razvrščanje in ga prilagodili za delo z usmerjenimi omrežji. Še naprej bomo sledili spremembam na področju orodij za analizo omrežij in na področju omrežnih datotečnih formatov ter morda razvili primeren in splošen jezik za opisovanje slogov omrežij. Izpopolniti moramo metodo za prikaze hierarhij.

Na aplikativnem področju se bomo ukvarjali z nadaljnjim razvojem našega orodja za analizo omrežij. Orodje je potrebno dopolniti, ga narediti bolj robustnega in ga približati širši skupini uporabnikov. Naredili bomo še kakšno bibliografsko oz. scientometrično analizo. Morda bomo razvili kak univerzalni pretvornik datotek različnih omrežnih datotečnih formatov.

Ključne besede: abstrakcija, analiza omrežij, bibliografska omrežja, hierarhija, interaktivnost, omrežni datotečni format, orodje za analizo omrežij, prikaz, promet v mobilnem omrežju, razpršeni prikaz, razvrščanje, scientometrika, topološki indeks, vizualizacija

University *of Ljubljana*
Faculty *of Computer and Information Science*
Jernej Bodlaj
Hierarchical visualization of large networks

ABSTRACT

In the dissertation we present some improvements of the network analysis and visualization approaches. One of the basic approaches in the network analysis is a clustering and between the numerous agglomerative clustering techniques we noticed some possibilities for improvements. These are the introduction of relational constraints into the agglomerative clustering and the introduction of new dissimilarity measures, which beside the structure of a network consider also its properties. Among the tools for network analysis and visualization, we noticed that only a few use a structured approach to the analysis. They are mostly designed to perform the analysis in a number of discrete steps, which may render the analysis in terms of overhead time more time consuming and may require the users to be well organized. Similar seems to apply for the visualization of networks. Using the introduction of elements of interactivity into the analysis and visualization it is possible to make these tools more user-friendly and efficient. To promote user experience in real time the abstraction alongside the interactivity has to be introduced as well. Over the course of research and development of our new clustering algorithm, we searched for a suitable network to test the algorithm and found out that no extensive bibliographic analysis has been performed in the field of topological indices. Besides obtaining the network for the evaluation of our algorithm, we prepared an interesting insight into the field of topological indices. While many subjects in the dissertation are tightly bound on the collection and on the storage of network data, we discuss most of the known network file formats in regard to these two aspects. In the same regard we highlight some tools for network analysis as well.

A hierarchical clustering is a process by which we identify internally strongly connected regions (cluster or groups) of nodes in a network. The idea can be adapted to a clustering of links in a network to identify groups of links instead. In doing so, groups of nodes belonging to groups of links overlap. We present a basic hierarchical agglomer-

ative algorithm for clustering nodes, which we adapt into new algorithm for clustering links in a network. Alongside we introduce some new dissimilarity measures, which, unlike the existing ones, considering only the structure of a network, take also properties (descriptions, attributes) of network nodes and/or links into account. Unlike some existing algorithms, our algorithm is only implicitly dependent on a line graph of the input network, thereby having lower complexity. Based on artificial and real networks with known characteristics, we statistically investigate space and time complexities of both algorithms. On simple networks we study them also analytically. We compare the results of the algorithm to the ground truth knowledge of real networks and this way validate and confirm the usefulness of our new algorithm.

Among networks for the evaluation of our algorithm we constructed a series of bibliographic networks from the field of topological indices. Bibliographic data on topological indices were obtained from the online service *Web of Science*. We performed a comprehensive scientometric analysis of the field of topological indices. We report results of general analysis of collected data. We then delve into more specific analyses of authors, articles, journals, sub-topics of topological indices and keywords. We also analyze the relations between combinations of authors and journals, journals and articles etc. Scientometric approaches we use are for example the strongest and the main citation path to expose the most important works in the field and to identify major subtopics in the field using sets of keywords of works on the main path. We analyze co-authorships, we search for islands of collaborating authors in the collaboration network, we identify two-mode cores in the network of authors and works etc. We focus our interest to find out how authors cite each other, we highlight the important journals and citations between works published in them, we reveal journals favoured by previously identified groups of authors and construct a hierarchy of similar authors based on keywords they use. We discuss results of temporal analysis on one specific journal as well.

During a development of the algorithm and the above analyses, we started to develop a new comprehensive tool for network analysis; the interactive graphical environment *net.Plexor* based on the framework/library for work with large networks. Besides various support methods and algorithms, the above algorithm was incorporated into the tool. The tool is designed to be simply extended with additional functionalities via plugins. In the dissertation we discuss only parts of the tool that were designed by the author of the dissertation alone. This covers the logic responsible for visualization

of the networks, the logic to plan and carry out the network analysis – the analysis planer and the interactive graphical user interface in general. An interactive design enabling users to organize the work into the so-called analysis plans with an aid of visual programming was not encountered in any other network analysis tool. Besides visual programming we introduce also the concept of changing the input parameters of the algorithms on the fly in real time, which is a novelty in network analysis as well. We incorporate network and hierarchy visualizations into the tool, both externally referenced views (the observer is not part of the display), as well as internally referenced views (observer takes an active part of the display and can travel throughout the view). The internally referenced visualizations represent a new approach in the field of network analysis. In essence, internally referenced visualizations represent an example of virtual reality.

As an independent example of abstraction we have developed some summarizing visualization techniques to display a traffic in mobile telecommunications networks. From a half-year period mobile traffic data, obtained from the renowned telecommunications provider for more than a thousand base stations in Côte d'Ivoire, we prepared the cumulative networks of traffic. The networks were extremely dense and wishing to visualize them, we developed some appropriate new procedures. The traffic was shown using various flat and geographically oriented visualizations. Initially, the networks were visualized by conventional node and line diagrams and later with a specialized dispersed line diagrams, displaying traffic in a more natural way. We have also developed the so-called star diagrams. The traffic was also presented with a type of skeletal visualization. We have prepared two animations of traffic evolution through the observation time and some density diagrams of base stations.

As a complement to the already described research, we focus on initial stages in network analysis. We expose network file formats used to store network data in the form of files on storage media. To a lesser extent, we focus on procedures, how to collect and convert network data. We show the characteristics of network file formats according to their needs such as: readability, portability, generality, space requirements, speed of encoding and decoding, reading and writing etc. We arrange network file formats according to their basic features and highlight important aspects, which need to be considered when working with them. According to these aspects we compare some of the most popular network file formats. We describe a few of them in details. We highlight the issues with storing and defining the network layout and style. In regard to

network file formats we expose existing tools for network analysis, including programs to collect and convert network data and also give advice on efficiently converting network data files using the noted programs or without them. The main purpose of our research was to obtain the information needed when designing our network analysis tool. As we were unable to find any detailed review of network file formats elsewhere, we prepared one in a form of a contribution to the Encyclopedia of social network analysis and mining.

In the future we are going to focus on the parallelization of our clustering algorithm and adapt it to work with directed networks. We plan to keep a track of changes in the field of network analysis tools and network file formats and we may try to develop a suitable declarative language to be able to describe the style of a network. Our method for the visualization of hierarchies has to be improved as well.

Besides theoretical work we are going to further develop our tool for network analysis. We need to incorporate much necessary functionalities into it and make it more robust. We have to bring it closer to a wider group of users. We are also looking forward to make some alternative bibliographic and scientometric analyses. We might try to implement an online converter for various network file formats as well etc.

Key words: abstraction, bibliographic networks, clustering, dispersed diagram, hierarchy, interactivity, layout, mobile network traffic, network analysis, network analysis tool, network file formats, scientometrics, topological index, visualization

ZAHVALA

Zahvaljujem se mentorju in profesorju Vladimirju Batagelju ter somentorju, profesorju Francu Solini, saj brez njiju pričujočega dela ne bi bilo. Zahvaljujem se vodstvu podjetij Hruške d.o.o. in Abeliuma d.o.o., ki sta mi v času podiplomskega študija nudila raziskovalno okolje. V Hruški sem bil zaposlen kot mladi raziskovalec iz gospodarstva. Zahvala gre strokovnim sodelavcem, med njimi posebej raziskovalki Moniki Cerinšek s katero sva vsak na svoji raziskovalni poti večkrat izmenjevala ideje glede tekočega in nadaljnjega dela ter Ajdi Korošec za njene spodbude. Zahvaljujem se mami in stricu, prijateljem ter vsem neimenovanim posameznikom, ki so mi tako ali drugače pomagali na poti do cilja pri podiplomskem študiju. Za preživljanje kvalitetnega prostega časa, ki mi je v času študija omogočalo odmisliti vse skrbi v povezavi z njim, pa se zahvaljujem vsem jadalno-padalskim kolegom.

Še enkrat bi se rad zahvalil profesorju Milanu Randiću in anonimnim recenzentom, ki so s svojimi pripombami in predlogi pomagali izboljšati oba članka, ki sva jih z mentorjem napisala.

Delo je bilo delno financirano s strani ARRS, projekt J5-5537, delno v okviru programa EUROCORES Programme EUROGIGA (na projektu GReGAS) Evropskega Znanstvenega Sklada. Avtor je bil financiran s strani Evropske Unije, Evropskega Socialnega Sklada, Slovenskega ministrstva za znanost, šport in promet in delno tudi iz Kompetenčnega centra KC CLASS, ki je bil financiran s strani Evropske Unije, Evropskega Regionalnega Sklada.

— Jernej Bodlaj, Ljubljana, februar 2015.

KAZALO

Povzetek *i*

Abstract *v*

Zahvala *ix*

1 Uvod *1*

1.1 Motivacija . . . 2
1.2 Znanstveni doprinosi . . . 4
1.3 Metodologija . . . 7
1.4 Pregled disertacije . . . 11
1.4.1 Notacija . . . 12
1.4.2 Definicije . . . 12

2 Analiza bibliografskih omrežij objav iz področja topoloških indeksov *15*

2.1 Topološki indeksi in scientometrika . . . 16
2.2 Sorodna dela . . . 17
2.3 Gradnja bibliografskih omrežij . . . 18
2.3.1 Omejitve pri pripravi množice citiranih del . . . 19
2.3.2 Osnovne lastnosti bazne množice del . . . 19
2.3.3 Robni problem . . . 20
2.4 Analize . . . 22
2.4.1 Poti citiranj . . . 23
2.4.2 Normalizirano število del . . . 26
2.4.3 Analiza podpore avtorjem – kako avtorji citirajo druge avtorje 27

2.4.4 Analiza soavtorstev del . . . 31
2.4.5 Analiza revij . . . 37
2.4.6 Revije in avtorji . . . 40
2.4.7 Ključne besede . . . 42
2.5 Razprava . . . 44
2.5.1 Rezultati drugih analiz . . . 46

3 Hierarhični prikazi *47*
3.1 Prikazi podatkov . . . 48
3.2 Pogosti prikazi drevesnih hierarhij . . . 50
3.2.1 Drevo z vozlišči in povezavami . . . 51
3.2.2 Sosednostni prikazi . . . 53
3.2.3 Prikazi z ogradami . . . 54
3.2.4 Hiperbolični prikazi . . . 56
3.2.5 Tri-razsežna drevesa stožcev . . . 58
3.2.6 Hierarhično združevanje povezav v snope . . . 59
3.3 Prikaz in preoblikovanje splošnih hierarhij . . . 60

4 Hierarhično razvrščanje v omrežjih *65*
4.1 Uvod . . . 66
4.2 Sorodna dela . . . 68
4.3 Razvrščanje vozlišč omrežja z združevanjem . . . 69
4.3.1 Osnovni algoritem . . . 70
4.4 Združevalni postopek za razvrščanje povezav . . . 71
4.4.1 Prilagojeni algoritem . . . 71
4.5 Mere različnosti . . . 75
4.5.1 Direktne mere različnosti . . . 76
4.5.2 Rekurzivno izračunljive mere različnosti . . . 80
4.6 Dokazi monotonosti mer različnosti . . . 82
4.6.1 Dokazi monotonosti mer različnosti D_{SF}, D_{SC} in D_{MF} . . . 82
4.6.2 Omejena različnost po minimalni metodi zagotavlja monotonost . . . 84
4.6.3 Omejena različnost po povprečni metodi zagotavlja monotonost 84
4.7 Zahtevnosti algoritmov . . . 85

4.8 Uporaba algoritma na omrežjih in evaluacija 90
4.8.1 Demonstracijsko umetno omrežje 91
4.8.2 Primer na realnem omrežju – Spletna trgovina 95
4.8.3 Primer realnega omrežja – Omrežje sodelovanj avtorjev na področju topoloških indeksov 96
4.9 Diskusija 98
4.9.1 Stabilnost prilagojenega algoritma 99
4.9.2 Paralelizacija 100
4.9.3 Zaključek 100

5 Prikaz mobilnega prometa D4D *101*
5.1 Uvod 102
5.2 Podatki mobilnega omrežja 102
5.3 Analiza 104
5.3.1 Pristop s klasičnimi prikazi vozlišč in povezav 104
5.3.2 Razpršeni porazdelitveni prikazi prometa 107
5.3.3 Prikaz prometa z zvezdnimi prikazi 112
5.3.4 Soodvisnost števila klicev in trajanja klicev 115
5.3.5 Število in trajanje lokalnih klicev v odvisnosti od velikosti področja baznih postaj 115
5.3.6 Trajanje in število klicev v povezavi z razdaljo klicev 115
5.3.7 Porazdelitev verjetnosti izbire daljice z znano dolžino v kvadratu 116
5.3.8 Prikazi izbranega prometa 117
5.3.9 Dnevno in tedensko obnašanje baznih postaj – prikaz skupin baznih postaj 120
5.4 Zaključki 128

6 Interaktivni grafični uporabniški vmesnik in knjižnica za analizo velikih omrežij 129
6.1 Orodja v analizi omrežij 130
6.1.1 Primeri programskih rešitev za analizo omrežij 131
6.1.2 *Pajek* – orodje za analizo velikih omrežij 132
6.2 Interaktivni grafični uporabniški vmesnik in knjižnica za analizo velikih omrežji – *net.Plexor* 133
6.2.1 Knjižnica, osnovno ogrodje 133

6.2.2 Interaktivni grafični uporabniški vmesnik 134
6.3 Interaktivnost v *net.Plexor*-ju 136
6.3.1 Uporabniški vmesnik kot končni avtomat 138
6.4 Operacijski gradniki v *net.Plexor*-ju 147
6.4.1 Prikazovalnik in urejevalnik omrežja 147
6.4.2 Algoritem za razvrščanje povezav omrežja in podporni gradniki 154
6.4.3 Prikazovanje hierarhij 156
6.4.4 Umeščanje omrežij s hitro multipolno metodo 159
6.4.5 *net.Plexor* v prihodnosti 160

7 Zaključek *163*
7.1 Glavni znanstveni doprinosi 164
7.2 Smernice za nadaljnje delo 166

A Dodatki *169*
A.1 Priprava bibliografskih omrežij in drugih konstruktov iz *Web of Science* 170
A.1.1 Množica sorodnih del 171
A.1.2 Priprava razbitja po vrstah del z orodjem *Wos2Pajek* 172
A.1.3 Težave z imeni avtorjev pri storitvi *Web of Science* in obdelavi z *Wos2Pajek* 173
A.2 Omrežni datotečni formati 174
A.2.1 Vrste omrežnih datotečnih formatov 174
A.2.2 Vidiki omrežnih datotečnih formatov 177
A.2.3 Vidiki nekaterih popularnih ODF-ov 182
A.2.4 ODF-i, ki se v praksi veliko uporabljajo 184
A.2.5 Pretvarjanje različnih formatov 187
A.2.6 Datotečni formati za grafično predstavitev omrežij 188
A.2.7 ODF-i in združljivi programi za analizo omrežij 189
A.2.8 Opombe 189
A.2.9 Omrežni datotečni formati v prihodnosti 194
A.3 Definicija hierarhičnega končnega avtomata 194
A.4 Hitra multipolna metoda v tehnologiji *OpenCL*® 198
A.4.1 Naivni algoritem 198
A.4.2 Izboljšan algoritem 199

A.4.3 Posplošeni algoritem v prostoru 200
A.4.4 Programska izvedba 200
A.4.5 Analiza časovne zahtevnosti 201

Literatura *205*

I

Uvod

1.1 Motivacija

Analizo omrežij že kar nekaj časa srečujemo v praksi in na mnogih področjih, predvsem na področju družbenih ved. Z napredkom tehnologije se njen spekter uporabe še bolj širi. V zadnjem obdobju se dogaja pravi razcvet socialnih omrežij in z njimi uporaba analize omrežij za odkrivanje raznih vzorcev, ki se v takšnih omrežjih porajajo. Naloge, kot so analiza trgov za podlago pri oglaševanju, promocijo, optimizacija iskanja na spletu, zaznavanje skupin, kritičnih področij v omrežjih in mnoge druge, vse temeljijo na analizi omrežij. V današnjih časih nam analiza omrežij odpira mnoge nove vidike in odkriva zakonitosti v množicah podatkov, ki so na voljo na modernih medijih. Često so podatki že po svoji naravi strukturirani v obliki omrežij, ali pa lahko omrežja iz njih konstruiramo. Dobljena omrežja so lahko velika, z več tisoč, več milijoni in tudi milijardami vozlišč. Običajni primeri takšnih omrežij so že omenjena socialna omrežja, bibliografska omrežja, omrežja denarnih transakcij, omrežja sopojavljanja besed v besedilih in mnoga druga. Da bi natančneje spoznali analizo omrežij in hkrati pridobili stvarne podatke, ki bi jih lahko uporabili za podlago pri analizi algoritma, ki smo ga razvili, smo se osredotočili na področje topoloških indeksov. Gre za primer znanstvene domene, na podlagi katere smo konstruirali bibliografska omrežja in jih analizirali.

Eden od temeljnih pristopov v analizi omrežij in na splošno v velikih podatkovjih je razvrščanje vozlišč oz. enot v hierarhijo skupin. S tem pristopom je možno identificirati področja v omrežjih, ki so v skladu z neko različnostjo blizu skupaj. Pri hierarhičnem razvrščanju v omrežjih obstajata dve osnovni metodi razvrščanja; prva, kjer začnemo s trivialno majhnimi skupinami vozlišč, ki jih združujemo v čedalje večje sestavljene skupine (združevalne metode) ali pa obratno, da začnemo z eno veliko skupino, t.j. celotnim omrežjem, ki jo/ga nato delimo v čedalje manjše skupine (delitvene metode). V obeh primerih dobimo hierarhijo skupin vozlišč omrežja. Hierarhične razvrstitve omrežja lahko dobimo tudi na primer s postopkom za določanje otokov. Ukvarjali smo se predvsem z algoritmi, ki razvrščajo z združevanjem. V splošnem se najbolj razlikujejo v izbiri primerne mere različnosti, ki določa, kateri skupini se v dani iteraciji algoritma združita. Pomemben je tudi način določanja različnosti med sestavljenimi skupinami (minimalna, maksimalna, povprečna, sredinska metoda, itd.)

V delu *Lehmann*-a in sodelavcev [1] je predlagan algoritem za razvrščanje povezav v omrežju. Za razliko od razvrščanja vozlišč je bistvena prednost razvrščanja povezav v tem, da v različnih povezanih skupinah lahko nastopajo ista vozlišča (vozlišče je lah-

ko krajišče večih povezav hkrati). Razvrščanje vozlišč pa vozlišča združuje v ločene skupine. V marsikaterem omrežju razvrščanje vozlišč v ločene skupine vsebinsko ni smiselno, zato takrat le-tega ne bi smeli uporabiti oz. ga zavestno vsilimo. Predlagana metoda za razvrščanje povezav je posredna. Za delovanje uporablja tradicionalni pristop hierarhičnega razvrščanja vozlišč z različnostjo po minimalni metodi na t.i. povezavnem grafu osnovnega omrežja. Dobljene rezultate metoda projicira nazaj na osnovno omrežje. Metodo smo predelali, da deluje direktno, brez prehoda na povezavni graf. Pri tem smo definirali nove različnosti, ki poleg strukture omrežja upoštevajo še lastnosti povezav in vozlišč omrežja.

Temeljni pristop v analizi omrežij je vizualizacija. Hierarhijo skupin omrežja, ki jo ob upoštevanju povezavne strukture omrežja in morebitnih drugih podatkov o elementih omrežja dobimo z razvrščanjem, lahko uporabimo za abstrakcijo v prikazih omrežja. Namesto z veliko množico vsebujočih vozlišč in povezav lahko na ta način velike skupine predstavimo poenostavljeno in bolj pregledno. Osnovni graf/omrežje torej razbijemo na posamezne skupine, ki vsebujejo kontekst, t.j. konkretne podrobnosti izvornega omrežja. Z metodo se dobi tudi model omrežja, t.j. graf povezanih skupin s hierarhijo oz. drevesom, ki predstavlja gnezdenje skupin.

S pomočjo interaktivnih postopkov je prikaze moč nadalje izboljšati. Za prikaz omrežja lahko uporabimo računalniški zaslon, na katerem lahko izbiramo (vključujemo/izključujemo) sestavine, ki naj bodo prikazane, premikamo pogled, povečujemo in zmanjšujemo zorni kot pogleda – uporaba lupe in na različnih nivojih povečave pogleda vizualiziramo z drugačno metodo, t.j. koncept nivoja podrobnosti. Uporabljamo lahko tudi vzporedne prikaze, kot so globalni in lokalni pogled ali celo avtomatično sledenje, sestopanje in ponavljanje vodenih poti ogledov ali morda prikaze z raznimi dodanimi opisi, legendami, mrežami, zemljevidi za orientacijo in možnostmi za različne poti nazaj na višje nivoje pogledov, prikaz različnih pomožnih podatkov ipd. Interaktivnost torej v prikaze vnaša dodatne razsežnosti, predvsem prostor in čas. Morda bi lahko vnesli tudi zvok. Predstavljene ideje so natančneje opisane v zadnjem razdelku *Batagelj*-evega dela [2].

Medtem, ko je v tehniki problem prikaza grafa usmerjen k iskanju najboljše slike, analiza omrežij spada v področje analize podatkov. Njen cilj je vpogled v strukturo in lastnosti danega omrežja, pa tudi vpogled kako ta struktura vpliva na procese, ki se dogajajo v omrežju. Navadno zato potrebujemo več različnih prikazov. V vseobsegajočem pogledu velikih omrežij se namreč podrobnosti izgubijo, nasprotno pa pri

natančnem vpogledu prikažemo le del strukture omrežja. Tudi v tem pogledu gre za načelo abstrakcije in interaktivnosti.

Pri prikazovanju izpostavimo še dva vidika. Prvi je vidik robustnosti, ki je ključnega pomena, da rezultate lahko uporabimo na več neodvisnih področjih. Poleg tehnološkega in teoretičnega ozadja vizualizacije omrežij pa je pomembno (drugi vidik) upoštevati tudi psihološki učinek izgleda slike oz. prikaza omrežja na uporabnika rezultatov. Zaželeno je, da so rezultati čim bolj informativni, prijetni na oko in zanimivi kot slike same, npr. če bi jih opazoval laik.

Poglejmo še področja, kjer bi zgornji algoritem in metode vizualizacije lahko uporabili. Širše gledano bi oboje lahko uporabili pri napovedovanju obnašanja omrežij, pri iskanju tipičnih ciljnih marketinških skupin, v preprečevanju izgub v energetskih omrežjih, pri preprečevanju goljufij in korupcije ali zgolj za inteligentno pomoč pri povezovanju in uporabi spletnih storitev glede na vedenje (obnašanje ter usmerjanje) njihovih uporabnikov, pri informacijski podpori v logistiki, pri programski izvedbi inteligentnih spletnih iskalnikov po lokalnih omrežjih in še marsikje. Uporabili bi ju lahko tudi v sistemih za analizo podatkov pri okoljskem ali meteorološkem monitoringu, pri analizi obnašanj uporabnikov v mobilnih omrežjih, v analizah računalniških omrežij, pri analizah semantičnega spleta, pri sledenju finančnih transakcij, za raziskovanje relacij med podatki v bazah znanj, predstavljenih v oblikah grafov itd. Poudariti je treba, da v analizi omrežij največkrat uporabimo več različnih pristopov, da dobimo predstavo o lastnostih opazovanega omrežja. Šele nato lahko z večjo gotovostjo interpretiramo rezultate specializiranih algoritmov in naš ni izjema. Predstavlja le dodatno orodje v analizi omrežij.

1.2 *Znanstveni doprinosi*

Prvi znanstveni doprinos je obsežna analiza podatkov v obliki nabora bibliografskih omrežij, ki smo jih sestavili iz podatkov na temo topoloških indeksov, dostopnih preko spletne storitve *Web of Science*. Podatki hkrati predstavljajo del testnega nabora omrežij za testiranje algoritma, ki smo ga razvili in ga izpostavljamo v naslednjem odstavku. Topološki indeksi predstavljajo področje v matematični kemiji, molekularni topologiji in v aplikativni teoriji grafov v kemiji. Ker bibliometrične analize dovolj dobro sovpadajo z analizo omrežij, ki predstavlja eno od temeljnih področij za uporabo zgornjega algoritma, smo analizo dobljenih bibliografskih omrežij razširili in posplošili

ter s tem pripravili obsežen scientometrični pregled področja topoloških indeksov. V preteklosti za področje topoloških indeksov takšen pregled še ni bil narejen in je kot tak nov. Na podlagi odziva recenzentov na članek, ki je nastal na podlagi rezultatov, se je izkazalo, da je tudi zanimiv in zaželen.

Drugi prispevek je algoritem za razvrščanje povezav omrežij, ki tradicionalno deluje z uporabo standardnih metod razvrščanja vozlišč na pripadajočem povezavnem grafu osnovnega omrežja in s projekcijo rezultatov nazaj na osnovno omrežje. Težave tradicionalnega pristopa nastopijo v gostih, močno povezanih omrežjih, kjer imajo vozlišča zelo velike stopnje. Pri porojenem povezavnem grafu vozlišča visokih stopenj vodijo v tvorbo polnih podgrafov (klik) reda stopenj teh vozlišč. Velikost oz. število povezav v polnem grafu narašča s kvadratom njegove stopnje. V praksi lahko to predstavlja problem. Naš algoritem težavo odpravi tako, da se izogne prehodu na celoten povezavni graf in razvrščanje opravi bolj ali manj direktno. S tem sta njegovi zahtevnosti (prostorska in časovna) manjši. Pri izvedbi algoritma smo razvili še nekaj novih različnosti, ki upoštevajo multivariantne podatke oz. simbolne objekte v omrežju (to so dodatni podatki, ki skupaj z grafovsko strukturo tvorijo omrežje), saj algoritem brez mer različnosti ni uporaben. Od mere različnosti je odvisno, kako bo potekalo razvrščanje skupin v hierarhijo. Algoritem iz *Lehmann*-ovega članka [1] uporablja enostavno mero različnosti, ki upošteva le strukturo omrežja. Naš algoritem že sam po sebi upošteva strukturo omrežja (združuje povezane dele omrežja), nove različnosti pa upoštevajo še lastnosti vozlišč in/ali povezav. Pri tem smo različnosti zasnovali tako, da zagotavljajo monotonost dobljenih hierarhičnih razvrstitev, kar smo podprli z ustreznimi dokazi.

Naslednji prispevek je celovita in inovativna programska rešitev v obliki orodja za analizo velikih omrežij *net.Plexor*, ki med drugim vključuje večino v tej disertaciji opisanih postopkov, kot so interaktivni pregledovalniki velikih omrežij in hierarhij s prvinami abstrakcije, prikazi z notranjimi pogledi, algoritmi za razvrščanje, umeščanje vozlišč omrežij v prostor itd. Pri zasnovi orodja smo v analizo omrežij vpeljali tri novosti. To sta podpora za vizualno programiranje in interaktivno krmiljenje algoritmov analize omrežij v realnem času, ki temeljita na namenskem hierarhičnem končnem avtomatu. Tretja novost je vpeljava prikazov omrežij z notranjimi pogledi, ki predstavljajo eno od osnovnih značilnosti navidezne resničnosti.

Na podlagi (npr. z uradno klasifikacijo) pridobljene hierarhične razvrstitve skupin omrežja smo skušali razviti načrt bolj ali manj dobre metode za hierarhično pregle-

dovanje velikih omrežij. Tu mislimo predvsem na postopke prikazovanja in skrivanja različnih podrobnosti omrežja, alternativne prikaze omrežja, začasne prikaze delov omrežja, tudi vodene oglede, in sicer na različnih nivojih hierarhije. Prikazi višjih nivojev vsebujejo le relevantne povezave, vozlišča omrežja, na globljih nivojih pa vse več podrobnosti. V disertaciji predlagamo način, kako s prepletanjem interaktivnosti in abstrakcije prikazovati oz. urejati velike množice hierarhično urejenih podatkov, kot so na primer hierarhije velikih omrežij, kot jih vrne naš algoritem. Za razliko od običajnih pregledovalnikov in urejevalnikov hierarhij, ki so omejeni na drevesne hierarhije, predlagamo teoretično osnovo, na kateri je mogoče zgraditi pregledovalnik ali urejevalnik, ki omogoča delo s splošnimi hierarhijami oz. z acikličnimi grafi. S poudarkom na učinkovitosti, robustnosti in splošnosti smo metodo okvirno realizirali v našem orodju za analizo velikih omrežij *net.Plexor*.

Izpostavimo še dva manjša prispevka. To sta, grobo povedano, razvoj metode za geografsko prikazovanje mobilnih omrežij in pregled datotečnih omrežnih formatov.

- Pri načrtovanju metode za geografsko prikazovanje je šlo predvsem za poskus vizualizacije mobilnega omrežja Slonokoščene obale v okviru izziva *D4D*. V okviru dela smo sicer opravili tudi nekaj splošnih analiz mobilnega omrežja, predvsem celičnih baznih postaj in prometa med njimi. Na podlagi rezultatov smo pripravili plakat in ga predstavili na konferenci *NetMob* 2013. Naša metoda prikazov mobilnih omrežij je tu zanimiva, ker jo je možno posplošiti. Lahko jo uporabimo kot primer načina prikaza hierarhij velikih omrežij, in sicer v skladu z zgornjimi idejami ter za prikaz poljubno velikih in gostih omrežij, če le imamo koordinate vozlišč. Pri tem prikaz vedno odraža celotno strukturo omrežja, saj metoda temelji na prikazu gostote omrežja. Kot taka je primerna za prikaz višjih nivojev v hierarhiji omrežja in kot bomo spoznali, predstavlja primer prikaza s povzetki.

- Omrežni datotečni formati na nek način predstavljajo temelj v moderni analizi omrežij, saj rešujejo problem, kako omrežne podatke shraniti na računalniških pomnilnih medijih. Bolj splošno gledano, v prispevku z njimi v povezavi navajamo še postopke, kako omrežne podatke zbrati in pripraviti ter na katere ovire lahko pri tem naletimo. S stališča omrežnih datotečnih formatov opišemo nekaj popularnih orodij za analizo omrežij. V obliki samostojnega poglavja je

prispevek objavljen v enciklopediji o analizi socialnih omrežij in pridobivanju omrežnih podatkov. V disertacijo smo ga vključili, ker so zbiranje, priprava in shranjevanje omrežnih podatkov začetni koraki v analizi omrežij. Če algoritem za razvrščanje povezav predstavlja primer vmesnega koraka analize omrežij in vizualizacija rezultatov končne korake, potem s prispevkom te alineje zgodbo primera analize omrežij združimo v smiselno celoto.

Povzetek:

- *Obsežen scientometrični vpogled v področje topoloških indeksov*
- *Izboljšan in razširjen algoritem za hierarhično razvrščanje povezav v omrežjih*
- *Načrt in programska izvedba interaktivnega orodja za analizo omrežij z vizualnim programiranjem, kontrolo v realnem času in prvinami navidezne resničnosti*
- *Načrt metode za prikazovanje velikih hierarhično urejenih omrežij*
- *Metoda geografskega prikaza za promet mobilnih omrežij*
- *Pregled omrežnih datotečnih formatov in nekaterih orodij za analizo omrežij*

1.3 Metodologija

Začetni in hkrati sprotni pristop k problematiki je bil študij literature, aktualnih člankov, priprava zapiskov in poročil, ki so nam služili pri pripravi doktorske disertacije. Verjetno nam bodo koristili tudi pri nadaljnjem delu. Sledile so raziskave in predvsem razvojno delo.

Algoritem smo razvili v obliki samostojne aplikacije – prototipa in pripravili enostavne testne primere, na katerih smo se prepričali v pravilnost njegovega delovanja. Algoritem smo vgradili v orodje za analizo omrežij, ki ga razvijamo in tako pridobili nekaj dodatnega maneverskega prostora za testiranje in izvajanje primerjalnih analiz ter pregledov in si omogočili hitrejši in avtomatiziran vpogled v rezultate algoritma. Opravili smo statistično analizo časovne in prostorske zahtevnosti algoritma. Pripravili smo nabore grafov – omrežij z znanimi parametri in obliko, za katere vemo, da dovolj dobro modelirajo omrežja, ki se redno pojavljajo v praksi in na njih zaganjali algoritem in

spremljali njegovo obnašanje. Za posamezne primere smo časovno in prostorsko zahtevnost oz. njihove zgornje meje določili analitično. Algoritem smo zasnovali tako, da ga je, do določene mere, enostavno paralizirati na več-jedrni strojni opremi. Dodatno smo se poglobili še v tehnologijo *OpenCL®*, ki ponuja možnost paralelizacije na GPE – grafični procesni enoti. Algoritma v tehnologiji *OpenCL®* sicer nismo implementirali, smo pa na primeru opazovali razliko v hitrosti procesiranja na GPE v primerjavi s CPE – centralno procesno enoto. Ker je algoritem namenjen splošni uporabi v analizi omrežij, smo gledali tudi na njegovo robustnost in stabilnost ter ga skušali optimizirati.

Za delovanje algoritma nujno potrebujemo mere različnosti, saj se na podlagi različnosti skupin v trenutni iteraciji algoritem odloči, kateri dve bo združil. Razvili smo več mer različnosti in z uporabo preizkusili ter opredelili njihovo uporabno vrednost pri analizi različnih testnih omrežij in tudi omrežij iz realnega sveta. Opise algoritma, različnosti in povzetke poskusov z njimi smo združili v članku, ki je v fazi pisanja disertacije oddan v objavo. Prejeli smo že tudi pozitiven odziv recenzentov in večino njihovih pripomb v disertaciji upoštevali. Algoritem je bil predstavljen na mednarodni konferenci in na več domačih seminarjih.

Skupaj s primernimi različnostmi smo algoritem uporabili pri analizi več konkretnih omrežij, tako naravnih, kot tudi umetnih. Algoritem smo poganjali na bibliografskih omrežjih, ki smo jih v ta namen pripravili. Bibliografska omrežja smo ustvarili iz podatkov, ki smo jih sprva morali pridobiti. Izbrali smo znanstveno področje – področje topoloških indeksov in iz spletne storitve *Web of Science* ročno pridobili podatke o publikacijah s tega področja. Za postopek tvorbe omrežij iz pridobljenih podatkov smo uporabili orodje *Wos2Pajek*, ki iz ročno prenesenih podatkov sestavi bibliografska omrežja in jih shrani v *Pajkovem* formatu *.net*. (*Pajek* je uveljavljeno orodje za analizo omrežij.)

Da bi lažje ovrednotili pravilnost našega algoritma, smo z uveljavljenimi postopki bibliografska omrežja temeljito preučili in si tako izoblikovali širše znanje o njih. Identificirali smo ključne skupine znanstvenikov – avtorjev, določili poglavitne revije, izvedli analizo ključnih besed v domeni topoloških indeksov, identificirali glavne poti citiranj... Analiza bibliografskih omrežij je kmalu prerasla v obsežno raziskavo. Večji del raziskave smo izvedli v *Pajku*, del pa tudi v našem programu *net.Plexor*. Nekatere dodatne in specifične, tudi primerjalne analize smo izvedli v drugih namenskih ali posebej razvitih orodjih. Rezultate analiz smo v obliki besedila, nabora tabel, slik, prikazov, hierarhij ter smiselnih interpretacij objavili v članku za katerega verjamemo,

da bo vpletenim avtorjem in drugim bralcem razširil pregled na področje topoloških indeksov. V članku navedemo še nekaj predlogov za izboljšanje konsistence podatkov v bibliografskih bazah. Tematika je bila predstavljena tudi na mednarodni konferenci in nekaj seminarjih.

V zvezi s problematiko prikazov velikih omrežij ter hierarhičnih prikazov le-teh smo najprej poiskali primere obstoječih prikazov za ta namen, pretežno na spletu. V skladu z obliko rezultatov algoritmov, ki ju obravnavamo, smo pripravili formalne opise hierarhij in načrtali funkcionalnosti, ki bi jih želeli na takšnih hierarhijah izvajati, t.j. pri pregledovanju in tudi urejanju hierarhij v našem programskem okolju za analizo omrežij. Pri načrtovanju smo upoštevali načela abstrakcije in interaktivnosti. Zasnovano metodo smo (delno) programsko izvedli in jo vgradili v naše orodje. Pri programski izvedbi smo pazili na učinkovitost. S tehnologijama *OpenGL*® in *OpenCL*® smo izkoristili prednosti, ki jih nudi moderna strojna oprema, t.j. strojno pospeševanje na grafičnih karticah.

Pri razvoju orodja *net.Plexor* smo si zadali nekaj glavnih ciljev. Razviti orodje, ki bo delovalo na več operacijskih sistemih, ki bo temeljilo na odprtokodnih knjižnicah, ki bo hitro in učinkovito in še kaj. Naravna izbira programskega jezika je bil *c++*. Ker je programiranje v *c++* relativno zahtevno, zamudno in ni najbolj primerno za prototipiranje, smo marsikatero funkcionalnost predhodno razvili v kakem drugem jeziku. Nekatere elemente, kot so zasnova hierarhičnega končnega avtomata in večino 3-razsežnostne grafike, smo razvili "na papirju". Pretežen del kodiranja je potekal neposredno. Pomembni orodji, ki ju uporabljamo v pomoč pri kodiranju, sta *wxFormBuilder* [1] za načrtovanje in generiranje kode uporabniškega vmesnika in *Sybase PowerDesigner* [2] za pomoč pri snovanju hierarhičnih končnih avtomatov za podporo različnim interaktivnostim. Tekom razvoja smo uporabljali še mnoga druga orodja, pri katerih smo upali na kakšno dodatno bližnjico pri kodiranju, na primer za avtomatsko generiranje kode hierarhičnih končnih avtomatov ipd., vendar se je izkazalo, da nam nobeno povsem ne ustreza. Na kratko lahko zapišemo, da je program zahteven in zapleten ter s tem narava programiranja; še posebno, ker mora razvijalec v *c++* sam skrbeti za nadzor nad pomnilnikom. Kot zanimivost dodajmo, da je avtor disertacije prispeval okoli 40.000 vrstic kode, kar trenutno v kodi orodja predstavlja okoli štiri

[1] http://sourceforge.net/projects/wxformbuilder
[2] http://www.sybase.com/products/modelingdevelopment/powerdesigner

petine vse ročno napisane kode.

Pri pripravi pregleda omrežnih datotečnih formatov smo pregledali glavni del dostopnih spletnih virov, ki vsebujejo informacije o podrobnostih različnih formatov, jih zbrali skupaj in naredili ustrezne primerjave. Informacije, predvsem o lastniških formatih, smo črpali iz navodil za uporabo programske opreme, ki posamezen format uporablja. V navodilih smo dobili tudi informacije o lastnostih orodij, ki smo jih v prispevku opisali, delno pa iz izkušenj pri delu z njimi za lastne potrebe in na delavnicah. Splošna priporočila glede zbiranja omrežnih podatkov in sestavljanja omrežij smo zapisali na podlagi opisov praks v literaturi in na podlagi lastnih izkušenj. Del pregleda temelji na rezultatih, do katerih smo prišli s pomočjo analize dvodelnega omrežja združljivosti obravnavanih formatov in orodij.

Pri raziskavi podatkov o mobilnih komunikacijah na Slonokoščeni obali smo se osredotočili predvsem na prikaze prometa. Zaradi velike količine podatkov smo sprva razvili namensko aplikacijo za predelavo podatkov v ustrezno obliko, nato pa se osredotočili na primarni cilj. Metodo prikazov smo razvili in implementirali od začetka. Pretežno je šlo za programiranje in grafično obdelavo. Pri tem omenimo, da se nismo posebej prepričali, če naši podobna metoda za prikaze omrežij že obstaja. Drži pa, da pri pregledu velikega števila programov v prejšnjem odstavku, v zvezi z omrežnimi datotečnimi formati, med katerimi jih veliko omogoča tudi risanje omrežij, nismo zasledili nobene podobne metode. Sorodne metode prikazov se sicer uporablja za prikaz gostote omrežij. Pri izdelavi plakata smo posebej upoštevali načela za pripravo kvalitetnih plakatov.

Povzetek:

- pridobivanje znanj iz literature v povezavi s hierarhičnim razvrščanjem, zbiranjem podatkov, analizo omrežja, prikazovanjem ustreznih podatkov in drugimi področji;
- uporaba znanj pri konstrukciji algoritma za hierarhično razvrščanje in ustreznih različnosti ter pri vrednotenju algoritma;
- uporaba znanj o zbiranju podatkov in analizi omrežij na primeru področja znanosti o topoloških indeksih;

- razvoj in programska izvedba večine delov nove programske opreme za analizo omrežij;
- eksperimentalno delo z novim algoritmom v novi programski opremi;
- interpretacija rezultatov algoritma in splošnih analiz omrežij;
- poročanje in priprava pregledov aktivnosti na področju raziskav in razvoja v obliki poročil, člankov, plakata, poglavja v knjigi; objava le-teh v priznanih znanstvenih revijah oz. knjigi in aktivna udeležba na mednarodnih konferencah in seminarjih.

1.4 Pregled disertacije

Disertacija je razdeljena na več poglavij. Vsako zase je smiselno zaključena celota. Poglavja skupaj tvorijo zgodbo o primeru analize omrežja od začetka, t.j. od zbiranja podatkov, do konca, t.j. do predstavitve ugotovitev o podatkih. Disertacija v grobem odgovori na vprašanje, kako priti do omrežnih podatkov in jih shraniti v primerno obliko. Opredeli več praktičnih primerov analiz na pridobljenih podatkih, tako konceptualno, kot na primeru zgrajenih realnih bibliografskih omrežij s področja, kjer podobna analiza še ni bila opravljena in predstavi zanimive ugotovitve. V disertaciji je natančno opredeljenih nekaj povsem novih postopkov v analizi omrežij in razloženo je, na kakšen način pametno zasnovati oz. kako smo zasnovali in realizirali programsko okolje, kamor je takšne in druge analitične postopke možno vgraditi in jih pozneje na prijazen način uporabiti. V disertaciji je prikazan še dodaten primer stvarne analize omrežij, kjer je poudarek na prikazih s povzetki in pa razdelava načrta, v katerem so opredeljeni novi postopki hierarhičnih prikazov omrežnih podatkov, in sicer kot primer v zaključni fazi analize omrežij.

V poglavju 2 podamo izsledke celostne scientometrične analize bibliografskih omrežij iz publikacij na področju topoloških indeksov. Podatke smo pridobili iz spletne storitve *Web of Science*. Postopek pridobivanja podatkov in izgradnje omrežij opišemo v dodatku A.1. V poglavju 4 opišemo osnovni pristop razvrščanja v analizi omrežij in ga razširimo v nov pristop, t.j. hierarhično razvrščanje povezav z relacijsko omejitvijo v omrežjih. Podamo tudi primerne mere različnosti. V poglavju 3 opišemo nekaj običajnih pristopov prikazovanja hierarhično urejenih podatkov in razvijemo načrt prikazov splošnih hierarhij. V poglavju 5 izdelamo primer prikaza s povzetki oz. z

zgostitvami, in sicer na podatkih prometa v mobilnem telekomunikacijskem omrežju na Slonokoščeni obali. V poglavju 6 podamo informacije o idejah, zasnovi, razvoju in delovanju naše inovativne programske opreme za analizo omrežij *net.Plexor*, v katerega smo vgradili večji del zgoraj opisanih postopkov. V dodatku A.3 podamo definicijo hierarhičnega končnega avtomata, v dodatku A.4 pa izsledke raziskave strojnega pospeševanja na modernih grafičnih karticah. Dodatka predstavljata osnovi, na katerih temelji interaktivnost orodja *net.Plexor*. V dodatku A.2 podamo napotke in težave, ki jih srečujemo pri shranjevanju omrežnih podatkov v datoteke. Opredelimo omrežne datotečne formate in jih med seboj primerjamo. Podamo tudi nekaj informacij o pretvarjanu le-teh.

1.4.1 Notacija

V disertaciji od bralca pričakujemo, da razume osnovne pojme, ki se uporabljajo pri opisih algoritmov, osnove logike, programiranja, računalniške grafike in splošno terminologijo v računalništvu in v analizi omrežij. Ta razdelek je namenjen zgolj vpeljavi notacije, ki se uporablja v besedilu.
Vpeljali bomo nekaj pogosto uporabljenih simbolov:

- $\exists$ je okrajšava za "obstaja",
- $\forall$ je okrajšava za "za vsak",
- $|\mathbf{A}|$ je okrajšava za moč množice, t.j. številnost množice $\mathbf{A}$,
- $\mathscr{P}(\mathbf{A})$ je okrajšava za množico vseh možnih množic elementov množice $\mathbf{A}$, t.j. potenčna množica množice $\mathbf{A}$,

1.4.2 Definicije

Na mnogih mestih v disertaciji omenjamo strukture, s katerimi so predstavljeni podatki v analizi omrežij. Tu navajamo njihove definicije.

Graf

Graf je urejen par $\mathbf{G} = (\mathbf{V}, \mathbf{L})$, kjer je $\mathbf{V}$ množica vozlišč in $\mathbf{L}$ množica povezav – parov elementov množice $\mathbf{V}$. $\mathbf{V}$ je množica vozlišč in $\mathbf{L} = \mathbf{A} \cup \mathbf{E}$ je množica povezav, sestavljena iz usmerjenih povezav $\mathbf{A}$ in neusmerjenih povezav $\mathbf{E}$. Velikosti množic navadno označimo z $n = |\mathbf{V}|$ in $m = |\mathbf{L}|$.

Omrežje

Omrežje je matematična struktura, ki jo sestavljata graf (1.4.2) in množica podatkov, dodeljenih vozliščem in povezavam grafa.

Formalno je omrežje podano z $\mathbf{N} = (\mathbf{V}, \mathbf{L}, \mathbf{P}, \mathbf{W})$, kjer je $\mathbf{G} = (\mathbf{V}, \mathbf{L})$ graf, $\mathbf{V}$ je množica vozlišč in $\mathbf{L} = \mathbf{A} \cup \mathbf{E}$ je množica povezav. Sestavljena je iz usmerjenih povezav $\mathbf{A}$ in neusmerjenih povezav $\mathbf{E}$. Velikosti množic navadno označimo z $n = |\mathbf{V}|$ in $m = |\mathbf{L}|$, $\mathbf{P}$ je množica funkcij vozlišč oz. njihovih lastnosti $p : \mathbf{V} \to \mathbf{A}$, $\mathbf{W}$ pa je množica funkcij povezav oz. njihovih uteži $w : \mathbf{L} \to \mathbf{B}$. Lastnosti vozlišč $\mathbf{P}$ in lastnosti povezav $\mathbf{W}$ lahko merimo v različnih lestvicah (v številskih, urejenostnih ali v imenskih).

Dvodelno omrežje

Dvodelno omrežje [3] definiramo z $\mathbf{N} = ((\mathbf{V}_1, \mathbf{V}_2), \mathbf{L}, \mathbf{P}, \mathbf{W})$. Množica vozlišč sestoji iz dveh tujih podmnožic $\mathbf{V}_1$ in $\mathbf{V}_2$: $\mathbf{V} = \mathbf{V}_1 \cup \mathbf{V}_2$, povezave v množici $\mathbf{L}$ pa morajo imeti eno krajišče v $\mathbf{V}_1$, drugo pa v $\mathbf{V}_2$. Množico vozlišč je pri njih torej možno ločiti na dve tuji podmnožici, tako da vse povezave potekajo izključno med njima.

Za primer enega izmed mnogih dvodelnih omrežij, s katerimi imamo opravka v poglavju 2, izpostavimo omrežje soavtorstev WA. Gre za omrežje na dveh tujih podmnožicah vozlišč – del ($\mathbf{W}$) in avtorjev ($\mathbf{A}$). Vsaka (usmerjena) povezava povezuje delo in enega od njegovih avtorjev. Omrežje WA je tehnično matrika del × avtorjev, ki ima ne-ničelne vrednosti (1) za vsakega izbranega avtorja (stolpec), če in samo če je on avtor izbranega dela (vrstica). Tudi v poglavju A.2.7 srečamo primer dvodelnega omrežja, omrežja omrežnih datotečnih formatov × orodij, ki omogočajo delo z njimi.

Časovno omrežje

Časovno omrežje [3] definiramo z $\mathbf{N}_T = (\mathbf{V}, \mathbf{L}, \mathbf{P}, \mathbf{W}, \mathbf{T})$, ki se od osnovne definicije (1.4.2) razlikuje v vpeljavi časa $\mathbf{T}$. $\mathbf{T}$ je množica časovnih točk. V časovnih omrežjih so vozlišča $v \in \mathbf{V}$ in povezave $l \in \mathbf{L}$ lahko prisotne oz. aktivne le v izbranih časovnih točkah. Če je povezava $l(u, v)$ aktivna ob času t, morata biti takrat hkrati aktivni tudi njeni krajišči u in v. Omrežje, ki ob danem času $t \in \mathbf{T}$ sestoji iz aktivnih povezav in vozlišč, označimo z $\mathbf{N}(t)$. Imenujemo ga časovna rezina ob času t. Pojem časovne rezine lahko razširimo na časovni interval med dvema časoma.

Skupina, razbitje, pokritje, razvrstitev, preureditev

Neprazno podmnožico $\mathbf{C} \subseteq \mathbf{V}$ imenujemo skupina (združba, gruča), neprazno množico skupin $\mathscr{C} = \{\mathbf{C}_i\}$ pa razvrstitev.
Razvrstitev je polna ali pokritje, če in samo če $\cup\mathscr{C} = \cup_i \mathbf{C}_i = \mathbf{V}$.
Razvrstitev $\mathscr{C} = \{\mathbf{C}_i\}$ je razbitje natanko tedaj, ko je pokritje in velja $\forall i \neq j : \mathbf{C}_i \cap \mathbf{C}_j = \varnothing$.
Razvrstitev $\mathscr{C} = \{\mathbf{C}_i\}$ je preureditev (permutacija), če in samo če je $\mathscr{C}$ razbitje in $|\mathbf{C}_i| = 1$.

Hierarhija

Polna razvrstitev $\mathscr{H}$ je osnovna hierarhija natanko tedaj, ko velja $\forall v \in \cup\mathscr{H} : \{v\} \in \mathscr{H}$. Relacija podmnožice $\subset$ namreč množico $\mathscr{H}$ strogo delno ureja in hkrati na razvrstitvi $\mathscr{H}$ porodi povezan aciklični graf njenih elementov $\mathbf{C}_i \in \mathscr{H}$, ki so med seboj usmerjeno povezani natanko tedaj, ko so v relaciji $\mathbf{C}_i \subset \mathbf{C}_j$.
Polna razvrstitev $\mathscr{H}$ je drevesna hierarhija, če in samo če velja $\forall i \neq j : \mathbf{C}_i, \mathbf{C}_j \in \mathscr{H} \wedge \mathbf{C}_i \cap \mathbf{C}_j \in \{\varnothing, \mathbf{C}_i, \mathbf{C}_j\}$.

V hierarhiji ločimo več vrst vozlišč. Notranja vozlišča z izhodno stopnjo večjo od 0, outdeg > 0, imeujemo vejišča. Končna vozlišča z izhodno stopnjo enako 0, outdeg $= 0$, imenujemo listi. V primeru drevesne hierarhije lahko določimo še korensko vejišče, koren ali vrh. To vozlišče ima vhodno stopnjo 0, indeg $= 0$. V splošnih hierarhijah je takšnih vozlišč lahko več, a v tem primeru taka vozlišča raje imenujemo izvori.

Hipergraf

Hipergraf [4] $\mathbf{H}$ je par $\mathbf{H} = (\mathbf{V}, \mathbf{E}_h)$ kjer $\mathbf{V}$ predstavlja množico vozlišč, $\mathbf{E}_h$ pa množico nepraznih podmnožic množice $\mathbf{V}$, ki jih imenujemo hiperpovezave oz. preprosto povezave. Množica $\mathbf{E}_h$ je podmnožica potenčne množice $\mathscr{P}(\mathbf{V})$ množice vozlišč $\mathbf{V}$, a brez prazne množice: $\mathscr{P}(\mathbf{V}) \setminus \{\varnothing\}$.

2

Analiza bibliografskih omrežij objav iz področja topoloških indeksov

2.1 Topološki indeksi in scientometrika

Prvi topološki indeks [5] imenovan *path number* [6], kasneje znan pod imenom *Wienerjev index* je bil predlagan že v letu 1947. Bolj konkretni začetki v znanosti topoloških indeksov se porajajo kasneje, in sicer v šestdesetih letih. Po dveh desetletjih je bil objavljen tudi prvi pregledni članek iz področja [7]. Zdaj lahko rečemo, da je področje zrelo in dobro uveljavljeno. Številne raziskovalne ustanove delujejo na tem področju. Področje topoloških indeksov se uvršča v matematično kemijo, molekularno topologijo in v aplikativno teorijo grafov v kemiji. Prva knjiga o aplikaciji teorije grafov v kemiji je izšla v letu 1976 [8]. Pod pojmom topološki indeks razumemo molekularni opisnik [9–11], ki predpisuje, kako lahko na podlagi molekularnega grafa kemične spojine izračunamo numerično vrednost – enoličen parameter grafa, ki opredeljuje njegovo topologijo [9–11]. Pomembnejši primeri topoloških indeksov so *Hosoyin indeks* [12] ali preprosto *topološki indeks*, že omenjeni *Wienerjev indeks*, *Randićev molekularni vezni indeks* [13], *Balaban-ov J indeks* [14] in drugi. V delu *R. Todeschini*-ja in sodelavcev [15] je na obsežnem naboru kemičnih spojin obrazložen in ovrednoten kemijski pomen mnogih topoloških indeksov.

Scientometrika je veda o merjenju in analizi znanosti kot take. Pretežno temelji na bibliometričnih postopkih in se osredotoča na analizo znanstvenih objav. Splošnejša dela v scientometriki obravnavajo različne vrste analiz citiranj, iskanje različnih vzorcev npr. kako znanstveniki medsebojno citirajo svoja dela in kako sodelujejo. Z uporabo analize citiranj in analize omrežij lahko odkrijemo omrežja znanstvenikov in njihovega sporočanja, relacij med njimi in razvoja znanstvenih področij skozi čas. Ker na področju topoloških indeksov takšna analiza še ni bila opravljena, jo bomo na tem mestu skupaj z izsledki predstavili.

Bibliografske podatke smo pridobili z uporabo storitve *Web of Science* [16]. Izvorne podatke smo pretvorili v množico bibliografskih omrežij in jih analizirali. *Web of Science* je spletna bibliografska in akademska podatkovna storitev, ki jo zagotavlja *Thomson Reuters*. Storitev nam omogoča dostop do podatkovnih baz in je namenjena interdisciplinarnemu raziskovanju in poglobljenemu raziskovanju specializiranih pod-področij znotraj akademskih ali znanstvenih strok. Indeksira večino pomembnih znanstvenih del, predvsem tistih, ki so bila objavljena po letu 1975.

V razdelku 2.2 predstavimo podobne bibliometrične raziskave omrežij na področju topološkim indeksom sorodnih tem. V razdelku 2.3 opišemo kako smo bibliografske

podatke pripravili in jih uredili v omrežja in uporabili v analizah. V razdelku 2.4 opišemo, kako smo podatke analizirali in predstavimo izbrane rezultate. V razdelku 2.5 razpravljamo o alternativnih pogledih na delo in predstavimo nekaj odprtih tem.

2.2 Sorodna dela

V povezavi s kemijo je splošneje gledano objavljenih precej scientometričnih raziskav. *J. M. Modak* je s sodelavci [17] analiziral scientometrične parametre objav kemijskega inženirstva. Geografsko po državah in ustanovah so primerjali števila objav v izbranih revijah in števila citatov. *P. Willett* [18] je na primer pregledal članke iz zvezkov 2 – 24 v reviji *Journal of Molecular Graphics and Modelling*. Osredotočil se je na spremembe tematike v aktualnem časovnem obdobju, na najbolj produktivne ter najbolj citirane avtorje in na ustanove, kjer so dela nastajala. V svojem drugem delu [19] je pregledal članke iz zvezkov 4 – 27 revije *Quantitative Structure-Activity Relationships and of QSAR & Combinatorial Science*. Raziskoval je objave člankov v revijah, citate teh člankov, najbolj produktivne avtorje in njihove države ter povezave med revijami in širšo kemijsko literaturo. *N. M. Builova* je s sodelavci [20] preiskovala članke v povezavi z nano-energijo in sorodnimi postopki. Za obdobje od 2000 do 2012 so opazovali ključna področja. Za najpomembnejše objave so predstavili indekse citiranja, geografski izvor avtorjev in potrebe po izdelkih v povezavi z nano-energijo. Za posamezne države so navedli področja rasti ter analizirali zanimanje za znanost na sorodnih področjih. *K. W. Boyack* in sodelavci [21] so na velikih bibliografskih množicah 30-letnega časovnega okvira s kemijo povezanih člankov uporabili bibliometrične in scientometrične pristope v želji, da bi odkrili podrobnosti v razvoju kemije in sorodnih znanostih kot so biologija, biokemija in bioinženirstvo. Bibliometrične postopke sta uporabila tudi *M. R. Davarpanah* in *S. Aslekia* [22]. Izvedla sta poglobljeno kvantitativno študijo produktivnosti, karakteristik globalnega publiciranja in drugih dejavnosti v knjižničarstvu in informacijskih znanostih.

Večina analiz podatkov in predvsem njihov izvor se v izpostavljenih objavah precej razlikuje od našega dela. Tehnično sorodno, a manj obsežno bibliometrično analizo podatkov iz *Web of Science* je opravila *N. Kejžar* s sodelavci [23]. Osredotočili so se na področje razvrščanja in klasifikacije (*Clustering and Classification*). Za zadnjih nekaj desetletij so z uporabo postopkov analize omrežij identificirali pomembne avtorje, dela in teme dotičnega področja. V tehničnem smislu podobno delo je opravil tudi *A. Ahmed* s sodelavci [24]. Analizirali so spletno bazo filmov *Internet Movie Databa-*

se. Predstavili so študijo za prikazovanje in analizo velikih in kompleksnih časovnih dvodelnih omrežij [3]. Z uporabo analize omrežij, med drugim z otoki za posamezne časovne rezine, so identificirali pomensko pomembna podomrežja. Znan članek na temo analize bibliografskih omrežij je objavil *M. Newman* [25]. V omrežjih sodelovanj s področja fizike, biomedicinskih raziskav in računalniških znanosti je preiskoval različne statistične lastnosti. Izvedel je analizo obstoja velikih skupin povezanih znanstvenikov, njihove velikosti in stopnje gručenja v omrežjih. Takšno analizo smo izvedli tudi v naši raziskavi. *J. Moody* [26] je preizkušal tri sociološke modele omrežij sodelovanj in ugotovil, da njihova strukturno skladna jedra v stroki določajo omrežja soavtorstev. *V. Batagelj* in *M. Cerinšek* [27] sta predlagala pristop kako z množenjem ustreznih omrežij pridobiti izpeljana omrežja in razpravljata o normalizaciji takšnih omrežij. Izpostavila sta problem ohranitve redkosti v izpeljanih omrežjih. Metodo sta uporabila na množici bibliografskih omrežij iz področja socialnih omrežij (*"social network"*), ki sta jih sestavila iz podatkov, dobljenih iz storitve *Web of Science*.

2.3 Gradnja bibliografskih omrežij

V skladu s postopkom, ki ga opisujemo v dodatku A.1, smo 15. decembra, 2011 preko spletne storitve *Web of Science* zbrali podatke za potrebe analiz. Iskali smo z iskalnim nizom ``topological indices'' OR ``topological index*'' in našli 2036 ustreznih primerkov. Umestili smo jih v t.i. "bazno" množico **B**. Množice so prikazane na sliki A.1. Na podlagi množice **B** smo pripravili še množico "uporabnikov" **U**, ki v našem primeru vsebuje 10.814 del. Poiskali smo tudi "samo citirana" dela iz množice del **B**. Umestimo jih v množico **C**. Pri snovanju množice **C** smo morali pripraviti 5.796 iskalnih nizov. 424 citiranih del v *Web of Science* nismo našli, ker so prestara, preostalih 880 del pa je monografskih, ki jih *Web of Science* prav tako ne indeksira. Za preostalih 4.492 citiranih del smo pripravili iskalne nize in dobljeno množico del programsko prečistili. Ostalo nam je 2.284 opisov del, kar predstavlja približno polovico (50,9%) vseh del, citiranih iz množice **B**. To pomeni, da je v *Web of Science* zavedenih približno polovico del na temo topoloških indeksov. 281.545 opisov (oz. samo imen) del pripada naši množici **D**. Zaradi obsežne naloge priprave množice **C** smo že v začetku ohranili le dela, ki so iz **B** citirana vsaj dvakrat. Postopek izdelave množic del je nekoliko bolj natančno opisan na spletu v dodatku [28]. Po redukciji opisanih množic del v povezavi z robnim problemom (razdelek 2.3.3) bi bila ta dela tako ali tako odstranjena. V množici **D** nam je ostalo 3.512 opisov. Množice del **E**

Tabela 2.1

Število entitet v posameznih bibliografskih omrežjih.

Entiteta	Število
Dela	294.939
Avtorji	120.953
Revije	18.566
Ključne besede	56.671
Polni opisi del	15.134
Odstranjeni duplikati	1.741

iz dveh razlogov nismo analizirali, konstruirali ali iskali. Kot smo zapisali v dodatku (A.1.1), je množica **U** običajno precej večja od množice **B**, v našem primeru približno petkrat. Za gradnjo množice **E** bi bili na podlagi množice **U** tako primorani pripraviti bistveno več iskalnih nizov in nato več časa iskati. Dela iz množice **E** bi hkrati lahko zasenčila rezultate, ki nam jih izkazujejo dela iz drugih množic. Še najbolje bi jih bilo analizirati ločeno. Za potrebe naših analiz smo jih pustili v obliki nerazločljive podmnožice množice **D**. Na rezultate zato posebnega vpliva nimajo.

Nabor bibliografskih vsebin smo pripravili z orodjem *Wos2Pajek*. Opisani so v dodatku, v tabeli A.1. Po čiščenju duplikatov in napak v koraku *Wos2Pajek* (A.1.2) je v naši množici **B** od izvornih 2.037 ostalo 2.036 opisov. V množici **U** nam je ostalo 9.074 od 10.814 opisov, v množicah **C** in **D** pa je ostalo vseh 2.284 oz. 3.512 opisov. Razsežnosti podatkov so prikazane v tabeli 2.1.

2.3.1 Omejitve pri pripravi množice citiranih del

Zaradi narave priprave nabora podatkov, natančneje opisane v dodatku [28], množica citiranih del **C** ne vsebuje opisov del, ki so citirana le enkrat ali dvakrat – množica **C′**. Ker opisov množice **C′** nimamo, ne moremo vedeti ali katero delo od tam povratno citira kakšno delo iz **B** ali **U**. Slednje se v množici **C′** izraža tako, da imajo dela namesto pravilne večje izhodne stopnje, le-to enako nič (0). Ne glede na opisano pričakujemo malo takšnih del v **C′**, če sploh katerega, ki bi nazaj citirala dela v **B** ali **U**.

2.3.2 Osnovne lastnosti bazne množice del

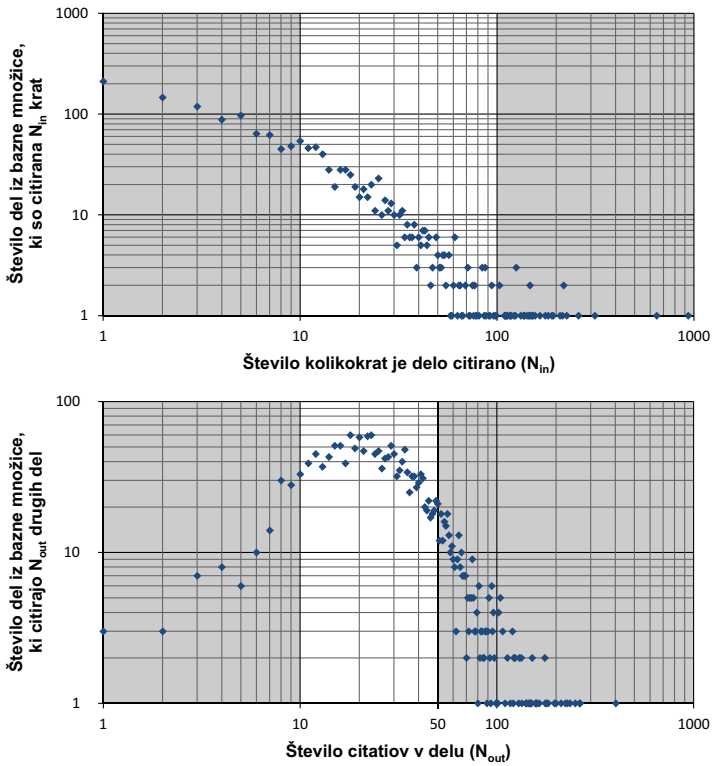
Poglejmo najprej, kako je s porazdelitvijo citiranj – vhodna stopnja v omrežju citiranj *Cite* (tabela A.1). Podana je na zgornjem prikazu na sliki 2.1. Porazdelitev citiranj del po pričakovanjih približno sledi t.i. potenčnemu zakonu (*power-law*) za katerega je v

log-log skali značilna premica. Porazdelitev po potenčnem zakonu je značilna za brezlestvična (*scale-free*) omrežja [29]. Na spodnjem delu slike 2.1 je podana porazdelitev števila citiranj v posameznem delu – izhodna stopnja del v omrežju citiranj *Cite*. O množici citiranih del **C** je zgoraj (2.3.1) navedena pomembna podrobnost. Večina del (67,5%) ni citirana več kot deset krat, $N_{in} \leq 10$. Gre za dela, ki so na zgornjem prikazu (vhodna stopnja) osenčena. Najbolj citiranih 1,9% del (desno osenčeno področje na prikazu vhodnih stopenj) je citiranih več kot sto krat, $N_{in} > 100$. V povprečju je delo citirano $N_{in} = 14,0$ krat. Z levo osenčenim področjem na spodnjem prikazu je predstavljenih 8,3% del z najmanjšimi števili citiranj. Preostala dela imajo največ deset citiranj, $N_{out} \leq 10$. Zadnjih 17,9% del z največjim številom citiranj citira več kot petdeset del, $N_{out} > 50$. Predstavljena so z desnim osenčenim področjem na spodnjem prikazu. Srednjih 73,8% del (svetlo področje na spodnjem prikazu) ima v povprečju 27,5 citiranj drugih del. Skupno povprečno število citiranj v delu je $N_{out} = 35,1$.

Deset najbolj citiranih del iz množice **B**, vidnih na skrajnem desnem delu zgornjega prikaza na sliki 2.1 je navedenih v tabeli 2.2. Najbolj citirana dela na področju so v splošnem bodisi najpomembnejši članki ali pa pregledni članki. V tabeli 2.2 najdemo predstavnike obeh skupin. Naj izpostavimo dve najbolj citirani deli, ki sicer nista navedeni v tabeli 2.2. Gre za *Wiener*-jevo delo [6], ki je v bazni množici **B** citirano kar 1.531 krat in *Randić*-evo delo [13], skupno citirano 1.436 krat v delih iz množice **B**. Del v tabeli ni, ker ju v *Web of Science* ni moč najti z našim iskalnim izrazom. Njuna avtorja termina "topološki indeks" še nista uporabljala.

2.3.3 *Robni problem*

Po konstrukciji bibliografskih omrežij smo iz njih odstranili "šum" in ohranili dela, ki so za področje "značilna". Problem izbora del, t.i. akterjev, ki naj bi bila za analizo pomembna, poznamo pod imenom robni problem v omrežjih [30]. Lastnosti robnega problema so razložene v delih *Kossinets*-a in *Marschall*-a [31, 32]. Iz omrežij smo odstranili dela v, ki nimajo citiranj in so citirana manj kot k krat: $(0 < \text{indeg}(v) < k) \wedge (\text{outdeg}(v) = 0)$. Takšna dela večinoma nimajo vpliva na izbrano znanstveno področje. k navadno izberemo [33] na mestu, kjer na prikazu predvidenega števila del za odstranitev v odvisnosti od k zasledimo stopnico. Ker v našem primeru nimamo izrazite stopnice, slika 2.2, smo izbrali vrednost $k = 3$. Izmed 16.906 del smo jih 501 odstranili. Novo omejeno množico del iz množice **D** označimo z **D′**, $|\mathbf{D'}| = 3.011$. Ker se v besedilu večkrat sklicujemo na množico vseh del, razen tistih iz množice **D′**,



Slika 2.1

Porazdelitev po vhodni (zgoraj) in izhodni stopnji vozlišč (spodaj) v omrežju citiranj *Cite* v log-log skali. 440 del v bazni množici ni citiranih v nobenem delu. Kot taka na zgornjem prikazu niso podana. 27 del iz bazne množice nima citatov in niso podana na spodnjem prikazu. Opisi osenčenih področij so v glavnem besedilu.

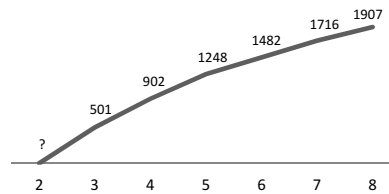
Tabela 2.2

Največkrat citirana dela v bazni množici **B**.

Število citatov dela	Delo
935	HOSOYA_H(1971)44:2332: Topological index – newly proposed quantity characterizing topological nature of structural isomers of saturated hydrocarbons; *Bulletin of the chem. society of Japan*
647	BALABAN_A(1982)89:399: Highly discriminating distance-based topological index; *Chemical physics letters*
314	BALABAN_A(1983)114:21: Topological indexes for structure-activity correlations; *Topics in current chemistry*
259	DOBRYNIN_A(2001)66:211: Wiener index of trees: Theory and applications; *Acta applicandae mathematicae*
226	BALABAN_A(1983)55:199: Topological indexes based on topological distances in molecular graphs; *Pure and applied chemistry*
218	MACCHI_P(2003)238:383: Chemical bonding in transition metal carbonyl clusters: complementary analysis of theoretical and experimental electron densities; *Coordination chemistry reviews*
218	RANDIC_M(1991)31:311: Resolution of ambiguities in structure – property studies by use of orthogonal descriptors; *Journal of chemical information and computer sciences*
215	MIHALIC_Z(1992)69:701: A graph-theoretical approach to structure property relationships; *Journal of chemical education*
209	RANDIC_M(1991)15:517: Orthogonal molecular descriptors; *New journal of chemistry*
193	SCHULTZ_H(1989)29:227: Topological organic-chemistry .1. Graph-theory and topological indexes of alkanes; *Journal of chemical information and computer sciences*

Slika 2.2

Prikaz števila del – na krivulji, ki bi jih odstranili pri izbrani meji k – na abscisi. Vključena so dela iz množice **D**. Ker v začetku izhajamo iz množice del, ki ne vsebuje del, ki so le enkrat citirana, je pri $k = 2$ zapisan vprašaj.



jih bomo označili z **F**. Gre za dela s polnimi opisi $\mathbf{F} = \mathbf{B} \cup \mathbf{U} \cup \mathbf{C}$.

2.4 Analize

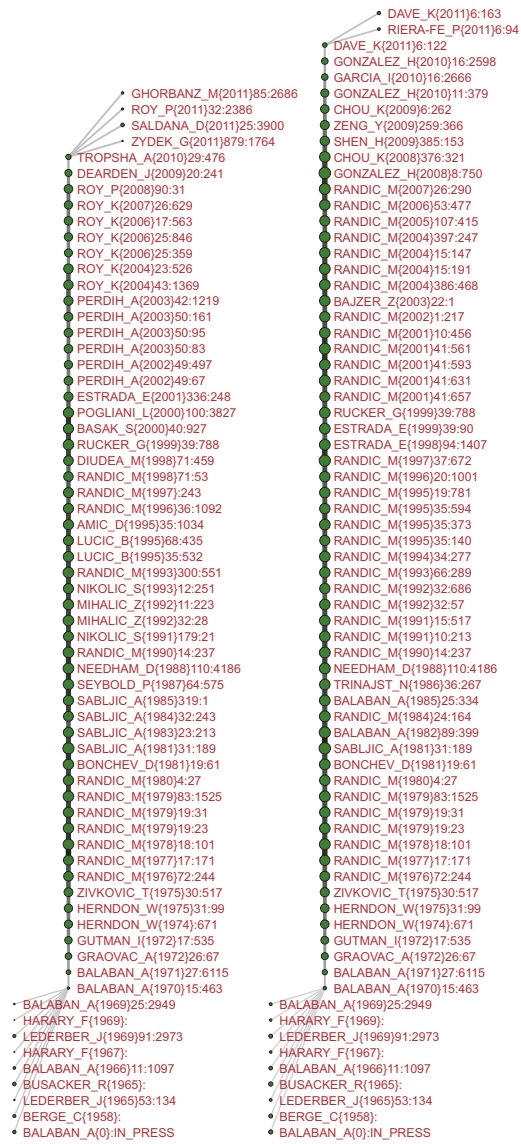
Za izvedbo analiz smo v večji meri uporabljali orodje za analizo omrežij *Pajek* [34, 35]. Zasnovano je za delo z velikimi omrežji, shranjenimi v obliki tekstovnih datotek, o katerih bralec lahko več prebere v poglavju o omrežnih datotečnih formatih A.2.4. Omrežja smo konstruirali z orodjem *Wos2Pajek*. Z vmesnimi rezultati smo pretežno upravljali v *Microsoft Excel*-u in raznih tekstovnih urejevalnikih. Pri nekaterih analizah smo si pomagali z našim programom za analizo velikih omrežij *net.Plexor*, opisanim v poglavju 6. Za nekatera področja analize smo razvili posebne programe. Slike smo risali

bodisi s programom *Pajek*, bodisi v *Excel*-u. Naknadno smo jih uredili s programom *Corel Draw*.

2.4.1 Poti citiranj

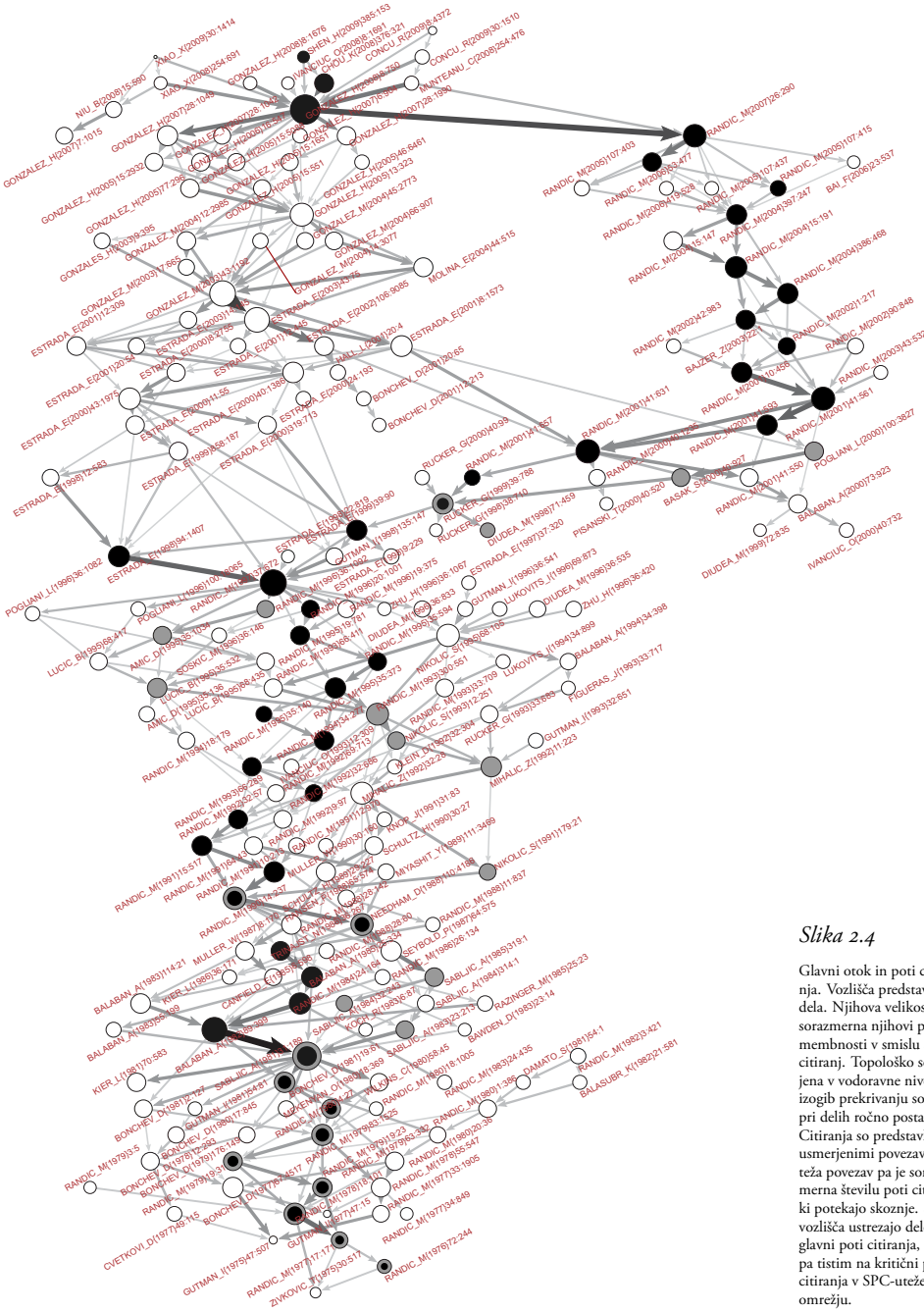
Da bi izpostavili pomembnost posameznih del v razvoju znanosti topoloških indeksov, smo na omrežju citiranj *Cite* uporabili algoritem za štetje iskalnih poti (*search path count*) – SPC [36, 37]. Algoritem vsakemu vozlišču in povezavi v omrežju dodeli število vseh možnih poti od izvora do ponora (SPC uteži). V omrežju citiranj, opremljenim s SPC utežmi, smo izrezali glavno pot citiranja [36]. Prikazana je levo na sliki 2.3. Postopek iskanja glavne poti citiranja se začne z najmočnejšo povezavo v SPC-uteženem omrežju citiranj, ki se ji nato v obeh smereh požrešno dodaja najmočnejše povezave; nazaj proti izvornemu vozlišču in naprej proti ponornemu vozlišču. Če v isti iteraciji obstaja več enakovrednih povezav z enakimi utežmi, se pot razveji na več vej – glavna pot citiranja ni nujno pot v pravem pomenu besede. V SPC-uteženem omrežju *Cite* smo določili glavni otok [38], ki glavno pot citiranja vsebuje. Prikazan je na sliki 2.4. (Kratka razlaga otokov je podana v razdelku 2.4.4.) Vozlišča na slikah glavne poti citiranja in glavnega otoka predstavljajo dela. Njihova velikost je sorazmerna kvadratnemu korenu njihove pomembnosti, izražene v številu citiranj. Tok citiranj predstavljajo usmerjene povezave. Njihova teža in nivo sivine sta sorazmerni kvadratnemu korenu števila poti citiranj, ki potekajo preko njih. Na otoku lahko sledimo delu glavne poti citiranj. Vozlišča na njem so označena s sivo barvo. Glavna pot v otok vstopi pri delu RANDIC_M(1976)72:244 in ga zapusti pri delu POGLIANI_L(2000)100:3827. Pred delom *M. Randić*-a iz leta 1976 je glavna pot pod "gladino" okoli otoka. Kot vidimo na sliki 2.3, glavna pot izvira pri delih, ki so jih objavili kemik *A. T. Balaban*, molekularni biolog in Nobelovec *J. Lederberg* in nekaj matematikov. Zadnji del glavne poti citiranj, t.j. po delu *L. Pogliani*-ja v letu 2000, je prav tako "pod" nivojem otoka in bi bil danes verjetno drugačen, saj so avtorji objavljali nova dela, dela objavljena po zajemu podatkov. Zadnji del glavne poti preprosto vodi do najpozneje objavljenega dela v našem naboru opazovanih del. Otok se pojavi v letu 1975 in se v letu 2001 razdeli na dve močnejši veji. V levi veji ima največji vpliv avtor *M. Randić*, v desni pa *E. Estrada* in *H. Gonzalez-Diaz*. Veji se v letu 2008 ponovno združita.

Ker se glavna pot citiranja ogne zgornjemu delu otoka, smo uporabili še metodo kritične poti (*critical path method* – CPM) [36, 39] s svojim izvorom v teoriji projektnega vodenja. Pogosto se uporablja tudi v bibliografski analizi. Pri projektnem



Slika 2.3

Izolirana glavna pot citiranja levo in kritična pot v SPC-uteženem omrežju na desni. Vozlišča predstavljajo dela. Njihova velikost je sorazmerna njihovi pomembnosti v smislu logaritma števila citiranj. Usmerjene povezave (prikazane brez puščic) predstavljajo najmočnejša citiranja. Teža povezav je sorazmerna logaritmu števila poti citiranj, ki tečejo preko njih.



Slika 2.4

Glavni otok in poti citiranja. Vozlišča predstavljajo dela. Njihova velikost je sorazmerna njihovi pomembnosti v smislu števila citiranj. Topološko so urejena v vodoravne nivoje. V izogib prekrivanju so napisi pri delih ročno postavljeni. Citiranja so predstavljena z usmerjenimi povezavami, teža povezav pa je sorazmerna številu poti citiranj, ki potekajo skoznje. Siva vozlišča ustrezajo delom na glavni poti citiranja, črna pa tistim na kritični poti citiranja v SPC-uteženem omrežju.

vodenju CPM upošteva trajanje posameznih nalog, v analizi omrežij pa analogno upošteva SPC uteži. Vozlišča v otoku, ki ležijo na kritični poti so označena s črno barvo, celotna kritična pot pa je prikazana desno na sliki 2.3. V primerjavi z glavno potjo, kritična pot lepše poteka preko otoka. Na spodnjem delu se popolnoma ujema z glavno potjo in jo zapusti pri delu SABLIJIC_A(1981)31:189. V sredinskem delu otoka se poti prepletata. Medtem ko glavna pot otok zapusti v sredinskem delu, kritična pot sledi desni veji vse do vrha otoka. Dela, ki ustrezata obema potema hkrati, smo označili s koncentričnima krogoma v črni in sivi. Le kritična pot je prisotna na celotnem otoku in hkrati pod nivo "gladine" njegove okolice ne potone.

Za dela na glavni poti smo naredili analizo ključnih besed. Ključne besede smo vzeli iz naslovov, povzetkov in seznamov ključnih besed posameznih del. Iz drsečega povprečja ključnih besed treh zaporednih del na glavni poti smo skušali določiti pod-področja. Sezname ključnih besed smo uredili po številu pojavitev posamezne ključne besede. Seznamov zaradi velikosti tu ne navajamo, lahko pa jih bralec najde v dodatku [28]. Identificirali smo sledeča pod-področja: v začetku, do leta 1975 so se znanstveniki pretežno ukvarjali z molekularno teorijo grafov. Do leta 1980 so sledile študije molekularne strukture in kemičnih lastnosti molekul, pojavil se je termin "topološki indeks". V začetku devetdesetih je v ospredje stopilo področje QSAR (*quantitative structure-activity relationships*), v sredini desetletja pa študije topoloških opisnikov. Razvili so napovedovalne modele [10]. Z analizo smo tu zaključili, saj glavna pot tu zapusti glavni veji otoka, slika 2.4. Kljub temu lahko sklepamo, da vsaka veja predstavlja neko pod-področje. Veji se v letu 2000 spet združita. Na kritični poti nismo opravili analize.

2.4.2 *Normalizirano število del*

Iz omrežja $AW = WA^T$ (tabela A.1) smo izračunali, koliko del w je posamezni avtor objavil. Večina avtorjev svoje delo objavlja v sodelovanju z drugimi avtorji. Zanimal nas je njihov individualni doprinos. Za vsako delo v AW smo normalizirali število avtorjev in dobili normalizirano omrežje AW_N. Pri tem smo predpostavili, da v primeru, da delo objavi k soavtorjev, vsak izmed njih prispeva enak delež $1/k$. Če seštejemo vse deleže doprinosov vsakega avtorja dobimo ekvivalentno število celih del, ki jih je posamezni avtor objavil. V tabeli 2.3 so na podlagi množice $\mathbf{B}$, upoštevajoč število del ($w_{\mathbf{B}}$), v padajočem vrstnem redu prikazana ekvivalentna števila del posameznih avtorjev. Pričujočo vrsto normalizacije bibliografskih omrežij sta predlagala *V. Batagelj* in

Tabela 2.3

Avtorji z največjim doprinosom. Doprinos je izražen s številoma $w_{\mathbf{B}}$ in $w_{\mathbf{F}}$ celih del v okviru množic **B** in **F**. Najboljših 16 izmed 18 avtorjev v ozir $w_{\mathbf{F}}$ je zapisanih z rdečo (razdelek 2.4.2). Pomen imen v oklepajih je razložen v dodatku A.1.3.

$w_{\mathbf{B}}$	$w_{\mathbf{F}}$	Avtor	$w_{\mathbf{B}}$	$w_{\mathbf{F}}$	Avtor
36,9	52,4	Ovidiu Ivanciuc	14,6	51,1	Ali Reza Ashrafi
35,0	147,5	Ivan Gutman	13,7	21,2	Andrey A. Dobrynin
34,0	72,9	Alexandru T. Balaban	13,0	29,9	Istvan Lukovits
33,1	146,5	Milan Randić	11,0	15,3	Denise Mills
31,7	41,5	Alina Pyka	11,0	11,0	Anton Perdih
29,6	44,2	P. V. Khadikar	10,8	30,1	Modjtaba Ghorbani
23,4	43,0	Subhash C. Basak	9,5	21,0	H. Gonzalez-Diaz
22,2	51,2	Ernesto Estrada	9,4	44,5	Francisco Torrens
20,4	57,2	Mircea V. Diudea	9,3	19,3	Sonja Nikolić
20,0	68,5	Nenad Trinajstić	9,3	39,6	Lionello Pogliani
17,0	27,8	Anil Kumar Madan	9,2	10,4	Hao Jun-Feng
16,9	39,6	Haruo Hosoya	9,1	38,5	Danail Bonchev
16,2	16,2	(Avat A. Taherpour)	8,9	42,6	Bo Zhou
14,8	29,0	(Luo-Nan Xu)	8,4	16,2	Nenad Raos
14,6	23,2	Vijay K. Agrawal	7,6	27,4	Kunal Roy

M. Cerinšek [27]. Šestnajst od devetnajstih avtorjev z največjim ekvivalentnim številom del ($w_{\mathbf{F}}$), izračunanih na podlagi množice **F**, t.j. celotne množice del, razen tistih iz $\mathbf{D}'$, je v tabeli 2.3 izpisanih z rdečo. Manjkajoča avtorja sta *K. C. Chou* ($w_{\mathbf{F}} = 63.2$) in *L. B. Kier* ($w_{\mathbf{F}} = 42.2$). V analizi ne upoštevamo del iz $\mathbf{D}'$, saj za dela v $\mathbf{D}'$ poznamo le prvega avtorja. Pri delih z več avtorji iz množice $\mathbf{D}'$ bi tako prvim avtorjem pripisali preveč zaslug.

Pomen rdečega besedila v tabelah

Razložimo pomen rdečega besedila v tabelah na konkretnem primeru. Tabela 2.3 prikazuje 30 najboljših avtorjev glede na $w_{\mathbf{B}}$. V želji, da bi prikazali tudi najboljše avtorje glede na $w_{\mathbf{F}}$ smo najboljše izmed njih zapisali z rdečo. Ker avtorja, ki visoko kotirata (peti in štirinajsti glede na $w_{\mathbf{F}}$), manjkata v tabeli, ju individualno navajamo. Manjkata preprosto zato, ker glede na $w_{\mathbf{B}}$ nista uvrščena med 30 najboljših. Glede na $w_{\mathbf{F}}$ ostali avtorji precej močneje zaostajajo.

2.4.3 *Analiza podpore avtorjem – kako avtorji citirajo druge avtorje*

V razdelku se v smislu normaliziranega ekvivalentnega števila celih del t osredotočimo na analizo kolikokrat posamezni avtor citira drugega avtorja. Predpostavimo, da imamo dve deli W_a in W_b kjer W_a citira W_b. Avtorji v W_a naj bodo $a_1, a_2, .., a_n$ in v W_b

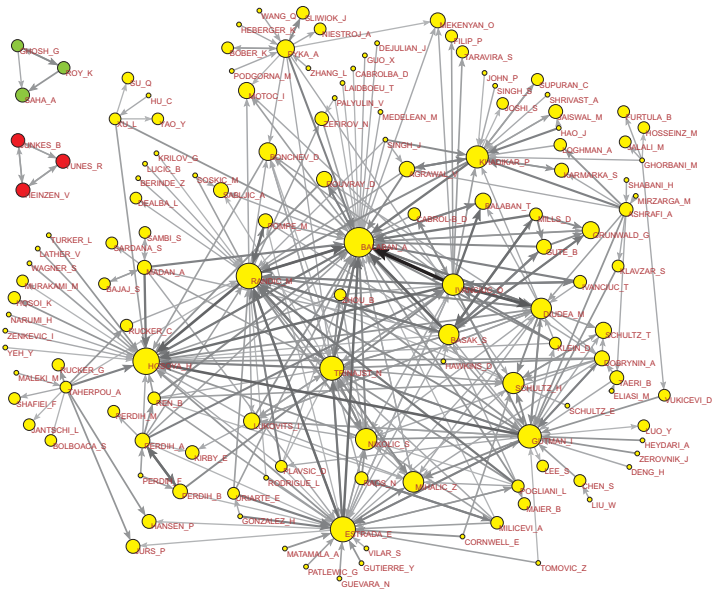
Tabela 2.4

Avtorji, ki najbolj izrazito citirajo drugega avtorja. $t_{\mathbf{F}}$ je število kolikokrat, upoštevajoč množico **F**, dani avtor citira drugega avtorja. Za $t_{\mathbf{B}}$ izhajamo iz množice **B**. Vrstice, ki ustrezajo petim od šest največjih vrednosti $t_{\mathbf{B}}$, so zapisane v rdeči. Zapis z rdečo je razložen na primeru v razdelku 2.4.2.

Avtor	Citirani avtor	$t_{\mathbf{F}}$	$t_{\mathbf{B}}$
Nenad Trinajstić	Milan Randić	151,5	20,5
Alexandru T. Balaban	Milan Randić	150,7	11,6
Ovidiu Ivanciuc	Alexandru T. Balaban	131,2	120,1
Ernesto Estrada	Milan Randić	123,1	24,2
Ivan Gutman	Haruo Hosoya	110,2	29,3
Lionello Pogliani	Milan Randić	109,6	16,5
Milan Randić	Alexandru T. Balaban	105,0	35,6
Milan Randić	Haruo Hosoya	103,1	32,2
Subhash C. Basak	Milan Randić	103,0	17,0
Milan Randić	Nenad Trinajstić	101,0	12,6
Ivan Gutman	Milan Randić	85,1	9,1
Ovidiu Ivanciuc	Mircea V. Diudea	82,6	48,5
Lowell H. Hall	Lemont B. Kier	81,3	0,0
Humberto González-Díaz	Kuo-Chen Chou	69,4	0,0
Dejan Plavšić	Milan Randić	68,8	12,5
Danail G. Bonchev	Milan Randić	63,5	2,6
Hong-Bin Shen	Kuo-Chen Chou	63,1	0,0
Xuan Xiao	Kuo-Chen Chou	63,1	0,0
Mircea V. Diudea	Milan Randić	61,7	10,0
Ali Reza Ashrafi	Mircea V. Diudea	61,3	5,0

$b_1, b_2, .., b_m$. Par avtorjev (a_i, b_j), $a_i \neq a_j$, naj pomeni, da avtor a_i citira avtorja b_j natanko $1/(n \cdot m)$ krat. Če na podatkih seštejemo ekvivalente citiranj za vse pare avtorjev za vse pare povezanih del dobimo rezultate, zapisane v tabeli 2.4. Tabela predstavlja pare avtorjev, ki druge avtorje citirajo najbolj. Števili $t_{\mathbf{F}}$ in $t_{\mathbf{B}}$ izračunani na podlagi množic **F** in **B** odražata kolikokrat izbrani avtor citira drugega avtorja. Pet od šestih najizrazitejših parov na podlagi množice **B** je zapisanih z rdečo. Manjkajoči par predstavljata avtorja (*S. C. Basak*, *A. T. Balaban*) s $t_{\mathbf{B}} = 33,0$. Ideja zapisa z rdečo barvo je razložena na primeru v razdelku 2.4.2.

Tri najizrazitejše skupine 135 avtorjev, določene na podlagi množice **B**, so prikazane na sliki 2.5. Skupine smo določili z uporabo algoritma za posplošene sredice (*generalized cores*) [40] pri meji 2, 2. Velikost vsakega vozlišča – avtorja je sorazmerna vsoti števil citiranj – vhodnih usmerjenih povezav. Ker porazdelitev števil citiranj sledi potenčnemu zakonu je smiselno prikazovati logaritmirane vrednosti citiranj (vrednosti povezav). Širina in odtenek usmerjenih povezav sta zato prikazani v sorazmerju z logaritmom števila citiranj. Gosto povezan del največje sredice, ki vsebuje avtorje s stopnjami najmanj deset, smo na sliki 2.6 prikazali še v matrični obliki. Vidimo lahko



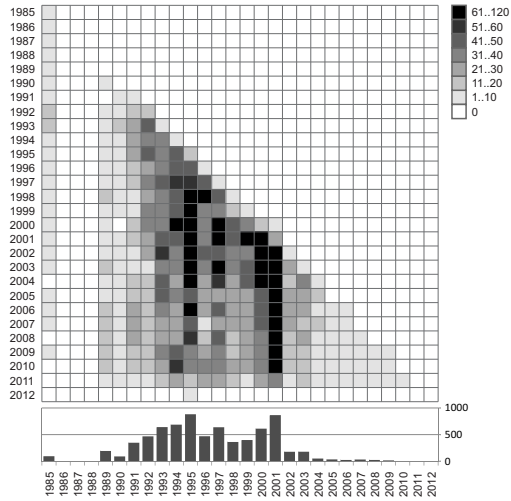
Slika 2.5

Posplošene sredice najbolj izrazitih avtorjev v omrežju citiranj (izbrani prag je 2,2) na množici **B**. Velikost posameznega vozlišča – avtorja je sorazmerna vsoti števil njegovih citatov. Vrednosti povezav so prikazane z lestvico sivin, le-ta pa je sorazmerna logaritmu dejanskega števila citiranj. Centralni del največje sredice je v matrični obliki prikazan tudi na sliki 2.6.

skupino avtorjev C_2, ki pretežno citirajo drug drugega in avtorje v skupini C_3. Avtorji skupine C_2 imajo precej citatov v skupini C_1, avtorji slednje pa se redko citirajo med seboj. Avtorji skupine C_3 so neodvisni od avtorjev ostalih skupin. Še več; neodvisni so znotraj lastne skupine. Citirajo jih avtorji skupine C_1, še bolj pa avtorji iz C_2. Posebej izpostavimo avtorja *H. Hosoya* v zadnjem stolpcu. Mnogo avtorjev ga citira, sam pa redko citira druge avtorje. Tudi avtorji *A. T. Balaban*, *N. Trinajstić* in *M. Randić* so pogosto citirani.

Analiza samo-citiranja

Poseben primer podpore avtorjem je samo-citiranje, t.j. podpora, ki jo avtor namenja samemu sebi oz. ko avtor a_i citira delo, katerega avtor je delno ali v celoti: $b_j = a_i$. Par avtorjev se v tem primeru izrodi v zanko (a_i, a_i). Zanimiv podatek o avtorju je kolikokrat le-ta v povprečju citira svoja dela. V tabeli 2.5 so navedeni podatki za v tem pogledu najbolj izrazite avtorje. $s_{\mathbf{F}}$ predstavlja ekvivalentno število samo-citiranj glede na množico **F**. $w_{\mathbf{F}}$ je število del. Enak izračun smo uporabili v okviru množice **B** in dobili sorodna podatka $s_{\mathbf{B}}$ in $w_{\mathbf{B}}$.



Slika 2.6

Centralni del največje posplošene sredice najbolj izrazitih avtorjev v omrežju citiranj (prag je $\mathbf{2,2}$) v množici $\mathbf{B}$. Avtorji s stopnjami manj kot 10 niso prikazani. Avtorji na levi citirajo avtorje spodaj. Odtenek posamezne celice v matriki je sorazmeren logaritmu števila citiranj. Celotna posplošena sredica je prikazana na sliki 2.5.

Tabela 2.5

Avtorji, ki najbolj citirajo lastna dela. $s_{\mathbf{F}}$ je normalizirano število samo-citiranj na množici $\mathbf{F}$. $w_{\mathbf{F}}$ je normalizirano število del na množici $\mathbf{F}$; $s_{\mathbf{B}}$ in $w_{\mathbf{B}}$ sta analogni vrednosti, izračunani na podlagi množice $\mathbf{B}$.

$s_{\mathbf{F}}$	$w_{\mathbf{F}}$	$s_{\mathbf{B}}$	$w_{\mathbf{B}}$	Avtor	$s_{\mathbf{F}}/w_{\mathbf{F}}$
1078,4	146,5	133,4	33,1	Milan Randić	7,36
262,6	39,6	12,3	9,3	Lionello Pogliani	6,63
180,1	34,0	0,0	0,0	Jun-Ichi Aihara	5,29
292,7	63,2	0,0	1,5	Kuo-Chen Chou	4,63
55,8	12,9	0,0	0,0	Farhad Gharagheizi	4,32
208,4	51,2	63,7	22,2	Ernesto Estrada	4,07
83,3	20,5	3,5	3,5	Aleksander Sabljić	4,06
115,8	28,8	7,5	6,5	Isaac Freund	4,02
192,3	52,4	160,7	36,9	Ovidiu Ivanciuc	3,67
39,5	11,0	39,5	11,0	Anton Perdih	3,59
36,5	10,2	20,7	7,3	Bo Ren	3,59
137,1	39,6	47,4	16,9	Haruo Hosoya	3,47
51,3	16,2	19,1	8,4	Nenad Raos	3,16
229,6	72,9	103,6	34,0	Alexandru T. Balaban	3,15
134,1	44,5	6,2	9,4	Francisco Torrens	3,01
83,9	31,9	0,6	2,4	Lowell H. Hall	2,63
21,0	8,0	0,0	0,0	Juan A. Lazzus	2,63
71,0	27,4	16,3	7,6	Kunal Roy	2,59
76,9	30,1	12,0	10,8	Modjtaba Ghorbani	2,56
39,3	15,5	0,0	0,5	Richard F. W. Bader	2,54

Tabela 2.6

Avtorji, ki najbolj sodelujejo. $c_{\mathbf{B}}$ in $c_{\mathbf{F}}$ sta skupni normalizirani števili del, ki jih dani avtor prispeva in so bila objavljena skupaj z drugimi avtorji v okviru množice **B** oz. **F**. Rezultati so omejeni na množico avtorjev, ki so prispevali vsaj eno delo v okviru množice **B** in vsaj 15 del v okviru množice **F**. Devet najboljših avtorjev v ozir $c_{\mathbf{F}}$ je zapisanih z rdečo. Pomen imena v oklepajih je opisan v razdelku A.1.3.

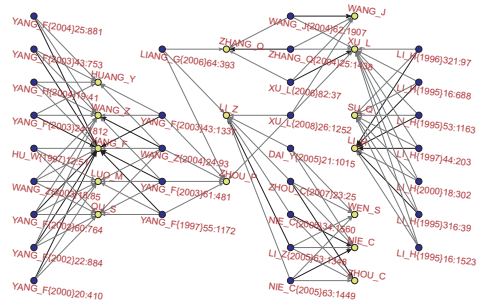
$c_{\mathbf{B}}$	$c_{\mathbf{F}}$	Avtor	$c_{\mathbf{B}}$	$c_{\mathbf{F}}$	Avtor
0,77	0,37	Kuo-Chen Chou	0,59	0,39	Lemont B. Kier
0,74	0,59	Claudiu T. Supuran	0,59	0,54	Taylor Schulz
0,74	0,74	Zimao Li	0,59	0,63	Humberto Gonzalez-Diaz
0,73	0,65	Ovanes Mekenyan	0,58	0,58	Ali Reza Ashrafi
0,72	0,69	Vijay K. Agrawal	0,58	0,58	Ali Iranmanesh
0,71	0,65	Jorge Galvez	0,56	0,36	Damir Vukičević
0,71	0,73	Eugenio Uiarte	0,55	0,52	Lowell H. Hall
0,70	0,70	Sonja Nikolić	0,54	0,55	Anil Kumar Madan
0,69	0,72	N. S. Zefirov	0,54	0,32	Aleksander Sabljić
0,69	0,61	Andrey A. Toropov	0,53	0,50	Kunal Roy
0,68	0,66	Alla P. Toropova	0,53	0,47	Douglas J. Klein
0,68	0,65	Xiaoling Zhang	0,52	0,54	(Heping Zhang)
0,67	0,67	P. V. Khadikar	0,51	0,41	Ivan Gutman
0,66	0,71	(Maykel Perez Gonzalez)	0,49	0,46	Chenzhong Cao
0,65	0,64	Eduardo A. Castro	0,48	0,43	Bo Zhou
0,65	0,66	Denise Mills	0,44	0,48	Danail Bonchev
0,64	0,73	Alan R. Katritzky	0,43	0,36	Aleksandar Ilić
0,63	0,62	Subhash C. Basak	0,42	0,43	Sandi Klavžar
0,63	0,63	Nenad Trinajstić	0,40	0,34	Nenad Raos
0,62	0,59	(Luo-Nan Xu)	0,40	0,47	Modjtaba Ghorbani

2.4.4 *Analiza soavtorstev del*

Na množici **F** smo za vsakega avtorja izračunali skupno normalizirano število del $c_{\mathbf{F}}$, ki jih je objavil skupaj z vsaj še enim soavtorjem. Skupno število del avtorja a je enako vsoti izhodnih vrednosti a v omrežju sodelovanj, normalizirani s skupnim številom vseh del, ki jih je a prispeval. Pred izračunom smo zanke v omrežju sodelovanj odstranili. Skupno število prispevanih del $c_{\mathbf{B}}$, pridobljeno na podlagi množice **B**, smo izračunali analogno. Rezultati za avtorje, ki so v okviru množice **F** skupaj prispevali vsaj 15 del in vsaj eno delo v okviru množice **B** so urejeni po številu $c_{\mathbf{B}}$ in prikazani v tabeli 2.6. Devet najboljših avtorjev glede na množico **F** je zapisanih z rdečo.

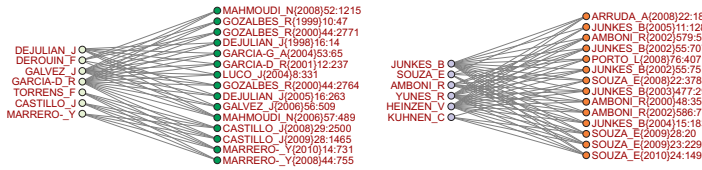
Dvovrstne sredice avtorjev in del

Z algoritmom za dvovrstne sredice [24] smo v omrežju avtorstev za množico del **B** identificirali izrazite dvovrstne sredice avtorjev in del. Prvi del takšne sredice je namenjen množici avtorjev in drugi množici del. Vsaka sredica povezuje množico avtorjev, ki so skupaj objavili zadostno količino oz. množico del. Identificirali smo 7 $(4, 3)$-sredic;



Slika 2.7

Druga največja (4, 3)-sredica v omrežju avtorstev, ki predstavlja skupino kitajskih avtorjev in njihovih del. Avtorji so označeni s svetlejšo barvo, dela s temnejšo.



Slika 2.8

Dve manjši (4, 3)-sredici v omrežju avtorstev. Avtorji so navedeni na levi strani vsake sredice.

vsako sestavljajo vsaj trije avtorji, ki so skupaj objavili vsaj štiri dela. Druga največja sredica predstavlja skupino kitajskih avtorjev in je prikazana na sliki 2.7. Dve manjši sredici sta v nekoliko drugačnem slogu prikazani na sliki 2.8. V prvi od slednjih dveh avtorji, razen *F. Derouin*, izhajajo iz Univerze v Valencii. V drugi zasledimo avtorje iz brazilske univerze *Federal University of Santa Catarina*. Parametra $p = 4$ in $q = 3$ smo za (p, q) sredice dobili empirično, in sicer s poskušanjem in opazovanjem dobljenih skupin. Pri manjših vrednostih smo dobili večje skupine, ki so vsebovale stotine avtorjev in del. Po drugi strani smo z večjimi vrednostmi dobili skupine z manj kot deset avtorji. Največja skupina je vsebovala 219 del in 76 avtorjev. Večina pričujočih avtorjev je prisotnih tudi v največjem otoku, omenjenem v razdelku 2.4.4 in prikazanim na sliki 2.10. Le-ta vsebuje najbolj izrazite avtorje in dela na področju topoloških indeksov.

Individualno objavljena dela

Poglejmo še avtorje, ki so samostojno objavili največ del $I_{\mathbf{F}}$. Gre za dela, ki imajo samo enega avtorja. V množici **B** je takšnih del 20, 0%, v množici **F** pa 16, 8%. Podatki so zapisani v tabeli 2.7. 15 od 17 najboljših samostojnih avtorjev glede na množico **F** je zapisanih z rdečo. Manjkajoča avtorja sta *J. I. Aihara* s 27 in *K. C. Chou* s 25

Tabela 2.7

Avtorji z največjim številom individualno objavljenih del. $I_{\mathbf{B}}$ je število del iz množice **B** z enim samim avtorjem. $P_{\mathbf{B}}$ podaja razmerje med samostojno objavljenimi članki in skupnim številom člankov izbranega avtorja v množici **B**. $I_{\mathbf{F}}$ in $P_{\mathbf{F}}$ predstavljata analogni količini $I_{\mathbf{B}}$ in $P_{\mathbf{B}}$, vendar v okviru množice **F**. 15 izmed 17 najboljših avtorjev z ozirom na $I_{\mathbf{F}}$ je zapisanih z rdečo (razlaga ideje zapisa z rdečo je v razdelku 2.4.2). Pomen imen v oklepajih je zapisan v dodatku A.1.3.

$I_{\mathbf{B}}$	$P_{\mathbf{B}}$	$I_{\mathbf{F}}$	$P_{\mathbf{F}}$	Avtor	$I_{\mathbf{B}}$	$P_{\mathbf{B}}$	$I_{\mathbf{F}}$	$P_{\mathbf{F}}$	Avtor
27	0,73	36	0,69	Ovidiu Ivanciuc	4	0,92	4	0,94	Zoita Berinde
24	0,76	27	0,65	Alina Pyka	4	1,00	4	1,00	E. L. Krasnykh
19	0,57	89	0,61	Milan Randić	4	1,00	4	1,00	Anton Perdih
17	0,50	36	0,49	Alexandru T. Balaban	3	0,73	16	0,73	Hanyuan Deng
12	0,74	12	0,74	(Avat A. Taherpour)	3	0,51	13	0,59	K. Balasubramanian
11	0,50	29	0,57	Ernesto Estrada	3	0,58	10	0,59	Adam Voelkel
10	0,73	15	0,71	Andrey A. Dobrynin	3	0,56	9	0,69	Danail Bonchev
9	0,97	38	0,96	Lionello Pogliani	3	0,33	8	0,23	Igor G. Zenkevich
9	0,69	21	0,70	Istvan Lukovits	3	0,73	5	0,67	Modjtaba Ghorbani
9	0,53	18	0,46	Haruo Hosoya	3	0,28	3	0,17	E. A. Smolenskii
9	0,98	9	0,87	Junfeng Hao	3	0,59	3	0,41	Anna Niestroj
8	0,39	22	0,38	Mircea V. Diudea	3	0,68	3	0,62	Dana Gheorghe
7	0,95	9	0,89	Bo Ren	2	1,00	10	1,00	Kinkar Ch. Das
6	0,92	23	0,80	Isaac Freund	2	0,46	8	0,61	Hongbo Hua
6	0,95	8	0,88	Lemi Turker	2	0,71	6	0,63	Nenad Raos
6	0,80	7	0,74	Abbas Heydari	2	0,24	6	0,37	Gj. Murphy
5	0,14	46	0,31	Ivan Gutman	2	1,00	6	1,00	S. Durr
5	0,53	30	0,67	Francisco Torrens	2	0,86	5	0,95	James Devillers
5	0,91	8	0,86	E. Cornwell	2	1,00	5	0,64	Ej. Kupchik
4	0,70	15	0,76	Dh. Rouvray	2	1,00	4	1,00	Seiichi Yamaguchi
4	0,73	5	0,69	Boris Hollas	2	1,00	4	1,00	Darren R. Flower

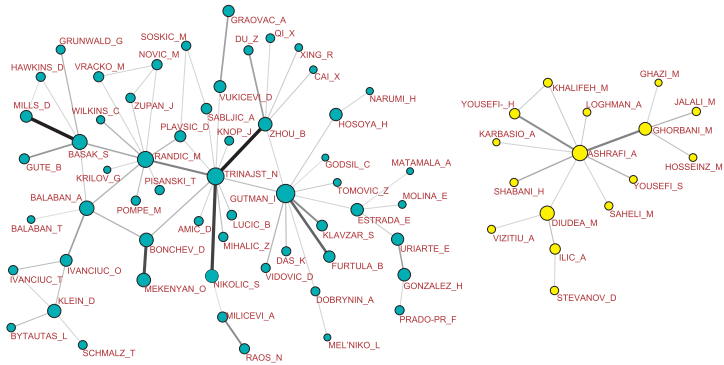
samostojnimi deli. Noben izmed njiju nima samostojno objavljenih člankov v množici **B**. Postopek za rdečimi zapisi je opisan v razdelku 2.4.2.

Posplošene sredice sodelujočih avtorjev

Na množici **F** smo poiskali šibko povezane skupine sodelujočih avtorjev. Največjo izmed njih sestavlja 11.462 avtorjev. Deset skupin sestavlja 20 do 43 avtorjev. Ostale so manjše. V največji skupini smo pri pragu 11 identificirali 73 posplošenih sredic [40]. Sredice avtorjev predstavljajo raziskovalne skupine, ki delujejo na skupni temi in svoja dela objavljajo kot soavtorji. Dve največji posplošeni sredici, prikazani na sliki 2.9, sta sestavljeni iz 58 oz. 16 avtorjev.

Otoki avtorjev v omrežju citiranj

V omrežju citiranj avtorjev smo našli 46 otokov velikosti od 5 do 100 avtorjev. Otoki [38] so šibko povezana podomrežja v omrežju. Povezanost znotraj otokov je v primerjavi s povezanostjo z okolico močnejša. Otoki v našem primeru predstavljajo skupine



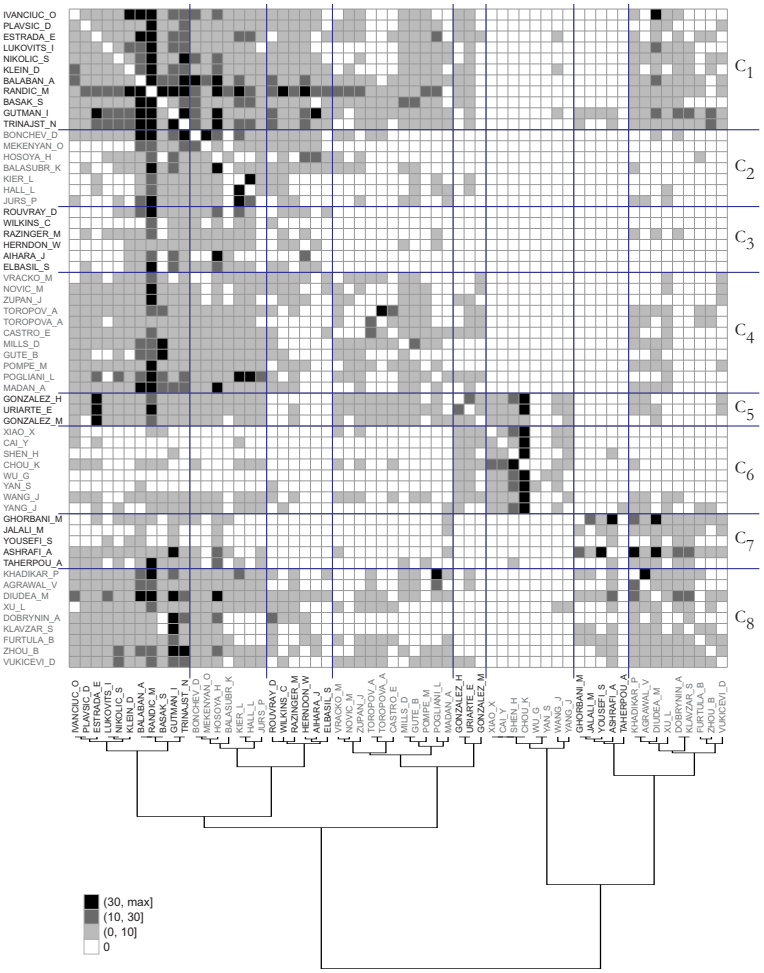
Slika 2.9

Dve največji sredici izmed 73 sredic, s pragom 11 dobljenih iz največje šibko povezane skupine sodelujočih avtorjev v normaliziranem omrežju. Utež in sivina povezav je sorazmerna obsežnosti sodelovanja med avtorjema v krajiščih povezave med njima. Velikost vozlišč – avtorjev je sorazmerna njihovemu skupnemu številu člankov, objavljenih skupaj z ostalimi avtorji.

avtorjev, ki se med seboj bolj citirajo. Vrh 60 avtorjev najmočnejšega otoka smo na sliki 2.10 prikazali v matrični obliki. Matriko smo v skladu s hierarhijo, ki smo jo dobili z *Ward*-ovim razvrščanjem [41, 42] in z nekaj ročnimi menjavami vej globlje v hierarhiji preuredili. Pod matriko je prikazan dendrogram, višina vejitev v njem pa se ujema z različnostmi vej hierarhije. Blizu skupaj razvrščeni avtorji tvorijo skupine, ki jih nakažemo z ločnicami. Iz hierarhije oz. dendrograma smo jih določili empirično. Pomen skupin avtorjev je, da le-te citirajo podobne množice avtorjev. Poglejmo npr. skupino C_6 na sredini matrike: gre za skupino kitajskih avtorjev. Podobno ugotovimo, da gre pri skupini C_7 za skupino iranskih avtorjev. Zanimiva je podobnost skupin C_5 in C_6 v njunem delu. C_5 vsebuje španske in kubanske avtorje. Čeprav bi avtorje obeh skupin lahko pojmovali kot skupno raziskovalno skupino, kitajski avtorji manj citirajo ostale avtorje, hkrati pa so manj citirani.

Otoki kot znanstvena pod-področja

Da bi razkrili s čim se ukvarjajo avtorji nekaterih skupin, smo analizirali ključne besede del, ki so jih avtorji izpostavljenih skupin objavili. Pod drobnogled smo vzeli največji otok in dva manjša otoka, ki smo jih opisali v razdelku 2.4.4. V besedilu jih imenujemo *glavni*, *srednji* in *mali* otok. V istem vrstnem redu vsebujejo 100, 20 in 7 avtorjev. Posebej smo analizirali skupino avtorjev C_7 znotraj največjega otoka. Skupino C_7 smo opisali v razdelku 2.4.4, prikazali pa smo jo na sliki 2.10. Relativno glede na število avtorjev, ki jih ima posamezno delo, smo izračunali kolikokrat se posamezna ključna beseda uporablja v delih, ki so jih avtorji v skupini objavili. Pod pojmom "relativno"



Slika 2.10

Matrični prikaz vrha glavnega otoka v omrežju citiranj avtorjev. Spodaj je prikazan dendrogram hierarhije podobnosti avtorjev. Matrika je urejena glede na hierarhijo, ki smo jo dobili z *Ward*-ovim razvrščanjem. Dejanske vrednosti povezav – polja matrike smo kodirali v štiri vrednosti, prikazane v legendi. Empirično smo identificirali osem podobnostnih skupin avtorjev: od C_1 do C_8.

Tabela 2.8

Najpogostejše ključne besede v treh izbranih otokih in skupina C_7 (stolpci). Vse strukture so opisane v razdelku 2.4.4. f je relativna frekvenca pojavnosti posamezne ključne besede v delih, ki so jih avtorji posamezne skupine objavili.

Glavni otok		*Srednji* otok		*Mali* otok		Skupina C_7	
Ključna beseda	f	Ključna beseda	f	Ključna beseda	f	Ključna beseda	f
index	0,66	dynamics	0,56	similarity	0,76	index	1,00
topological	0,51	use	0,50	quantum	0,71	graph	0,99
molecular	0,50	topological	0,50	molecular	0,66	paper	0,83
graph	0,46	dynamical	0,50	measure	0,65	compute	0,82
use	0,44	chaotic	0,44	use	0,60	g	0,81
descriptor	0,32	attractor	0,39	set	0,58	topological	0,80
model	0,30	symmetry	0,33	qsar	0,51	nanotube	0,67
number	0,29	index	0,33	application	0,48	define	0,63
structure	0,27	generate	0,33	relationship	0,45	fullerene	0,63
study	0,26	mapping	0,33	study	0,44	number	0,60
property	0,25	nonlinear	0,33	descriptor	0,41	wiener	0,56
chemical	0,23	result	0,33	approximation	0,40	pi	0,56
qsar	0,23	model	0,33	property	0,40	molecular	0,53
base	0,23	1	0,28	shell	0,39	vertex	0,50
relationship	0,22	2	0,28	function	0,39	polyhex	0,48
matrix	0,22	chaos	0,28	model	0.38	polynomial	0.46
result	0,21	time	0,28	density	0.38	sigma	0.44
wiener	0.21	allow	0,28	atomic	0,37	family	0,43
set	0,21	observe	0,28	present	0,37	n	0,42
vertex	0,21	parameter	0,28	structure	0,35	infinite	0,42

mislimo na normalizacijo s številom soavtorjev posameznega članka; dobljene količine smo normalizirali z normaliziranim številom člankov, ki so jih avtorji posamezne skupine objavili. Končne količine predstavljajo relativno frekvenco, kolikokrat je bila ključna beseda v povprečju v članku uporabljena. Upoštevana so le dela, ki so jih avtorji posamezne skupine objavili. Na podlagi najpogostejših ključnih besed smo določili pod-področja s katerimi se avtorji posamezne skupine ukvarjajo. V tabeli 2.8 je za vsako obravnavano skupino prikazan seznam 20 najpogostejših ključnih besed.

Področje *glavnega* otoka se ujema s področjem teme molekularnih topoloških indeksov. Je relativno velik in specifične ključne besede zato ostajajo v ozadju. Pri petkrat manjšem, *srednjem* otoku na površje splava tema o topološki analizi kaotičnih dinamičnih sistemov, v še manjšem, *malem* otoku pa opazimo področje kvantnih molekularnih podobnostnih metrik v QSAR modelih itd. V nasprotju z *glavnim* otokom, v njegovi skupini C_7 zaznamo specifično področje o računski nanoznanosti fulerenov ter nanocevk.

Tabela 2.9

Najpopularnejše revije (v stolpcih) med avtorji otokov. Vrstni red sledi odstotku del, objavljenih v posamezni reviji.

Glavni otok		*Srednji* otok		*Mali* otok	
revija	%	revija	%	revija	%
J. Chem. Inf. Comput. Sci.	8,4	Phys. Rev. E	38,9	J. Chem. Inf. Comput. Sci.	15,1
MATCH Commun. Math. Comput. Chem.	7,6	Chaos	27,8	J. Math. Chem.	13,9
Croat. Chem. Acta	4,0	Int. J. Bifurcation. Chaos	5,6	Int. J. Quantum. Chem.	9,1
J. Math. Chem.	3,1	J. Phys. A: Math. Theor.	5,6	J. Comput. Chem.	7,2
Chem. Phys. Lett.	2,5	Physica D	5,6	J. Mol. Struct.: Theochem.	7,1
J. Mol. Struct.: Theochem.	2,1	Rev. Mod. Phys.	25,6	J. Comput. Aid. Mol. Des.	5,0
Int. J. Quantum. Chem.	2,0	Siam. Rev.	5,6	Adv. Molec. Simil.	3,4
Rev. Roum. Chim.	2,0	Sol. Phys.	5,6	Sar. QSAR Environ. Res.	3,4
Sar. QSAR. Environ. Res.	1,9			Adv. Quantum. Chem.	3,0
Bioorgan. Med. Chem.	1,7			J. Chem. Inf. Model.	2,3
Druge revije	64,7	Druge revije	0,0	Druge revije	30,5

Značilne revije otokov

V tem razdelku preučujemo v katerih revijah so avtorji posameznih otokov, opisanih v razdelku 2.4.4, objavljali. Za vsak otok je v tabeli 2.9 prikazanih deset najbolj značilnih revij. Urejene so po odstotku člankov, ki so jih avtorji posameznih otokov v njih objavili. Avtorji v *srednjem* otoku so objavljali samo v osmih revijah.

2.4.5 Analiza revij

Zaradi nestandardnega zapisa imen revij je kazalo, da so bila nekatera dela objavljena v več revijah, kar je v nasprotju z etiko publiciranja v znanosti. Primerov objave kateregakoli dela v več kot dveh revijah nismo našli. Imena revij v *Web of Science* sestojijo iz pogosto dvoumnih okrajšav besed polnega imena revije. Na primer okrajšavi J SOL-GEL SCI TECHN in J SOL-GEL SCI TECHNOL predstavljata isto revijo *Journal of Sol-Gel Science and Technology*. Iz primerov del, ki so bila na videz objavljena v dveh različnih revijah smo identificirali sumljive pare okrajšav imen revij. Programsko smo jih preverili in jih skušali poenotiti. Za vsak sumljiv par okrajšanih imen smo preverili, če se prva črka posamezne besede v prvem imenu ujema z enakoležno okrajšavo v drugem okrajšanem imenu. Vse razen devetih parov revij smo uspeli poenotiti. V ostalih devetih primerih so pari predstavljali različne revije. Kje natančno so bila dela objavljena smo ročno preverili in ugotovili, da je v vseh primerih šlo za napake v podatkih, ki smo jih dobili iz *Web of Science*. Prav vsa dela so bila objavljena v eni sami reviji.

Tabela 2.10

Najpopularnejše revije v množici del **B** in **F**. $wj_{\mathbf{B}}$ in $wj_{\mathbf{F}}$ sta števili del, ki so bila objavljena v izbranih revijah. V rdeči (glejte analogni primer, opisan v razdelku 2.4.2) je 8 od 9 najboljših revij glede na celotno množico del **F**.

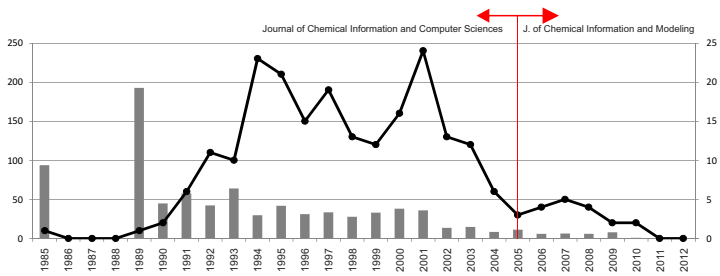
$wj_{\mathbf{B}}$	$wj_{\mathbf{F}}$	Revija
205	929	Journal of Chemical Information and Computer Sciences
104	531	MATCH-Communications in Math. and in Computer Chemistry
75	267	Croatica Chemica Acta
54	246	Journal of Mathematical Chemistry
39	196	Journal of Molecular Structure (Theochem)
39	118	JPC – Journal of Planar Chromatography
39	87	Revue Roumaine de Chimie
38	170	SAR and QSAR in Environmental Research
38	85	Indian Journal of Chemistry – Section A
35	202	Bioorganic & Medicinal Chemistry
32	74	Optoelectronics and Advanced Materials: Rapid Communications
31	47	Acta Chimica Slovenica
27	58	Acta Chimica Sinica
24	40	Oxidation Communications
22	181	Chemical Physics Letters
21	164	QSAR & Combinatorial Science
20	240	Journal of Chemical Information and Modeling
20	98	Journal of Molecular Graphics and Modelling
20	50	Journal of Computational and Theoretical Nanoscience
19	182	International Journal of Quantum Chemistry

Menimo, da bi opisan postopek lahko v *Web of Science* bazi izvedli v splošnem.

Popularne revije

Revije z največ objavami na področju topoloških indeksov smatramo za najpopularnejše revije tega področja. Prikazane so v tabeli 2.10. V okviru množice **F** in množice **B** z wj_F in wj_B označimo število člankov za posamezno revijo. Osredotočimo se na bazno množico **B**. 225 izmed 2.036 (11,1%) del je bilo objavljenih v različnih posameznih revijah. V vsaki izmed 1% najpopularnejših revij (20 revij) je bilo objavljenih vsaj 19 del. V teh revijah je bilo objavljenih kar 44,3% vseh člankov. Osem izmed devet najpopularnejših revij v okviru množice **F** je v tabeli 2.10 zapisanih v rdeči. Manjka revija *Journal of Medicinal Chemistry* z $wj_F = 248$ člankov. Razlaga postopka zapisa z rdečo je podana v razdelku 2.4.2.

Časovne analize podatkov v splošnem nismo naredili, smo pa na predlog profesorja *M. Randić*-a podrobneje raziskali življenjski cikel najpopularnejše revije *Journal of Chemical Information and Computer Sciences* – JCICS. Povedal nam je, da je bilo v letu 2005 ime revije spremenjeno. Dodelili so ji novo ime *Journal of Chemical Information*



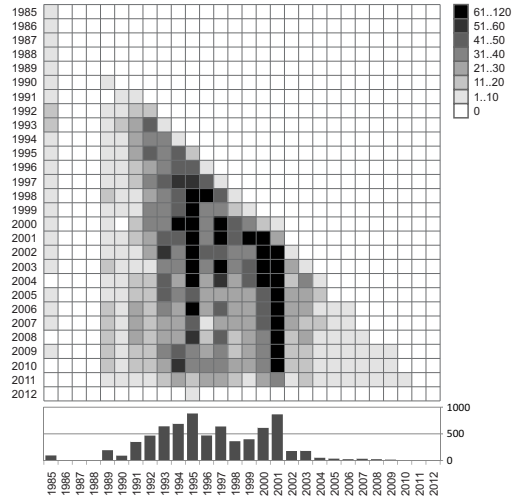
Slika 2.11

Število objavljenih del na temo topoloških indeksov v reviji *Journal of Chemical Information and Computer Sciences* v letnih razdelkih pred in po letu 2005, ko so revijo preimenovali v *Journal of Chemical Information and Modeling* – lomljenka s skalo na levi. Histogram s skalo na desni prikazuje normalizirano število citiranj iz del množice **F** na dela v povezavi s topološkimi indeksi v obeh inkarnacijah revije. Normalizacijo smo naredili glede na število del na temo topoloških indeksov, letno objavljenih v reviji.

and Modeling, kar podatki natančno odražajo. Del, ki bi bila po letu 2004 zmotno zavedena pod objavo v reviji s starim imenom, nismo našli. Podobno tudi del, ki bi bila v reviji z novim imenom objavljena pred 2005, ni bilo. Upad števila člankov na temo topoloških indeksov vidimo v prikazu na sliki 2.11. Proti letu 2005 se število objavljenih člankov vidno zmanjša. Na sliki 2.12 je po letih prikazano število citatov člankov na temo topoloških indeksov, objavljenih v opazovani reviji. Gre za podmnožico del iz množice **B**, ki so bila objavljena v JCICS. Na dnu vidimo histogram števil skupnih citatov člankov po letih. Dela, ki citirajo (na levi) so dobljena iz množice **F**. Na sliki 2.12 je možno videti, da proti letu 2005 število citatov močno upada in v prihodnosti ostane nizko. Normalizirana števila citatov so prikazana na skali histograma na sliki 2.11. Tudi pri njih je proti letu 2005 viden upad, kar nakazuje, da se je relativno število citatov del v reviji na temo topoloških indeksov zmanjšalo. S tem sovpada sprememba usmeritve revije. Ker se upad pojavlja že v letu 2002, je možno, da je sprememba imena revije posledica širše spremembe njene usmeritve. Delno zaradi zmanjšanja popularnosti revije na področju topoloških indeksov.

Citiranje med revijami

Poglejmo, kako v delih, objavljenih v izbrani reviji, citirajo dela, objavljena v drugih revijah. Povedano z drugimi besedami; kako revije citirajo druge revije. V omrežju citiranj med revijami $JJ = WJ^T \times Cite \times WJ$ (tabela A.1) smo najprej poiskali povezane



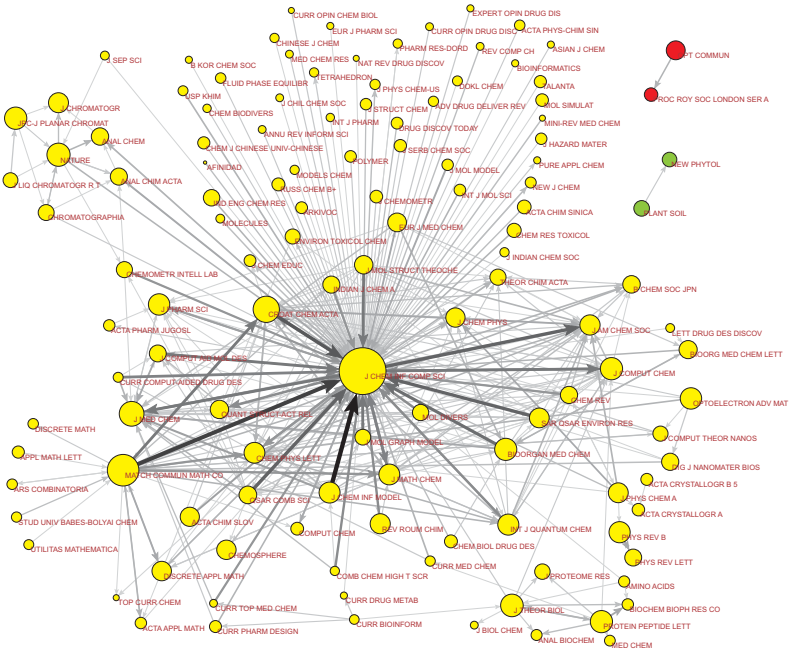
Slika 2.12

Število citatov po letih (levo) iz del v množici **F** na dela objavljena v reviji *Journal of Chemical Information and Computer Sciences* in na temo topoloških indeksov (podmnožica del množice **B**, objavljenih v JCICS) po letih (spodaj). Pod matriko je prikazan še histogram citatov. Legenda: da bi poudarili kontrast, smo natančne vrednosti matrike prekodirali v osem intervalov. Najvišja originalna vrednost v matriki je bila 120.

skupine. Našli smo le dve ne-trivialni. Prva vsebuje tri revije, druga pa 1.616 revij. Z algoritmom za posplošene sredice in primerno izbiro velikosti smo poiskali sredice v drugi skupini. Izbrane sredice smo sestavili tako, da revije v njih druga drugo citirajo v vsaj 40 delih. Sredice so prikazane na sliki 2.13. Usmerjene povezave na prikazu predstavljajo citiranja iz skupin del, objavljenih v eni reviji, na skupine del, objavljene v drugi reviji. Uteži in sivina povezav je sorazmerna kvadratnemu korenu dejanske količine citiranj. Premer vozlišč – revij je sorazmeren kvadratnemu korenu števila citiranj v delih iz izbrane revije na skupino del v isti reviji – samo-citiranje revije. Razvidno je, da so članki v reviji *Journal of Chemical Information and Computer Sciences* najbolj citirani. Sledijo jim članki iz revije *Journal of Medicinal Chemistry* in iz *Journal of the American Chemical Society*. Revija z najvišjo stopnjo samo-citiranja je prav tako *Journal of Chemical Information and Computer Sciences*, sledita pa ji reviji *MATCH-Communications in Mathematical and in Computer Chemistry* in *Croatica Chemica Acta*.

2.4.6 Revije in avtorji

V tabeli 2.11 navajamo avtorje, ki so svoja dela objavljali v največ različnih revijah. $N_{\mathbf{F}}$ in $N_{\mathbf{B}}$ sta števili različnih revij v katerih je posamezni avtor objavljal. Prvo število je izračunano na podlagi celotne množice del **F**, drugo pa le na bazni množici **B**. Vre-



Slika 2.13

Sredice v največji povezani skupini revij. Sredice so sestavljene iz vozlišč – revij, ki citirajo vsaj 40 del v katerikoli drugi, vendar isti reviji. Usmerjene povezave predstavljajo citate v člankih ene revije člankov v drugi. Uteži in odtenek povezav je sorazmeren kvadratnemu korenu števila dejanskih citatov. Velikost vozlišč je sorazmerna kvadratnemu korenu števila citiranj v delih izbrane revije na dela v isti reviji – samo-citiranje revije.

Tabela 2.11

Avtorji, ki objavljajo v največ različnih revijah. $N_{\mathbf{F}}$ in $N_{\mathbf{B}}$ sta števili v koliko revijah dani avtor objavlja. Prvo vključuje dela iz množice **F**, drugo pa tiste iz **B**. Pomen avtorjev v oklepajih je zapisan v dodatku A.1.3.

$N_{\mathbf{B}}$	$N_{\mathbf{F}}$	Avtor	$N_{\mathbf{B}}$	$N_{\mathbf{F}}$	Avtor
27	73	Subhash C. Basak	13	42	Haruo Hosoya
27	50	P. V. Khadikar	13	33	(Luo-Nan Xu)
26	88	Ivan Gutman	12	24	Sonja Nikolić
26	75	Alexandru T. Balaban	12	18	Francisco Prado-Prado
21	49	Ali Reza Ashrafi	12	17	Florencio M. Ubeira
20	70	Nenad Trinajstić	11	47	Andrey A. Toropov
20	43	Humberto Gonzalez-Diaz	11	44	Eduardo A. Castro
19	35	Anil Kumar Madan	11	21	Sneha Karmarkar
18	41	Ramon Garcia-Domenech	11	16	Jyoti Singh
18	39	Eugenio Uiarte	11	14	Augusto Rosendo Yunes
18	33	Vijay K. Agrawal	11	12	Vilma Edite Fonseca Heinzen
17	78	Milan Randić	10	36	Kunal Roy
17	69	Francisco Torrens	10	33	Bo Zhou
17	50	Ernesto Estrada	10	32	Claudiu T. Supuran
16	52	Danail Bonchev	10	31	N. S. Zefirov
16	42	Jorge Galvez	10	27	Istvan Lukovits
16	26	Denise Mills	10	26	Chenzhong Cao
14	42	Mircea V. Diudea	10	19	Mehdi Eliasi
14	39	Ovidiu Ivanciuc	10	13	Cristian Robert Munteanu
13	73	Zimao Li	9	52	Ovanes Mekenyan

dnosti smo izračunali iz dvodelnega omrežja (1.4.2), ki smo ga dobili z množenjem normaliziranega omrežja avtorjev in del ter omrežja del in revij, $AJ = AW_N \times WJ$; glejte tabelo A.1.

2.4.7 *Ključne besede*

Kot za ostala omrežja, smo omrežje del in ključnih besed konstruirali s pomočjo orodja *Wos2Pajek*. Le-ta za analizo ključnih besed uporablja knjižnico *MontyLingua* [43], ki fraze, imena in druge jezikovne konstrukte izlušči iz besedila in jih normalizira v osnovno obliko – lema. Vse ključne besede nad katerimi izvajamo analizo se tako že nahajajo v normalni obliki. V našem primeru smo upoštevali ključne besede, ki se nahajajo v naslovih del, v seznamih ključnih besed del in v njihovih povzetkih. Fraze ključnih besed se v *Wos2Pajek* razbije na posamezne besede. Pogoste besede, kot so členi, zaimki, pogosti predlogi, pogosti glagoli idr., ki jih skupno imenujemo manjpomenske besede *Wos2Pajek* avtomatsko odstrani in jih v analizi ne upoštevamo.

Tabela 2.12

40 najbolj zastopanih ključnih besed. $R_{\mathbf{B}}$ in $R_{\mathbf{F}}$ sta relativni frekvenci uporab posameznih ključnih besed v povprečnem delu v množici **B** oz. **F**. Najbolj pogostih 24 ključnih besed glede na $R_{\mathbf{F}}$ je zapisanih v rdeči.

$R_{\mathbf{B}}$	$R_{\mathbf{F}}$	Ključna besed	$R_{\mathbf{B}}$	$R_{\mathbf{F}}$	Ključna beseda
1,000	0,357	index	0,244	0,244	set
1,000	0,253	topological	0,227	0,211	quantitative
0,557	0,509	use	0,226	0,246	prediction
0,543	0,385	molecular	0,226	0,207	chemical
0,395	0,196	graph	0,222	0,178	new
0,381	0,366	model	0,218	0,071	wiener
0,364	0,331	study	0,213	0,176	parameter
0,362	0,264	descriptor	0,212	0,144	regression
0,327	0,267	relationship	0,210	0,103	matrix
0,325	0,249	property	0,206	0,210	2
0,314	0,313	structure	0,201	0,089	distance
0,308	0,295	result	0,200	0,174	value
0,287	0,198	number	0,198	0,189	1
0,270	0,201	QSAR	0,193	0,148	structural
0,268	0,263	compound	0,191	0,151	correlation
0,262	0,228	base	0,188	0,185	molecule
0,252	0,101	connectivity	0,180	0,128	paper
0,251	0,207	obtain	0,174	0,081	vertex
0,249	0,287	method	0,171	0,167	present
0,244	0,257	analysis	0,167	0,171	different

Pogoste ključne besede

Najbolj pogosto zastopane ključne besede smo identificirali in uredili glede na njihovo relativno frekvenco $R_{\mathbf{B}}$ uporabe v povprečnem delu v množici **B**. Frekvence na množici **F** označimo z $R_{\mathbf{F}}$. Rezultati so prikazani v tabeli 2.12. 24 najpogostejših ključnih besed glede na $R_{\mathbf{F}}$ je zapisanih rdeče. Pri izračunu $R_{\mathbf{B}}$ smo absolutne frekvence normalizirali s celotnim številom del v množici **B** (2.036), pri $R_{\mathbf{F}}$ pa s številom del v množici **F** (13.394). Za dela iz množice $\mathbf{D}'$ (2.3.3) imamo podane le ključne besede iz naslovov. Zanje namreč nimamo polnih opisov, t.j. povzetki in seznami ključnih besed nam za ta dela manjkajo. Analize ključnih besed teh del zato nismo naredili. Rezultati samo na podlagi naslovov bi bili pristranski.

Množice ključnih besed, ki jih uporabljajo avtorji člankov v bazni množici

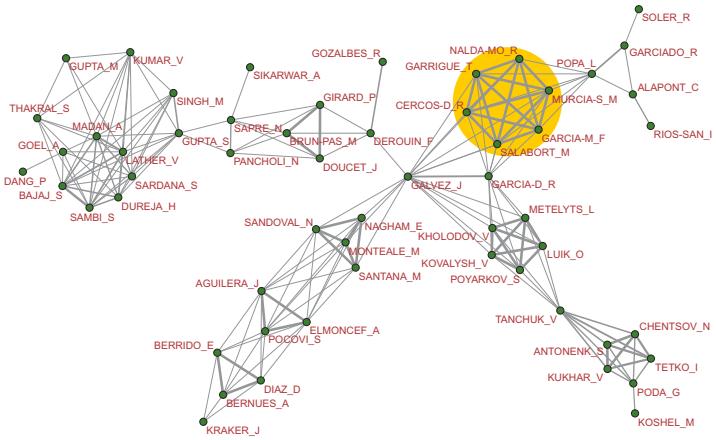
Ključne besede smo razvrstili glede na njihovo zastopanost pri avtorjih v množici **B**. Vrstni red smo določili na omrežju avtorjev in ključnih besed, $AK = WA^T \times WK$ (tabela A.1). Najbolj značilne ključne besede posameznih avtorjev smo določili s *tf-idf* urejanjem [44]:

$$\begin{aligned} \text{tfidf}(k,a) &= \text{tf}(k,a)\cdot\text{idf}(k) \\ \text{tf}(k,a) &= \frac{\text{pomembnost ključne besede } k \text{ avtorju } a}{\sum_i(\text{pomembnost ključne besede } k \text{ avtorju } i)} \\ \text{idf}(k) &= \log\frac{\text{število avtorjev}}{\text{število avtorjev, ki uporablja ključno besedo } k} \end{aligned} \tag{2.1}$$

Za vsakega avtorja smo ohranili $n = 15$ najbolj značilnih oz. vse ključne besede, če jih uporablja manj. Vsem ključnim besedam smo v nadaljevanju pripisali enako pomembnost, neodvisno od vrstnega reda. Z množenjem reduciranega omrežja avtorjev in ključnih besed s svojo transpozicijo smo dobili omrežje avtorjev s podobnim interesom glede na podobnost naborov zanje značilnih ključnih besed. V omrežju smo določili otoke, t.j. podomrežja oz. skupine avtorjev, ki uporabljajo podobne nabore ključnih besed. Otoki z manj kot štirimi avtorji so večinoma polna podomrežja avtorjev, ki dela objavljajo skupaj. Večji otoki, vendar manjši od 12 avtorjev so pretežno sestavljeni iz ene (oz. v približno četrtini primerkov iz dveh) povezanih polnih podomrežij avtorjev, ki uporabljajo skupni nabor ključnih besed. Pri večjih otokih struktura zajema več povezanih polnih podomrežij ali skoraj polnih podomrežij avtorjev. Rezultati so v skladu s pričakovanji, saj je mediana števila avtorjev, ki skupaj objavljajo članke enaka 3. Drugi največji otok avtorjev je prikazan na sliki 2.14. Polna podomrežja avtorjev so dobro vidne v strukturi otoka. Poglejmo največjo (osenčeno) polno podomrežje otoka na sliki 2.14. Vsebuje šest avtorjev: *M. Murcia-Soler*, *R. Nalda-Molina*, *M. T. Salabort-Salvador*, *F. J. García-March*, *R. A. Cercós-del-Pozo* in *T. M. Garrigues*. Vsi našteti avtorji uporabljajo enak nabor značilnih ključnih besed: *agent*, *confirm*, *diagram*, *discriminant*, *distribution*, *fit*, *goodness*, *hypoglycemic*, *identification*, *LDA*, *PDD*, *pharmacological*, *rat*, *stability* in *visualize*. Ni presenetljivo, da so skupaj objavili članek; *QSAR analysis of hypoglycemic agents using the topological indices* – MURCIA-S_M(2001)41:1345.

2.5 Razprava

Rezultati analize omrežij temeljijo na izvirnih podatkih in na izboru primernih analitičnih metod. Analizo se navadno izvaja progresivno na rezultatih ene od predhodnih analiz. Postopek navadno sproti prilagajamo, glede na vmesne rezultate, možnosti pa je veliko. Ko smo na primer določili množico otokov v omrežju sodelovanj, bi lahko vsak otok posebej analizirali naprej. Izbrali smo jih nekaj, a se moramo zavedati, da so vsi enako pomembni in po svoje zanimivi. Eden od pristopov v analizi omrežij je časovna



Slika 2.14

Drugi največji otok avtorjev, ki uporabljajo sorodne množice ključnih besed. Primer polnega podomrežja 6 avtorjev je osenčen.

analiza. Kljub temu, da z orodjem *Wos2Pajek* izluščimo časovne podatke, t.j. vektor let objave del, smo v splošnem časovno analizo izpustili. Pogledali smo le dogajanje v zvezi z najbolj popularno revijo. Čeprav *Wos2Pajek* že upošteva večino podatkov iz *Web of Science*, bi bile nekatere dodatne funkcionalnosti dobrodošle. Lahko bi na primer upoštevali regijske podatke o avtorjih. Upoštevanje dejstva, da mnogi avtorji delujejo v več državah oz. ustanovah in se pogosto selijo, bi analizo precej otežilo. Tudi glede elektronskih naslovov avtorjev bi lahko rekli podobno itd. Vsekakor bi bilo zanimivo raziskati, kako se je tema razvijala prostorsko skozi čas. Na področju ključnih besed iz naslovov, povzetkov in dejanskih seznamov ključnih besed del je v orodju *Wos2Pajek* možna izboljšava. Dobljeni seznami ključnih besed so nemalokrat dolgi in vsebujejo mnoge splošne besede, npr.: uporabiti, analiza, študija, metoda, rezultat idr. Takšne besede bi bilo dobro izločiti iz množic ključnih besed del, lahko na enak način, kot je že implementirano za t.i. manjpomenske besede. Množico splošnih besed bi seveda morali določiti vnaprej. Iz velikega števila znanstvenih del bi lahko zbrali nabor najbolj pogostih besed, ki bi služila kot filter ozadja pri iskanju bolj informativnih in specifičnih ključnih besed za posamezna dela. *Wos2Pajek* oz. bolje *MontyLingua* besede že postavi v njihovo osnovno obliko – lema, ne združuje pa sinonimov in besed z enakimi koreni. Tudi takšne primere različnih besed bi pogosto brez škode lahko nadomestili z

eno samo besedo.

Nekaj besed o validaciji rezultatov: postopki, ki smo jih uporabili v naših analizah se razmeroma pogosto uporabljajo v analizi omrežij, a se njihovi rezultati iz primera v primer lahko precej razlikujejo, niso razločni in/ali enolični. Posebne in eksplicitne validacije rezultatov zato ne moremo izvesti. Lahko bi jih kvečjemu primerjali z rezultati podobnih postopkov na alternativnih omrežjih, ki bi jih konstruirali iz drugih virov podatkov, npr. iz baz kot sta *Google Scholar* ali *Scopus*, itd. *Google Scholar* je še posebej zanimiv, saj za razliko od *Web of Science* indeksira tudi opise knjig in njihovih poglavij. Orodje za konstrukcijo bibliografskih omrežij iz podatkov *Google Scholar* zaenkrat še ne obstaja.

Še beseda o *Web of Science*. Pri iskanju množice del **C** (slika A.1) smo našli približno polovico del, ki so bila citirana iz bazne množice del **B**. Pri tem ne upoštevamo starejših člankov, knjig in drugih vrst del, ki jih *Web of Science* ne indeksira. Opažanje nam vzbudi občutek, kako popolna pravzaprav je baza *Web of Science*, če bi v njej iskali poljubno delo na temo topoloških indeksov. Na enak način bi za poljubno znanstveno področje lahko določili popolnost baze *Web of Science* ali katere druge. Slabost postopka je predvsem v časovni zahtevnosti, saj zahteva veliko ročnega dela. V tej smeri nismo dodatno raziskovali. V primeru, da bi dobili direktni dostop do baze, bi bilo metodo vsekakor koristno uporabiti.

2.5.1 Rezultati drugih analiz

Za bralce, ki bi si želeli ogledati popoln nabor rezultatov, vmesne rezultate in tudi nekaj analiz, ki jih v tem poglavju nismo izpostavili, smo na spletu pripravili dodatek [28]. Vmesne rezultate v obliki omrežij, razbitij in vektorjev v formatu *Pajek*, tekstovnih datotek in *Excel*-ovih tabel je mogoče prenesti. Program za združevanje okrajšanih imen in program za ustvarjanje iskalnih nizov za *Web of Science* smo razvili v jeziku *C#* in ju je prav tako možno prenesti.

3

Hierarhični prikazi

3.1 Prikazi podatkov

Eden glavnih ciljev vizualizacije podatkov je, da nam pomaga pri njihovem razumevanju. Izkorišča človekov dominantni čut – vid. Pogosto slišimo, da sta v vid vpleteni dve tretjini možganske skorje. Petina možganske skorje naj bi bila izključno namenjena vidu, preostali del pa kombinaciji vida z drugimi funkcijami možganov kot so tip, motorika, pozornost, prostorska orientacija, interpretacija itd. Centri za posamezne naloge možganov sicer niso ostro definirani. Z vizualizacijo skušamo v čim večji meri izkoristiti sposobnost vidnega zaznavanja [45], ki pri človeku med drugim vključuje izostreno zaznavanje vzorcev, trendov, nepravilnosti, odstopanj ipd. Dobro oblikovan prikaz lahko zajema veliko količino podatkov ter bistvo izpostavi tako, da ga je, namesto s poglobljenim razumevanjem, možno na naraven način, s t.i. zaznavnim sklepanjem hitro "zaznati". Lahko ga uporabimo tudi kot alternativno podajanje podatkov oz. kot povzetek, ki olajša procese razumevanja podatkov, pomnenje in odločanje na podlagi podatkov. S prikazi lahko podatke približamo širšemu krogu bralcev in v njih spodbudimo zanimanje. Izziv vizualizacije je v pripravi pomenljivih, jasnih in zanimivih prikazov, ki ustrezajo podatkom, na katerih temeljijo in o njih govorijo resnico [46].

Pri izdelavi prikazov moramo sprejeti in pretehtati nekatere odločitve. Odločiti se moramo: katere podatke izbrati, kaj bi s prikazom radi sporočili in na katera vprašanja bi radi odgovorili? Nato se moramo odločiti za način prikaza in njegove grafične lastnosti: barve, oblike, velikosti, položaja itd. Glavna težava je v tem, da je izbor prikazov izjemo velik. V skladu s tem so znanstveniki iz področij računalništva, psihologije in statistike skušali doumeti, kako določen grafični prikaz podatkov omogoča razumevanje podatkov. Številčne podatke je na primer ugodneje podati v obliki položajev, na primer v razpršenem ali v stolpičnem prikazu, za razliko od predstavitve, ki vrednosti podaja v obliki kotov, s površino ali, še slabše, z barvno intenziteto. Na različne dražljaje smo namreč različno občutljivi [47]. Ni presenetljivo, da je večina prikazov v literaturi kodiranih s položaji: razpršeni prikaz, linijski prikaz, histogrami itd. Ker procesa vidne zaznave še ne razumemo dovolj dobro in ker se od posameznika do posameznika razlikuje, moramo pri pripravi prikazov upoštevati tudi druge vidike, kot je na primer estetika [46].

Pri odločitvah o ustreznosti prikazov moramo upoštevati lastnosti podatkov, kot so velikost, gostota in dejavnost (deterministični, naključni). Na izbiro prikazov ključno vpliva tudi velikost ter narava medija, na katerem jih želimo prikazati. Velikih množic

podatkov navadno na majhnih in/ali statičnih medijih v celoti ni mogoče prikazati, saj bi se podrobnosti podatkov izgubile. Pri medijih, kot je na primer list papirja, se moramo zateči k prikazom s povzetki ali pa se moramo omejiti na posamezne dele podatkov oz. na t.i. delne prikaze podatkov. V primeru, da izberemo dinamični medij, kot je zaslon, lahko prikaz realiziramo na interaktivni način, s prepletanjem obeh načinov.

Prikazom s povzetki je skupno, da vključujejo celoten nabor podatkov, vendar na njih ne moremo ločiti vseh podrobnosti. Uporabljamo jih, ko navkljub velikosti želimo podatke prikazati v celoti ali če opazovalca želimo seznaniti s splošnimi lastnostmi podatkov. Ločimo več primerov prikazov s povzetki. Tipična sta dva: prikazi z zgostitvami in prikazi s skeleti. Prikazi z zgostitvami ne prikazujejo posameznih podatkov. V njih podatke nadomestimo z vzorci, ki jih v prostoru prikaza glede na lastnosti podatkov razporedimo blizu skupaj. Tako pride do zgostitev tam, kjer se nahaja več podatkov s podobnimi lastnostmi. Preprost primer prikaza z zgostitvijo je histogram. Lahko si predstavljamo, da v histogramu vsakemu stolpcu pripada koš, katerega robovi predstavljajo mejne vrednosti intervalov lastnosti podatkov, ki sodijo v ta koš. V koše nato razvrstimo vse podatke. Višina stolpcev v histogramu je odvisna od števila podatkov v pripadajočih koših in odraža gostoto podatkov z lastnostmi, ki jih ustrezen koš zajema. V poglavju 5 smo opisali primere precej bolj naprednih prikazov z zgostitvami, stvarni primer takega prikaza pa je podan na sliki 5.4. Na prikazih s skeleti, v želji po večji preglednosti, množice sorodnih podatkov nadomestimo z izbranimi predstavniki, ki jih najbolje opredeljujejo. Na primeru omrežij lahko namesto celotnega omrežja prikažemo le omrežju vpeto minimalno drevo. Povedano drugače, prikazi s povzetki vpeljujejo abstrakcijo podatkov. S problematiko zgostitev se na primer ukvarjajo *Elmqvist* in sodelavci [48].

Delnim ali lokalnim prikazom je prav tako skupno, da ne prikazujejo celotne množice podatkov. Pri prikazih iz te skupine zajemamo posamezne dele podatkov, vendar jih, strogo gledano, prikažemo z vsemi podrobnostmi. Gre za prikaze izsekov. Na primer 2-okolico nekega vozlišča v poljubno velikem omrežju stopnje do 10 lahko na papirju A4 prikažemo z vsemi podrobnostmi.

Interaktivnost nam omogoča, da zgornja dva postopka prepletemo oz. združimo – vzporedni prikazi. Prikaze s povzetki lahko uporabimo za globalni pregled nad podat-

ki, podrobnosti pa opazujemo z delnimi prikazi. Pri tem je zaželeno in pomembno, da so meje delnega prikaza vidne na globalnem prikazu – povezave med prikazi. Na primeren način seveda lahko vpeljemo več nivojev abstrakcije oz. nivojev podrobnosti, ki se interaktivno prelivajo glede na stopnjo povečave, s katero podatke opazujemo. Po potrebi lahko dodamo še začasne sestavine, kot so razne oznake, pojasnila, zastavice ipd. Z interaktivnostjo razbremenimo opazovalca, saj zmanjšamo njegovo kognitivno obremenitev, podatke oz. njihovo informacijo "oživimo", sam postopek pregledovanja pa z interaktivnostjo postane bolj prijeten in omogoča enostavnejše dojemanje podatkov v celoti [49].

Ker se naše delo pretežno nanaša na prikaze hierarhij v omrežjih, bomo v nadaljevanju predstavili nekaj uveljavljenih in pogostih primerov prikazovanja drevesnih hierarhij [50], nato pa predstavili načrt postopka za prikaz osnovnih oz. splošnih hierarhij. Formalno definicijo različnih vrst hierarhij smo podali v okviru definicij v uvodu 1.4.2.

3.2 *Pogosti prikazi drevesnih hierarhij*

V praksi pogosto srečamo podatke, ki so že po svoji naravi ali pa dodatno oz. umetno urejeni hierarhično. Naravne hierarhije so na primer urejenost občin v pokrajine, pokrajine v države in le-te naprej po kontinentih in v svetu. Vsebina v strokovni literaturi je navadno organizirana hierarhično: odstavek, podrazdelek, razdelek, poglavje, zvezek, knjiga. Nekateri podatki na videz niso urejeni hierarhično, vendar jih kljub temu lahko v hierarhijo razvrstimo. Avtorje v bibliografskih omrežjih lahko hierarhično uredimo glede na podobnost množic ključnih besed, ki jih uporabljajo v svojih delih (2.4.7). Interaktivni zemljevidi v digitalni obliki so navadno urejeni hierarhično, saj bi celotna karta zasedla preveč prostora, da bi jo lahko obravnavali v enem kosu, hkrati pa se tudi prikaz na posameznih nivojih lahko razlikuje. Pri umetnih umestitvah v hierarhijo število nivojev ni nujno vnaprej določeno.

Za prikazovanje hierarhij so bili razviti mnogi postopki, ki nam omogočajo pregledovaje na več nivojih. Ideja je v tem, da podatke lahko opazujemo na visokem makro nivoju, po potrebi pa se poglobimo v podrobnosti na nižjih mikro nivojih oz. do posameznih elementov v listih. Večina hierarhij je drevesnih, vse bolj pa se kažejo potrebe po podpori za pregledovanje splošnih hierarhij.

V prvih treh razdelkih (3.2.1, 3.2.2, 3.2.3) v nadaljevanju opišemo nekaj osnovnih prikazov drevesnih hierarhij. Slike smo povzeli iz vzorcev knjižnice *Prefuse Flare* [51]. Gre za knjižnico *ActionScript* za vizualizacijo podatkov v predvajalniku *Adobe Flash Pla-*

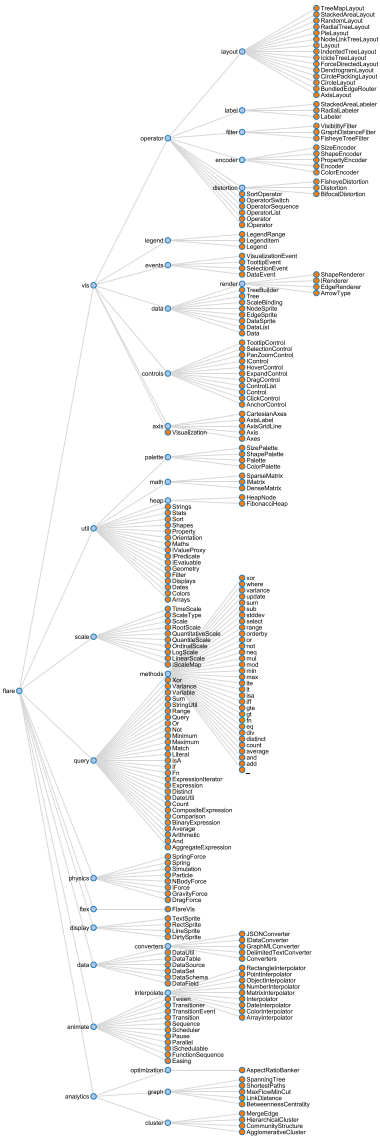
yer. Tu omenimo še podobno odprto-kodno knjižnico *Data Driven Documents* (D3) [52], ki omogoča pripravo podobnih prikazov in še mnogo več. Temelji na posebnem opisnem (deskriptivnem) jeziku, s katerim se da do potankosti opisati poljuben prikaz. Podobno kot *Prefuse Flare* je knjižnica *Data Driven Documents* zasnovana za rabo na spletu. Njena velika prednost je v odlični podpori za interaktivno rabo. Zanimiva spletna storitev na področju vizualizacij je tudi *Many Eyes* [53]. Po sposobnostih je primerljiva s prejšnjima dvema, poleg ustvarjanja prikazov pa omogoča še spletno gostovanje izdelanih prikazov. V naslednjih dveh razdelkih (3.2.4, 3.2.5) se osredotočimo na nekoliko bolj napredna postopka za prikazovanje drevesnih hierarhij. V primerjavi s prejšnjimi sta bolj primerna za interaktivno pregledovanje hierarhij na računalniškem zaslonu. V razdelku 3.2.6 opišemo primer postopka, ki prikaze drevesnih hierarhij dopolnjuje s prikazi morebitne povezanostne strukture med končnimi vozlišči hierarhije, t.j. med elementarnimi podatki, ki so v hierarhijo razvrščeni.

3.2.1 Drevo z vozlišči in povezavami

Termina drevo in hierarhija v praksi večkrat zamenjujemo, čeprav ne predstavljata nujno enake strukture. Kasneje, v razdelku 3.3 bomo videli, da termin hierarhija v splošnem označuje strukturo, kateri ustreza aciklični graf. V kolikor je hierarhija drevo, zanjo lahko uporabimo ravninski dvo-razsežen prikaz. Primer drevesa vidimo na sliki 3.1. V konkretnem primeru gre za prikaz drevesa, katerega veje so bile razvrščene z *Reingold-Tilford*-ovim algoritmom [54], ki rezultat optimizira tako, da le-ta porabi čim manj prostora. Vozlišča v drevesu razporedi tako, da se le-ta glede na njihovo globino v hierarhiji nahajajo v nivojih, kar pomeni, da so lahko listi v prikazu na različnih nivojih.

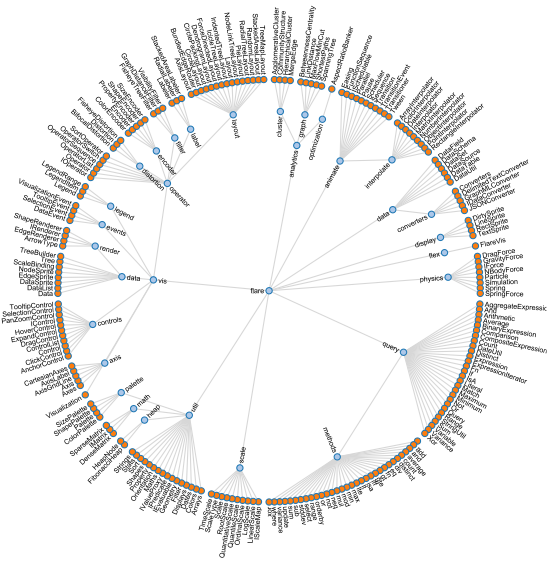
Na sliki 3.2 je drevo s slike 3.1 prikazano še v krožni obliki. Kot vidimo, so v njem listi drevesa (oranžna vozlišča) prikazani na zunanji krožnici, t.j. na istem nivoju. Ta lastnost je značilna za alternativni prikaz, ki ga poznamo pod imenom drevo razvrstitev ali t.i. dendrogram.

Primer bolj običajnega prikaza dendrograma najdemo na sliki 6.8. Za razliko od prikaza na sliki 3.2, kjer so vsa vmesna vozlišča od zunanje krožnice v notranjost pomaknjeni sorazmerno glede na dolžino poti do njihovih listov, se vmesna vozlišča v dendrogramu običajno nahajajo med korenom in listi na različnih višinah, določenih z mero različnosti med vejami danega vejišča hierarhije.



Slika 3.1

Primer klasičnega prikaza drevesa oz. hierarhije z vozlišči in povezavami.



Slika 3.2

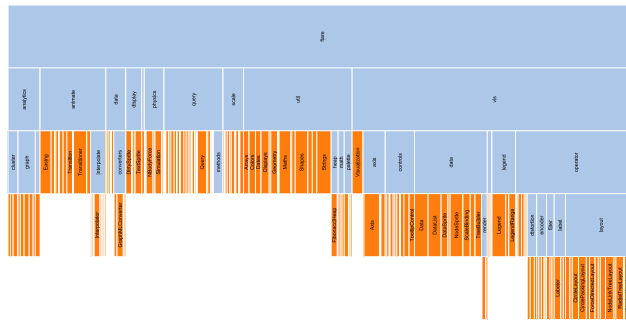
Primer krožnega prikaza drevesa oz. hierarhije z vozlišči in povezavami.

3.2.2 *Sosednostni prikazi*

Sosednostni prikazi *adjacency diagrams* se od prikazov vozlišč in povezav ne razlikujejo veliko. Med starši in potomci v hierarhiji se prostor, ki je namenjen povezavam, zapolni, bodisi s pravokotniki, bodisi z odseki kolobarjev, kot je na primeru drevesa s slike 3.1 na slikah 3.3 in 3.4 prikazano. Prikaza 3.1 in 3.3 sta si podobna v tem, da je koren v obeh primerih na vrhu, spodaj pa so izrisani potomci. Ker so vozlišča v drugem primeru prikazana z zapolnjenimi pravokotniki, lahko z njihovo širino zakodiramo poljubno količino. Slednje bi bilo v prikazu z vozlišči in povezavami težje prikazali.

Na slikah 4.6 in 4.7 smo hierarhiji podali z različico sosednostnih prikazov. V našem prikazu so vsi listi enako široki. Širina vejišč ustreza vsoti širin potomcev in robov, ki služijo estetskim namenom. Njihova višina ni enakomerna. Prilagojena je različnosti skupin, ki ustrezajo njihovim podvejitvam. Pri našem prikazu delno zasledimo tudi postopek prikaza z vsebovanostjo, ki je opisan v razdelku 3.2.3.

Če v krožnem prikazu vozlišč in povezav zapolnimo prostore, ki sicer ustrezajo povezavam, dobimo krožno različico sosednostnega prikaza. Na sliki 3.4 je podan takšen prikaz. Nivo njegovih listov in notranja vozlišča, za razliko od prikaza na sliki 3.2,



Slika 3.3

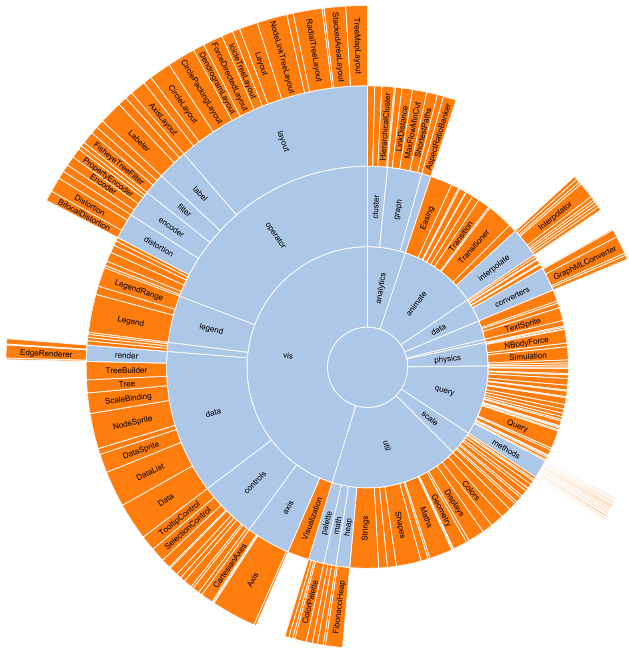
Primer klasičnega pravokotnega sosednostnega prikaza hierarhije z zapolnjenimi vejitvami.

tokrat glede na središče niso na prirejenih razdaljah. Vsak nivo hierarhije je glede na oddaljenost od korena predstavljen s svojim kolobarjem.

3.2.3 Prikazi z ogradami

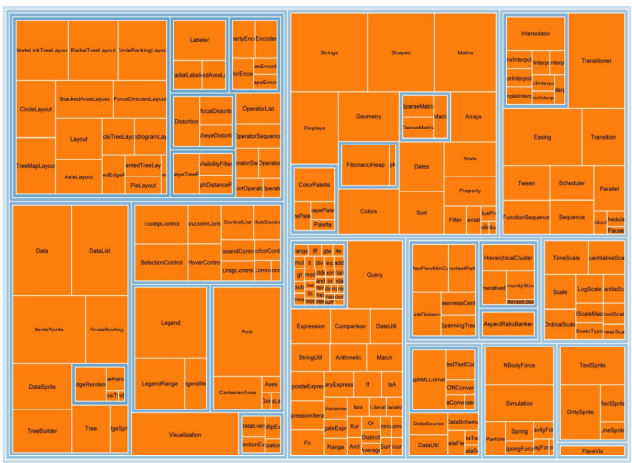
Prikazi z ogradami (*enclosure diagrams*) so podobno kot sosednostni prikazi zasnovani tako, da zapolnjujejo prostor. Namesto s postopkom sosednosti se v njih hierarhija izraža z vsebovanostjo. Pod imenom gnezdene ograde (*tree-map*) [55] sta jih uvedla *Johnson* in *Shneiderman*. Postopek deluje tako, da se začetni pravokotnik rekurzivno deli na manjše pravokotnike, ki predstavljajo vejišča. Razdelitev pravokotnika starša ni nujno enakomerna, pravokotnik se lahko deli skladno s pomembnostjo posameznega dela. Primer pravokotnega prikaza z gnezdenimi ogradami je podan na sliki 3.5. Kot je s svetlo modrimi obrobami prikazano na sliki 3.5, je pred razdelitvijo pravokotnike priporočljivo obrobiti. Hierarhija tako postane bolj nazorna.
Poleg opisanega postopka z gnezdenimi ogradami redkeje srečamo še *Voronoi*-jeve in druge postopke.

Z rekurzivnim vstavljanjem krogov dobimo alternativno možnost prikazov z ogradami. Po izgledu je prikaz v tem načinu nekoliko bolj organski, a žal hkrati slabše izkorišča prostor. Primer prikaza je podan na sliki 3.6. Prazen prostor v prikazu nazorno izraža hierarhijo. Prednost pred prikazom z gnezdenimi ogradami je tudi v tem, da oko lažje loči med površinami različnih krogov, katerih površina je določena z enim parametrom (polmer), kot med površinami pravokotnikov, katerih površina je določena z dvema parametroma (višina in širina).



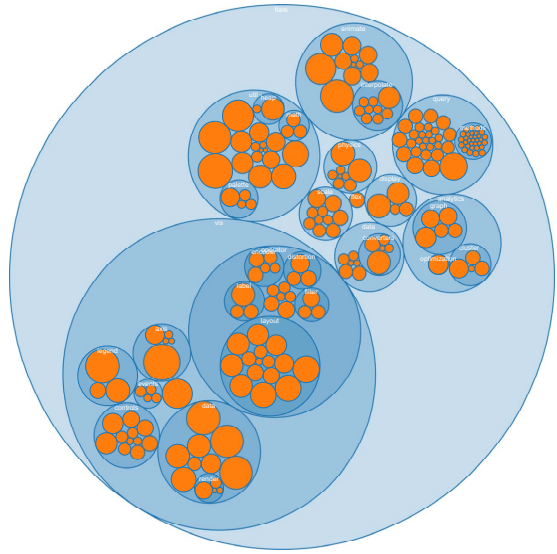
Slika 3.4

Primer krožnega sosednostnega prikaza hierarhije z zapolnjenimi vejitvami.



Slika 3.5

Primer prikaza neke hierarhije z gnezdenimi ogradami (*tree-map*).



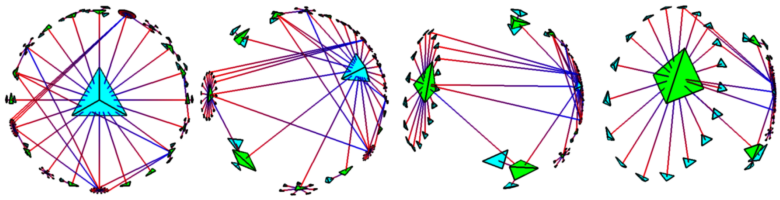
Slika 3.6

Primer prikaza hierarhije z ogradami z rekurzivnim vstavljanjem krogov.

3.2.4 Hiperbolični prikazi

Hiperbolični prikazi dreves [56], tudi hiperbolična drevesa predstavljajo eno izmed različic prikazov grafov oz. dreves. Pri prikazovanju hierarhij v obliki dreves lahko hitro zmanjka prostora, saj lahko število vej eksponentno narašča z globino. Že dvojiško drevo ima na globini n 2^n vej. Na prvi pogled so hiperbolični prikazi dreves podobni krožnim prikazom dreves 3.2.1, vendar za razliko od *Evklidskega* prostora slikajo v hiperbolični prostor. Če v *Evklidskem* prostoru drevo v celoti želimo prikazati, potrebujemo eksponentno veliko prostora, v hiperboličnem pa linearno. Če v *Evklidskem* prostoru linearno povečujemo polmer krožnice s središčem v izhodišču, njen obseg narašča linearno. V hiperboličnem prostoru bi njen obseg naraščal eksponentno. Drevesa zato mnogo lažje umestimo v hiperbolični prostor.

Pregledovanje hiperboličnih dreves je realizirano z *Möbius*-ovimi transformacijami hiperboličnega v *Evklidski* prostor. Z njimi dele drevesa v (interaktivno) izbranem žarišču povečamo, medtem ko se deli proč od žarišča pomanjšajo in pomaknejo proti robom prikaza. Navadno v žarišče postavimo izbrano vozlišče in njegova okolica postane dobro vidna. Njemu oddaljena vozlišča so potisnjena k robu. Gre pravzaprav za



Slika 3.7

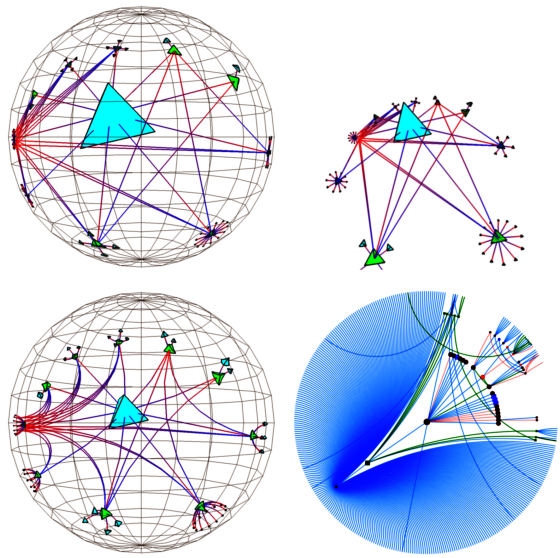
Primer spremembe pogleda na omrežje med dvema sosednjima vozliščima v projektivnem hiperboličnem prikazu "od zunaj". Vir: www.geom.uiuc.edu/docs/research/webviz/webviz/node2.html#models.

eno izmed izvedb prikazov z "ribjim očesom".

Tro-razsežne hiperbolične prikaze lahko realiziramo v notranjosti krogle. Za prikaz uporabimo ravne črte, a se razdalje ne bodo ohranjale. Površina krogle predstavlja neskončnost glede na izhodišče. Elementi blizu središča krogle oz. prikaza dajejo vtis, da so narisani v *Evklidskem* prostoru, vendar se bližje robovom hitro popačijo in pomanjšajo. Takšnemu prikazu hiperboličnega prostora pravimo projektivni ali tudi *Klein*-ov prikaz hiperboličnega prostora. Prednost projektivnega prikaza je v tem, da ga lahko pripravimo z matrično preslikavo velikosti 4×4, kar je izjemno prikladno za realizacijo na standardnih modernih grafičnih karticah, katerih cevovodi temeljijo na transformacijah reda 4×4. Interaktivna programska izvedba projektivnih prikazov je tako razmeroma preprosta. Na sliki 3.7 so v projektivnem načinu podani štirje prikazi prehoda med pogledi na dve sosednji vozlišči v omrežju.

Na sliki 3.8 zgoraj levo je še enkrat prikazan primer projektivnega prikaza nekega omrežja **R**. Prikaz vsebuje informativni obris krogle, v katerem se nahaja celotni hiperbolični prostor, na katerega gledamo "od zunaj". Če se pomaknemo v notranjost *meja* hiperboličnega prostora, t.j. v kroglo, dobimo prikaz projektivnega hiperboličnega prostora s pogledom "od znotraj". Za enako smer opazovanja znotraj krogle in za omrežje **R** je slednji podan na sliki 3.8 zgoraj desno.

S preprosto transformacijo je projektivni prikaz mogoče predelati v t.i. konformni ali *Poincaré*-ov prikaz [57]. Za omrežje **R** je podan spodaj levo na sliki 3.8. V konformnem prikazu so ravne črte prikazane s krožnimi loki, ravne ploskve pa z odseki površine krogel. Velikost kotov se ohranja. Za konformni prikaz preslikava z matriko 4×4 žal ne zadošča. Za dosego izrisa ukrivljenih elementov je le-te potrebno večkrat subdividirati. Namesto subdividiranja lahko uporabimo tudi zlepke. Na primeru nekega drugega omrežja je spodaj desno na sliki 3.8 prikazan primer konformnega prikaza z zlepki. Programska izvedba konformnih prikazov je nekoliko bolj zahtevna, so pa estetsko najbolj privlačni.



Slika 3.8

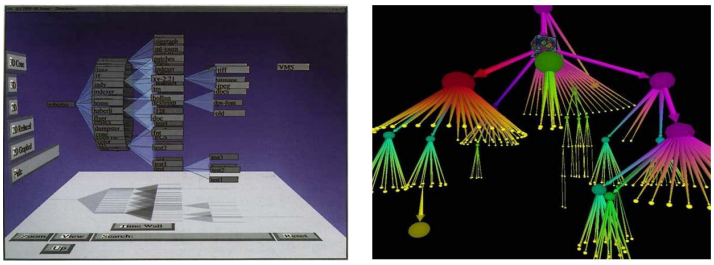
Različni hiperbolični prikazi. Zgoraj levo je primer projektivnega prikaza "od zunaj" in desno "od znotraj". Spodaj levo je podan primer konformnega prikaza "od zunaj". Na obeh prikazih "od zunaj" sta v obliki krogel informativno izrisani meji hiperboličnega prostora. Spodaj desno je podan primer konformnega prikaza alternativnega omrežja z zlepki. Vira: www.geom.uiuc.edu/docs/research/webviz/webviz/node2.html#models, www.ontology4.us/hypertree/?subject=a.

3.2.5 *Tri-razsežna drevesa stožcev*

Drevesa stožcev *cone trees* [49] lahko uporabimo za interaktivno prikazovanje hierarhij v treh razsežnostih. Hierarhijo prikažemo v obliki enako visokih stožcev. V korenu hierarhije je starš – vrh stožca prvega nivoja hierarhije. Njegovi potomci so nanizani v krogu po obodu stožca. Ti nato rekurzivno vsak zase predstavljajo stožce na nižjih nivojih v hierarhiji. Širina stožcev na višjih nivojih mora zadostiti širinam vseh stožcev potomcev. Stožci so prikazani prosojno, da poleg njih lahko vidimo še strukturo za njimi. Na sliki 3.9 levo je podan primer opisanega prikaza.

Z interaktivnim izbiranjem – s klikom poljubnega potomca v drevesu stožcev se v prikazu poskrbi, da se stožec v katerem potomec nastopa in vsi stožci njegovih staršev zavrtijo tako, da pridejo v ospredje. Vrtenje na višje ležečih nivojih je izvedeno hkratno, in sicer z animacijo po najkrajši poti vrtenja. Podobno kot pri hiperboličnih prikazih (3.2.4), so tri-razsežna drevesa stožcev še en primer pogleda z "ribjim očesom", saj v vsakem trenutku vidimo celotno strukturo hierarhije. Pri tem je izbrana veja oz. vejišče, ki ga opazujemo v žarišču oz. v ospredju.

S tri-razsežnimi drevesi stožcev je na razmeroma majhnem prostoru možno prikazati



Slika 3.9

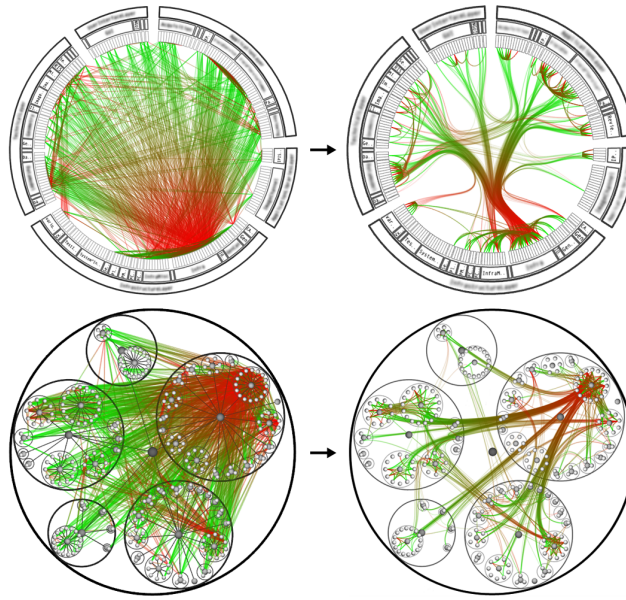
Dva primera prikazov s tri-razsežnimi drevesi stožcev. Levo: izvirni prikaz dreves stožcev [49], desno: izvedenka prikaza dreves stožcev v programskem paketu *Tulip* [58].

hierarhije z močno razvejenimi vejitvami, niso pa najbolj primerni za prikaze globokih hierarhij ali za prikazovanje na statičnih medijih. Z izrisom sence drevesa hierarhije lahko prikaz delno izboljšamo. Na sliki 3.9 desno je podan alternativni primer prikaza hierarhije s tri-razsežnimi drevesi stožcev.

3.2.6 Hierarhično združevanje povezav v snope

Pri hierarhično urejenih podatkih imamo navadno podano tudi kakšno relacijo med posameznimi podatki. Često na podlagi ene izmed takšnih relacij hierarhijo pravzaprav sestavimo, lahko pa so podatki v hierarhijo razvrščeni vnaprej. Na primeru bibliografskih omrežij lahko dela razvrstimo v hierarhijo po oddelkih, inštitutih in univerzah, kjer so le-ta nastala. Dobljeno hierarhijo lahko prikažemo z enim od opisanih postopkov, vendar bi dodatno želeli prikazati še povezave med deli, ki na primer predstavljajo citate drugih del. Za ta namen je smiselno uporabiti t.i. tehniko hierarhičnega združevanja povezav v snope (*hierarchical edge bundling*) [59]. Tehnika temelji na običajnih prikazih hierarhične strukture urejenosti vozlišč. Če bi v skladu z relacijo med posameznimi vozlišči (citiranje), le-ta preprosto povezali z daljicami, bi prikaz hitro postal nepregleden in zasičen. Pri hierarhičnih prikazih s snopi povezav povezave zato raje združujemo na način kot na primer žice v električnih vezjih in sistemih. Električne povezave (žice) med deli sistema, katerih izvori in ponori se nahajajo blizu skupaj, združimo v snope. Po izvorih žice združimo v snope, žice v snopih nato vodijo v bližino njihovih ponorov in se tam ponovno razvejijo. Pri oblikovanju snopov povezav upoštevamo hierarhijo krajišč oz. povezav, v katero so urejene. Namesto z daljicami povezave v snopih navadno narišemo z zlepki, kontrolne točke zlepkov pa določimo glede na hierarhijo urejenosti krajišč oz. povezav.

Prednosti hierarhičnega združevanja povezav v snope so naslednje:



Slika 3.10

Dva primera uporabe hierarhičnega združevanja povezav v snope. Zgoraj: uporaba postopka na obrnjenem krožnem sosednostnem prikazu hierarhije, spodaj: uporaba postopka na prikazu hierarhije z ogradami z rekurzivnim vstavljanjem krogov. Slike smo povzeli iz članka avtorjev postopka [59] in jih ustrezno priredili.

- postopek lahko uporabimo na večini obstoječih postopkov za prikazovane hierarhično urejenih podatkov,
- s postopkom se zmanjša zasičenost prikazov omrežij in
- postopek zagotavlja intuitiven in robusten način združevanja povezav. Za nižje nivoje hierarhije je izbrana moč združevanja manjša, na višjih pa večja.

Na sliki 3.10 sta podana dva primera uporabe hierarhičnega združevanja povezav v snope. Zgornji primer ilustrira uporabo postopka na obrnjenem krožnem sosednostnem prikazu hierarhije, opisanem v 3.2.2. Spodnji primer ilustrira uporabo postopka na prikazu hierarhije z ogradami z rekurzivnim vstavljanjem krogov, ki smo ga opisali v razdelku 3.2.3.

3.3 Prikaz in preoblikovanje splošnih hierarhij

Algoritem za razvrščanje povezav, opisan v poglavju 4.4.1, na vhodnem povezanem omrežju ustvari drevesno hierarhijo nad povezavami. Deluje z združevanjem skupin

povezav, začenši z vsako povezavo v svoji skupini. Izhodna oblika hierarhije, ki jo ustvari algoritem, je dvojiško usmerjeno drevo (povezave kažejo na starša) z utežmi na vejiščih. Vsak list predstavlja skupino, ki vsebuje posamezno povezavo v izvornem omrežju, vsako vejišče pa skupino, ki vključuje skupine oz. povezave skupin njegovih vej. Korensko vejišče predstavlja skupino, ki vključuje vse povezave. Utež na posameznem vejišču predstavlja različnost med skupinami, ki jih predstavljajo njegove veje. Listi imajo utež 0.

S postopkom ploščenja hierarhije (4.8.1) v dobljenem drevesu (hierarhiji) vejišča na zaporednih nivojih (globinah) z enako vrednostjo uteži še združimo v eno samo vejišče. Na ta način dobimo drevo, katerega stopnje vejišč so večje ali enake 2. Ker v omrežju brez zank, ki predstavlja vhod za naš algoritem, vsaka povezava predstavlja dve vozlišči, to pomeni, da vsakemu listu drevesa pripadata dve vozlišči. Vsako vozlišče, ki je v vhodnem omrežju na preseku k povezav, je v drevesu od korena dostopno v k listih. Če želimo takšno hierarhijo obravnavati na enak način kot hierarhijo vozlišč, ki nam jo vrne osnovni algoritem, opisan v razdelku 4.3, imamo dve možnosti:

- Na množici povezav, ki jo razširimo z dodatnim nivojem krajišč povezav, uporabimo drevesno hierarhijo; nivo, ki ga dodamo, vsaki povezavi na dnu hierarhije (do zdaj listu) za vsako krajišče povezave, t.j. vozlišče, dodeli novo vejo. Ker ne želimo, da bi se vozlišča v novih listih podvajala, naj na zadnjem nivoju vsako vozlišče nastopa samo enkrat. Enako kot v drevesni hierarhiji osnovnega algoritma. Če vozlišče – krajišče nastopa v več povezavah vhodnega omrežja, bo iz korena hierarhije dosegljivo prek več alternativnih poti. Struktura, ki zadošča predstavitvi takšne hierarhije, je aciklični usmerjeni graf. Ker v pregledovalniku želimo pregledovati hierarhije povezav kot tudi iz njih izpeljane hierarhije vozlišč, bomo za osnovo predstavitve hierarhije vzeli aciklični graf, ki ga označimo z $\mathbf{G} = (\mathbf{W}, \mathbf{A}, \tau)$. $\mathbf{W}$ predstavlja množico vozlišč v grafu, posredno torej skupin povezav oz. vozlišč, ki pripadajo povezavam. $\mathbf{A}$ predstavlja množico povezav, ki v hierarhiji vedno kažejo na starša. Starš predstavlja skupino unije povezav oz. vozlišč vhodnega omrežja, ki ustrezajo posameznim skupinam njegovih potomcev. τ je preslikava, ki vsakemu vozlišču iz $\mathbf{W}$ določa opis D: $\tau : \mathbf{W} \rightarrow D$. Kot je zapisano v poglavju 4, so to lahko na primer množice ključnih besed oz. njihove frekvence.

  Množico $\mathbf{W}$ nadalje razdelimo na ločeni množici skupin $\mathbf{C}$ in terminalov $\mathbf{V}$:

$\mathbf{W} = \mathbf{C} \cup \mathbf{V}$, $\mathbf{C} \cap \mathbf{V} = \emptyset$, povezavo v $\mathbf{A}$ pa zapišemo kot $(v_1, v_2) \in \mathbf{A} \equiv v_2 \in \mathbf{C} \wedge ((v_1 \in \mathbf{C} \wedge v_1 \sqsubseteq v_2) \vee (v_1 \in \mathbf{V} \wedge v_1 \in \text{ext}(v_2)))$. Povezave v $\mathbf{A}$ določajo hierarhijo vsebovanosti elementov skupin. Za terminale je τ že podan v vhodnem omrežju, na skupinah $\mathbf{C}$ pa ga lahko določi uporabnik.

Da $\mathbf{G}$ predstavlja hierarhijo, mora biti acikličen in hkrati v normalni obliki. $\mathbf{G}$ je v normalni obliki, če obstajata en sam izvor s in en sam ponor t tako da: $\mathbf{W}' = \mathbf{W} \cup \{s, t\}$ in $\mathbf{A}' = \mathbf{A} \cup \{(s, s_i) : s_i \in \mathbf{W} \wedge \text{indeg}(s_i) = 0\} \cup \{(t_i, t) : t_i \in \mathbf{W} \wedge \text{outdeg}(t_i) = 0\}$, pri tem pa je $\mathbf{G}'$ definiran na $\mathbf{W}'$ in $\mathbf{A}'$ povezan in acikličen. Glejte tudi definicijo osnovne hierarhije v razdelku 1.4.2.

- Uporabimo osnovno hierarhijo nad skupinami vozlišč, ki pripadajo posameznim povezavam, razširjeno z dodatnim nivojem posameznih vozlišč; $\{\mathbf{C}_i = V(\mathbf{L}_i)\}$ in dodatni nivo $\{\{v\} : v \in \mathbf{V}\}$. Zadnji nivo hierarhije v tej alineji, t.j. listi, ustreza zadnjemu nivoju hierarhije v prejšnji alineji. Vmesna vejišča hierarhije v tej alineji so podobna vmesnim vejiščem v zgornji alineji, le da namesto skupin povezav tu vsebujejo povezavam pripadajoča vozlišča.

Oba načina predstavitve rezultatov prilagojenega algoritma in tudi rezultate osnovnega algoritma je torej možno predstaviti z osnovnimi hierarhijami vozlišč.

Nad skupinami iz razvrstitve $\mathscr{H}$ vpeljemo relacijo neposrednega potomca $\mathbf{C}_i \sqsubset \mathbf{C}_j$ ali z besedami: $\mathbf{C}_i$ je neposredni potomec (otrok) $\mathbf{C}_j$, s predpisom:

$$\mathbf{C}_i \sqsubset \mathbf{C}_j \equiv \mathbf{C}_i \subset \mathbf{C}_j \wedge \neg\exists \mathbf{C}_k \in \mathscr{H} : \mathbf{C}_i \subset \mathbf{C}_k \subset \mathbf{C}_j. \tag{3.1}$$

Ker je relacija $\subset$ aciklična, je aciklična tudi relacija $\sqsubset$. Tedaj je razvrstitev $\mathscr{H}$ popolna hierarhija natanko tedaj, ko velja $\forall \mathbf{C}_i \in \mathscr{H} : (\exists \mathbf{C}_j \in \mathscr{H} : \mathbf{C}_i \sqsubset \mathbf{C}_j \Rightarrow \exists \mathscr{G} \subseteq \mathscr{H} : (\forall \mathbf{C} \in \mathscr{G} : \mathbf{C} \sqsubset \mathbf{C}_j \wedge \cup \mathscr{G} = \mathbf{C}_j))$. Skupina $\mathbf{C}_v \in \mathscr{H}$ je vrh hierarhije $\mathscr{H}$ natanko tedaj, ko $\forall \mathbf{C} \in \mathscr{H} : \mathbf{C} \sqsubset^* \mathbf{C}_v$, pri čemer je $\sqsubset^*$ tranzitivna in refleksivna ovojnica relacije $\sqsubset$.

V popolni hierarhiji $\mathscr{H}$ za vsak par skupin $\mathbf{C}_i$, $\mathbf{C}_j$, pri čemer velja $\mathbf{C}_i \subset \mathbf{C}_j$, obstaja vsaj ena veriga $\mathbf{C}_i = D_0 \sqsubset D_1 \sqsubset .. \sqsubset D_k = \mathbf{C}_j$, tako da so vsi $D_s \in \mathscr{H}$, $s = 0, 1, .., k$. V drevesni hierarhiji je za vsak par $\mathbf{C}_i \subset \mathbf{C}_j$ veriga enolično določena.

Zanimale nas bodo predvsem osnovne, polne in popolne hierarhije z vrhom. Naštejmo primere operacij na takšnih hierarhijah, ki naj jih urejevalnik hierarhij omogoča:

- odpiranje vejišča – približevanje (*zoom-in*),

- zapiranje vejišča – oddaljevanje (*zoom-out*),
- odpiranje vejišča v ločenem pogledu za primerjavo,
- prikaz staršev vejišča,
- prikaz potomcev vejišča,
- izpis velikosti preseka paru izbranih vejišč pripadajočih skupin $\mathbf{C}_1$ in $\mathbf{C}_2$ in elementov,
- nastavitve lastnosti vejišča (ime, utež),
- prikaz informacij o vejišču:
  - ime,
  - velikosti,
  - oznake,
  - opombe,
  - globine vejišča – dolžina najdaljše poti od izvora,
  - izpis števila elementov skupine, ki pripada vejišču,
- združevanje skupin izbranih vejišč,
- razdruževanje skupin izbranega vejišča,
- premikanje vejišča v druga vejišča,
- označevanje vejišč, skok na označeno vejišče,
- prikaz sledi iskanja,
- zrcaljenje oz. preurejanje podvej v izbrani veji ter
- topološko urejanje oz. test acikličnosti.

Opisane operacije omogočajo interaktivno pregledovanje in preoblikovanje splošnih hierarhij. Poleg tega, da je opisan postopek možno uporabiti na splošnih hierarhijah, se od postopkov prikaza drevesnih hierarhij (3.2) razlikuje v tem, da je pravzaprav sestavljen iz večih različnih prikazov, ki jih medsebojno interaktivno povezuje in dopolnjuje. Postopek omogoča pregledovanje velikih hierarhij.

Predlagan postopek prikaza za pregledovanje in urejanje splošnih hierarhij smo delno programsko izvedli v našem orodju za analizo velikih omrežij *net.Plexor* (6.4.3). V prihodnosti bomo prikaz dogradili in izpopolnili, trenutna izvedba pa je predvsem namenjena pregledovanju hierarhij, izdelanih z algoritmoma za razvrščanje, ki sta predmet poglavja 4. Skupaj s kompleksnim uporabniškim vmesnikom s prikazi omrežij in hierarhij ter z razvrščanjem gradnikov omrežja je (bo) tako realiziran splošno namenski sistem za interaktivni vpogled v strukturo velikih omrežij na več nivojih.

4

Hierarhično razvrščanje v omrežjih

4.1 Uvod

Hierarhično razvrščanje omrežij je postopek s katerim iščemo notranje povezane skupine (gruče oz. združbe) vozlišč v omrežju. Skupina je poljubna neprazna podmnožica elementov (vozlišč ali povezav) omrežja. Idejo razvrščanja v omrežju lahko prilagodimo na razvrščanje povezav v omrežju. Namesto skupin vozlišč iščemo skupine povezav. Pri tem načinu se skupine vozlišč oz. krajišč povezav lahko prekrivajo. Čeprav pristop s prekrivajočimi skupinami vozlišč ni vedno naravna izbira, ga lahko uporabimo na mnogih področjih.

V tem poglavju bomo predstavili dva hierarhična algoritma za razvrščanje vozlišč oz. povezav v neusmerjenih omrežjih. Algoritem za razvrščanje povezav je nov. Obstoječi algoritmi za razvrščanje povezav temeljijo na enostavnih merah različnosti in izkoriščajo le strukturne lastnosti omrežij. Opisana algoritma dodatno upoštevata še lastnosti (opise, atribute) vozlišč in povezav. Pri tem je drugi na ustrezen povezavni graf (*line graph*) vezan le implicitno, s čimer je njegova zahtevnost, prostorska in časovna, manjša.

Glede na to, da obstaja veliko različnih pomenov združbe [60], bomo za naše potrebe privzeli, da je združba skupina elementov omrežja, ki so glede na njihove opise v sorodu, hkrati pa v danem omrežju porodijo povezano podomrežje. V besedilu večinoma uporabljamo pojem skupine, vendar zaradi delovanja algoritmov ta skoraj univerzalno predstavlja združbo. Struktura združbe naj bi hkrati bila tudi kompaktna in interno močneje povezana kot navzven oz. med različnimi združbami. V primeru prekrivajočih združb se sicer lahko zgodi, da je povezanost navzven močnejša. Zaradi slednjega so *Lehmann* in sodelavci [1] predlagali postopek za odkrivanje prekrivajočih združb, in sicer z razvrščanjem povezav namesto vozlišč. Na vhodnem omrežju naj algoritem zgradi hierarhijo povezav namesto vozlišč. Alternativno metodo, ki temelji na sorodni ideji, so predlagali *Lambiotte* in sodelavci [61]. Njihov postopek temelji na uporabi poljubnega klasičnega algoritma za razvrščanje vozlišč na povezavnem grafu vhodnega omrežja. V povezavnem grafu so povezave izvornega grafa oz. omrežja predstavljene z vozlišči; par vozlišč je v povezavnem grafu povezan natanko tedaj, ko imata v izvornem omrežju pripadajoči povezavi skupno krajišče. Skupina vozlišč v povezavnem grafu določa skupino povezav v izvornem grafu omrežja. Obe metodi poiščeta ločene skupine povezav, ki določajo prekrivajoče skupine vozlišč (krajišč povezav) v izvornem omrežju.

Oba algoritma temeljita na združevalnem (aglomerativnem) razvrščanju vozlišč oz. povezav, in sicer v skladu z relacijsko omejitvijo povezanosti – združujeta samo sosednje skupine vozlišč ali povezav in tako določita povezane skupine (podomrežja). Osnovni algoritem združuje sosednje skupine vozlišč, prilagojeni pa skupine povezav, ki imajo vsaj eno skupno vozlišče. Pri obeh algoritmih lahko uporabljamo tako rekurzivno izračunane, kot tudi direktno izračunane mere različnosti, ki upoštevajo lastnosti vozlišč oz. povezav. Rekurzivno izračunane mere različnosti upoštevajo samo različnosti med začetnimi skupinami, ki sestojijo iz posameznih vozlišč ali povezav. Različnost sestavljenih skupin se rekurzivno izračuna iz različnosti skupin pred združitvijo. Direktno izračunane mere različnosti upoštevajo lastnosti vozlišč in povezav, ki pripadajo sestavljenim skupinam, njihovo vrednost pa se vsakič izračuna "od začetka". Direktno izračunane mere različnosti upoštevajo lastnosti skupin kot celote. Rezultat obeh algoritmov so hierarhije skupin. Osnovni vrne hierarhijo vozlišč, prilagojeni pa hierarhijo povezav. Na izbranem nivoju lahko iz hierarhij vozlišč izločimo razbitja neprekrivajočih skupin vozlišč. V primeru hierarhij povezav lahko na enak način izločimo razbitja neprekrivajočih skupin povezav, katerim pa pripadajo prekrivajoče skupine vozlišč (krajišč povezav). Tako prekrivajoče, kot neprekrivajoče skupine vozlišč predstavljajo združbe vozlišč oz. povezav, katerih lastnosti so podobne. Čeprav je pri obeh algoritmih možno uporabiti različnosti, ki temeljijo na poljubni merski lestvici, smo se osredotočili na različnosti, ki temeljijo na imenskih lestvicah (ključne besede, oznake).

Prilagojeni algoritem smo smiselno uporabili na primeru omrežja sodelovanj avtorjev. Za vsakega avtorja smo poiskali nabor ključnih besed, ki jih uporablja v svojih delih, nato pa na podlagi podobnosti množic ključnih besed (opisov vozlišč) z algoritmom identificirali podrobnejšo strukturo omrežja sodelovanj. Poleg tega, da algoritem odkrije skupine sodelujočih avtorjev, lahko zaznamo tudi področja na katerih avtorji delujejo in podskupine avtorjev, ki delujejo na bolj sorodnih področjih. Drugi primer predstavlja omrežje skupaj kupljenih/ogledovanih izdelkov, ki so na voljo v spletni trgovini. Sestavili smo ga na podlagi relacije "ostali so kupili/izbirali izdelke", ki jo na spletnih straneh izdelkov najdemo v obliki povezav na sorodne izdelke. Na podlagi opisov izdelkov smo z algoritmom identificirali skupine *najbolj* povezanih in združljivih izdelkov. Kupec bi nemara pregledoval sorodne vendar nezdružljive izdelke. Imejmo na primer kupca, ki izbira med dvema fotoaparatoma, prvega z režo za pomnilniško kartico *SD* in drugega z režo za kartico *CF*. Kupec naj izbira tudi med pomnilniškima karticama obeh vrst. Ker izdelki bolj ali manj ustrezajo isti skupini, bi bilo omrežje

izbiranja danih izdelkov najverjetneje polno. Če bi takšno omrežje brez opisov posredovali algoritmu, bi zaznal le eno skupino povezanih izdelkov. Če mu posredujemo še opise, lahko loči med dvema nezdružljivima podskupinama.

Za analizo algoritmov smo uporabili omrežja, ki so jih pripravili *Yang* in sodelavci [62]. Ker njihova omrežja ne vsebujejo informacij o dodatnih lastnostih, smo nekaj večjih dodatnih omrežij pripravili sami (glejte poglavje 2). Oba algoritma lahko uporabimo tudi na omrežjih za katera nimamo dodatnih opisov. Za vsako vozlišče na primer predpostavimo, da ima enoličen opis. Algoritma se na takšnih omrežjih obnašata tako, da dajeta prednost združevanju skupin vozlišč/povezav na način, da se število vozlišč/povezav v združenih skupinah čim počasneje povečuje. Posledično algoritma gradita hierarhije tesno povezanih skupin.

V naslednjem razdelku izpostavimo nekaj sorodnih del. V razdelku 4.3 opišemo hierarhični združevalni algoritem za razvrščanje vozlišč in ga v poglavju 4.4 prilagodimo za združevanje povezav. V razdelku 4.5 za oba algoritma vpeljemo primerne mere različnosti skupin. V razdelku 4.6 za mere različnosti podamo ustrezne dokaze monotonosti. V razdelku 4.7 se ukvarjamo s časovno in prostorsko zahtevnostjo obeh algoritmov, v razdelku 4.8 pa algoritma uporabimo na konkretnih omrežjih in ovrednotimo njune rezultate.

O programski izvedbi algoritma pišemo v razdelku 6.4.2 poglavja 6. Algoritem smo realizirali v obliki operacijskega gradnika in ga vgradili v knjižnico za delo z velikimi omrežji *net.Plexor*.

4.2 Sorodna dela

V *Fortunato*-vem preglednem članku [42] lahko najdemo obsežen pregled metod razvrščanja. Avtor poda obsežno razpravo; od osnovne definicije problema razvrščanja do podrobnih opisov eksotičnih metod razvrščanja v omrežjih. Poseben poudarek namenja metodam, ki so jih za svoje potrebe razvili fiziki. Zastavi in odgovori na vprašanja o pomembnosti razvrščanja v praksi, poda informacije, kako metode razvrščanja ovrednotiti, kako jih primerjati med seboj ter opiše primere uporabe algoritmov razvrščanja na realnih omrežjih. Najboljše tehnike določanja skupin so opisane tudi v delu *J. Xie*-a in sodelavcev [63]. Avtorji definirajo metrike kvalitete in učinkovitosti ter jih podrobno opredelijo za štirinajst različnih algoritmov. Algoritme razvrstijo glede na naravo pristopa, kot so razširjanje po polnih podgrafih (*clique percolation*) [64], postopek lokalne širitve (*local expansion*) in optimizacija [65, 66], algoritem oz. postopek mehkega

odkrivanja (*fuzzy detection*) [67], agentski dinamični postopek oz. algoritem (*agent based dynamic algorithm*) [68] ter na postopke za razvrščanje povezav v povezavnem grafu, ki so sorodni naši metodi. Nedavno je *Yang* s sodelavci objavil članek [62], ki naslavlja problem ovrednotenja metod za zaznavanje skupin. Številne postopke so primerjali glede na podobnost skupin, ki jih metode na dejanskih omrežjih najdejo. Rezultate primerjajo tudi z vnaprej znano strukturo skupin v omrežjih.

Metoda zaznavanja skupin, ki je botrovala nastanku našega prispevka je med drugimi t.i. *Louvain*-ska metoda [69]. Gre za enostavno in učinkovito metodo zaznavanja skupin v velikih omrežjih. Lahko jo je programsko izvesti, njena osnovna ideja pa je v požrešni optimizaciji modularnosti razvrstitve omrežja [70]. Naše delo temelji tudi na delu *Ferligoj*-eve in *Batagelj*-a [71], ki sta standardne metode združevalnega razvrščaja prilagodila, da delujejo z relacijskimi omejitvami. Hiter algoritem za razvrščanje na velikih omrežjih je namreč smiselno zasnovati tako, da upošteva oz. se omeji na obstoječo povezanost omrežja [72]. V njunem drugem članku [73] metode razširita na nesimetrične relacije. Zanimivo metodo je razvil tudi *G.-J. Kim* [74]. Pri zaznavi skupin povezanih spletnih strani med izvajanjem razvrščanja, ki podobno kot naš prilagojeni algoritem, temelji na razvrščanju povezav, upošteva semantične lastnosti. V svojem delu opiše postopek, kako na inteligenten način določiti in pregledati omejeno in semantično zaokroženo množico spletnih strani. Ključno delo, na katerega se opira naš prilagojeni algoritem, je še delo *F. Murtagh*-a [76], ki v svoji knjigi predstavi postopke večrazsežnostnega hierarhičnega razvrščanja. Algoritem, na katerega se v nadaljevanju opiramo, bomo navajali pod *Murtagh*-ovim imenom, vendar nam ni povsem znano, ali je *F. Murtagh* dejanski avtor algoritma.

4.3 Razvrščanje vozlišč omrežja z združevanjem

Osnovni algoritem sledi klasičnemu združevalnemu postopku za razvrščanje. Poleg različnosti upošteva dodatno relacijsko omejitev; išče skupine v katerih morajo biti vozlišča povezana [71]. Hierarhijo vozlišč, ki jo algoritem zgradi, lahko na primer interpretiramo kot večnivojsko abstrakcijo podobnih povezanih delov v omrežju. Postopek lahko na primer uporabimo za določanje bližnjic med podobnimi povezanimi deli v omrežju in posledično za optimizacijo postopkov iskanja v njem [75].

4.3.1 Osnovni algoritem

Vhodno omrežje osnovnega algoritma označimo z $\mathbf{N}$. Predpostavili bomo, da je povezano, neusmerjeno in brez zank (povezav z istim začetnim in končnim vozliščem). Definicija omrežja je podana v razdelku 1.4.2, vendar ker gre za neusmerjeno omrežje, bomo njegove množice označili drugače: $\mathbf{N} = (\mathbf{V}, \mathbf{E}, \mathbf{P}, \mathbf{W})$. Razlika je v množici povezav $\mathbf{E}$, ki so tu izključno neusmerjene (*edges*). Množica $\mathbf{V}$ je množica vozlišč in skupaj z množico $\mathbf{E}$ tvorita neusmerjen graf $\mathbf{G} = (\mathbf{V}, \mathbf{E})$. Množici lastnosti vozlišč $\mathbf{P}$ in povezav $\mathbf{W}$ graf razširjata v omrežje. Številnost množice vozlišč bomo označili z $n = |\mathbf{V}|$ in številnost množice povezav z $m = |\mathbf{E}|$. Neprazna podmnožica $\mathbf{C}$ množice $\mathbf{V}$, $\emptyset \subset \mathbf{C} \subseteq \mathbf{V}$ je skupina vozlišč. Razvrstitev, t.j. množico skupin bomo označili s $\mathscr{C}$; $\mathscr{C} = \{\mathbf{C}_1, \mathbf{C}_2, .., \mathbf{C}_k\}$ v skladu z definicijo v razdeleku 1.4.2. Z $D(\mathbf{C}_s, \mathbf{C}_t)$ bomo označili različnost med skupinama $\mathbf{C}_s$ in $\mathbf{C}_t$ in z množico $\mathbf{D}$ (seznamom, slovarjem ali redko matriko) različnosti med skupinami v procesu algoritma. Različnost D navadno izrazimo z izbrano različnostjo d med vozlišči.

Vhod algoritma je omrežje $\mathbf{N}$ in mera različnosti D. Relacijsko omejitev označimo s $\psi(\mathbf{C}_s, \mathbf{C}_t) \equiv \exists u \in \mathbf{C}_s, v \in \mathbf{C}_t : (u : v) \in \mathbf{E}$. Izpolnjena (resnična) je natanko tedaj, ko v omrežju $\mathbf{N}$ obstaja povezava, ki veže par vozlišč, katerega vsako pripada eni od skupin $\mathbf{C}_s$ in $\mathbf{C}_t$.

$k \leftarrow 1; \mathscr{C}_1 \leftarrow \{\{v\} : v \in \mathbf{V}\}$
for all $\mathbf{C} \in \mathscr{C}_1$ *do* $h(\mathbf{C}) \leftarrow 0$
for all $(u : v) \in \mathbf{E}$ *do* $\mathbf{D}[\{u\}, \{v\}] \leftarrow d(u, v)$
while $\exists \mathbf{C}_i, \mathbf{C}_j \in \mathscr{C}_k : (i \neq j)$ *do*
  $(\mathbf{C}_p, \mathbf{C}_q) \leftarrow \text{argmin}\{\mathbf{D}[\mathbf{C}_i, \mathbf{C}_j] : i \neq j \wedge \psi(\mathbf{C}_i, \mathbf{C}_j)\}$
  $\mathbf{C} \leftarrow \mathbf{C}_p \cup \mathbf{C}_q; k \leftarrow k + 1$
  $h(\mathbf{C}) \leftarrow \mathbf{D}[\mathbf{C}_p, \mathbf{C}_q]$
  $\mathscr{C}_k \leftarrow \mathscr{C}_{k-1} \backslash \{\mathbf{C}_p, \mathbf{C}_q\} \cup \{\mathbf{C}\}$
  for all $\mathbf{C}_s \in \mathbf{C}_k \backslash \{\mathbf{C}\} : \psi(\mathbf{C}, \mathbf{C}_s)$ *do* $\mathbf{D}[\mathbf{C}, \mathbf{C}_s] \leftarrow D(\mathbf{C}, \mathbf{C}_s)$
  iz $\mathbf{D}$ odstrani vse zapise, ki vsebujejo $\mathbf{C}_p$ ali $\mathbf{C}_q$
end while

Algoritem vrne zaporedje razbitij $\mathscr{C}_1, \mathscr{C}_2, .., \mathscr{C}_n$. Njihova unija $\mathscr{H} = \cup_i \mathscr{C}_i$ je drevesna hierarhija, ki je definirana v uvodu disertacije 1.4.2. Funkcija $h(\mathbf{C})$, ki jo dobimo

poleg, je nivojska funkcija hierarhije $\mathscr{H}$. V obliki para $(\mathscr{H}, h)$ skupaj s hierarhijo $\mathscr{H}$ določata drevo razvrstitve ali dendrogram.

4.4 Združevalni postopek za razvrščanje povezav

Prilagojeni algoritem deluje podobno kot osnovni algoritem. Predvsem namesto vozlišč vhodnega omrežja v hierarhijo razvršča povezave. Mere različnosti za prilagojeni algoritem temeljijo na lastnostih vozlišč in/ali povezav in na strukturi omrežja. Pri združevanju povezav v skupine tudi prilagojeni algoritem upošteva relacijsko omejitev ter s tem ohranja povezanost skupin. Združuje le skupine povezav, katerih povezave imajo vsaj eno skupno krajišče oz. t.i. aktivno vozlišče.

4.4.1 Prilagojeni algoritem

Namesto s skupinami in razvrstitvami vozlišč pri prilagojenem algoritmu operiramo s skupinami in razvrstitvami povezav. Skupino povezav označimo z $\mathbf{L}$, predstavlja pa naj neprazno povezano podmnožico, t.j. združbo povezav množice $\mathbf{E}$; $\emptyset \subset \mathbf{L} \subseteq \mathbf{E}$. Razvrstitev povezav je množica skupin povezav, razbitje povezav pa razvrstitev povezav v ločene skupine, ki pokriva celotno množico $\mathbf{E}$, 1.4.2. Razvrstitev povezav bomo označili z $\mathscr{L}$; $\mathscr{L} = \{\mathbf{L}_1, \mathbf{L}_2, .., \mathbf{L}_p\}$, $s \neq t \Rightarrow \mathbf{L}_s \cap \mathbf{L}_t = \emptyset$ in $\bigcup_{L \in \mathbf{L}} L = \mathbf{E}$. Funkcija $\text{ext}(e) = \{u, v\}$ naj določa množico krajišč povezave $e = (u : v) \in \mathbf{E}$. Definirali bomo še skupino krajišč povezav $\mathbf{C}_t = \text{ext}(\mathbf{L}_t) = \bigcup_{e \in \mathbf{L}_t} \text{ext}(e)$. S $\mathbf{C}_t$ jo označimo, ker predstavlja skupino vozlišč, ki ustreza skupinam vozlišč v osnovnem algoritmu. Uporabili bomo tudi pomožne množice aktivnih vozlišč $\mathbf{A}_t$, $\mathbf{A}_t \subseteq \mathbf{C}_t$, ki nam olajšajo iskanje skupin, ki si aktivna vozlišča delijo – so si sosednje; $\mathbf{A}_t = \{v \in \mathbf{C}_t; \exists s \neq t : v \in \mathbf{C}_s\}$.

Vhod prilagojenega algoritma je enak vhodu osnovnega algoritma: omrežje $\mathbf{N}$ in mera različnosti D. Mera različnosti D mora biti tokrat monotona. Monotone mere različnosti opišemo v razdelku 4.5. Namesto hierarhije vozlišč omrežja $\mathbf{N}$ prilagojeni algoritem ustvari hierarhijo povezav. Relacijsko omejitev prilagojenega algoritma označimo s $\psi_a(\mathbf{L}_s, \mathbf{L}_t) \equiv \mathbf{A}_s \cap \mathbf{A}_t \neq \emptyset$, ki je enakovredna $\mathbf{C}_s \cap \mathbf{C}_t \neq \emptyset$. $\psi_a(\mathbf{L}_s, \mathbf{L}_t)$ je izpolnjena (resnična) natanko tedaj, ko si skupini vozlišč $\mathbf{C}_s$ in $\mathbf{C}_t$, ki pripadata ustreznima skupinama povezav $\mathbf{L}_s$ in $\mathbf{L}_t$, delita aktivno vozlišče. Vsebina glavne zanke algoritma je podobna osnovnemu algoritmu, delovanje pa je nekoliko drugačno. Algoritem temelji na *Murtagh*-ovem algoritmu [76], ki je v osnovi namenjen razvrščanju

vozlišč. Mi ga uporabimo na implicitno konstruiranem povezavnem grafu oz. podgrafu osnovnega omrežja. S tem ohranimo časovno zahtevnost in hkrati zmanjšamo prostorsko zahtevnost, ki bi se ob uporabi običajnega pristopa na celotnem povezavnem grafu močno povečala. Več o zahtevnosti prilagojenega algoritma je opisano v razdelku 4.7.

Ideja *Murtagh*-ovega algoritma je v tem, da trivialne skupine – vozlišča oz. sestavljene skupine združenih vozlišč omrežja razvršča lokalno, in sicer na koncu verige (poti obiskanih skupin v omrežju), ki jo iterativno gradi iz poljubno izbrane skupine vozlišč. V našem algoritmu verigo označimo s *chain*. Operacije na njej pa so *Push* za dodajanje elementa na njen konec, *Pop* za odstranjevanje njenega zadnjega elementa, ki hkrati ta element vrne in *Last*, ki le vrne zadnji element verige. Verigo predstavimo s skladom. *Murtagh*-ov algoritem verigo gradi v smeri najbolj padajoče vrednosti različnosti sosednjih skupin vozlišč, torej skupin vozlišč, ki paroma ustrezajo relacijski omejitvi. V kolikor v nobeni smeri od trenutne zadnje skupine vozlišč v verigi ne obstaja skupina s strogo manjšo različnostjo zadnji, je algoritem dosegel zaporedni skupini vozlišč, ki sta si vzajemno najmanj različni. Takemu paru pravimo par vzajemno najbližjih sosed/skupin – PVNS (*Reciprocal Nearest Neighbours*). Zaradi monotonosti izbrane mere različnosti, ki je pogoj za pravilno delovanje *Murtagh*-ovega algoritma, je možno pokazati [76], da pri združitvi skupin vozlišč PVNS različnosti nove združene skupine vozlišč z ostalimi skupinami vozlišč, torej s sosedami izhodiščnega para skupin PVNS, kvečjemu narastejo – že zgrajena veriga ostane v veljavi. Kot bomo videli v razdelku 4.5, gre za *Bruynooghe*-jev pogoj za skrčljivost. Algoritem nato iz konca verige odstrani obe zdaj združeni skupini vozlišč in verigo nadaljuje od zadnje, t.j. od prej pred-pred-zadnje skupine vozlišč v verigi naprej, če v skrajšani verigi sploh ostane kakšna skupina vozlišč. V nasprotnem primeru poljubno izbere novo neobiskano skupino vozlišč in z novo verigo na enak način razvrsti njeno okolico. Algoritem se ustavi, ko združi zadnji skupini vozlišč. Postopek je učinkovitejša izvedba osnovnega algoritma iz razdelka 4.3.1, vendar deluje pravilno le pod dodatnim pogojem, da uporabimo monotono mero različnosti. Pod temi pogoji je postopek pravzaprav optimalen [76].

Pri algoritmu za razvrščanje povezav izkoriščamo lokalno delovanje *Murtagh*-ovega algoritma. Teoretično sicer lahko pripravimo omrežje, ki bo algoritem vodilo v eno samo dolgo verigo preko celotnega omrežja, vendar se v praksi to ne dogaja. Med delovanjem algoritma povezavni graf konstruiramo inkrementalno, sproti v okolici verige skupin povezav vhodnega omrežja, ki jo gradi algoritem. Vozlišča povezavnega

grafa ustrezajo skupinam povezav v vhodnem omrežju, povezave v povezavnem grafu pa izražajo sosednost v smislu relacijske omejitve ψ_a in hkrati različnost skupin povezav vhodnega omrežja. V primeru, da algoritem dokončno razvrsti okolico trenutne verige skupin povezav, lahko njej pripadajoči lokalni povezavni podgraf skupaj z izračunanimi vrednostmi različnosti sprostimo iz pomnilnika. Slednje lahko storimo tudi v primeru, da nam začne zmanjkovati pomnilnika. V drugem primeru bomo seveda morali del nekoč že zgrajenega povezavnega grafa na novo konstruirati. V manjšem obsegu velja to tudi v prvem primeru. S primerno optimizacijo, npr. z ohranjanjem čim večjega povezavnega podgrafa v pomnilniku in z začetki novih verig v območju že izračunanega povezavnega podgrafa, hkrati pa z aktivnim izogibanjem skupin povezav s krajišči visokih stopenj v vhodnem omrežju, ki v povezavnem grafu porodijo polne podgrafe, je možno vnovično konstruiranje omejiti in posledično zmanjšati porabo pomnilnika. V našem algoritmu inkrementalni povezavni podgraf označimo z *ils* (*incremental line subgraph*), operacije na njem pa so *GetCalculateD*($\mathbf{L}_s, \mathbf{L}_t$), ki iz pomnilnika vrne povezavo oz. različnost med skupinama povezav vhodnega omrežja $\mathbf{L}_s$ in $\mathbf{L}_t$, v kolikor se tam že nahaja, sicer jo izračuna z $D(\mathbf{L}_s, \mathbf{L}_t)$, shrani in vrne njeno vrednost. Operacija *SetD*($\mathbf{L}_s, \mathbf{L}_t$) izračuna različnost med skupinama $D(\mathbf{L}_s, \mathbf{L}_t)$ in jo shrani oz. posodobi v pomnilniku, operacija *ClearD*($\mathbf{L}_s$) iz pomnilnika odstrani vse povezave oz. različnosti skupin, ki imajo v paru skupino $\mathbf{L}_s$, operacija *Clear* pa iz pomnilnika odstrani celoten povezavni podgraf z vsemi različnostmi. Za učinkovito izvedbo inkrementalnega povezavnega podgrafa lahko uporabimo par struktur: slovar različnosti parov skupin povezav vhodnega omrežja, t.j. slovar povezav povezavnega podgrafa, ki predstavljajo pare povezanih skupin povezav in različnosti med njimi v vhodnem omrežju) ter slovar množic sosednih skupin povezav vhodnega omrežja, ki nam služi za učinkovito iskanje skupini povezav sosednjih skupin povezav v povezavnem podgrafu. V pomoč algoritmu v vsakem vozlišču vhodnega omrežja hranimo še množico skupin povezav, ki se v tem vozlišču dotikajo (za katere velja ψ_a). Torej, če je v množici dotikajočih skupin izbranega vozlišča več skupin, je takšno vozlišče aktivno. Z $\mathbf{U}$ označimo množico še neobiskanih skupin povezav in funkcijo *PickFrom*($\mathbf{U}$), ki iz množice $\mathbf{U}$ odstrani nek element in ga vrne.

$k \leftarrow 1$; $\mathscr{L}_1 \leftarrow \{\{e\}; e \in \mathbf{E}\}$
for all $\mathbf{L} \in \mathscr{L}_1$ *do* $g(\mathbf{L}) \leftarrow 0$
$\mathbf{U} \leftarrow \mathscr{L}_1$

chain $\leftarrow$ []
while $k < m$ *do*
  if *chain* = [] *then*
    ils.*Clear*
    $\mathbf{L}_x \leftarrow$ *PickFrom*($\mathbf{U}$)
  else
    $\mathbf{L}_x \leftarrow$ *chain*.*Pop*
  end if
  chain.*Push*($\mathbf{L}_x$)
  while veriga *chain* se ne konča s PVNS *do*
    s skupino $\mathbf{L}'$ pri veljavni relacijski omejitvi ψ_a(*chain*.*Last*, $\mathbf{L}'$), v smeri strogo padajoče različnosti *ils*.*GetCalculateD*(*chain*.*Last*, $\mathbf{L}'$), podaljšaj verigo *chain*
  end while
  $\mathbf{L}_p \leftarrow$ *chain*.*Pop*; $\mathbf{L}_q \leftarrow$ *chain*.*Pop*
  $\mathbf{L} \leftarrow \mathbf{L}_p \cup \mathbf{L}_q$; $k \leftarrow k + 1$
  $g(\mathbf{L}) \leftarrow$ *ils*.*GetCalculateD*($\mathbf{L}_p, \mathbf{L}_q$)
  $\mathscr{L}_k \leftarrow \mathscr{L}_{k-1} \backslash \{\mathbf{L}_p, \mathbf{L}_q\} \cup \{\mathbf{L}\}$
  $\mathbf{U} \leftarrow \mathbf{U} \backslash \{\mathbf{L}_p, \mathbf{L}_q\} \cup \{\mathbf{L}\}$
  for all $\mathbf{L}' \in \mathscr{L}_k \backslash \{\mathbf{L}\} : \psi_a(\mathbf{L}', \mathbf{L})$ *do* *ils*.*SetD*($\mathbf{L}', \mathbf{L}$)
  ils.*ClearD*($\mathbf{L}_p$); *ils*.*ClearD*($\mathbf{L}_q$)
end while

Prilagojeni algoritem vrne zaporedje razbitij povezav $\mathscr{L}_1$, $\mathscr{L}_2$, .., $\mathscr{L}_m$ in preslikavo $g(\mathbf{L})$. Unija razbitij povezav $\mathscr{G} = \cup_i \mathscr{L}_i$ je drevesna hierarhija povezav (1.4.2), preslikavo $g(\mathbf{L})$ pa imenujemo nivojska funkcija hierarhije $\mathscr{G}$. V obliki para $(\mathscr{G}, g)$ predstavljata drevo razvrstitve (dendrogram) povezav. V programski izvedbi lahko hierarhijo skupin predstavimo s seznamom kazalcev na starše, različnost med skupinami pa s seznamom vrednosti.

V kolikor smo uporabili monotono mero različnosti, lahko tudi z osnovnim algoritmom dobimo natanko enak rezultat rezultatu prilagojenega algoritma. To storimo tako, da osnovni algoritem zaženemo na celotnem povezavnem grafu izvornega omrežja. Pri tem moramo v osnovnemu algoritmu za različnost med vozlišči oz. skupinami vozlišč v povezavnem grafu izbrati različnost, ki je identična različnosti med vozliščem povezavnega grafa pripadajočimi povezavami oz. skupinami povezav v izvornem omrežju.

Osnovni algoritem v tem primeru vrne hierarhijo skupin vozlišč v povezavnem grafu, ki natanko predstavlja hierarhijo skupin povezav v izvornem omrežju, ki bi jo dobili s prilagojenim algoritmom. Slaba stran konstrukcije celotnega povezavnega graf je, da postopek zahteva veliko prostora. Vsako vozlišče izvornega omrežja stopnje k se v pripadajočem povezavnem grafu pretvori v polni podgraf reda k. Prilagojeni algoritem se na realnih podatkih temu koraku v veliki meri izogne in posledično potrebuje manj prostora, je pa v izrednih primerih enako potraten.

4.5 Mere različnosti

Potek in rezultat algoritma razvrščanja je močno odvisen od izbire mere različnosti, saj se v vsaki iteraciji algoritma par najmanj različnih skupin vozlišč ali povezav združi v novo skupino. Mere različnosti delimo na dve skupini:

- mere različnosti, ki se računajo direktno iz opisov skupin elementov (razdelek 4.5.1) in
- mere različnosti, ki se računajo rekurzivno iz različnosti med manjšimi skupinami oz. na podlagi različnosti trivialnih skupin (razdelek 4.5.2).

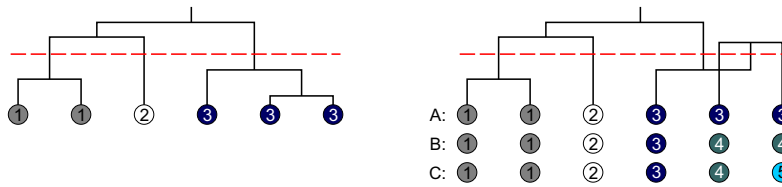
Pogosta pomanjkljivost mer različnosti je v tem, da niso monotone. To pomeni, da obstaja možnost, da bo združevalni algoritem z njihovo uporabo v hierarhiji ustvaril obrate, nekaterih algoritmov pa z nemonotonimi merami različnosti sploh ne moremo uporabiti. Pri obratih gre za pojav, ki nastopi, ko je različnost skupin, ki sta v hierarhiji združeni višje proti korenu, manjša, kot različnost dveh njunih podskupin nekje globlje v hierarhiji. Za pomanjkljivost gre predvsem zato, ker so hierarhije z obrati dvoumne; na izbranem nivoju v hierarhiji je možno dobiti več različnih razbitij na skupine elementov omrežja. Dendrograma primerov monotone in nemonotone hierarhije sta prikazana na sliki 4.1.

Če za mero različnosti obstaja takšna nivojska funkcija h, da pri vseh združevanjih parov skupin $\mathbf{C}_p$ in $\mathbf{C}_q$ v skupine $\mathbf{C}$ velja $\max(h(\mathbf{C}_p), h(\mathbf{C}_q)) \leq h(\mathbf{C})$, pravimo, da je mera različnosti monotona. V teoriji razvrščanja poznamo enakovreden pogoj za monotonost mere različnosti D, t.i. *Bruynooghe*-jev pogoj za skrčljivost (*reducibility property*) [77]:

$$D(\mathbf{C}_p, \mathbf{C}_q) \leq D(\mathbf{C}_p, \mathbf{C}) \ \wedge \ D(\mathbf{C}_p, \mathbf{C}_q) \leq D(\mathbf{C}_q, \mathbf{C}) \ \Rightarrow \\ D(\mathbf{C}_p, \mathbf{C}_q) \leq D(\mathbf{C}_p \cup \mathbf{C}_q, \mathbf{C}), \tag{4.1}$$

Slika 4.1

Primera dendrogramov hierarhije monotone različnosti (levo) in nemonotone različnosti (desno). Pri rezanju leve hierarhije je razvrstitev v skupine 1, 2 in 3 enolično določena, pri desni pa razvrstitev ni enolično določena. Odločiti se moramo za eno izmed možnosti A, B ali C.



kjer so $\mathbf{C}_p$, $\mathbf{C}_q$ in $\mathbf{C}$ skupine vozlišč ali povezav. Enačba (4.1) izraža dejstvo, da če sta si skupini $\mathbf{C}_p$ in $\mathbf{C}_q$ vsaj tako podobni, kot sta vsaka od njiju podobni skupini $\mathbf{C}$, potem sta si skupini $\mathbf{C}_p$ in $\mathbf{C}_q$ najmanj tako podobni, kot je združena skupina $\mathbf{C}_p \cup \mathbf{C}_q$ podobna skupini $\mathbf{C}$. To pomeni, da takoj, ko se dve najmanj različni skupini združi, skupine, ki bi bila združeni skupini podobna bolj, kot je bila ostalim skupinam pred združitvijo, ni več možno najti.

4.5.1 Direktne mere različnosti

Direktne mere različnosti temeljijo na lastnostih vozlišč in/ali povezav omrežja, ki pripadajo danim skupinam. Njihova vrednost je neodvisna od različnosti med trivialnimi skupinami s posameznimi elementi. V skupino direktnih mer različnosti spada dobršen del mer različnosti, ki jih predstavimo v nadaljevanju.

Preproste mere različnosti med skupinami

Preprosta mera različnosti, ki upošteva le strukturo omrežja, temelji na *Jaccard*-ovi korelacijski meri [78] in jo definiramo z enačbo:

$$D_J(\mathbf{L}_p, \mathbf{L}_q) = D_J(\mathbf{C}_p, \mathbf{C}_q) = 1 - \frac{|\mathbf{C}_p \cap \mathbf{C}_q|}{|\mathbf{C}_p \cup \mathbf{C}_q|} = \frac{|\mathbf{C}_p \oplus \mathbf{C}_q|}{|\mathbf{C}_p \cup \mathbf{C}_q|}. \tag{4.2}$$

$\mathbf{C}_p$ in $\mathbf{C}_q$ sta skupini krajišč, ki pripadata skupinam povezav $\mathbf{L}_p$ in $\mathbf{L}_q$. Operator $\oplus$ predstavlja simetrično razliko množic. Mera ni nujno monotona.

V smislu monotonosti se nekoliko bolje obnese utežena različica *Jaccard*-ove različnosti:

$$D_w(\mathbf{L}_p, \mathbf{L}_q) = \frac{|\mathbf{C}_p \oplus \mathbf{C}_q|}{|\mathbf{C}_p \cup \mathbf{C}_q|} \cdot |\mathbf{L}_p \cup \mathbf{L}_q|. \tag{4.3}$$

Če želimo D_w omejiti na interval $[0, 1]$, lahko desno stran enačbe (4.3) delimo s številom povezav v omrežju m. V glavnini primerov dobimo hierarhije z manj obrati kot sicer. Kljub temu utežena *Jaccard*-ova različnost ni monotona.

Meri D_J in D_w sta direktno izračunljivi in predstavljata naš prvi poskus pri uporabi v naših algoritmih.

Mere različnosti z množicami ključnih besed

Ideja izvira iz analize bibliografskih omrežij, kar bomo razložili na primeru omrežja $\mathbf{N} = (\mathbf{V}, \mathbf{E}, \boldsymbol{\kappa}, \varnothing)$ sodelovanj avtorjev $\mathbf{V}$, $\kappa : \mathbf{V} \to [\mathbf{K}]$, ki uporabljajo nabore ključnih besed $\mathbf{K}$. Za vsakega avtorja poznamo nabor ključnih besed in frekvence s katerimi jih uporabljajo (lahko uporabimo tudi kako drugo nenegativno mero pomembnosti). Omrežje sodelovanj navadno sestavimo za neko področje in za določen čas. Glede na ključne besede bi želeli podrobneje raziskati skupine povezanih avtorjev oz. v njih odkriti podskupine sodelujočih avtorjev, ki uporabljajo najbolj podobne ključne besede kar najverjetneje pomeni, da delujejo na podobnih podpodročjih. Glede na sezname ključnih besed podskupin torej s postopkom lahko odkrijemo znanstvena podpodročja. Za vsako ključno besedo $k \in \mathbf{K}$, ki jo avtor s uporablja, v enačbi (4.4) definiramo množico $\mathbf{F}_s$ z elementi oblike $(k : f_{s,k})$, kjer je $f_{s,k}$ frekvenca posamezne ključne besede $k \in \mathbf{K}$, ki jo avtor $s \in \mathbf{V}$ uporablja. Za skupino avtorjev $\mathbf{C}$ analogno definiramo seznam ključnih besed $\mathbf{F}_\mathbf{C}$ in frekvence ključnih besed v njem $f_{\mathbf{C},k}$.

$$\begin{aligned} \mathbf{F}_v &= \{(k : f_k); k \in \mathbf{K}\}, v \in \mathbf{V} \\ \mathbf{F}_C &= \{(k : f_{\mathbf{C},k}); k \in \mathbf{K}\}, \quad f_{\mathbf{C},k} = \textstyle\sum_{s \in \mathbf{C}} f_{s,k} \end{aligned} \tag{4.4}$$

Če avtor s ne uporablja določene ključne besede k, je njej ustrezna frekvenca enaka 0, $f_{s,k} = 0$. Analogno velja za skupino $\mathbf{C}$, če v njej ključna beseda k ni prisotna: $f_{\mathbf{C},k} = 0$. V programski izvedbi je pare $(k : f_k)$, kjer je $f_k = 0$, smiselno izpustiti. Primerna podatkovna struktura za takšno predstavitev je slovar.

V enačbi (4.5) definicijo (4.4) razširimo na seznam frekvenc za posamezno ključno besedo k, ki jo sodelujoča avtorja s in t paroma uporabljata oz. se pojavlja v paru skupin $\mathbf{C}_p$ in $\mathbf{C}_q$.

$$\begin{aligned} \mathbf{F}_s \sqcup \mathbf{F}_t &= \{(k : f_{s,k} + f_{t,k}); k \in \mathbf{K}\} \\ \mathbf{F}_{\mathbf{C}_p} \sqcup \mathbf{F}_{\mathbf{C}_q} &= \{(k : f_{\mathbf{C}_p,k} + f_{\mathbf{C}_q,k}); k \in \mathbf{K}\} \end{aligned} \tag{4.5}$$

V spodnji enačbi je podana absolutna razlika frekvenc posameznih ključnih besed, ki jih uporabljata avtorja oz. se nahaja v skupinah avtorjev:

$$\begin{aligned}\mathbf{F}_s \sqcap \mathbf{F}_t &= \{(k : \left|f_{s,k} - f_{t,k}\right|); k \in \mathbf{K}\},\\ \mathbf{F}_{\mathbf{C}_p} \sqcap \mathbf{F}_{\mathbf{C}_q} &= \{(k : \left|f_{\mathbf{C}_p,k} - f_{\mathbf{C}_q,k}\right|); k \in \mathbf{K}\}.\end{aligned} \tag{4.6}$$

Z enačbo (4.7) je podana še končna oblika mere različnosti. Gre za normalizirano različico absolutne razlike frekvenc ključnih besed, ki jih uporabljata dva avtorja oz. jo zasledimo v paru skupin avtorjev.

$$\begin{aligned}D_k(\mathbf{C}_p, \mathbf{C}_q) &= \frac{\left|\mathbf{F}_{\mathbf{C}_p} \sqcap \mathbf{F}_{\mathbf{C}_q}\right|}{\left|\mathbf{F}_{\mathbf{C}_p} \sqcup \mathbf{F}_{\mathbf{C}_q}\right|} \text{ in}\\ d_k(s,t) = D_k(\{s\},\{t\}) &= \frac{|\mathbf{F}_s \sqcap \mathbf{F}_t|}{|\mathbf{F}_s \sqcup \mathbf{F}_t|},\end{aligned} \tag{4.7}$$

kjer je $|\mathbf{F}| = \sum_{f \in \mathbf{F}} f$. Mero d_k lahko uporabimo v eni izmed rekurzivno izračunljivih mer različnosti, opisanih v razdelku 4.5.2, mero D_k pa lahko uporabimo neposredno. Mera D_k ni monotona.

Obe meri d_k in D_k sta primerni za uporabo v osnovnem algoritmu. Z uporabo rekurzivnih postopkov (4.5.2) lahko mero d_k uporabimo tudi v prilagojenem algoritmu, a jo moramo nekoliko prilagoditi. Namesto seznamov ključnih besed $\mathbf{F}_s$ in $\mathbf{F}_t$ posameznih avtorjev s in t moramo operirati s seznamoma ključnih besed $\mathbf{F}_u$ in $\mathbf{F}_v$ povezav u in v.

Preprost način kako dobimo seznam ključih besed $\mathbf{F}_e$ za povezavo e je, da zanj vzamemo unijo ključnih besed, ki ju s povezavo povezana avtorja uporabljata. Za vsako povezavo $e = (s : t) \in \mathbf{E}$ tako dobimo seznam ključnih besed $\mathbf{F}_e = \mathbf{F}_s \sqcup \mathbf{F}_t$. Za potrebe testiranja prilagojenega algoritma smo množice ključnih besed pripravili na tak način. Namesto, da sezname ključnih besed povezav (opise) določamo na podlagi njihovih krajišč, bi lahko uporabili kake druge opise povezav, za katere ni nujno, da izvirajo iz opisov vozlišč.

Nekaj monotonih direktno izračunljivih mer različnosti

Za primer ključnih besed iz razdelka 4.5.1 v enačbi (4.12) podajamo tri monotone mere različnosti. Najprej definirajmo pomožno funkcijo χ:

$$\chi(c) = \begin{cases} 1 & \text{če pogoj } c \text{ velja} \\ 0 & \text{sicer} \end{cases}. \tag{4.8}$$

Za skupini $\mathbf{C}_p$ in $\mathbf{C}_q$ nato definirajmo:

$$S_{SF}(\mathbf{C}_p, \mathbf{C}_q) = \sum_{k \in \mathbf{K}} \max(f_{\mathbf{C}_p,k}, f_{\mathbf{C}_q,k}), \tag{4.9}$$

$$S_{SC}(\mathbf{C}_p, \mathbf{C}_q) = \sum_{k \in \mathbf{K}} \max(\chi(f_{\mathbf{C}_p,k} > 0), \chi(f_{\mathbf{C}_q,k} > 0)) \text{ in} \tag{4.10}$$

$$S_{MF}(\mathbf{C}_p, \mathbf{C}_q) = \max_{k \in \mathbf{K}} (f_{\mathbf{C}_p,k}, f_{\mathbf{C}_q,k}). \tag{4.11}$$

Različnosti $D_x(\mathbf{C}_p, \mathbf{C}_q)$, kjer je x *SF*, *SC* in *MF* pa določimo takole:

$$D_x(\mathbf{C}_p, \mathbf{C}_q) = \begin{cases} S_x(\mathbf{C}_p, \mathbf{C}_q) & p \neq q \\ 0 & p = q \end{cases}. \tag{4.12}$$

Prva mera, D_{SF}, favorizira združevanje skupin s podobnimi množicami ključnih besed in s podobnimi frekvencami posameznih ključnih besed. Druga mera, D_{SC}, je poenostavitev prve mere. Namesto frekvenc upošteva le prisotnost posamezne ključne besede. Tretja mera, D_{MF}, deluje obratno od druge. Upošteva le frekvence ključnih besed. Frekvence ključnih besed niso nujno cela števila.

Delovanje mer različnosti lahko utemeljimo na sledeč način. Ko velja $\mathbf{C} = \mathbf{C}_p \cup \mathbf{C}_q$, velja tudi $\mathbf{F}_\mathbf{C} = \mathbf{F}_{\mathbf{C}_p} \sqcup \mathbf{F}_{\mathbf{C}_q}$. Če je v poteku algoritma na primer nabor ključnih besed skupine $\mathbf{C}_p$ nadmnožica nabora ključnih besed neke sosednje skupine $\mathbf{C}_q$ in različnost med $\mathbf{C}_p$ in $\mathbf{C}_q$ najmanjša, se v skladu z mero D_{SF} skupini $\mathbf{C}_p$ frekvence ključnih besed povečajo za vrednost frekvenc ključnih besed skupine $\mathbf{C}_q$ ali pa v skladu z merama D_{SC} in D_{MF} ostanejo nespremenjene: $\mathbf{C}_q \subseteq \mathbf{C}_p \Rightarrow D_x(\mathbf{C}_p, \mathbf{C}_q) = S_x(\mathbf{C}_p, \mathbf{C}_p)$; x je *SC* ali *MF*. Na učinek vseh treh mer različnosti lahko gledamo kot na "požiranje" skupin s podmnožicami ključnih besed. Takoj, ko zaradi nezdružljivosti skupin "požiranje" ni več primerno, pa mere različnosti favorizirajo združevanje sosednjih skupin z najmanj različnimi množicami ključnih besed.

Mere različnosti D_{SF}, D_{SC} in D_{MF} lahko uporabimo tudi v prilagojenem algoritmu, a jih moramo nekoliko prilagoditi. Namesto seznamov ključnih besed $\mathbf{F}_{\mathbf{C}_p}$ in $\mathbf{F}_{\mathbf{C}_q}$ skupin avtorjev $\mathbf{C}_p$ in $\mathbf{C}_q$ moramo operirati s seznamoma ključnih besed $\mathbf{F}_{\mathbf{L}_p}$ in $\mathbf{F}_{\mathbf{L}_q}$ za skupini povezav $\mathbf{L}_p$ in $\mathbf{L}_q$. Sezname ključnih besed za skupine povezav dobimo podobno, kot jih dobimo za skupine avtorjev oz. vozlišč pri meri različnosti d_k.

4.5.2 *Rekurzivno izračunljive mere različnosti*

Pri merah iz te skupine lahko direktno izračunamo le različnosti parov trivialnih skupin, t.j. iz mere različnosti med posameznimi elementi omrežja d. Različnost $D(\mathbf{C}_p, \mathbf{C}_q)$ med netrivialnimi ločenimi skupinami $\mathbf{C}_p$ in $\mathbf{C}_q$, ki vsebujejo več elementov omrežja, se vedno izračuna rekurzivno iz različnosti med združenimi skupinami. Na primer pri tradicionalni minimalni metodi je različnost skupin določena z najmanjšo različnostjo para elementov omrežja, kjer vsak posebej pripada drugi podskupini:

$$D_m(\mathbf{C}_p, \mathbf{C}_q) = \min_{u \in \mathbf{C}_p, v \in \mathbf{C}_q} d(u, v). \tag{4.13}$$

Za zgornji primer izpeljimo rekurzivno relacijo:

$$\begin{aligned} D_m(\mathbf{C}, \mathbf{C}_p \cup \mathbf{C}_q) &= \min_{u \in \mathbf{C}, v \in \mathbf{C}_p \cup \mathbf{C}_q} d(u, v) \\ &= \min(\min_{u \in \mathbf{C}, v \in \mathbf{C}_p} d(u, v), \min_{u \in \mathbf{C}, v \in \mathbf{C}_q} d(u, v)) \\ &= \min(D_m(\mathbf{C}, \mathbf{C}_p), D_m(\mathbf{C}, \mathbf{C}_q)). \end{aligned} \tag{4.14}$$

Pri definiciji različnosti D_m potrebujemo celotno matriko različnosti d na množici $\mathbf{V} \times \mathbf{V}$ – najmanj $O(n^2)$ prostora. Za velika in redka omrežja je to preveč. Učinkovitejši postopek dosežemo tako, da definicijo D omejimo na različnosti parov krajišč povezav v omrežju [72]. Naj bo $E(\mathbf{C}_p, \mathbf{C}_q) = \{e \in \mathbf{E} : \text{ext}(e) \cap \mathbf{C}_p \neq \emptyset \wedge \text{ext}(e) \cap \mathbf{C}_q \neq \emptyset\}$. Različnost D za vozlišči u and v nato definiramo kot:

$$D(\{u\}, \{v\}) = \begin{cases} d(u, v) & (u : v) \in \mathbf{E} \\ \infty & \text{sicer} \end{cases}. \tag{4.15}$$

Različnosti po minimalni metodi (4.13) lahko analogno določimo omejeno različnost po minimalni metodi:

$$D_{mc}(\mathbf{C}_p, \mathbf{C}_q) = \min_{(u:v) \in E(\mathbf{C}_p, \mathbf{C}_q)} d(u, v). \tag{4.16}$$

Enostavno lahko preverimo, da velja:

$$D_{mc}(\mathbf{C}, \mathbf{C}_p \cup \mathbf{C}_q) = \min(D_{mc}(\mathbf{C}, \mathbf{C}_p), D_{mc}(\mathbf{C}, \mathbf{C}_q)). \tag{4.17}$$

Na podoben način vpeljemo omejeno različnost po maksimalni metodi:

$$\begin{aligned} D_{Mc}(\mathbf{C}_p, \mathbf{C}_q) &= \max_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v), \\ D_{Mc}(\mathbf{C}, \mathbf{C}_p \cup \mathbf{C}_q) &= \max(D_{Mc}(\mathbf{C}, \mathbf{C}_p), D_{Mc}(\mathbf{C}, \mathbf{C}_q)) \end{aligned} \tag{4.18}$$

in tudi omejeno različnost po povprečni metodi:

$$\begin{aligned} D_{ac}(\mathbf{C}_p, \mathbf{C}_q) &= \tfrac{1}{w(E(\mathbf{C}_p,\mathbf{C}_q))} \sum_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v), \\ D_{ac}(\mathbf{C}, \mathbf{C}_p \cup \mathbf{C}_q) &= \tfrac{w(E(\mathbf{C},\mathbf{C}_p))}{w(E(\mathbf{C},\mathbf{C}_p\cup\mathbf{C}_q))} D_{ac}(\mathbf{C}, \mathbf{C}_p) + \tfrac{w(E(\mathbf{C},\mathbf{C}_q))}{w(E(\mathbf{C},\mathbf{C}_p\cup\mathbf{C}_q))} D_{ac}(\mathbf{C}, \mathbf{C}_q), \end{aligned} \tag{4.19}$$

kjer je z $w : \mathbf{E} \to \mathbb{R}^+$ določena utež posamezne povezave, $w(\mathbf{L}) = \sum_{l\in\mathbf{L}} w(l)$, $\mathbf{L} \subseteq \mathbf{E}$ in $w(\mathbf{E}(\mathbf{C}, \mathbf{C}_p \cup \mathbf{C}_q)) = w(\mathbf{E}(\mathbf{C}, \mathbf{C}_p)) + w(E(\mathbf{C}, \mathbf{C}_q))$ za $E(\mathbf{C}, \mathbf{C}_p) \cap E(\mathbf{C}, \mathbf{C}_q) = \emptyset$. Za $E(\mathbf{C}_p, \mathbf{C}_q) = \emptyset$ definiramo $D_x(\mathbf{C}_p, \mathbf{C}_q) = \infty$, kjer je x *mc*, *Mc* oz. *ac*.

Na polnih omrežjih (grafih) velikosti n, $\mathbf{N} \equiv K_n$, se postopki za izračun različnosti z relacijsko omejitvijo izrodijo v ustrezne tradicionalne postopke za izračun različnosti.

Obstaja še precej alternativnih postopkov za izračun različnosti in mnogi so opisani v *Fortunato*-vem preglednem članku [42], ni pa nam znano, ali se da njihove omejene različice izraziti rekurzivno.

Če želimo mere D_{mc}, D_{Mc} in D_{ac} uporabljati v prilagojenem algoritmu, jih moramo prilagoditi za računanje različnosti skupin povezav $\mathbf{L}_p$ in $\mathbf{L}_q$. Namesto funkcije $E(\mathbf{C}_p, \mathbf{C}_q)$ moramo tokrat upoštevati funkcijo aktivnih vozlišč $A(\mathbf{L}_p, \mathbf{L}_q) = \{v \in \mathbf{V} : v \in \mathbf{C}_p; \exists u \neq v : u \in \mathbf{C}_q\}$, kjer sta $\mathbf{C}_p$ in $\mathbf{C}_q$ skupini vozlišč, ki pripadata ustreznim skupinam povezav, kot je opisano v razdelku 4.4. Za povezavi u in v tako določimo:

$$D(\{u\}, \{v\}) = \begin{cases} d(u,v) & \text{ext}(u) \cap \text{ext}(v) \neq \emptyset \\ \infty & \text{sicer} \end{cases}. \tag{4.20}$$

Prilagojena omejena različnost po minimalni metodi za skupine povezav zgleda tako:

$$D_{mcl}(\mathbf{L}_p, \mathbf{L}_q) = \min_{u\in\mathbf{L}_p, v\in\mathbf{L}_q : \text{ext}(u)\cap A(\mathbf{L}_p,\mathbf{L}_q)\neq\emptyset \wedge \text{ext}(v)\cap A(\mathbf{L}_p,\mathbf{L}_q)\neq\emptyset} d(u,v). \tag{4.21}$$

Lehmann-*ova mera različnosti*

Primer rekurzivno izračunljive mere različnosti za izračun različnosti med skupinami povezav temelji na *Jaccard*-ovi meri različnosti. Uporabljajo jo v *Lehmann*-ovem delu [1]. Različnost med sosednjima povezavama u in v je določena z:

$$d_L(u,v) = \frac{|\mathrm{n}(u) \oplus \mathrm{n}(v)|}{|\mathrm{n}(u) \cup \mathrm{n}(v)|}. \tag{4.22}$$

$\mathrm{n}(u)$ predstavlja množico sosedov vozlišča u, t.j. množico vozlišč, ki so iz vozlišča u dostopna preko ene povezave. V kolikor povezavi u in v nimata skupnega vozlišča (nista sosednji) velja $d_L(u,v) = 1$. Razlika med *Jaccard*-ovo in *Lehmann*-ovo mero različnosti je v pravilu določanja razdalje med skupinami. V *Lehmann*-ovi za izračun uporabljajo minimalno metodo:

$$D_L(\mathbf{L}_p, \mathbf{L}_q) = \min_{u \in \mathbf{L}_p, v \in \mathbf{L}_q} d_L(u,v). \tag{4.23}$$

Lehmann-ova mera različnosti je zato monotona. Soroden dokaz monotonosti je opisan v razdeleku 4.6.2.

4.6 *Dokazi monotonosti mer različnosti*

4.6.1 *Dokazi monotonosti mer različnosti D_{SF}, D_{SC} in D_{MF}*

Vzemimo tri skupine vozlišč ali povezav $\mathbf{C}_p$, $\mathbf{C}_q$ in $\mathbf{C}$ in njim pripadajoče sezname lastnosti (ključne besede) $\mathbf{F}_{\mathbf{C}_p}$, $\mathbf{F}_{\mathbf{C}_q}$ in $\mathbf{F}$. Predpostavimo, da je algoritem pravkar združil skupini $\mathbf{C}_p$ in $\mathbf{C}_q$. Seznam ključnih besed združene skupine $\mathbf{C}_p \cup \mathbf{C}_q$ bo tako enak $\mathbf{F}_{\mathbf{C}_p} \sqcup \mathbf{F}_{\mathbf{C}_q}$.

Monotonost mere D_{SF}: če za D_{SF} velja lastnost skrčljivosti (4.1), velja tudi:

$$D_{SF}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{SF}(\mathbf{C}_p, \mathbf{C}) \tag{4.24}$$

in s tem:

$$D_{SF}(\mathbf{C}_p, \mathbf{C}_q) = \sum_{k \in \mathbf{K}} \max(f_{\mathbf{C}_p,k}, f_{\mathbf{C}_q,k}) \leq \sum_{k \in \mathbf{K}} \max(f_{\mathbf{C}_p,k}, f_{\mathbf{C},k}). \tag{4.25}$$

Ker velja $f_{\mathbf{C}_q,k} \geq 0$, $f_{\mathbf{C}_p,k} \leq f_{\mathbf{C}_p,k} + f_{\mathbf{C}_q,k}$, velja tudi:

$$\leq \sum_{k \in \mathbf{K}} \max(f_{\mathbf{C}_p,k} + f_{\mathbf{C}_q,k}, f_{\mathbf{C},k}) = D_{SF}(\mathbf{C}_p \cup \mathbf{C}_q, \mathbf{C}) \tag{4.26}$$

in posledično $D_{SF}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{SF}(\mathbf{C}_p \cup \mathbf{C}_q, \mathbf{C})$. Mera D_{SF} je monotona. □

Monotonost mere D_{SC}: če za D_{SC} velja lastnost skrčljivosti (4.1), velja tudi:

$$D_{SC}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{SC}(\mathbf{C}_p, \mathbf{C}) \tag{4.27}$$

in s tem:

$$\begin{gathered} D_{SC}(\mathbf{C}_p, \mathbf{C}_q) = \textstyle\sum_{k \in \mathbf{K}} \max(\chi(f_{\mathbf{C}_p,k} > 0), \chi(f_{\mathbf{C}_q,k} > 0)) \leq \\ \textstyle\sum_{k \in \mathbf{K}} \max(\chi(f_{\mathbf{C}_p,k} > 0), \chi(f_{\mathbf{C},k} > 0)). \end{gathered} \tag{4.28}$$

Ker velja $f_{\mathbf{C}_q,k} \geq 0$, $f_{\mathbf{C}_p,k} \leq f_{\mathbf{C}_p,k} + f_{\mathbf{C}_q,k}$, velja tudi:

$$\leq \sum_{k \in \mathbf{K}} \max(\chi(f_{\mathbf{C}_p,k} + f_{\mathbf{C}_q,k} > 0), \chi(f_{\mathbf{C},k} > 0)) = D_{SC}(\mathbf{C}_p \cup \mathbf{C}_q, \mathbf{C}) \tag{4.29}$$

in posledično $D_{SC}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{SC}(\mathbf{C}_p \cup \mathbf{C}_q, \mathbf{C})$. Mera D_{SC} je monotona. □

Monotonost mere D_{MF}: če za D_{MF} velja lastnost skrčljivosti (4.1), velja tudi:

$$D_{MF}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{MF}(\mathbf{C}_p, \mathbf{C}) \tag{4.30}$$

in s tem:

$$D_{MF}(\mathbf{C}_p, \mathbf{C}_q) = \max_{k \in \mathbf{K}}(f_{\mathbf{C}_p,k}, f_{\mathbf{C}_q,k}) \leq \max_{k \in \mathbf{K}}(f_{\mathbf{C}_p,k}, f_{\mathbf{C},k}). \tag{4.31}$$

Ker velja $f_{\mathbf{C}_q,k} \geq 0$, $f_{\mathbf{C}_p,k} \leq f_{\mathbf{C}_p,k} + f_{\mathbf{C}_q,k}$, velja tudi:

$$\leq \max_{k \in \mathbf{K}}(f_{\mathbf{C}_p,k} + f_{\mathbf{C}_q,k}, f_{\mathbf{C},k}) = D_{MF}(\mathbf{C}_p \cup \mathbf{C}_q, \mathbf{C}) \tag{4.32}$$

in posledično $D_{MF}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{MF}(\mathbf{C}_p \cup \mathbf{C}_q, \mathbf{C})$. Mera D_{MF} je monotona. □

4.6.2 Omejena različnost po minimalni metodi zagotavlja monotonost

Predpostavimo, da za skupine $\mathbf{C}_p$, $\mathbf{C}_q$ in $\mathbf{C}$ velja:

$$D_{mc}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{mc}(\mathbf{C}_p, \mathbf{C}) \text{ in } D_{mc}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{mc}(\mathbf{C}_q, \mathbf{C}). \tag{4.33}$$

S tem velja:

$$\begin{aligned} \min_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v) &\leq \min_{(u:v)\in E(\mathbf{C}_p,\mathbf{C})} d(u,v) \text{ in} \\ \min_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v) &\leq \min_{(u:v)\in E(\mathbf{C}_q,\mathbf{C})} d(u,v). \end{aligned} \tag{4.34}$$

Iz neenakosti (4.34) sledi:

$$\begin{gathered} \min_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v) \leq \\ \min\left(\min_{(u:v)\in E(\mathbf{C}_p,\mathbf{C})} d(u,v), \min_{(u:v)\in E(\mathbf{C}_q,\mathbf{C})} d(u,v)\right) = \\ \min_{(u:v)\in E(\mathbf{C}_p\cup\mathbf{C}_q,\mathbf{C})} d(u,v), \end{gathered} \tag{4.35}$$

kar dokazuje, da za omejeno različnost po minimalni metodi velja pogoj skrčljivosti. Omejena različnost po minimalni metodi je monotona. □

Dokaz za monotonost omejene različnosti po maksimalni metodi (4.18) je zelo podoben. Monotonost tradicionalne različnosti po minimalni metodi (4.13) sledi iz monotonosti omejene različnosti po minimalni metodi, če vhodno omrežje $\mathbf{N}$ vpnemo v polno omrežje (graph) K_n. Monotonost *Lehmann*-ove različnosti (4.23) sledi iz monotonosti omejene različnosti po minimalni metodi za $d = d_l$.

4.6.3 Omejena različnost po povprečni metodi zagotavlja monotonost

Monotonost omejene različnosti po povprečni metodi (4.19) dokažemo podobno, kot monotonost omejene različnosti po minimalni metodi. Predpostavimo, da za disjunktni skupini $\mathbf{C}_p$, $\mathbf{C}_q$ in skupino $\mathbf{C}$ velja:

$$D_{ac}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{ac}(\mathbf{C}_p, \mathbf{C}) \text{ in } D_{ac}(\mathbf{C}_p, \mathbf{C}_q) \leq D_{ac}(\mathbf{C}_q, \mathbf{C}). \tag{4.36}$$

Z množenjem neenačb (4.36) z $w(E(\mathbf{C}_p, \mathbf{C})) \cdot w(E(\mathbf{C}_p, \mathbf{C}_q))$ oziroma z $w(E(\mathbf{C}_q, \mathbf{C})) \cdot w(E(\mathbf{C}_p, \mathbf{C}_q))$ dobimo:

$$\begin{aligned} w(E(\mathbf{C}_p,\mathbf{C}))\sum_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v) &\leq w(E(\mathbf{C}_p,\mathbf{C}_q))\sum_{(u:v)\in E(\mathbf{C}_p,\mathbf{C})} d(u,v) \text{ in} \\ w(E(\mathbf{C}_q,\mathbf{C}))\sum_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v) &\leq w(E(\mathbf{C}_p,\mathbf{C}_q))\sum_{(u:v)\in E(\mathbf{C}_q,\mathbf{C})} d(u,v). \end{aligned} \tag{4.37}$$

Obe levi in obe desni strani neenačb (4.37) seštejemo in dobimo neenačbo:

$$\begin{aligned} &(w(E(\mathbf{C}_p,\mathbf{C}))+w(E(\mathbf{C}_q,\mathbf{C})))\sum_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v) \leq \\ &w(E(\mathbf{C}_p,\mathbf{C}_q))\left(\sum_{(u:v)\in E(\mathbf{C}_p,\mathbf{C})} d(u,v)+\sum_{(u:v)\in E(\mathbf{C}_q,\mathbf{C})} d(u,v)\right), \end{aligned} \tag{4.38}$$

ki je enakovredna:

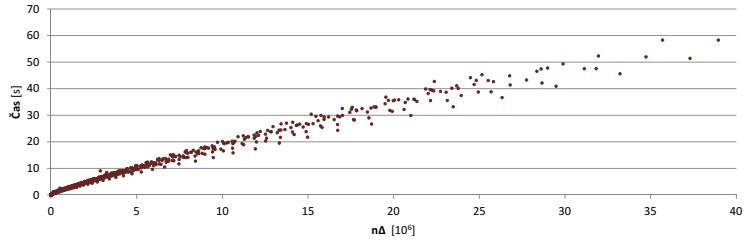
$$w(E(\mathbf{C}_p\cup\mathbf{C}_q,\mathbf{C}))\sum_{(u:v)\in E(\mathbf{C}_p,\mathbf{C}_q)} d(u,v) \leq w(E(\mathbf{C}_p,\mathbf{C}_q))\sum_{(u:v)\in E(\mathbf{C}_p\cup\mathbf{C}_q,\mathbf{C})} d(u,v). \tag{4.39}$$

Z deljenjem neenačbe (4.39) z $w(E(\mathbf{C}_p,\mathbf{C}_q))\cdot w(E(\mathbf{C}_p\cup\mathbf{C}_q,\mathbf{C}))$ dobimo $D_{ac}(\mathbf{C}_p,\mathbf{C}_q) \leq D_{ac}(\mathbf{C}_p\cup\mathbf{C}_q,\mathbf{C})$ in omejena različnost po povprečni metodi je s tem monotona. □

4.7 Zahtevnosti algoritmov

Osnovni algoritem iz razdelka 4.3 je relativno učinkovit. Njegova časovna zahtevnost je omejena s številom vozlišč vhodnega omrežja n in njegovo maksimalno stopnjo Δ, $O(n\cdot\Delta)$. V najslabšem primeru je $\Delta = n$ in zahtevnost algoritma postane kvadratična. V praksi omrežja s tako veliki stopnjami srečamo redko. Slednje velja tudi za brezlestvična omrežja, ki so pretežno predmet našega zanimanja. Algoritem smo zagnali na primerih brezlestvičnih omrežij [79, 80] velikosti med 10 in 10000 vozlišči, ki smo jih za ta namen pripravili. Največja stopnja v kateremkoli izmed njih je bila 6780. Ustvarili smo jih v programskem paketu za analizo omrežij *NetworkX* [81]. Vsakemu vozlišču v vsakemu omrežju smo določili edinstveno ključno besedo in sicer niz zaporedne številke vozlišča v omrežju.

Na sliki 4.2 je podan prikaz povprečnega trajanja izvedbe algoritma v odvisnosti od produkta velikosti omrežja in njegove maksimalne stopnje. Glede na velikost vhodnega



Slika 4.2

Prikaz časov izvedbe osnovnega algoritma v odvisnosti od produkta števila vozlišč in maksimalne stopnje (Δ) v naključno ustvarjenih brezlestvičnih omrežjih. Vsakemu vozlišču smo priredili eno in unikatno ključno besedo.

omrežja nastopi najbolj neugoden primer, ko je vhodno omrežje zvezda ($\Delta = n-1$). V vsaki iteraciji algoritma je potrebno preračunati vse različnosti do skupin nezdruženih vozlišč. Časovna zahtevnost v tem primeru postane kvadratična $O(n^2)$. Prostorska zahtevnost je glede na število povezav m v vhodnem omrežju linearna: $O(m)$.

Algoritem začne z n skupinami vozlišč $\mathbf{C}_i$, $i = 1..n$, kjer vsako vozlišče pripada svoji skupini. Prednostno vrsto Q napolni z razdaljami med posameznimi povezanimi skupinami vozlišč. Algoritem tako v Q začne z natanko m različnostmi. Vsaki povezavi $(u : v) \in \mathbf{E}$ ustreza različnost $d(\mathbf{C}_u, \mathbf{C}_v)$. V vsaki iteraciji algoritem združi skupini vozlišč $\mathbf{C}_u$ in $\mathbf{C}_v$ v novo skupino vozlišč $\mathbf{C}$, vse pripadajoče povezave med vozlišči združenih skupin (vsaj ena) pa se v združeno skupino absorbirajo. Različnosti med predhodnima skupinama združene skupine $\mathbf{C}_u$ oz. $\mathbf{C}_v$ in ostalimi skupinami se iz prednostne vrste odstrani. Najmanjša časovna zahtevnost nastopi, ko je vhodno omrežje pot, saj moramo takrat ne glede na izbiro skupin v vsaki iteraciji preračunati največ dve različnosti. V tem primeru je zahtevnost enaka $O(n)$. Nasprotno moramo, ne glede na izbiro združenih skupin, v zvezdi preračunati razdalje do vseh ostalih skupin. Pri skupinah moramo voditi evidenco ključnih besed, ki ustrezajo vozliščem v njih. Če je število ključnih besed veliko in mera različnosti upošteva vse, se časovna zahtevnost lahko poveča.

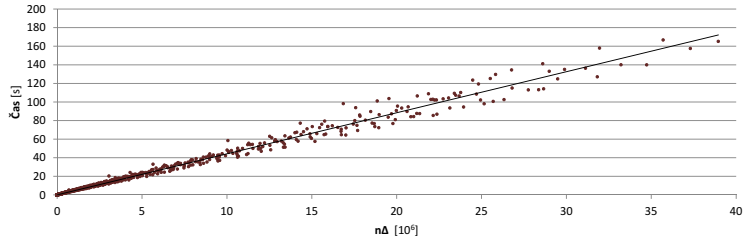
Prilagojeni algoritem za razvrščanje povezav

Nekoliko težje je oceniti zahtevnost prilagojenega algoritma, saj je bolj kompleksen in uporablja dodatne podatkovne strukture. Kot bomo videli v nadaljevanju, je njegova zahtevnost močno odvisna od strukture ter lastnosti vhodnega omrežja, pomemben vpliv pa ima tudi izbira mere različnosti. Ni se težko prepričati, da algoritem največ virov potrebuje na polnem grafu oz. omrežju. V izogib zadregi je zato dobro vedeti, ali se v vhodnem omrežju nahajajo polna podomrežja. Z uporabo *k-cores* algoritma [82]

lahko preverimo prisotnost polnih podomrežij v omrežju in posledično lažje predvidimo obnašanje algoritma. Zahtevnost *k-cores* algoritma je linearna $O(m)$. Glavno jedro, t.j. jedro največjega reda lahko vsebuje polne grafe velikosti k, ne pa tudi $k+1$. Na zahtevnost prilagojenega algoritma pravzaprav možno vpliva že prisotnost posameznih vozlišč visokih stopenj, saj se pri razvoju v povezavni graf le-ta razvijejo v polne podgrafe. V naš algoritem ni vgrajena kakršnakoli strategija izogibanja vozliščem visokih stopenj. Razmeroma očitno je, da je bolje najprej razvrstiti tiste povezave omrežja, ki takšna vozlišča obidejo, saj se ob združitvi povezav v okolici vozlišč visokih stopenj, le-tem stopnja zmanjša. Za poti in polna omrežja smo zahtevnosti prilagojenega algoritma določili analitično. Obliki predstavljata obe skrajnosti možnih vhodnih omrežij. Na običajnih omrežjih smo z merjenjem časa izvedbe njegovo zahtevnost določili statistično.

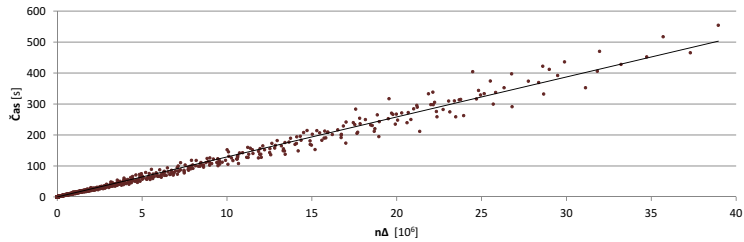
Časovno zahtevnost prilagojenega algoritma smo določili na poteh dolžine $m = n-1$ povezav. Pri uporabi mere različnosti D_{SF}, opisane v razdelku 4.5.1, je le-ta enaka $O(m)$. Zahtevnost izračuna različnosti D_{SF} za konstantni nabor ključnih besed $\mathbf{K}$ je namreč enaka $O(1)$, algoritem pa v vsaki iteraciji združi po dve sosednji skupini povezav in hkrati preračuna največ dve različnosti do sosednjih skupin povezav na vsaki strani. Časovno zahtevnost smo analizirali tudi za primer polnih grafov z n vozlišči, ki je v tem primeru enaka $O(n \cdot \Delta^2)$ oz. $O(n^3)$. Takšna časovna zahtevnost nastopi, ko algoritem začne verigo graditi v enem od vozlišč inkrementalnega povezavnega podgrafa, t.j. v eni od povezav osnovnega omrežja s katero prične graditi verigo in v najslabšem primeru nadaljuje preko vseh sosednjih povezav te povezave (v okviru skupnega krajišča v vhodnem omrežju). Na vsakem koraku mora algoritem izbrati povezavo, katere različnost z zadnjo povezavo v verigi je najmanjša. Zahtevnost tega koraka je $O(\Delta)$ oz. $O(n)$, saj je povezava v polnem grafu v vsakem vozlišču povezana z $\Delta - 1$ oz. $n-1$ drugimi povezavami, t.j. $O(\Delta)$ oz. $O(n)$. Algoritem zato potrebuje $O(n^2)$ časa, da obdela eno vozlišče in ker je vozlišč n, je celotna časovna zahtevnost enaka $O(n \cdot \Delta^2)$ oz. $O(n^3)$. Ker je izračun različnosti para povezav v splošnem lahko zahtevnejši od $O(1)$, se zahtevnost celotnega algoritma v obeh primerih lahko poveča.

Z uporabo preproste mere različnosti D_{SF} smo zahtevnost prilagojenega algoritma statistično izmerili na istih brezlestvičnih omrežjih kot pri osnovnem algoritmu. Z uporabo mere iz razdelka 4.5.1 bi zahtevnost prilagojenega algoritma sorazmerno s številom ključnih besed narasla. Rezultati za različna nabora ključnih besed so podani na prikazih 4.3 in 4.4. Rezultati so podobni. Čas računanja predvsem variira v odvi-



Slika 4.3

Prikaz časov izvajanja prilagojenega algoritma v odvisnosti od produkta števila vozlišč in maksimalne stopnje (Δ) v naključno ustvarjenih brezlestvičnih omrežjih. Vsako vozlišče je imelo eno in edinstveno ključno besedo.



Slika 4.4

Prikaz časov izvajanja prilagojenega algoritma v odvisnosti od produkta števila vozlišč in maksimalne stopnje (Δ) v naključno ustvarjenih brezlestvičnih omrežjih. Iz nabora 200 ključnih besed je bilo vsakemu vozlišču dodeljenih 10 naključno izbranih ključnih besed.

snosti od strukture vhodnih omrežij, ki so v obeh primerih enaka. Občuten vpliv na zahtevnost ima tudi večje število ključnih besed v drugem primeru, ki algoritem efektivno upočasni za $\sim 3{,}3$ krat, a je upočasnitev praktično enakomerna. Ocena časovne zahtevnosti algoritma na obeh množicah omrežij je $O(n \cdot \Delta)$. Videli bomo, da je v brezlestvičnih omrežjih navadno malo vozlišč z visokimi stopnjami, le ta pa bistveno vplivajo na povečanje časovne zahtevnosti. Ocena zahtevnosti je zato smiselna. Maksimalna dolžina verige v katerem koli iz obravnavanih omrežij je bila 15 skupin, največji inkrementalni povezavni podgraf pa reda 4.930.881 različnosti oz. povezav in 13.625 skupin oz. vozlišč.

Čas izvajanja prilagojenega algoritma smo merili tudi na realnih omrežjih, kot je na primer *Amazon*-ovo omrežje nakupov sorodnih produktov. Pripravili so ga *Yang* in sodelavci [62]. Iz njihovega omrežja smo izolirali štiri podomrežja in na njih zaganjali algoritem ter merili čase izvajanja. Podomrežja v originalnem omrežju predstavljajo največje povezane komponente, katerih najmanjša stopnja vozlišča je večja od izbrane meje. Največja komponenta z minimalno stopnjo 2 predstavlja približno 80% originalnega omrežja. V tabeli 4.1 so navedeni povprečni časi izvedb algoritma. Časi izvedbe algoritma so razmeroma visoki, vendar se moramo zadevati, da je algoritem v vseh primeru uporabil manj kot 400.000 povezav oz. manj kot 50.000 vozlišč velik inkre-

Tabela 4.1

Časi izvajanja prilagojenega algoritma na največjih komponentah $\mathbf{S_1}..\mathbf{S_4}$ v omrežju sorodnih produktov (vozlišč) spletne storitve *Amazon* z omejeno minimalno stopnjo. Najdaljša zgrajena veriga v katerem koli primeru je vključevala 24 skupin. Najkrajša izmed najdaljših verig za posamezen primer je vključevala 14 skupin. Največ shranjenih različnosti (povezav) v inkrementalnem povezavnem podgrafu izmed vseh primerov je bilo 345.764. V tem primeru je bilo v povezavnem podgrafu tud največ skupin (vozlišč), in sicer 47.525. Podatke smo izmerili na računalniku z 2,5GHz *Intel Core2Duo* CPE in 4GB RAM-a.

Omrežje	$\mathbf{S_1}$	$\mathbf{S_2}$	$\mathbf{S_3}$	$\mathbf{S_4}$
Minimalna stopnja	10	5	3	2
Maksimalna stopnja	52	172	298	361
Število vozlišč, n	21.765	145.592	268.380	309.154
Število povezav, m	48.077	455.093	822.523	900.163
Povprečni čas (vsako vozlišče svoja ključna b.) [s]	17	1.198	4.417	5.401
Povprečni čas (vsako vozlišče 10 izmed skupaj 200 ključnih b.) [s]	26	1.205	4.418	5.486

Tabela 4.2

Lastnosti nekaterih brezlestvičnih omrežij. Pomen omrežij je opisan v glavnem besedilu.

Omrežje	n	m	Δ	γ
Amazon	334.863	925.872	549	3,0
Brightkite	58.228	428.156	2.268	1,9
Wordnet	146.005	656.999	1.008	2,2
Topološki indeksi	120.953	209.522	382	2,6
Slovenska spletna trgovina	61.095	71.651	1.167	2,7

mentalni povezavni podgraf, ki predstavlja prostorsko najbolj kritičen del algoritma. V naši programski izvedbi je povezavni podgraf zasedel manj kot $100MB$ pomnilnika.

Znano je, da imajo brezlestvična omrežja lahko veliko maksimalno stopnjo [80], a je število vozlišč z veliko stopnjo, t.i. žarišč (*hubs*), navadno majhno. Število vozlišč p_k stopnje k namreč sledi porazdelitvi po potenčnemu zakonu (*power-law*) [80]: $p_k \sim k^{-\gamma}$, kjer je γ eksponent stopnje porazdelitve, katerega vrednosti so navadno med 2 in 3. Zgornje meje zahtevnosti našega algoritma temeljijo na maksimalni stopnji omrežja, vendar maksimalna zahtevnost nastopi le, če ima večina vozlišč visoko stopnjo, kar za brezlestvična omrežja ni značilno. Za občutek strukture brezlestvičnih omrežij v tabeli 4.2 podajamo nekaj lastnosti konkretnih primerov. Izpostavili smo že omenjeno *Amazon* omrežje, področno usmerjeneo socialno omrežje *Brightkite*, ki ga najdemo v množici podatkov *SNAP Datasets* [83], leksikografsko omrežje besed *Wordnet* iz podatkovne množice avtorja *Fellbaum* [84], omrežje soavtorstev na področju topoloških indeksov (poglavje 2), ki ga je avtor disertacije med drugimi skupaj z mentorjem pripravil za potrebe analize algoritmov [85] in omrežje sorodnih nakupov produktov ene izmed slovenskih spletnih trgovin, ki ga je prav tako konstruiral avtor disertacije.

Prostorska zahtevnost prilagojenega algoritma

Prostorska zahtevnost prilagojenega algoritma je pogojena z dolžino verige, ki jo algoritem zgradi in velikostjo povezavnega podgrafa, ki verigo obdaja. V najslabšem primeru se nam lahko zgodi, da algoritem konstruira verigo vseh povezav v osnovnem omrežju oz. verigo vseh vozlišč v povezavnem grafu, ki v primeru, da je vhodno omrežje poln graf, zahteva $O(m)$ oz. $O(n^2)$ prostora. Ker si prilagojeni algoritem spotoma "zapomni" tudi vse različnosti med povezavami osnovnega omrežja, ki jih je izračunal tekom napredovanja z verigo, to pomeni, da v najslabšem primeru zgradi in shrani tudi celoten povezavni graf. Prostorska zahtevnost konstrukcije povezavnega grafa nad polnim vhodnim omrežjem je reda $O(n^3)$, kar je tudi največja prostorska zahtevnost prilagojenega algoritma. Glavnino prostora torej predstavlja povezavni graf vhodnega omrežja. Ker imamo v praksi redko opravka z zelo gostimi omrežji, še redkeje pa se zgodi, da bi se nam pri izvajanju prilagojenega algoritma pojavljale dolge verige, je pričakovana prostorska zahtevnost le-tega nižja. Mnogo bolj pogosto imamo opravka z na primer brezlestvičnimi omrežji, v katerih ima razmeroma majhno število vozlišč visoko stopnjo. Če na takšnem omrežju zaženemo prilagojeni algoritem, se prej ali slej zgodi, da algoritem z verigo naleti na povezavo, katere krajišče ima visoko oz. maksimalno stopnjo Δ. V tem primeru se nam v inkrementalnem povezavnem podgrafu zgradi poln podgraf velikosti $O(\Delta^2)$. Ker so verige navadno kratke, vozlišč z visoko stopnjo pa je v dotičnih omrežjih razmeroma malo, je pričakovana prostorska zahtevnost prilagojenega algoritma enaka $O(k \cdot \Delta^2)$, $k << m$. Ob vsaki vnovični gradnji verige namreč povezavni podgraf sprostimo.

Prostor za shranjevanje skupinam pripadajočih seznamov aktivnih vozlišč ne vpliva na povečanje prostorske zahtevnosti. Vozlišče z najvišjo stopnjo Δ je lahko skupno največ Δ skupinam. V najslabšem primeru imamo n takšnih vozlišč v omrežju, kar pomeni, da potrebujemo $\Omega(\Delta \cdot n)$ prostora za hrambo seznamov aktivnih vozlišč. Slednje ne vpliva na red prostorske zahtevnosti algoritma.

4.8 Uporaba algoritma na omrežjih in evaluacija

Glavni problem pri vrednotenju ustreznosti rezultatov algoritmov za razvrščanje je v tem, da stroga definicija skupine (združbe) ne obstaja [62]. Dokler takšna definicija ne obstaja, je nemogoče enotno primerjati rezultate različnih algoritmov za razvrščanje.

V optimizacijskih algoritmih, ki na primer temeljijo na kriteriju modularnosti in

podobnih, je združba algoritmično določena na podlagi kriterija samega. Vrednotenje algoritma na podlagi primerjave njegovih rezultatov v obliki združb z rezultati nekega drugega algoritma, ki temelji na drugačnem kriteriju, je zato vprašljiv. Testiranje algoritmov za iskanje združb zato velikokrat poteka z opazovanjem njihovih rezultatov na manjših empiričnih omrežjih in na umetnih testnih omrežjih [86]. Predvsem se jih vrednoti v smislu lastnosti, za katere so bili razviti. Dobimo pravzaprav odgovor na vprašanje, katera kriterijska funkcija je za dano nalogo boljša. Primerjava nehierarhičnih in hierarhičnih algoritmov lahko predstavlja dodaten izziv. Primerjavo lahko izvedemo na primer med razbitjem nehierarhičnega algoritma in razbitjem hierarhičnega algoritma, ki vsebuje enako število skupin. V primerjavo lahko vzamemo tudi razbitje hierarhičnega algoritma, za katerega je gostota razbitij [70] največja. Pri tem se moramo zavedati, da lahko skupine kakšnega drugega razbitja hierarhičnega algoritma bolje sovpadajo s skupinami razbitja nehierarhičnega algoritma. Z opisanimi težavami se ukvarjajo *Tibély* in soavtorji [86].

Z uporabo *Lehmann*-ove različnosti (4.23) naš prilagojeni algoritem postane enakovreden njihovemu postopku [1]. Za izbrani nesosednji povezavi u in v je namreč njihova različnost vedno enaka maksimalni možni ($d_L(u, v) = 1$), kar je zaradi relacijske omejitve v našem algoritmu predpogoj za enakovrednost. Na sosednjih povezavah ob uporabi *Lehmann*-ove različnosti pa naš algoritem deluje enako. Zaradi podobnosti obeh algoritmov pri izbiri običajnih različnosti tu primerjave z uveljavljenimi algoritmi ne bomo izvajali, saj avtorji [1] svoj algoritem že primerjajo z uveljavljenimi metodami, kot so razširjanje po polnih podgrafih, s požrešno optimizacijo modularnosti in z algoritmom *Infomap*. Ker z uporabo novih različnosti naš algoritem poleg strukture upošteva tudi lastnosti elementov omrežja, njegovi rezultati niso kompatibilni z rezultati algoritmov, ki teh lastnosti ne upoštevajo. Za ovrednotenje rezultatov našega algoritma se zato opremo na preprosto metodo primerjanja prekrivajočih razbitij [87]. Rezultate algoritma v obliki razbitij na skupine primerjamo s skupinami, ki so jih v primernih realnih omrežjih z dodatnimi lastnostmi njegovih elementov vnaprej določili ljudje [62]. Rezultate primerjave nato empirično interpretiramo.

4.8.1 Demonstracijsko umetno omrežje

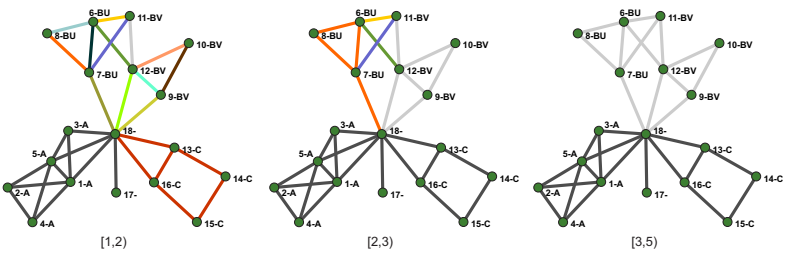
Preden se osredotočimo na primer realnega omrežja pokažimo, da je algoritem pravzaprav sposoben razvrščati sorodna območja v omrežju. Sestavili smo majhno omrežje, trikrat prikazano na sliki 4.5. Sestoji iz 18 vozlišč in 31 povezav, razporejenih v bolj

povezane dele, ki se združijo v centralnem (18.) vozlišču. Vozliščem v zgornjem delu na levi smo dodelili ključni besedi $\{B, U\}$, tistim na desnem delu $\{B, V\}$. Spodnji del omrežja je sestavljen iz treh krakov, vozliščem na levem kraku smo dodelili ključno besedo $\{A\}$, srednjemu kraku nismo dodelili ključnih besed, desnemu kraku pa smo dodelili ključno besedo $\{C\}$. Stično vozlišče v sredini nima ključnih besed. Prikazi omrežja na sliki 4.5 imajo z barvo kodirane skupine povezav kot jih dobimo iz razbitij, določenih z rezanjem pri različnih višinah v hierarhiji. Hierarhijo smo dobili z uporabo prilagojenega algoritma in mere različnosti (4.12), D_{SC}. Vozlišča smo označili z oznakami, katerim sledijo nizi ključnih besed. Na sliki 4.6 je prikazana sploščena hierarhija. Sploščenje je postopek, ko rekurzivno od korena hierarhije v vsaki vejitvi ω združimo še vse njej neposredne podvejitve z enako različnostjo kot jo ima vejitev ω.

Po pričakovanjih bi moral algoritem v omrežju zaznati povezana področja s podobnimi ključnimi besedami. Na podlagi prikaza hierarhije se v slednje lahko prepričamo. Sicer drži, da rez na fiksnem nivoju preko celotne hierarhije rezultira v razbitje skupin povezav oz. razvrstitve skupin vozlišč, ki imajo v skladu z uporabljeno mero različnosti D_{SC} manj podobne množice ključnih besed. Na začetku je vsaka povezava v svoji skupini (na sliki 4.5 to stanje ni prikazano). Pri različnosti 1 se sosednje skupine povezav z vozlišči z enakimi ključnimi besedami združijo v novo skupino. (Povezava (17– : 18–) se lahko razporedi v katerikoli skupino z eno ključno besedo, t.j. v skupino s ključno besedo $\{A\}$ ali v skupino s ključno besedo $\{C\}$.) V sledečih iteracijah se pri različnosti 2 združijo skupine povezav z dvema različnima ključnima besedama, kar se odraža v združeni skupini povezav s ključnima besedama $\{B, U\}$ in v združeni skupini s ključnima besedama $\{B, V\}$. Skupini povezav s ključno besedo $\{A\}$ in $\{C\}$ se nato združita v skupino s ključnima besedama $\{A, C\}$. Pri različnosti 3 se združijo skupine s tremi različnimi ključnimi besedami... Pri različnosti 5 se končno vse skupine združijo v eno samo.

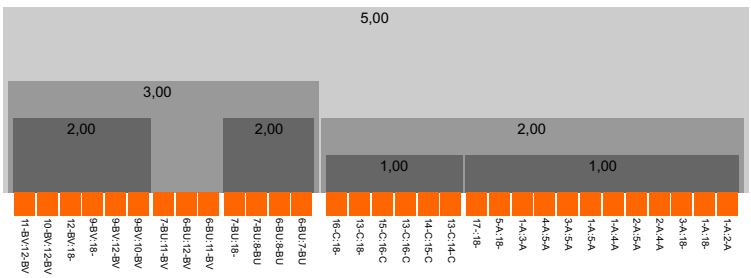
Mera različnosti D_{SC} favorizira združevanje manjših skupin, ki imajo inherentno manjše število različnih ključnih besed. Pri njej je pomembno le število različnih ključnih besed, in sicer ne glede na število skupnih ključnih besed. Na primer različnost množic s ključnimi besedami $\{A\}$ in $\{B\}$ je 2, različnost med $\{B, U\}$ in $\{B, V\}$ pa 3.

Poglejmo isti primer še z uporabo mere različnosti v enačbi (4.12), D_{SF}. Prikaz dobljene hierarhije je prikazan na sliki 4.7. Za vsako različno razvrstitev, ki temelji na vseh možnih vrednostih rezanja dobljene hierarhije, smo uporabili mero odstopanja δ za izračun odstopanja med dobljenimi razvrstitvami $\mathscr{C}_k = \{\mathbf{C}_{k,1}, \mathbf{C}_{k,2}, .., \mathbf{C}_{k,|\mathscr{C}_k|}\}$ in



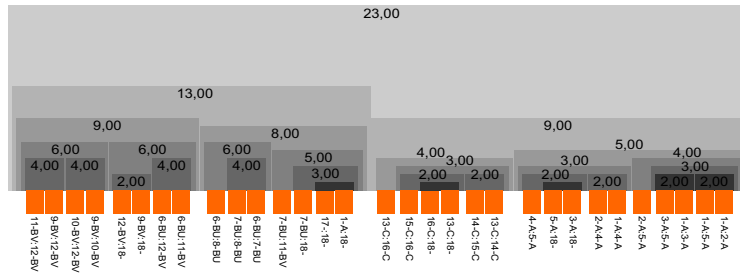
Slika 4.5

Tri razbitja na skupine povezav v demonstracijskem omrežju. Vozlišča so označena z zaporednimi številkami, katerim sledijo nizi njihovih ključnih besed. Skupine smo dobili z rezanjem hierarhije, prikazane na sliki 4.6, ki jo je ustvaril naš prilagojeni algoritem. Intervali različnosti pri posameznem rezu so prikazani pod sliko omrežij. Kodirali so jih z barvami.



Slika 4.6

Sploščena hierarhija povezav na demonstracijskem omrežju. Hierarhijo smo dobili s prilagojenim algoritmom in mero različnosti D_{SC}. Vrednosti različnosti so zapisane nad posameznimi vejitvami. Na dnu slike so prikazane povezave, in sicer v obliki parov sosednjih vozlišč. Sliko smo pripravili v našem orodju *net.Plexor* 6.4.3.



Slika 4.7

Sploščena hierarhija povezav demonstracijskega omrežja, ki smo jo dobili s prilagojenim algoritmom in mero različnosti D_{SF}. Vrednosti različnosti so prikazane na vrhu vejitev. Povezave so v obliki parov sosednjih vozlišč prikazane na dnu. Sliko smo pripravili v našem orodju *net.Plexor* 6.4.3.

vnaprej določeno (naravno) razvrstitvijo $\mathscr{G} = \{\mathbf{G}_1, \mathbf{G}_2, .., \mathbf{G}_{|\mathscr{G}|}\}$. Odstopanja so podana na prikazu 4.8. Izbrano mero za določanje odstopanja med razvrstitvami $\delta(\mathscr{C}_k, \mathscr{G})$ poznamo pod imenom *Best Match* [87]. Z uporabo števila premikov μ, potrebnih za pretvorbo skupine $\mathbf{C}$ v $\mathbf{C}'$:

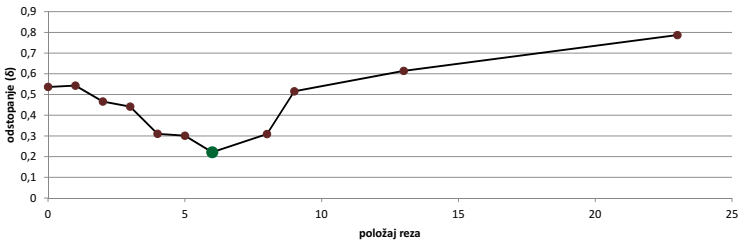
$$\mu(\mathbf{C}, \mathbf{C}') = |\mathbf{C} \oplus \mathbf{C}'| \tag{4.40}$$

lahko za vsako skupino $\mathbf{C}_{k,i} \in \mathscr{C}_k$ izračunamo njegovo najboljšo predstavnico $\mathbf{G}^* \in \mathscr{G}$ tako, da $\mathbf{G}^* = \operatorname{argmin}_{\mathbf{G} \in \mathscr{G}} \mu(\mathbf{C}_{k,i}, \mathbf{G})$. Naj opomnimo, da z normalizacijo $\mu(\mathbf{C}, \mathbf{C}')$ z $|\mathbf{C} \cup \mathbf{C}'|$ dobimo *Jaccard*-ovo različnost $D_J(\mathbf{C}, \mathbf{C}')$, podano v enačbi (4.2). Vsota števil premikov za pretvorbo vsake skupine razvrstitve $\mathscr{C}_k$ v njene najboljše predstavnice v razvrstitvi $\mathscr{G}$ predstavlja kako dobro $\mathscr{G}$ predstavlja $\mathscr{C}_k$. Ker bi želeli, da je mera odstopanja simetrična, je smiselno prišteti še vsoto števil premikov pretvorb skupin iz $\mathscr{G}$ v njihove najboljše predstavnice v $\mathscr{C}_k$. Tako dobimo:

$$\delta(\mathscr{C}_k, \mathscr{G}) = \sum_{\mathbf{C} \in \mathscr{C}_k} \min_{\mathbf{G} \in \mathscr{G}} \mu(\mathbf{C}, \mathbf{G}) + \sum_{\mathbf{G} \in \mathscr{G}} \min_{\mathbf{C} \in \mathscr{C}_k} \mu(\mathbf{G}, \mathbf{C}). \tag{4.41}$$

Mero *Best Match* in podobne so razvili za potrebe kvantitativnega določanja odstopanja med pari razvrstitev, katerih skupine se lahko prekrivajo. V naprej znana razvrstitev vozlišč iz našega primera je na primer določena s petimi skupinami: vozlišča s ključno besedo $\{A\}$, vozlišča s ključnima besedama $\{B, U\}$, vozlišča s ključnima besedama $\{B, V\}$ in vozlišča s ključno besedo $\{C\}$, in sice ne glede na povezanost. Vozlišča brez ključnih besed so bila dodeljena peti ločeni skupini.

Na demonstracijskem omrežju smo poiskali referenčne vrednosti rezov za katere je odstopanje med razvrstitvami vozlišč prilagojenega algoritma in v naprej znano razvr-



Slika 4.8

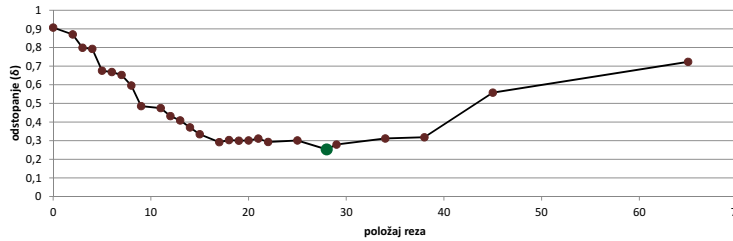
Prikaz odstopanj δ med vnaprej določenimi razvrstitvami in razvrstitvami, dobljenimi z enakomernim rezanjem v hierarhiji, ki smo jo na demonstracijskem omrežju izračunali s prilagojenim algoritmom. Najboljši rez dobimo pri vrednosti 6 in $\delta = 0,221$. Označen je z močnejšo točko.

stitvijo nizko, da jih lahko primerjamo z vrednostmi rezov pri večjih realnih omrežjih. Prikaz z večjim realnim omrežjem je podan na sliki 4.9 v razdelku 4.8.2. Razlaga prikazov na slikah 4.8 in 4.9 sledi na koncu razdelka 4.8.2.

4.8.2 Primer na realnem omrežju – Spletna trgovina

Na podatkih slovenske spletne trgovine, ki med drugim ponuja elektronsko opremo, smo konstruirali omrežje. Podoben postopek so na podatkih spletne storitve *Amazon* uporabili *Yang* in ostali raziskovalci [62]. Omrežje smo sestavili glede na izdelke, ki so jih stranke kupile oz. preiskovale skupaj z drugimi izdelki. Za osnovo nam je služila razvrstitev izdelkov v kategorije v katere so bili izdelki razvrščeni. Vsak izdelek je opredeljen z lastnostmi, kot so na primer barva, vrsta, velikost, oblika... V našem konkretnem primeru smo opazovali elektronsko merilno opremo, pretežno sonde osciloskopov različnih vrst. Vsak izdelek je bil v trgovini razvrščen v eno od štirih vrst. Nad opazovanimi izdelki smo konstruirali omrežje, tako da smo izdelke povezali glede na relacijo, kako so kupci iskali sorodne izdelke. Trgovina je zasnovana tako, da na podlagi iskanj izdelkov spremlja, če v primeru, da obstaja dovolj velik interes (število kupcev) za določen izdelek, pri tem hkrati obstaja tudi zanimanje za alternativni izdelek. Če temu je tako, se alternativni izdelek pri prvem izdelku dodeli v seznam "drugi so izbirali med". Na sestavljenem omrežju smo zagnali prilagojeni algoritem in na dobljeni hierarhiji poiskali primerno višino reza, s katerim smo dobili skupine, ki dobro pokrivajo posamezna področja izdelkov. Primerjava rezultatov je podana s prikazom na sliki 4.9.

Prikaza 4.8 in 4.9 izražata pričakovano. Če vse povezave razvrstimo v eno samo skupino (levo v prikazih, kjer rezanje nastopi pri nivoju 0) je odstopanje do podane naravne razvrstitve visoka. Če rezanje hierarhije nastopi pri višjih nivojih, postopoma dobimo optimalni rez, kjer se dobljene skupine najmanj razlikujejo od naravnih oz.



Slika 4.9

Prikaz odstopanja δ med naravno razvrstitvijo in razvrstitvami, ki smo jih z rezanjem dobili iz hierarhije, pridobljene s prilagojenim algoritmom na omrežju sond za osciloskope. Najboljši rez se nahaja pri vrednosti 28 in $\delta = 0,253$ in je označen s poudarjeno točko.

vnaprej določenih. S povečevanjem nivoja reza proti največji višini, kjer se vsako povezavo razvrsti v lastno skupino, odstopanje dobljene razvrstitve in naravne razvrstitve znova narašča.

Iz demonstracijskih razlogov smo v hierarhijah rezali le pri enakomernih nivojih različnosti. Zavedati se moramo, da bi z dobro izbranim ne-enakomernim nivojem rezanja (t.j. veje v hierarhiji bi rezali na različnih nivojih) najverjetneje dobili razvrstitve, ki bi še manj odstopale od vnaprej določenih.

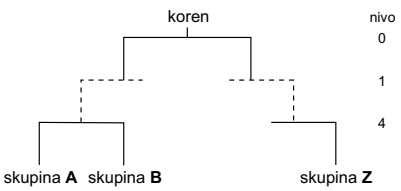
4.8.3 *Primer realnega omrežja – Omrežje sodelovanj avtorjev na področju topoloških indeksov*

V članku [85] smo preiskovali bibiliografska omrežja na temo topoloških indeksov. Omrežja smo zgradili (2.3) iz podatkov spletne storitve *Web of Science*. Več o tem lahko bralec najde v poglavju 2. Osredotočili smo se na omrežje sodelovanj avtorjev. V omrežju sodelovanj so avtorji predstavljeni z vozlišči, njihova sodelovanja, izražena v obliki skupnih del pa s povezavami. Na podlagi ključnih besed prispevkov avtorjev smo avtorje (vozlišča) opremili s frekvencami oz. utežmi za posamezno ključno besedo. (Z utežmi bi alternativno lahko opremili povezave.) Uteži smo izračunali s *tf-idf* uvrščanjem [44]. Glede na frekvence ključnih besed posameznih avtorjev smo s prilagojenim algoritmom izdelali hierarhijo podobnosti avtorjev. Za primer navajamo tri množice ključnih besed, ki pripadajo izbranim vejam v dobljeni hierarhiji. Prikazane so na sliki 4.10. Množici $\mathbf{A}$ in $\mathbf{B}$ ustrezata vejam s skupnim vejiščem, množica $\mathbf{Z}$ pa pripada oddaljeni veji, ki se s prejšnjima sreča šele v korenu hierarhije. ($|\mathbf{A}| = 491$, $|\mathbf{B}| = 533$, $|\mathbf{Z}| = 300$). Po pričakovanjih imata množici $\mathbf{A}$ in $\mathbf{B}$, v nasprotju z množico $\mathbf{Z}$, podoben nabor ključnih besed, kar je s slike 4.11 razvidno: $|\mathbf{A} \cap \mathbf{B}| = 227$, $|\mathbf{A} \cap \mathbf{Z}| = 59$. Čeprav bi bila množica $\mathbf{Z}$ dvakrat večja, tako da bi imela približno enako besed kot

Tabela 4.3

Seznam avtorjev, ki pripadajo izpostavljenim skupinam. Pozoren bralec bo opazil, da se avtor *I. Gutman* (GUTMAN_I) pojavlja v vseh treh skupinah.

A	AHMADI_A, AIRES-DE_J, ALIPOUR_M, ALIZADEH_Y, AMINI_K, AREZOOMA_M, ASHRAFI_A, AZAD_A, AZARI_M, BADAKHSH_-L, BAHRAMI_A, CIOSLOWS_J, CVETKOVI_, DARAFSHE_M, DAS_K, DOROSTI_N, DOSLIC_T, DU_Z, DURDEVIC_J, ELBASIL_S, ELIASI_M, ESTRADA_E, FAGHANI_M, FARHAMI_P, GHAZI_M, GHOLAMI-_F, GHOLAMI_N, GHOLIZAD_S, GHORBANI_M, GILANI_A, GODSIL_C, GOJAK_S, GUTMAN_I, HAMZEH_A, HEMMASI_M, HEYDARI_A, HOSSEIN-_S, HOSSEINZ_M, HOU_Y, ILIC_A, IRANMANE_-A, JADDI_M, JALALI_M, KAFRANI_A, KATONA_G, KHAKI_A, KHAKSAR_A, KHALIFEH_M, KHORMALI_O, LAM_P, LI_X, LINERT_-W, LOGHMAN_A, MAHMIANI_A, MAIMANI_H, MANDLOI_M, MANOOCHE_B, MATELJEV_M, MESGARAN_H, MILUN_M, MINAILIU_O, MIRZAIE_S, MIRZARGA_M, MOGHARRA_M, NADJAFI-_M, NAGY_C, NAGY_K, NIKZAD_P, PAKRAVES_Y, PARV_B, PESEK_I, POP_M, RADENKOV_S, REZAEI_F, ROUVRAY_D, SAATI_H, SABAGHIA_H, SAHELI_M, SALEHI_N, SEYEDALI_S, SHABANI_H, SHAKERAN_S, SHI_Y, SHIU_W, SOLEIMAN_B, STANKOVI_S, TAERI_B, TAHERKHA_B, TEHRANIA_A, TOMOVIC_Z, TRINAJST_, WAGNER_S, YAN_W, YANG_B, YAZDANI_J, YEH_Y, YOUSEFI-_H, YOUSEFI_S, ZAEEMBAS_A, ZERAATKA_M, ZHOU_B, ZIGERT_P, ZIVKOVIC_T
B	ABELEDO_H, AOUCHICH_M, ARAUJO_O, ARSIC_B, ASHRAFI_A, CAPOROSS_G, CHEPOI_V, CIGHER_S, CLARK_L, CMILJANO_N, DAS_-K, DELAPENA_J, DOBRYNIN_A, DOMOTOR_G, DOSLIC_T, DU_Z, DURDEVIC_J, FATH-TAB_G, FENG_L, FURTULA_B, GHOLAMI-_F, GOJAK_S, GORDEEVA_E, GRAOVAC_A, GUEVARA_N, GUTMAN_I, HAMZEH_A, HANSEN_P, HEMMASI_M, HOSSEIN-_S, HOSSEINZ_M, ILIC_A, ILIC_M, JUVAN_M, KARBASIO_A, KLAVZAR_S, KOVSE_M, LEPOVIC_M, LINERT_W, LIU_B, LO_S, LUO_Y, MARKOVIC_Z, MARSHALL_G, MEL'NIKO_L, MELOT_H, MILJKOVI_O, MILOSAVL_S, MOGHARRA_M, MOHAR_B, MORALES_D, MOTOC_I, NADJAFI-_M, NAGY_C, PAVLOVIC_L, PESEK_I, PLAVSIC_D, POP_M, POPOVIC_L, RADA_J, RADENKOV_S, RAJAPAKS_A, SAATI_H, SAHELI_-M, SALEM_K, SEITZ_W, SHAO_J, SHRIVAST_A, SOSKIC_M, STANKOVI_S, STEVANOV_D, TOMOVIC_Z, URSU_O, VIDOVIC_D, VIZITIU_A, XU_K, YARAHMAD_Z, YU_G, ZEROVNIK_J, ZHANG_F, ZHAO_H, ZIGERT_P
Z	AGRAWAL_V, BABIC_D, BALABAN_A, BALABAN_T, BASAK_S, BRISSEY_G, GRUNWALD_G, GUTMAN_I, HARARY_F, IVANCIUC_O, JAKLIC_G, KHADIKAR_P, KOPECKY_K, KRILOV_G, LEPOVIC_M, LERS_N, MEDELEAN_M, MILICEVI_A, MINAILIU_O, NATARAJA_R, PAVAN_M, PISANSKI_T, POMPE_M, POPOVIC_L, RANDIC_M, RUCKER_C, SABLJIC_A, STANKOVI_S, TRINAJST_N, VIDOVIC_D, VRACKO_M, VUKICEVI_D, YAMAGUCH_T

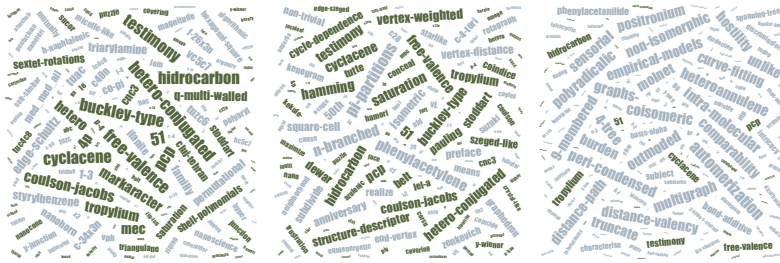


Slika 4.10

Shema področij skupin avtorjev v hierarhiji, ki smo jo iz omrežja sodelovanj avtorjev dobili s prilagojenim algoritmom.

množici **A** oz. **B**, bi z njima v preseku imela komaj pol toliko skupnih besed, kot jih ima množica $\mathbf{A} \cap \mathbf{B}$. Ne smemo pozabiti na naslednje. Čeprav lahko pričakujemo podobnost množic besed, ki pripadajo bližnjim vejam v hierarhiji, nasprotno ni nujno. Za množici, ki v hierarhiji pripadata oddaljenim vejam, ni nujno, da sta močno različni. V izvornem omrežju se namreč lahko zgodi, da obstajajo oddaljena, vendar podobna neodvisna področja, ločena z dolgimi potmi. Zaradi relacijske omejitve algoritma v hierarhiji lahko dobimo oddaljene veje, katerih pripadajoče množice ključnih besed si utegnejo biti podobne.

Ključne besede temnejše barve v levem in srednjem kvadratu na sliki 4.11 pripadajo ključnim besedam množice $\mathbf{A} \cap \mathbf{B}$. Temnejše ključne besede v desnem kvadratu pripadajo množici $\mathbf{A} \cap \mathbf{Z}$. Opisi avtorjev [85], ki pripadajo posameznim izpostavljenim skupinam, so navedeni v tabeli 4.3.



Slika 4.11

Množice ključnih besed, ki pripadajo posameznim skupinam avtorjev v hierarhiji, ki smo jo iz omrežja sodelovanj avtorjev dobili s prilagojenim algoritmom. Zelene (temnejše) ključne besede na levi in srednji sliki ustrezajo ključnim besedam iz preseka skupin $\mathbf{A} \cap \mathbf{B}$. Temnejše ključne besede na desni sliki ustrezajo skupini $\mathbf{A} \cap \mathbf{Z}$.

4.9 Diskusija

Prilagojeni algoritem smo prvenstveno zasnovali za uporabo na neusmerjenih omrežjih, a ga je možno predelati tako, da bo upošteval usmeritev povezav. Pri združevanju najbližjih skupin v smislu mere različnosti bi lahko poleg povezanosti upoštevali še dodatne relacijske omejitve [88]. V osnovi algoritem ohranja šibko povezanost, saj se v vsaki iteraciji združita dve skupini povezav, če in samo če sta povezani, t.j. če imata skupno (aktivno) vozliče. Če bi upoštevali še omejitve kot na primer povezanost do izbranega središčnega vozlišča, strogo povezanost ali kakšno drugo pravilo, bi usmerjenost povezav dobila pomen.

Zanimivo vprašanje se pojavi tudi v zvezi z nepovezanimi vhodnimi omrežji. Za vsako povezano komponento lahko na primer pripravimo ločeno hierarhijo in jih na najvišjem nivoju združimo s skupnim korenom. Ker se v disertaciji ukvarjamo tako z vizualizacijo omrežij, kot tudi s hierarhičnim razvrščanjem omrežij na podlagi opisov elementov omrežij, bi bralca utegnilo zanimati delo *M. Štajdohar*-ja in sodelavcev [89], v katerem izpostavljajo prav problem prikazov oz. umestitve ločenih povezanih komponent omrežij na podlagi opisov poleg strukture omrežja.

V razdelku o različnostih 4.5 smo se v veliki meri osredotočili na različnosti, ki temeljijo na podlagi ključnih besed avtorjev na primer v omrežju sodelovanja, vendar na tem mestu še enkrat poudarimo, da lahko mere različnosti izberemo poljubno na podlagi drugih lastnosti. Rekurzivno izračunljive mere različnosti temeljijo na različnosti d med posameznimi elementi omrežja, ki jo lahko določimo praktično povsem svobodno, in sicer na podlagi poljubnih lastnosti elementov kot so barva, velikost, oblika ali katera koli druga lastnost v poljubni merski lestvici, ne nujno imenski. Pri direktnih merah različnosti prav tako lahko upoštevamo poljubne lastnosti elementov omrežja,

le nekoliko bolj moramo biti pazljivi, če želimo ohraniti monotonost. Pri opisanih vzorcih rekurzivnih različnostih je le-ta v primerjavi z direktnimi različnostmi že avtomatično zagotovljena.

Pri merah različnostih še enkrat izpostavimo pomen monotonosti. Monotonost je izredno pomembna lastnost različnosti, saj hierarhije, ki jih z njimi dobimo, pri katerikoli vrednosti rezanja zagotavljajo enolično določene razvrstitve. Pomembna je pri našem prilagojemen algoritmu, saj pravilnost le-tega pri uporabi nemonotonih mer ni zagotovljena. Monotonost je pomembna tudi pri obeh vidikih združevalnega algoritma, ki ju obravnavamo v sledečih razdelkih, t.j. pri stabilnosti algoritma (4.9.1) in pri paralelni različici algoritma (4.9.2). Medtem, ko klasični postopki rekurzivnih izračunov različnosti monotonost avtomatično zagotavljajo, ji moramo pri načrtovanju direktno izračunljivih mer različnosti nameniti posebno pozornost.

4.9.1 Stabilnost prilagojenega algoritma

Če pri svojem delovanju prilagojeni algoritem naleti na več parov skupin v prednostni vrsti, med katerimi je različnost enaka, se ne glede na ostale alternativne izbire prvi par skupin požrešno izbere. Zaporedje parov z istimi različnostmi na koncu prednostne vrste je algoritmično določeno z zaporedjem navedbe povezav v vhodni datoteki omrežja ter s programsko izvedbo prednostne vrste, ki jo algoritem uporablja. Gre za znano problematiko v razvrščanju, ki jo *Morgan* s sodelavci obravnava v svojem delu [90].

Pri analizi stabilnosti algoritma smo opazovali razlike med rezultati, ki smo jih dobili, če smo algoritem zagnali na enakem omrežju, katerega zaporedji vozlišč in povezav smo večkrat naključno premešali. Uporabili smo bibliografsko omrežje avtorjev iz področja topoloških indeksov, ki smo ga srečali že v poglavju 4.8.3. Hierarhije, ki jih je algoritem vrnil, so se razlikovale največ do preureditev njihovih vej. To pomeni, da se je vrstni red vej v hierarhiji od primera do primera razlikoval, vendar so bile strukturno vse hierarhije enake. Z drugimi besedami: z obrati vej v vejitvah dobljenih hierarhij je bilo hierarhije možno preurediti eno v drugo. Vrednost različnosti na posameznih vejitvah v hierarhiji so bile v vseh primerih enake. Vse rezultirajoče hierarhije formalno predstavljajo isto kanonsko obliko hierarhije. Do določene mere to pomeni, da če je vhodno omrežje veliko in/ali imajo njegovi elementi dovolj različnih ključnih besed, algoritem na njem deluje stabilno. Z uporabo na primer manj diskriminatorne mere različnosti na kakšnem drugem omrežju, bi seveda dobili manj skladne, vendar še vedno podobne rezultate. Natančnejše analize stabilnosti algoritma nismo izvajali.

Prikladen in za programsko izvedbo enostaven dodatek algoritmu bi bila možnost informiranja uporabnika o prisotnosti enakovrednih alternativnih izbir na prednostni vrsti in s tem obveščanje o morebitnih alternativnih rešitvah. Še boljšo rešitev problema vpeljujeta avtorja *Fernández* in *Gómez* [91], ki v postopku razvrščanja predlagata implicitno združevanje vej hierarhije z enakimi različnostmi v skupna vejišča. Namesto združevanja parov vej lahko tako pride do združevanja večih v t.i. multidendrogram oz. sploščeno hierarhijo, ki smo jo omenili v razdelku 4.8.1.

4.9.2 Paralelizacija

Glede na ralacijsko omejitev s katero algoritem deluje, se oddaljena področja v izvornem omrežju ne morejo združiti v skupino v manj iteracijah, kolikor je dolžina najkrajše poti med temi področji. Algoritem bi zato lahko razdelili med več niti, ki bi razvrščanje oddaljenih področij opravile paralelno. Predpogoj za pravilno delovanje paralelnega algoritma, tako osnovnega kot prilagojenega, je uporaba monotonih mer različnosti. Pri prilagojenem algoritmu je to pravzaprav že osnovna zahteva. V nasprotnem primeru bi lahko prišlo do težav pri združevanju delov hierarhije, ki bi jih ustvarile posamezne niti. Vozliščni (*node-centric*) postopek, podoben opisanemu, je bil omenjen v delu *Yang*-a [62].

4.9.3 Zaključek

V pričujočem poglavju smo predstavili osnovni združevalni algoritem za združevanje vozlišč v omrežjih in ga prilagodili za razvrščanje povezav v omrežju. Predstavili smo nekaj ustreznih mer različnosti, ki jih pri delovanju obeh algoritmov nujno potrebujemo. Mere različnosti smo razdelili v dve skupini, in sicer v skupino t.i. rekurzivno izračunljivih in v skupino direktno izračunljivih. Predstavili smo tudi nekaj novih mer različnosti, ki so poleg strukture omrežja sposobne upoštevati še pomen lastnosti elementov omrežij. Algoritmom za razvrščanje tako omogočijo boljše oz. bolj natančno razločevanje med skupinami globlje v hierarhiji. Na primerih tako umetnih, kot tudi na realnih omrežjih smo nato prikazali delovanje algoritmov, analizirali smo njihovi zahtevnosti in pokazali, da so njuni rezultati smiselni. Prikazali smo tudi njuno uporabno vrednost v praksi.

Nekaj dodatnih podrobnosti o programski izvedbi prilagojenega algoritma lahko bralec najde v poglavju 6 o naši knjižnici za delo z velikimi omrežji, *net.Plexor*.

5

Prikaz mobilnega prometa D4D

5.1 Uvod

V poglavju bomo prikazali rezultate našega dela [92] v povezavi z izzivom *D4D*, ki ga je organiziral francoski mobilni operater – *Orange*. Na podlagi informacij o klicih v mobilnem omrežju na Slonokoščeni obali so pripravili podatke in nami jih posredovali. V prvi meri smo se osredotočili na statičen geografski prikaz mobilnega prometa med baznimi postajami. V skladu z obliko podatkov smo prikazali promet oz. trajanje in število klicev med pari baznih postaj. Da nebi povsem zapostavili časovnega vidika dogajanja v mobilnem omrežju, smo konstruirali dva videa s katerima prikazujemo dogajanje v omrežju skozi čas. Raziskali smo še odvisnost med vrstama prometa, t.j. kako sta trajanje klicev in število klicev povezana ter kako promet sovpada z razdaljo parov baznih postaj med katerimi je potekal. Prikazali smo tudi nekaj rezultatov razvrščanja baznih postaj po času delovanja in vključili nekaj prikazov, ki prikazujejo najbolj izrazit promet v obeh načinih.

Na tem mestu izpostavimo, da je delo v ozadju tega poglavja v veliki meri eksperimentalno in hipotetično. Konkretnejših zaključkov o podatkih zato ne navajamo, smo pa v skladu z dobrimi praksami teorije prikazov [45, 46] razvili novo metodo za prikazovanje velikih omrežij, s katero želimo podati alternativni pogled na podatke.

Metodo lahko uporabimo tudi kot enega izmed načinov pregledovanja hierarhij. Pri tem je za uporabo predpogoj, da imamo na voljo podatke o koordinatah vozlišč v omrežju, ki bi ga želeli pregledovati. V primeru posredovanih podatkov so lokacije baznih postaj – vozlišč znane. Dodatno naj omenimo, da je opisana metoda časovno precej potratna in je v realnem času, razen za prikaze zelo majhnih omrežij, ne moremo uporabljati. Povsem mogoče pa jo je uporabiti tako, da se ustrezne slike pripravi vnaprej, pregledovanje pa kasneje opravi na bazi shranjenih slik. Metoda je predvsem primerna za pregledovanje omrežij na višjih nivojih, v obliki zemljevida omrežja, ki nam služi za globalno oz. višjenivojsko orientacijo, medtem ko omrežje na nižjih nivojih pregledujemo oz. prikazujemo s kakšno drugo metodo. Z estetskega vidika verjamemo, da metoda ustvari lepe prikaze in je hkrati dober primer prikaza omrežja s povzetki. Osnove prikazov omrežij s skeleti in povzetki so podane v poglavju 3.

5.2 Podatki mobilnega omrežja

V opisu podatkov [93] se nahajajo opisi štirih množic podatkov, ki smo jih prejeli od organizatorjev v analizo. Nabori podatkov so sledeči:

1. promet med baznimi postajami v obliki števila in trajanja klicev za celotno pol-letno obdobje opazovanja,

2. sledi mobilnih uporabnikov v visoki ločljivosti (do natančnosti posamezne bazne postaje) v 14-dnevnih časovnih intervalih,

3. sledi mobilnih uporabnikov v nizki ločljivosti (do natančnosti vnaprej določenih skupin baznih postaj) v celotnem pol-letnem obdobju,

4. komunikacijska egocentrična omrežja mobilnih uporabnikov.

V analizah se pretežno ukvarjamo s podatki iz prvega nabora na podlagi katerih smo pripravili statične prikaze. O njih pišemo v sledečih razdelkih. V razdelku 5.3.1 promet prikažemo s klasičnimi diagami z daljicami. V razdelku 5.3.2 opišemo pomanjkljivosti klasičnih prikazov in vpeljemo novo vrsto prikazov, ki jih imenujemo razpršeni prikazi. Le-ti so v osnovi različica prikazov gostote omrežja – porazdelitveni prikazi omrežij (*density map diagrams*), a smo jih prilagodili konkretnemu problemu vizualizacije omrežja mobilnega prometa. V naslednjem razdelku 5.3.3 vpeljemo način prikaza prometa s t.i. zvezdnimi prikazi. Osnovna ideja obeh vrst prikazov je v aditivnem izrisovanju vzorcev oz. "tekstur" gostote prometa.

Preko celotnega pol-letnega obdobja nabora podatkov smo združili trajanja in števila klicev med baznimi postajami in skupne vsote uporabili za vrednosti uteži povezav – komunikacij med pari baznih postaj in tako sestavili dve omrežji. Obe omrežji sestojita iz baznih postaj – vozlišč, združene vsote trajanj pogovorov med njimi oz. združene vsote števil klicev pa predstavljajo moč povezav. V kolikor komunikacije za dani par baznih postaj ni bilo, tam povezave ni. Omrežji sta predstavljali osnovo za izris opisanih prikazov in analiz v razdelkih 5.3.4, 5.3.5, 5.3.6, 5.3.8, kjer smo opazovali odvisnost med trajanjem in številom oz. razdaljo klicev. Na podoben način smo pripravili omrežja za izdelavo videa oz. posameznih sličic videa. Namesto celotnega obdobja opazovanja smo pri združevanju podatkov za izgradnjo omrežij trajanj in števila klicev upoštevali le enodnevna obdobja opazovanja. Posamezna sličica v videu tako predstavlja aktivnost med baznimi postajami za posamezen dan.

V razdelku 5.3.9 opazujemo dnevno in tedensko obnašanje baznih postaj. Prikaze smo ustvarili iz podatkov, navedenih v drugi alineji nabora podatkov [93]. V postopku priprav prikazov smo uporabili metodo razvrščanja po *Ward*-u [41]. Za izdelavo prika-

zov smo uporabili postopke, sorodne tistim v prejšnjih odstavkih, le da je tu poudarek na vizualizaciji vozlišč in prometa – povezav.

Analize podatkov iz tretje in četrte alineje nabora podatkov tu ne bomo izpostavljali, ker se je z njimi pretežno ukvarjala *M. Cerinšek* [94]. Med drugim je določila naravo baznih postaj po razvrščanju z metodo določanja voditeljev [95] na podlagi katerih smo z uporabo tu opisanih metod izdelali prikaze.

5.3 Analiza

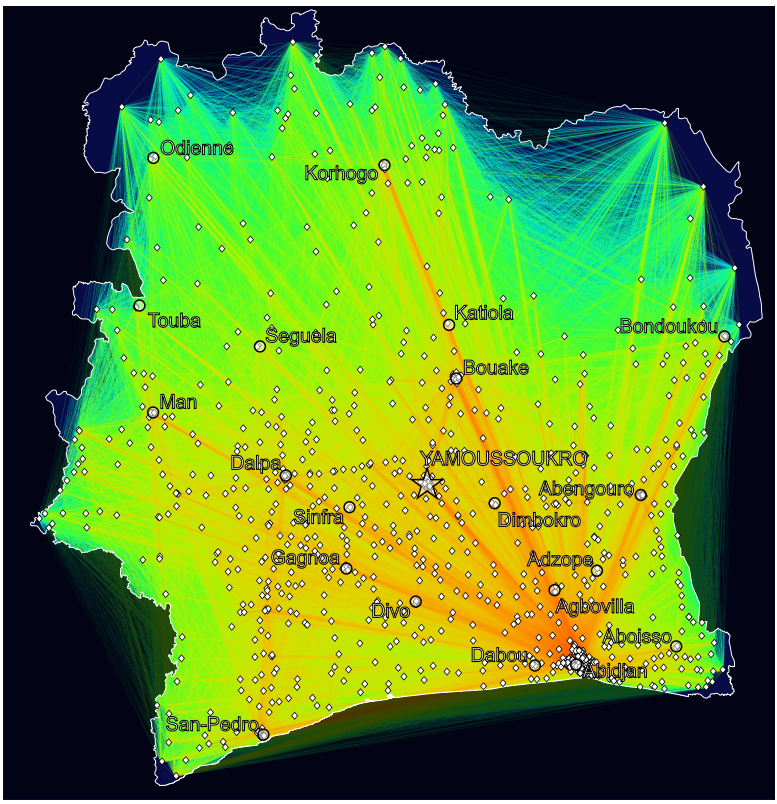
5.3.1 Pristop s klasičnimi prikazi vozlišč in povezav

Klasične prikaze omrežij v našem primeru razširimo tako, da povezave med vozlišči izrisujemo aditivno in ne kot le enobarvne daljice. S to tehniko narisan prikaz imenujemo porazdelitveni prikaz. Na presekih daljic je intenziteta v porazdelitvenem prikazu višja, enaka vsoti intenzitete posameznih daljic. Shema primera štirih sekajočih se daljic je prikazana na sliki 5.7. Na sliki je sicer prikazan postopek izdelave razpršenih vzorcev, o katerih bomo več povedali kasneje, vendar je ideja enaka. Intenziteta piksov na preseku daljic je enaka vsoti intenzitet daljic, ki potekajo skoznje.

Porazdelitvene prikaze števila in trajanja klicev med baznimi postajami smo osnovali geografsko, glede na združen promet med lokacijami posameznih parov baznih postaj za celotno pol-letno obdobje opazovanja. Vsak par baznih postaj povežemo z daljico, katere intenziteta je linearno sorazmerna količini prometa (številu klicev oz. trajanju klicev) za izbrani par baznih postaj. Slikovni element (piksel) na prikazu države tako predstavlja količino prometa, ki navidezno poteka skozi območje, ki ga v geografskem smislu pokriva. Ker je Slonokoščena obala približno kvadratne oblike, smo pripravili kvadratne prikaze (velikosti 5000×5000 pikslov). Slaba lastnost klasičnih porazdelitvenih prikazov je v tem, da imajo redkejša območja blizu skupaj (v skrajnem primeru sosednji piksli na prikazu) lahko zelo različne vrednosti. Slednje se na obeh prikazih izraža v obliki ostrih robov, kar je še posebej očitno na obrobju omrežja, kjer je prometa manj. Prikaza trajanja in števila klicev sta podana na slikah 5.1 oz. 5.2.

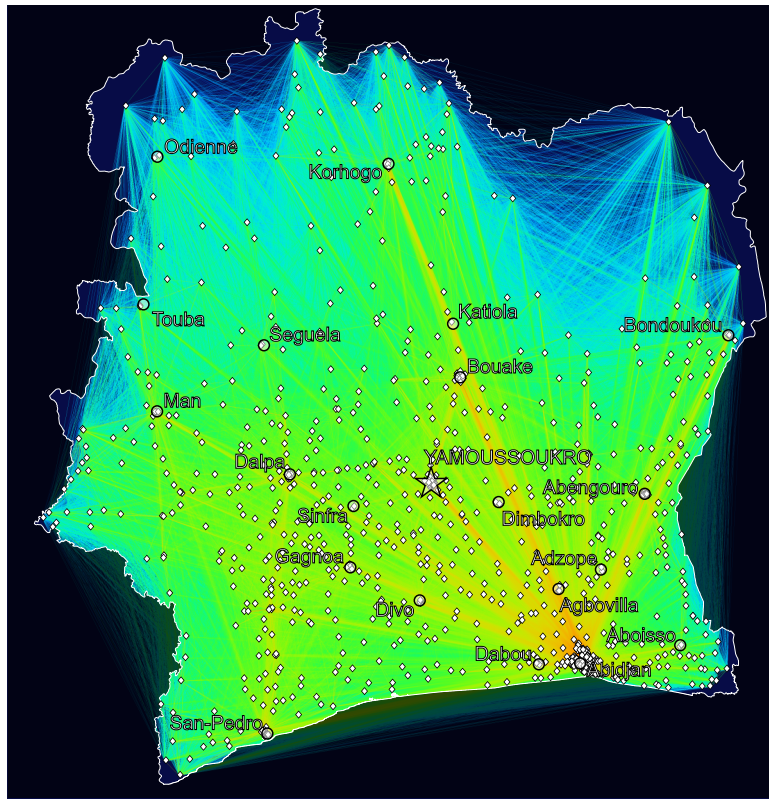
Animirani porazdelitveni prikazi prometa

Omrežja za posamezne sličice v animacijah smo pripravili iz podatkov, združenih za posamezen dan. Postopek izrisa sličic je bil enak kot pri statičnih prikazih, opisanih v razdelku 5.3.1, smo pa izbrali nižjo ločljivost. Omenimo še eno razliko. Prikazi za



Slika 5.1

Porazdelitev trajanja klicev med baznimi postajami, podana z osnovnim linijskim prikazom v logaritemski lestvici (slika 5.3).



Slika 5.2

Porazdelitev števila klicev med pari baznih postaj, podana z osnovnim linijskim prikazom v logaritemski lestvici (slika 5.3).

Slika 5.3

Barvna lestvica, ki jo uporabljamo v prikazih prometa. Temno modra barva predstavlja nizke vrednosti, svetlo rdeča pa visoke.

celotno obdobje opazovanja so normalizirani glede na maksimalno intenziteto piksla na sliki. Pri animacijah so vse sličice normalizirane glede na maksimalno vrednost piksla na katerikoli izmed sličic za posamezen dan.

Animaciji časovnega razvoja omrežij trajanja in števila klicev sta dostopni na spletu [1]. Vsaka sličica v animaciji predstavlja skupni promet v posameznem dnevu. V levem zgornjem kotu animacij sta na vsaki sličici prikazana datum in dan v tednu, ki mu sličica ustreza.

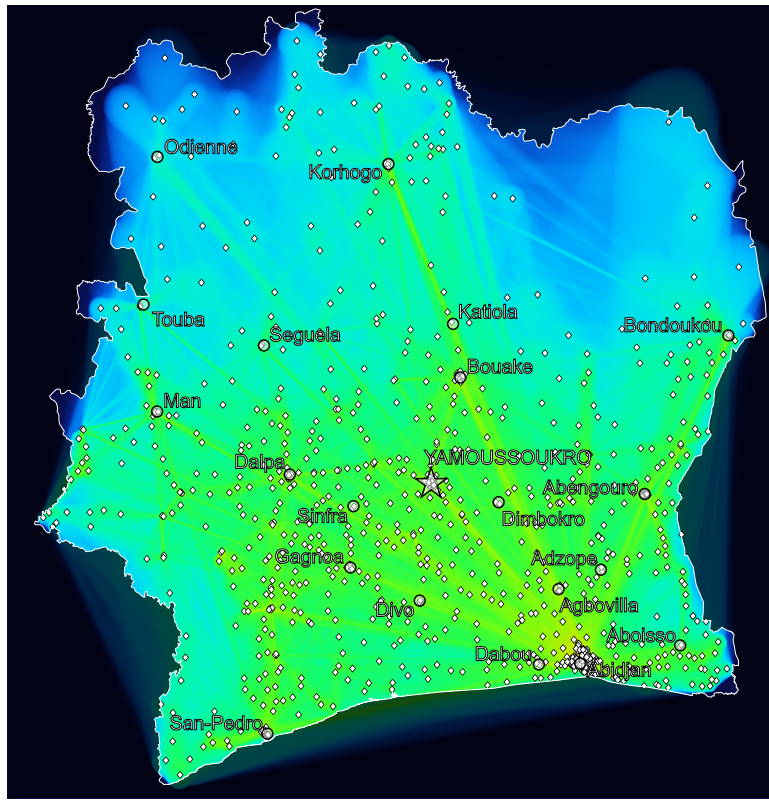
5.3.2 *Razpršeni porazdelitveni prikazi prometa*

V tem razdelku opišemo idejo razpršenih porazdelitvenih prikazov s katerimi rešujemo težavo ostrih robov, ki se pojavljajo na osnovnih linijskih prikazih. Razpršeni porazdelitveni prikazi se od osnovnih linijskih razlikujejo po tem, da med pari baznih postaj v njih namesto daljic aditivno izrisujemo razpršene verjetnostne ali katere druge (5.3.3) vzorce. Vzorci so pravzaprav posebne teksture, ki jih med pari vozlišč napenjamo namesto daljic.

Verjetnostni vzorci, ki jih opišemo v razdelku 5.3.2, promet modelirajo bolje kot daljice. Za razliko od daljice, ki vsako komunikacijo modelira kot promet med položaji para ustreznih baznih postaj tu predpostavimo, da se klicoči in klicani (dejanski izvor in ponor prometa) nahajata vsak v območju svoje bazne postaje in ne točno na lokaciji posameznih baznih postaj. V tem primeru promet opazujemo na nivoju uporabnikov mobilnega omrežja in ne na nivoju prometa med baznimi postajami. Ker podatkov o točnih lokacijah izvorov in ponorov klicev nimamo, predpostavimo, da se klicoči naključno nahaja nekje znotraj področja izvorne bazne postaje, klicani pa znotraj območja sprejemne bazne postaje. Hipotetični klic oz. njegovo trajanje tako ne bo več predstavljeno z daljico med parom baznih postaj, pač pa razpršeno med območjema para baznih postaj, znotraj katerih se klicoči in klicani naključno nahajata. Tako kot na osnovnih linijskih prikazih iz prejšnjega razdelka (5.3.1) tudi tu prikazujemo globalni promet med pari različnih baznih postaj. Razpršena porazdelitvena prikaza prometa, sestavljena iz verjetnostnih vzorcev, sta podana na slikah 5.4 in 5.5.

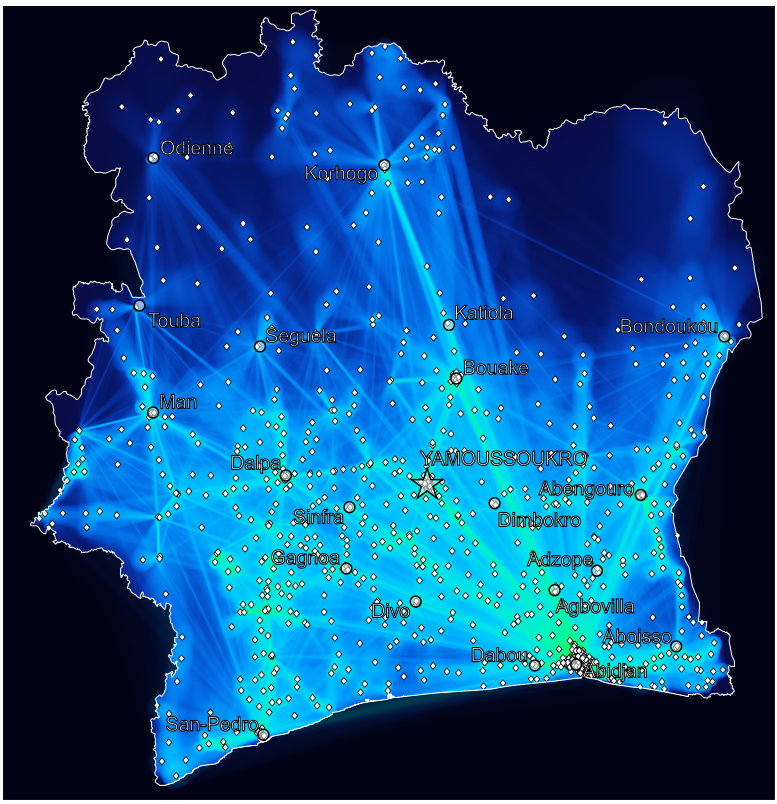
Na sliki 5.6 je prikazano glavno mesto Slonokoščene obale Abidjan in njegova bližnja okolica. Glede na to, da je gostota baznih postaj znotraj mest bistveno večja kot na obrobju in še posebej na podeželju, smo glavno mesto in bližnja okoliška mesta pri-

[1] http://zvonka.fmf.uni-lj.si/netbook/lib/exe/fetch.php?id=&media=project:d4d:pub:d4dvideos.rar



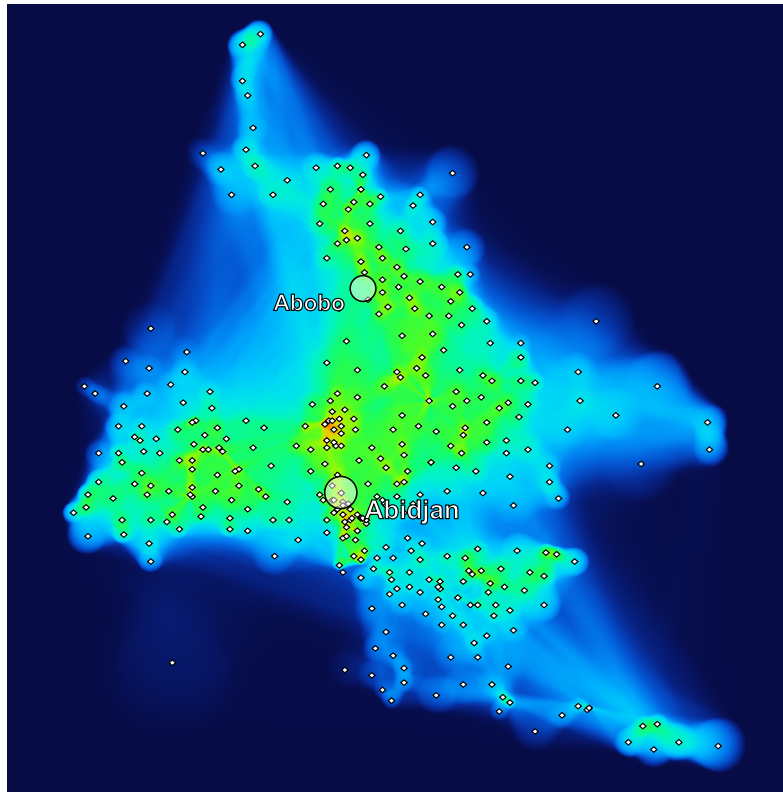
Slika 5.4

Prikaz trajanja klicev med baznimi postajami, sestavljen iz razpršenih vzorcev in prikazan v logaritemski lestvici (slika 5.3).



Slika 5.5

Prikaz števila klicev med baznimi postajami, sestavljen iz razpršenih vzorcev in prikazan v logaritemski lestvici (slika 5.3).



Slika 5.6

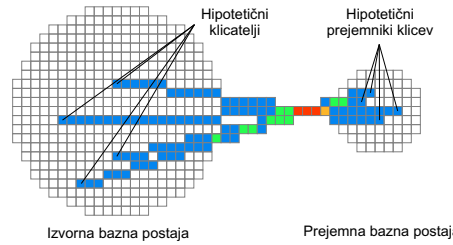
Prikaz števila klicev med baznimi postajami v okolici prestolnice Slonokoščene obale – Abidjan, sestavljen iz razpršenih vzorcev in prikazan v logaritemski lestvici (slika 5.3).

kazali posebej. Razpršeni prikaz mesta Abidjan predstavlja prikaz enega izmed nižjih nivojev v hierarhiji omrežja mobilnega prometa. Prikaz smo sestavili na enak način kot prikaza 5.4 in 5.5, le da je območje primerno povečano. Poleg tega so vključene samo bazne postaje, ki spadajo v izbrano območje. Na prikazu ni podan promet v in iz okolice.

Priprava verjetnostnih vzorcev prometa

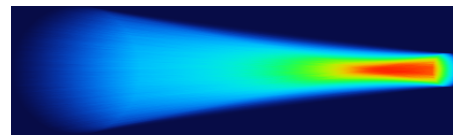
Če aditivno narišemo "dovolj" veliko število daljic med naključno postavljenima hipotetičnima klicateljem in prejemnikom, pri čemer je vsak znotraj svojega področja bazne postaje, za izbrani par baznih postaj dobimo nenormaliziran približek idealnega

vzorca. Vzorec oz. intenzitete njegovih pikslov nato normaliziramo na skupno vsoto 1. "Dovolj" pomeni toliko daljic, da pri dodajanju novih daljic v vzorcu ne vidimo več sprememb. Ker se bodo pri izrisu razpršenih prikazov uporabljali takšni vzorci, je kvečjemu manj verjetno, da bi pri samih prikazih opazili, da vzorci niso popolni. Prikaz para baznih postaj, skupaj z njunima območjema in velikim številom naključno izbranih položajev hipotetičnih klicateljev in prejemnikov znotraj območij svojih baznih postaj in komunikacijami med njimi, predstavljenim s piksli na daljicah, je podan na sliki 5.7. Ker daljice, t.j. promet rišemo aditivno, je intenziteta presečišča daljic po seštevanju (piksel oz. več pikslov) takšna kot vsota intenzitet pikslov na vsaki daljici posebej. Vzorce bi alternativno lahko ustvarili analitično, a zaradi preproste predpostavke ne potrebujemo velike natančnosti. Še več; predpostavka o krožnih območjih okrog baznih postaj je verjetno precej grob približek dejanskega območja delovanja bazne postaje, vendar je brez natančnejših informacij o mobilnem omrežju smiseln in najbolj verjeten. Polmere območij baznih postaj smo za vsako bazno postajo intuitivno določili, in sicer kot 70% razdalje do njene najbližje sosednje postaje. Ker so območja baznih postaj različno velika, hkrati pa tudi razdalje med baznimi postajami, bi morali v idealnem primeru za praktično vsak par baznih postaj pripraviti edinstven vzorec. Pripraviti bi morali več kot milijon različnih vzorcev, kar je precej zamudno. Namesto tega smo v okviru pričakovanih omejitev pripravili približno 5500 vzorcev in jih shranili v bazo. Omejitve so: polmer območja prve bazne postaje r_1, (5 – 140 pikslov), polmer območja druge bazne postaje r_2, ki naj bo vedno manjši od območja prve bazne postaje (2 – 107 pikslov) ter razdalja med baznima postajama d (5 – 1203 pikslov). Vzorec s parametri $r_1 = 60$, $r_2 = 16$, $d = 348$ je prikazan na sliki 5.8. Pri umeščanju vzorca na prikaz za vsak dejanski par baznih postaj iz baze izberemo vzorec, ki najbolje ustreza dejanskim parametrom, ga "raztegnemo" oz. "skrčimo" na pravo velikost in šele nato dodamo v prikaz. V primeru, da je območje druge bazne postaje večje od območja prve, vzorec obrnemo. Omenimo, da smo vzorce glede na intervale omejitev pripravili neenakomerno, in sicer po potenčnem zakonu (*power-law*) porazdelitvi; posledica je, da je takšnih z manjšimi razdaljami d, r_1 in r_2 več, manj pa takšnih z večjimi d, r_1 in r_2. Pri razširjanju daljših vzorcev za neko absolutno dolžino namreč storimo manjšo napako kot pri razširjanju krajših vzorcev za enako absolutno dolžino.



Slika 5.7

Grafični prikaz priprave vzorcev. Par baznih postaj je prikazan s svojima premeroma. Dve naključni daljici s krajišči znotraj posameznih območij baznih postaj predstavljata komunikaciji med naključnima klicateljema in klicanima. V presečišču daljic imajo piksli dvakrat večjo intenziteto kot piksli na daljicah, saj daljice dodajamo aditivno.

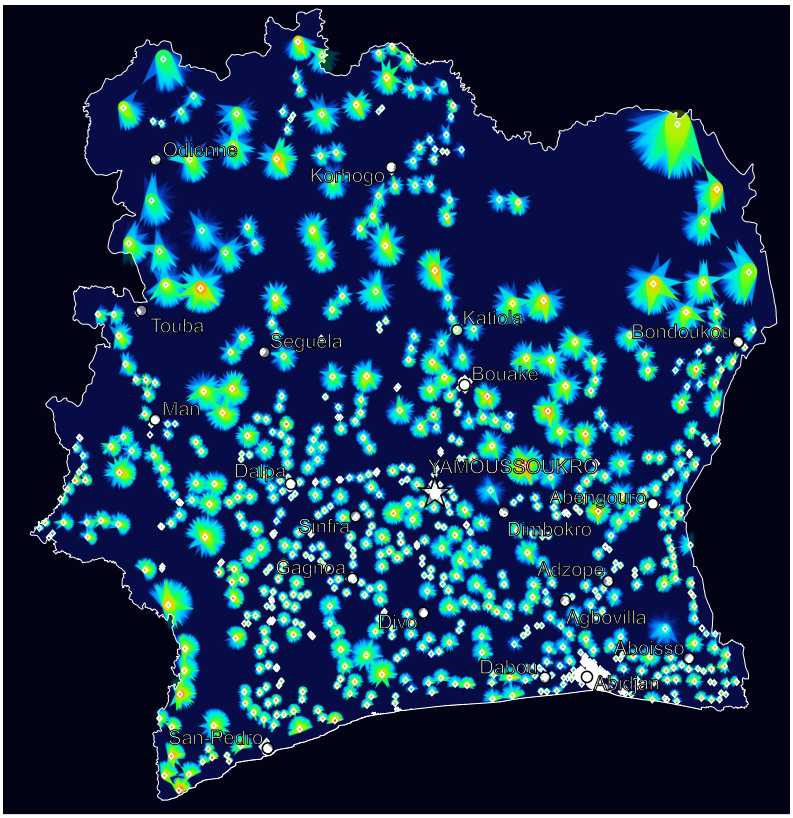


Slika 5.8

Primer verjetnostnega vzorca prometa z $r_1 = 60, r_2 = 16, d = 348$. Vsi parametri so izraženi v pikslih.

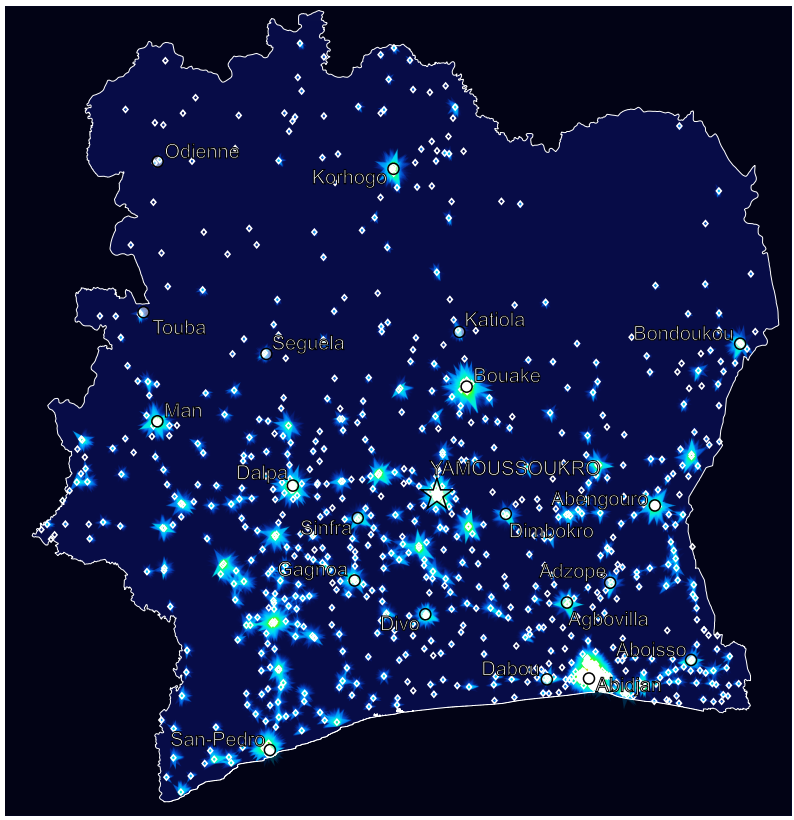
5.3.3 Prikaz prometa z zvezdnimi prikazi

Prikaze na slikah 5.9 in 5.10 smo ustvarili na enak način kot osnovne prikaze z daljicami, le da smo namesto vsake daljice aditivno izrisali usmerjeni žarek oz. teksturo v obliki žarka, podobno kot rišemo teksture verjetnostnih vzorcev pri razpršenih prikazih. V tem primeru za vse povezave zadošča enaka tekstura žarka – komunikacije med baznimi postajami, prilagoditi moramo le velikost in intenziteto. Faktor povečave žarka je v našem primeru sorazmeren celotnemu oddanemu prometu njegove bazne postaje, torej vsoti vseh klicev oz. trajanj klicev iz te bazne postaje, faktor intenziete pa je enak moči povezave – komunikacije, ki jo žarek predstavlja. Intenziteta izrisanega žarka bo torej takšna, kot bi bila pri ustrezni daljici na osnovnem prikazu. Primer para baznih postaj z vzajemnima žarkoma je prikazan na sliki 5.11. Ker želimo z velikostjo žarkov prikazati absolutno moč bazne postaje, z njihovo orientacijo pa smer v katero postaja oddaja promet, je oblika žarkov in njihova osnovna intenziteta na teksturi lahko poljubna. Za vse bazne postaje namreč uporabimo isto teksturo. Dobljeno sliko na koncu normaliziramo. Preden rišemo moramo izbrati še relativno velikost žarkov, da slika ne bo preveč na gosto posejana in hkrati ne bo delovala prazno. Pri tem mora biti izhodiščna tekstura žarka dovolj velika, da pri pripravi prikaza ne izgubimo na ločljivosti, t.j. teksturo le pomanjšujemo.



Slika 5.9

Zvezdni prikaz trajanja klicev med baznimi postajami, pripravljen v logaritemski lestvici, podani na sliki 5.3. Posamezen žarek kaže proti bazni postaji kamor komunikacija poteka. Velikost zvezd je sorazmerna vsoti vseh trajanj klicev njenih baznih postaj.

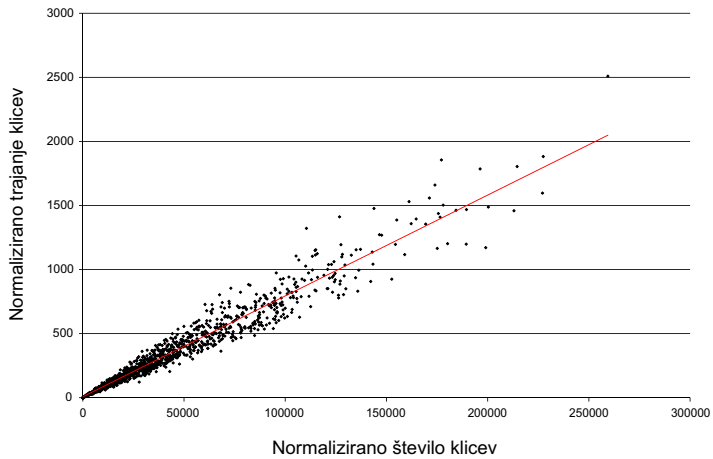


Slika 5.10

Zvezdni prikaz števila klicev med baznimi postajami, pripravljen v logaritemski lestvici, vidni na sliki 5.3. Žarki kažejo proti baznim postajam, kamor poteka komunikacija, ki jo predstavljajo. Velikost zvezde je sorazmerna vsoti vseh števil klicev bazne postaje v njenem središču.

Slika 5.11

Primer para žarkov, ki ponazarjata promet med dvema baznima postajama. Posamezen žarek se nahaja na položaju bazne postaje, s smerjo in intenziteto nakazuje smer in količino prometa bazne postaje, njegova velikost pa je sorazmerna celotnemu prometu bazne postaje. Žarke uporabljamo pri konstrukciji zvezdnih prikazov.



Slika 5.12

Razpršeni prikaz števila klicev v odvisnosti od trajanja klicev za vse bazne postaje.

5.3.4 Soodvisnost števila klicev in trajanja klicev

Na sliki 5.12 je podan razpršen prikaz, ki prikazuje odvisnost med normaliziranim številom klicev in normaliziranim trajanjem klicev. Za vsako bazno postajo smo določili celoten kumulativni promet (klici in trajanje) v opazovanem pol-letnem obdobju in ga normalizirali glede na število baznih postaj do katerih poteka. Prikaz kaže na linearno odvisnost količin.

5.3.5 Število in trajanje lokalnih klicev v odvisnosti od velikosti področja baznih postaj

Pri lokalnih klicih gre za klice znotraj ene same bazne postaje. Z drugimi besedami to pomeni, da sta klicoči in klicani znotraj iste bazne postaje, promet pa je v naših omrežjih izražen z zankami. Področje bazne postaje je tisto, ki ga bazna postaja pokriva. Določili smo jo kot kvadrat razdalje od opazovane bazne postaje pa do najbližje druge bazne postaje. Povezave med številom oz. trajanjem lokalnih klicev in površino, ki jo bazna postaja pokriva nismo zaznali.

5.3.6 Trajanje in število klicev v povezavi z razdaljo klicev

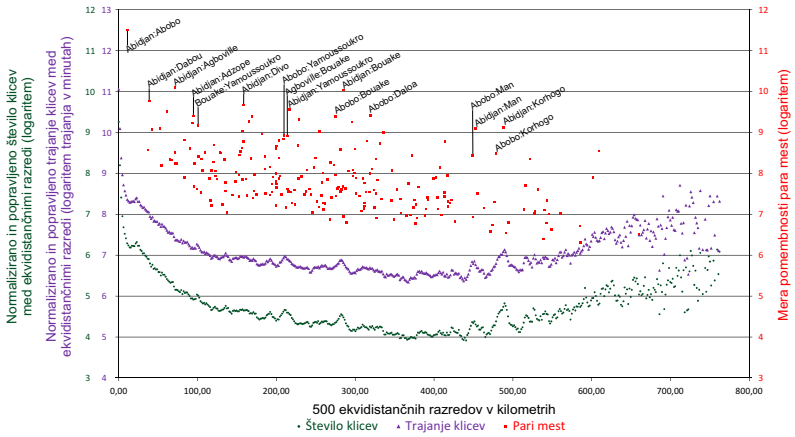
Tu navajamo rezultate analize odvisnosti prometa od razdalje klicev. Pare baznih postaj smo združili v 500 ekvidistančnih razredov, v katerih se nahajajo pari baznih postaj,

ki so približno enako oddaljene med seboj. Razrede smo predstavili z intervali razdalj med $0m$ in $d_{max} = 764km$, kar je enako razdalji med najbolj oddaljenima baznima postajama. Za vsak razred smo sešteli trajanja klicev in dobljene vrednosti normalizirali s številom parov v pripadajočem razredu in dodatno še z verjetnostjo pojavitve takšnega razreda. Verjetnosti pojavitev razredov smo aproksimirali z verjetnostmi izbiranj daljic z razredom ustreznimi dolžinami iz kvadrata v velikosti Slonokoščene obale. Verjetnost izbire daljice z naključno izbranima krajiščema, t.j. modeliranima lokacijama baznih postaj znotraj kvadratnega območja velikosti Slonokoščene obale, je razloženo v razdelku 5.3.7.

Na prikazu normaliziranih trajanj klicev pri določenih razredih zasledimo izrazite vrhove – vijolična krivulja na sliki 5.13. Predvidevali smo, da vrhovi ustrezajo parom večjih mest, v katerih se bazne postaje nahajajo blizu skupaj. Posledično tako dobimo več parov baznih postaj na podobnih razdaljah, ki pripadajo ločenim razredom. Pari večjih mest so na prikazu podani z rdečimi točkami. Pričakovali smo, da je skupno trajanje klicev med parom večjih mest sorazmerno populaciji obeh mest in morda obratno sorazmerno razdalji med mestoma. Predvideno smo preverili z vpeljavo in izračunom metrike za pomembnost para mest. Metrika pomembnosti para mest je na primer produkt populacij para mest in obratne vrednosti razdalje med njima. Na prikazu na sliki 5.13 so z višino rdečih točk podane vrednosti metrike. Po predpostavki bi morali najti primere parov mest, ki sovpadajo z vrhovi na prikazu. Nekaj parov je precej očitnih. Gre za pare velikih mest: Abidjan:Korhogo, Abidjan:Man, Abidjan:Bouake... Enako logiko kot pri trajanju klicev, smo uporabili še pri ugotavljanju odvisnosti števila klicev od razdalje klicev. Na sliki 5.13 vidimo, da ima krivulja števila klicev (zelena) nekoliko bolj izrazite vrhove.

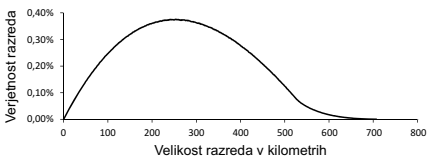
5.3.7 *Porazdelitev verjetnosti izbire daljice z znano dolžino v kvadratu*

Ker je Slonokoščena obala približno kvadratne oblike, smo njeno območje aproksimirali s kvadratom s stranico s. s je približek širine oz. višine države. Diagonalo kvadrata označimo z $d = \sqrt{2s^2}$. Ker so postaje pomaknjene nekoliko v notranjost države, smo aproksimirali še kvadrat, ki je vsem baznim postajam očrtan. Privzeli smo, da je njegova stranica za 5% manjša od s: $a = s \cdot (1 - 0,05)$, $d' = \sqrt{2a^2}$. d' tako dobro ustreza največji razdalji med dvema baznima postajama: $d' \approx d_{max}$ (glejte razdelek 5.3.6). V osnovni kvadrat s stranico s smo umestili veliko število (m) daljic z naključno izbranimi krajišči. Daljice, katerih dolžina je presegala d', smo izpustili (m_i daljic). Dolžine



Slika 5.13

Prikaz odvisnosti trajanja klicev in števila klicev od razdalje klicev. Črno – razred razdalje in razdalja med mesti v metrih, zeleno – normalizirano število klicev glede na razred razdalj, popravljen z verjetnostjo pojavitve takšnega razreda, vijolično – normalizirano trajanje klicev glede na razred razdalj, popravljen z verjetnostjo pojavitve takšnega razreda, rdeče – mera pomembnosti para mest.



Slika 5.14

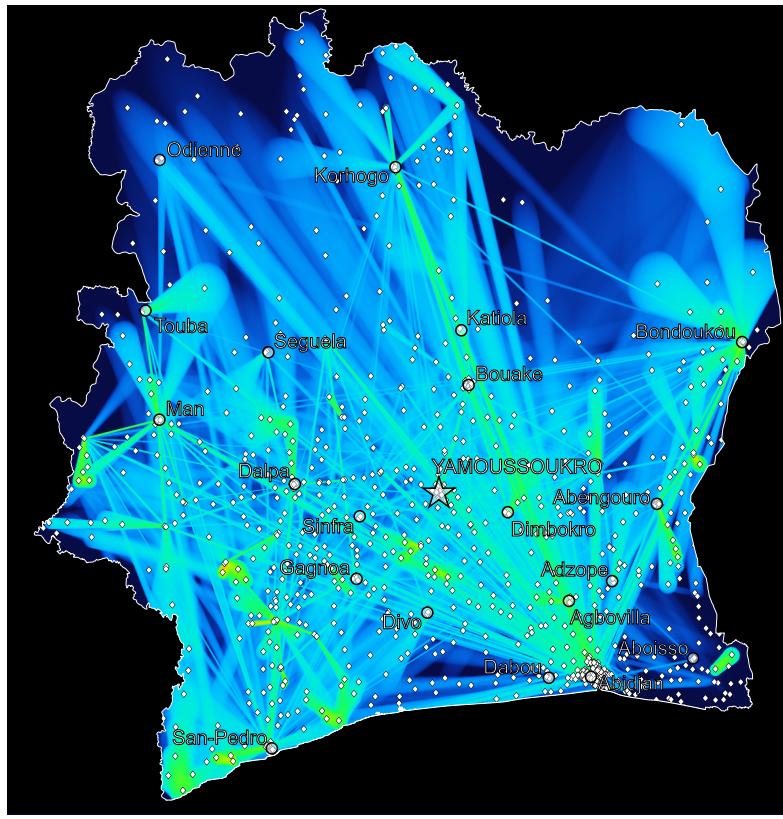
Približek verjetnosti naključnega izbora daljice z dolžino, ki pripada enemu izmed 500 ekvidistančnih razredov dolžin na intervalu $[0, d']$, iz kvadrata s stranico dolžine s. Če je izbrana daljica daljša od d', izbor razveljavimo in izbiranje ponovimo.

preostalih daljic smo razvrstili v 500 ekvidistančnih razredov, določenih na intervalu od 0 pa do dolžine diagonale pomanjšanega kvadrata: $[0, d']$. Števila daljic v posameznih razredih smo normalizirali s številom vseh daljic v pomanjšanem kvadratu $m - m_i$ in dobili približke verjetnosti izbir daljic za posamezen razred. Vrednosti smo uporabili kot normalizacijske oz. korekcijske faktorje v analizi iz poglavja 5.3.6. Na opisan način dobljena verjetnostna porazdelitev naključnega izbora daljice z znano dolžino iz osnovnega oz. pomanjšanega kvadrata je prikazana na sliki 5.14.

5.3.8 *Prikazi izbranega prometa*

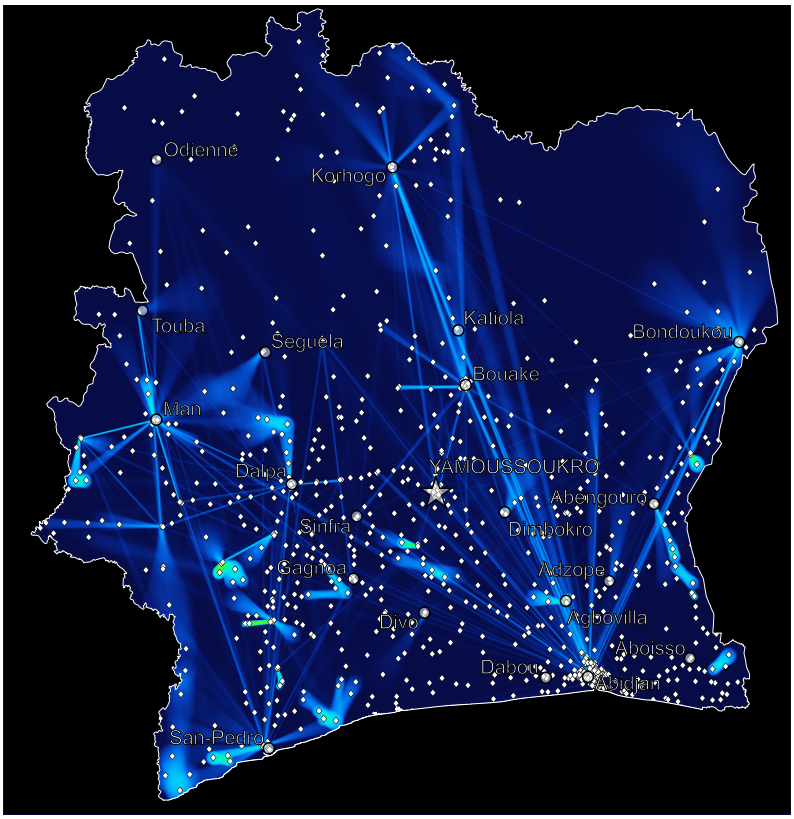
Za vsak ekvidistančni razred, definiran v razdelku 5.3.6, smo opazovali najbolj izrazita trajanja in števila klicev. Iz vsakega razreda smo izbrali po tri najbolj izrazite pare baznih postaj, torej tiste z največjim prometom in jih prikazali na slikah 5.15 in 5.16.

Zanimiv pogled na omrežje baznih postaj dobimo tudi, če obdržimo le del najbolj



Slika 5.15

Tri najizrazitejša trajanja klicev med pari baznih postaj za vsak ekvidistančni razred izmed 500, prikazana v logaritemski skali (slika 5.3).

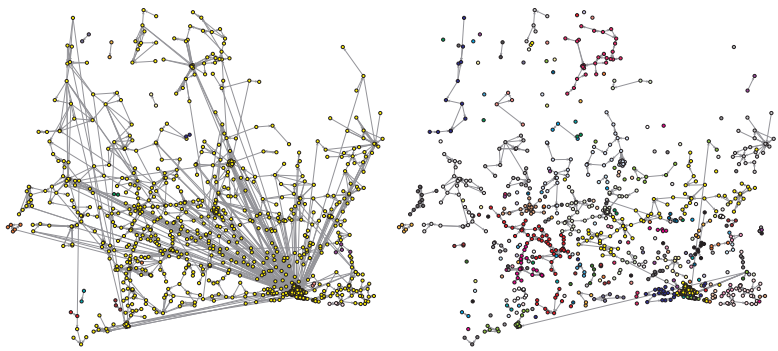


Slika 5.16

Tri najizrazitejša števila klicev med pari baznih postaj za vsak ekvidistančni razred izmed 500, prikazana v logaritemski skali (slika 5.3).

Slika 5.17

Na levi je prikazano omrežje trajanj klicev. V okviru posamezne bazne postaje so bile odstranjene vse povezave, katerih vrednost je manjša od 1/4 najmočnejše povezave v okviru iste bazne postaje. Na desni vidimo omrežje števil klicev. V okviru posamezne bazne postaje so tudi tu bile odstranjene vse povezave, katerih vrednost ni presegla 1/4 vrednosti najmočnejše povezave v okviru iste bazne postaje. Skupine preostalih povezanih baznih postaj so prikazane z drugo barvo.

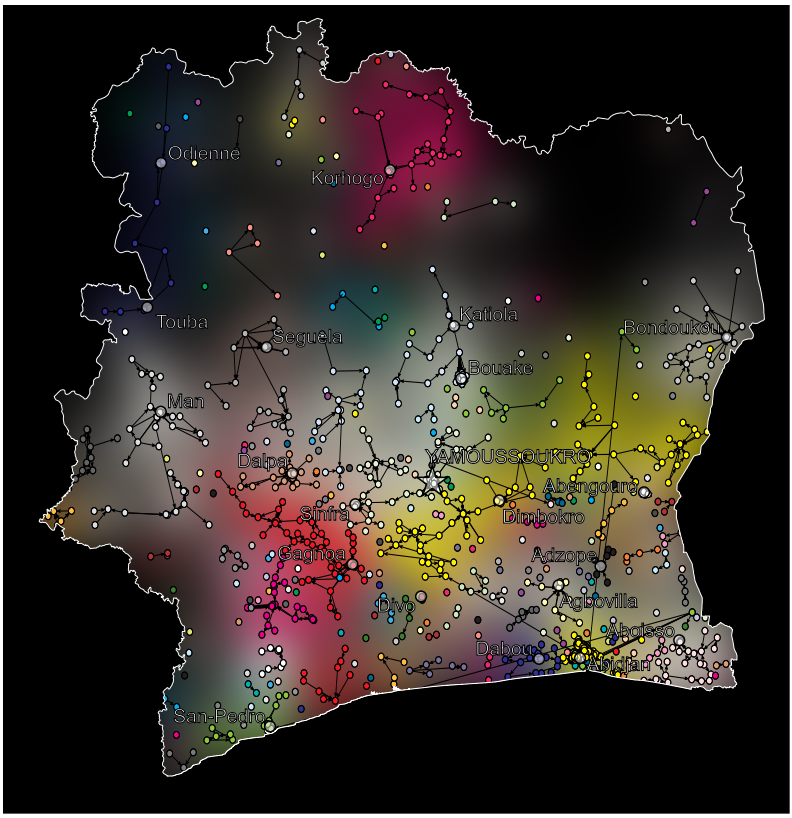


izrazitega prometa med posamezno bazno postajo in njenimi sosedami glede na najmočnejši promet opazovane bazne postaje. Na sliki 5.17 je prikazano omrežje baznih postaj in prometa, ki je večji od četrtine najmočnejšega prometa posamezne bazne postaje. Kot vidimo, je meja pri eni četrtini največjega prometa precej nizka. Skoraj vse povezave imajo vrednosti pod to mejo in so kot take odstranjene. Omrežje povezav med baznimi postajami razpade. Tvorijo se povezane komponente, ki smo jih na prikazu 5.18 predstavili z različnimi barvami.

5.3.9 *Dnevno in tedensko obnašanje baznih postaj – prikaz skupin baznih postaj*

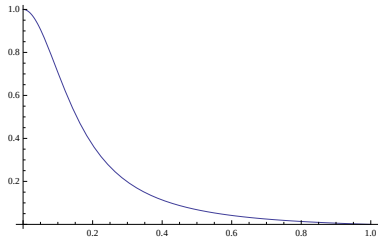
V tem razdelku obravnavamo še en način prikazov omrežij. Tokrat z razpršenimi vzorci prikazujemo vozlišča omrežja – bazne postaje in ne povezav – prometa. V tem smislu gre za bolj tradicionalni pristop prikazov gostote omrežja.

Posamezne bazne postaje smo glede na njihovo obnašanje razvrstili v skupine [95] in jih izrisali na slike velikosti 6000×6000 pikslov. Vsako bazno postajo smo predstavili z okroglim vzorcem premera 1000 pikslov, in sicer v barvi njej določene skupine. Pri dodajanju vzorcev na položaje baznih postaj, se le-ti lahko prekrivajo s tistimi iz iste skupine, lahko pa tudi s tistimi iz drugih skupin. Na takšen način lahko geografsko prikažemo, kje posamezna skupina prevladuje. Vzorce smo pripravili na podlagi funkcije f (5.1). Prikazana je na sliki (5.1). x predstavlja odmik od središča vzorca ($x = 0$) proti robu vzorca ($x = 1$). V želji, da bi končni prikazi izgledali čim bolje, smo pri izbiri funkcije f upoštevali dva kriterija: želeli smo, da ima zvonasto obliko in da se



Slika 5.18

Področja z močnim lokalnim prometom, določena na podlagi števila klicev v celotnem obdobju opazovanja.



Slika 5.19

Graf funkcije f (5.1) na intervalu $x \in [0, 1]$.

njena vrednost, ko gre x proti 1 počasi, a vztrajno približuje ničli.

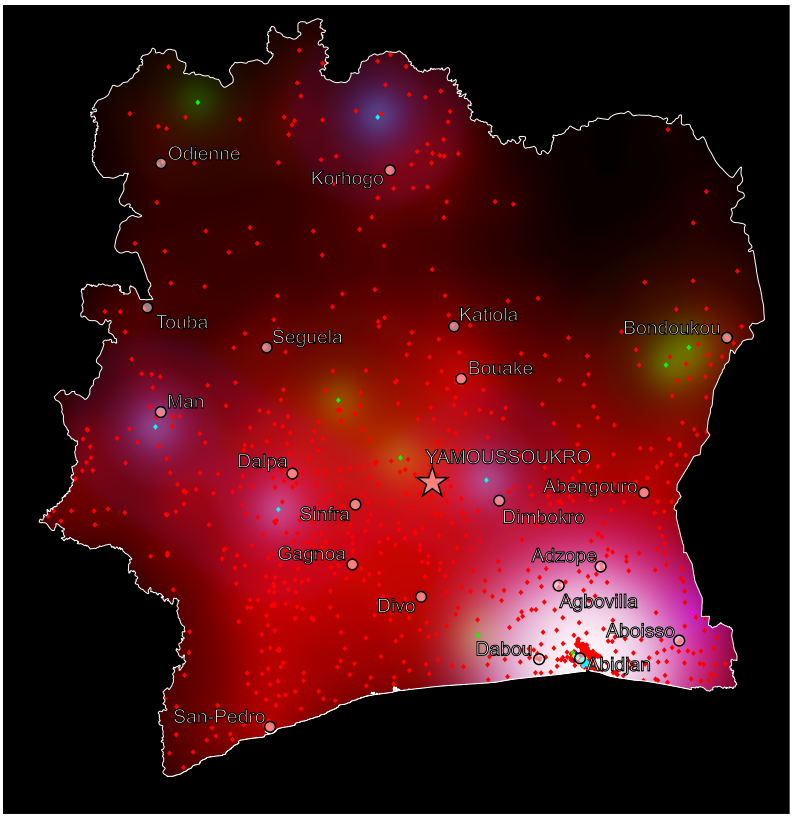
$$f(x) = \frac{1 - x^2}{40 \cdot x^2 + 1}, x \in [0, 1] \tag{5.1}$$

Po "seštevanju" vzorcev s središči na položajih baznih postaj na sliki, smo posamezne barvne komponente rezultata (R – rdeča (*red*), G – zelena (*green*), B – modra (*blue*)) normalizirali z maksimumi posameznih barvnih komponent in ga šele nato prikazali v logaritemski skali.

Če iz drugega nabora podatkov (druga alineja v razdelku 5.2) vzamemo podatke o klicih, lahko poleg trajektorij uporabnikov analiziramo tudi aktivnost baznih postaj. Ker imamo točne čase vsakega klica posebej, lahko aktivnost bazne postaje opazujemo po posameznih urah v dnevu ali po izbranem dnevu v tednu. Baznih postaj brez aktivnosti nismo vključili v analizo.

Podatke iz drugega nabora smo predelali v dvodelna omrežja (1.4.2, A.2.2). Bazne postaje nastopajo v enem delu dvodelnih omrežij, dnevi tedna pa v drugem. Vsaka bazna postaja je povezana z dnevom tedna natanko tedaj, ko je bil na izbrani dan v tednu opravljen vsaj en klic iz te bazne postaje. Utež na povezavi med bazno postajo in dnevom v tednu je enak številu klicev, ki so bili opravljeni na izbrani dan preko izbrane bazne postaje. Ker je v tednu sedem dni, je vsaka bazna postaja povezana s kvečjemu toliko vozlišči.

Stopnja vozlišč, ki predstavljajo bazne postaje, nam pove minimalno število dni v katerih je bila bazna postaja aktivna. Zemljevid skupin baznih postaj je prikazan na sliki 5.20. Bazne postaje so prikazane s pikami različnih barv, njihove okolice pa v skladu z opisanimi vzorci. Takšen način prikaza nam pomaga prepoznati katerim skupinam bazne postaje pripadajo. Področje okoli glavnega mesta Abidjan je skoraj belo, saj se tam nahaja veliko baznih postaj, ki pripadajo različnim skupinam. Barve skupin



Slika 5.20

Bazne postaje v barvah skupin, ki opredeljujejo njihovo delovanje po številu dni v tednu.

zasedajo vse tri barvne komponente in se tako seštejejo v belo. Rdeče pike prevladujejo, predstavljajo pa bazne postaje, ki delujejo preko celega tedna. 3,54% je baznih postaj, ki niso aktivne preko celega tedna. Zasledimo jih v jugovzhodnem delu države, t.j. v Abidjanu in njegovi okolici. Zelene pike predstavljajo bazne postaje, ki so aktivne do 6 dni v tednu, modre največ 5 dni, rumene 4 dni, škrlatne 3 dni, modrozelene 2 dni in vijolične največ en dan na teden.

Množica vozlišč drugega dvodelnega omrežja, ki smo ga pripravili, je sestavljena iz baznih postaj v enem delu in ur dneva v drugem. Vsaka bazna postaja ima največ 24 povezav. Želeli smo prikazati, kako so bazne postaje razporejene glede na število ur

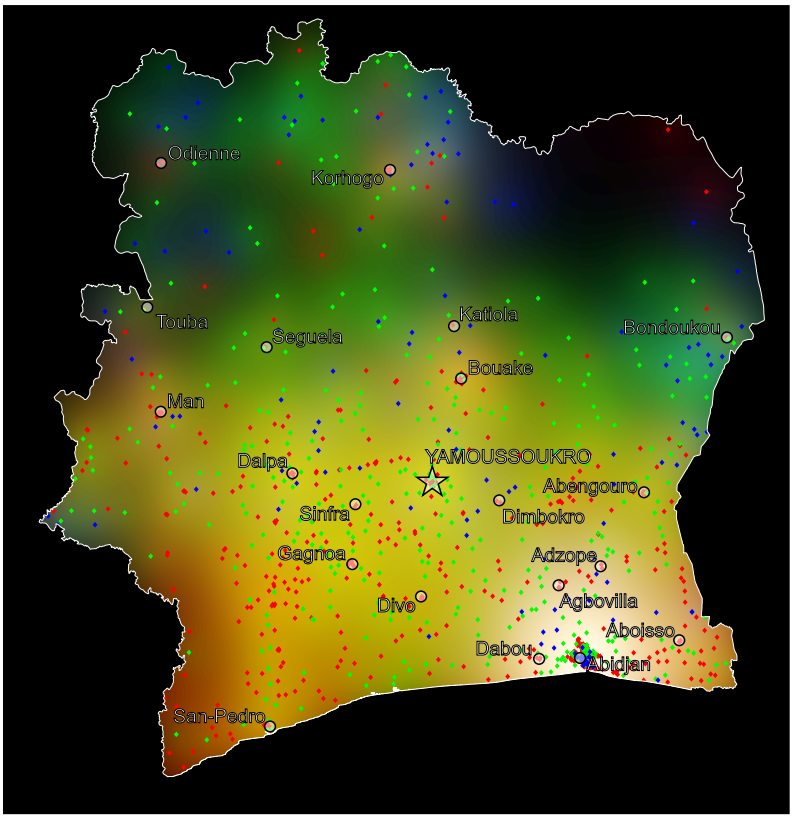
delovanja v dnevu.

Stopnja vozlišča, ki predstavlja bazno postajo, je enaka minimalnemu številu ur v katerih je bazna postaja aktivna. Vsaj en klic je bil torej iz te bazne postaje opravljen. Barve baznih postaj glede na njihovo dnevno aktivnost so prikazane na sliki 5.21. Bazne postaje so prikazane s pikami različnih barv in prav tako njihova okolica. Modra vozlišča predstavljajo bazne postaje, ki so bile aktivne do vključno 20 ur na dan. Z zeleno so predstavljene tiste, ki so bile aktivne največ 23 na dan, z rdečo pa tiste, ki so bile aktivne vse ure.

Področje v okolici Abidjana je tudi na tem prikazu praktično belo, ker je tam veliko baznih postaj iz različnih skupin. Na prvi pogled barve Slonokoščeno obalo razdeljujejo na tri dele. Severna polovica je zelena, jugovzhodni del rdečkast, ostala področja pa rumena. Rumena barva je pravzaprav mešanica rdeče in zelene. Ni presenetljivo, da veliko baznih postaj tekom dneva ni aktivnih ves čas. Ponoči je klicev bistveno manj: 46,5% baznih postaj je takšnih.

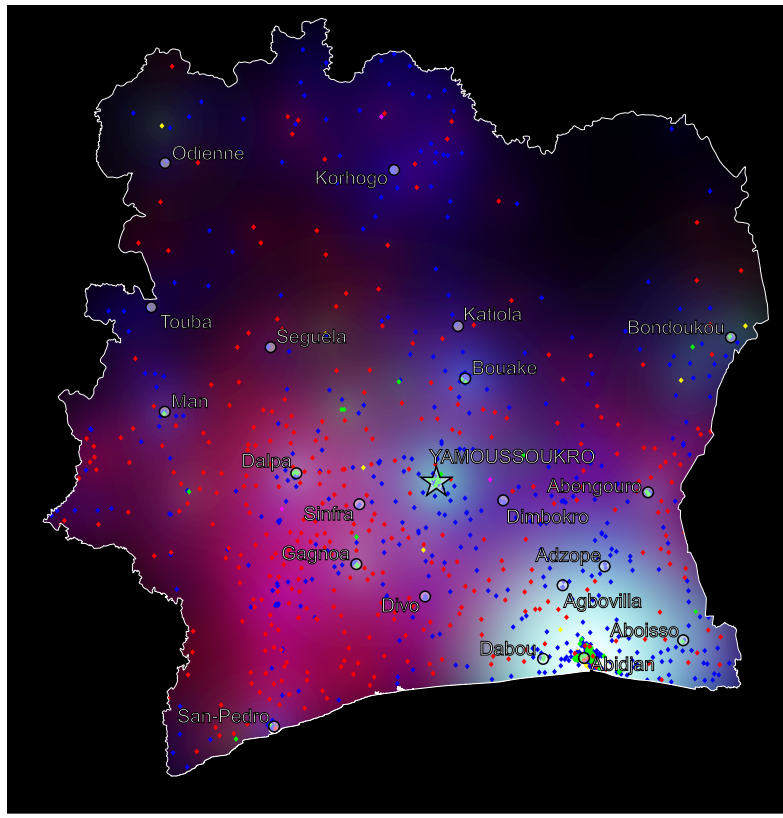
Zanimiv pogled na bazne postaje dobimo, če jih razvrstimo glede na podobnost po aktivnosti. Izračunali smo normalizirano omrežje baznih postaj in ur. Vrednosti povezav smo zamenjali na način, da vsota uteži izhodnih povezav vsake bazne postaje postane enaka 1, porazdelitev uteži pa ustreza porazdelitvi števila klicev iz posamezne bazne postaje. Z *Evklidsko* [42] medodo smo izračunali različnosti med baznimi postajami in uporabili *Ward*-ovo [41] razvrščanje in dobili hierarhijo baznih postaj. Iz hierarhije smo nato vzeli pet karakterističnih skupin baznih postaj in jih prikazali na sliki 5.22. V državi prevladujeta dve skupini – rdeča in modra. Jugovzhodni del je bolj moder, jugozahoden bolj rdeč. Ostale skupine so komaj opazne.

Identičen postopek smo uporabili še za določitev različnosti baznih postaj na podlagi njihove tedenske aktivnosti. Uporabili smo omrežje baznih postaj in dni, zamenjali vrednosti povezav, izračunali različnosti in analizirali hierarhijo. Izpostavili smo pet skupin in jih prikazali na sliki 5.23. Na sliki prevladujeta rdeča in zelena skupina. Rdeča prevladuje na vzhodnem delu države z mestom Abidjan. Zelena skupina prevladuje na zahodni strani in v jugovzhodnem delu mesa Aboisso. Za razliko od redkih rumenih in škrlatnih predstavnic skupin baznih postaj, najdemo nekaj več predstavnic modre skupine.



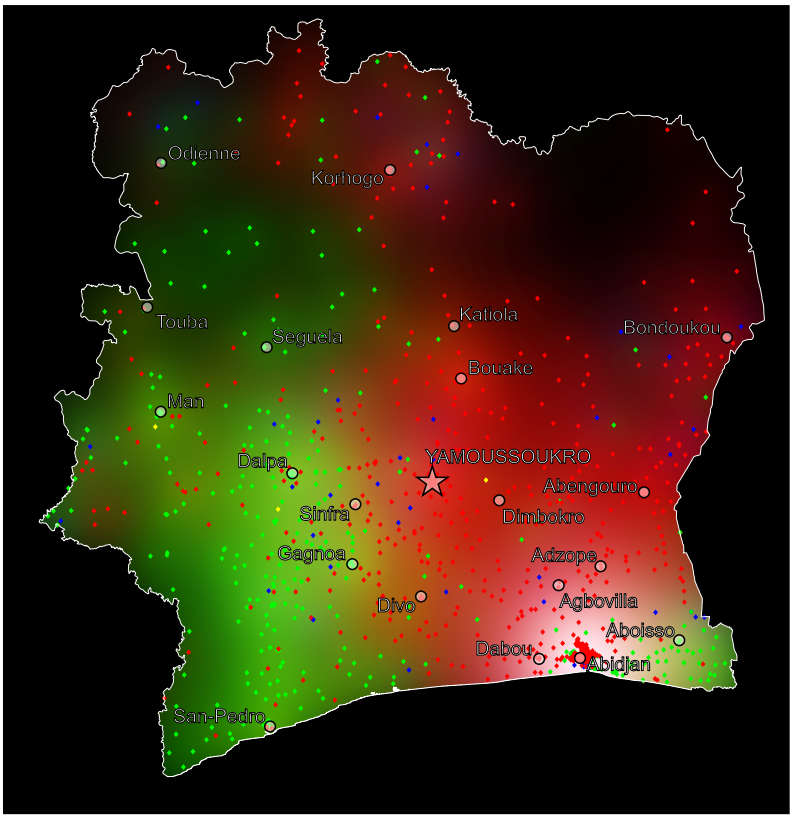
Slika 5.21

Bazne postaje v različnih barvah skupin, ki določajo njihovo aktivnost po urah v dnevu.



Slika 5.22

Podobnost baznih postaje glede na dnevno aktivnost. Podobne postaje so enake barve.



Slika 5.23

V barvne skupine razvrščene bazne postaje. Skupine smo določili glede na podobnost tedenske aktivnosti baznih postaj.

5.4 Zaključki

V poglavju smo opisali postopke, ki smo jih razvili za vizualizacijo mobilnega prometa med baznimi postajami na Slonokoščeni obali. Pripravili smo različne slike in prikaze vključno s specializiranimi razpršenimi in zvezdnimi prikazi. Nekateri prikazi upoštevajo celotni nabor podatkov, drugi le del podatkov in v določenem smislu bolj zanimive podatke. Pripravili smo dva videa prometa, da bi poudarili dinamiko sprememb prometa. Raziskali smo nekatere odvisnosti podatkov in presenetljivo ugotovili, da trajanje klicev dobro sovpada s številom klicev. Nasprotno pričakovanjem smo ugotovili, da število in trajanje lokalnih klicev nista povezana z velikostjo področja, ki ga opazovana bazna postaja pokriva. Identificirali smo skupine obnašanja baznih postaj glede na dni v tednu in ure v dnevu. Skupine smo prikazali na razpršenih prikazih.

Naše delo se je pretežno nanašalo na podatke, ki izvirajo iz organizatorjevega nabora. Skušali smo uporabiti druge vire podatkov, vendar jih, razen splošnih o Slonokoščeni obali in njenih mestih, na koncu nismo. Skozi organizatorjeve podatke bi bilo zanimivo raziskovati npr. geografske, zgodovinske, verske, politične in druge vidike. Namen organizatorjev tekmovanja je navsezadnje bil, da se skozi njihove podatke pogleda še na kakšno drugo dogajanje v državi.

Prikazane metode prikazovanja prometa v mobilnih omrežjih lahko posplošimo na prikaze poljubnih velikih omrežij. Pogoj za uporabo je poznavanje koordinat vozlišč omrežja, ki ga želimo prikazati. Koordinate vozlišč lahko dobimo vnaprej ali pa jih določimo s primerno metodo umeščanja na primer s hitro multipolno metodo, ki smo jo kot primer učinkovitosti procesiranja na GPE prikazali v dodatku v poglavju A.4. Opisane metode iz tega poglavja so časovno potratne in kot take niso najbolj primerne za interaktivno prikazovanje omrežij, so pa ena izmed najboljših izbir za pripravo zemljevidov, ki jih lahko uporabimo za globalno navigacijo v omrežjih pri pregledovanju. Primerne so tudi za vizualizacijo gostih delov omrežij, kjer nas bolj kot posamezne povezave zanima njihova gostota. V hierarhičnih prikazih so najbolj uporabne na višjih nivojih, saj prikaz odraža celostno strukturo opazovanega omrežja, hkrati pa zabriše podrobnosti omrežja, ki na višjih nivojih niso potrebne.

6

Interaktivni grafični uporabniški vmesnik in knjižnica za analizo velikih omrežij

6.1 Orodja v analizi omrežij

Z razvojem modernih tehnologij analiza omrežij v znanosti vztrajno pridobiva na pomenu. Že dolgo jo srečujemo na primer v sociologiji, zadnje čase pa se pojavlja na vse več področjih, predvsem v družbenih vedah, v ekonomiji, v biologiji, lingvistiki, na raznih tehnoloških področjih in na mnogih drugih. Potrebo po analizi omrežij v praksi pogosto zasledimo. Analiziramo lahko realna omrežja že v primarni obliki, kot so na primer omrežja na spletu, omrežja infrastrukture cest, električnih vodov, omrežja v računalništvu, bibliografska omrežja idr. Poleg tega z omrežji lahko modeliramo marsikatero strukturo, ki v svoji naravi ni omrežje, na primer procesi, načrti, interakcije ljudi, sistemov, lingvistična omrežja itd. Z omrežji lahko tudi računamo in izpeljemo umetna in abstraktna omrežja. Z razvojem, predvsem spletne tehnologije omenimo še, da se v praksi pojavlja veliko *velikih* omrežij. Gre za omrežja velikosti od nekaj tisoč pa do več milijonov in več vozlišč oz. takšnih, ki jih še lahko v celoti shranimo v delovni pomnilnik računalnika (običajno osebnega). V tem smislu je pojem velikih omrežij šibko definiran, vendar zadošča. Da bi lahko s takšnimi in drugačnimi omrežji delali, so se pojavila mnoga orodja s katerimi lahko izvajamo analizo omrežij, jih prikazujemo, urejamo, shranjujemo..., a se v praksi pokaže, da redka omogočajo analizo *pravih* velikih omrežij.

Poglejmo v grobem katere vrste orodij pri analizi omrežij srečujemo. Najbolj pogosto orodje je verjetno kar tekstovni urejevalnik ali urejevalnik podatkovnih tabel s katerim si pomagamo pri zbiranju podatkov omrežij, konstrukciji omrežij in urejanju datotek z omrežji. Eno od orodij so tudi pretvorniki različnih oblik datotek omrežij v druga in namenska orodja za zajem podatkov omrežij. O teh orodjih več pišemo v dodatku A.2. Naslednja veja orodij je namenjena procesiranju omrežij in "računanju" z njimi. Zajema različne knjižnice algoritmov s katerimi je na različne načine mogoče priti do lastnosti omrežij, omrežja predelovati, jih razumeti, obvladovati, skratka priti do uporabnih spoznanj o omrežjih. Pomembna veja orodij so pregledovalniki omrežij in sorodnih podatkov s katerimi podatke lahko bolj ali manj interaktivno opazujemo, slogovno oblikujemo, vizualno analiziramo in pripravimo za izvoz slik, animacij, ki prikazujejo rezultate analiz. Nenazadnje v analizi omrežij pogosto uporabljamo tudi splošno namenska programska okolja, v katerih lahko razvijamo lastne specifične postopke in aplikacije, ki nam olajšajo delo z omrežnimi podatki. V zadnjem času srečujemo tudi sestavljene programske pakete, ki zgornje lastnosti združujejo v zaključeno

celoto. Uporabniku naj bi omogočala izvedbo celostnega postopka analize od začetka do konca v obliki primerno predstavljenih rezultatov. Poleg tega, da so bolj ali manj samozadostna, je prednost nekaterih integriranih orodij v organizaciji vmesnih rezultatov in v organizaciji delovnega procesa analize. Mnoga so zasnovana modularno in omogočajo dodajanje t.i. vtičnikov, ki jih uporabniki lahko samostojno razvijajo.

Naš cilj je bil razviti novo orodje, t.j. celostno interaktivno grafično okolje *net.Plexor*, ki bi temeljilo na knjižnici – ogrodju za delo z velikimi omrežji. Hkrati bi vgradili tudi nove metode, ki smo jih predstavili v prejšnjih poglavjih. Ker smo orodje razvijali skupaj z drugimi razvijalci, bomo v tem poglavju opisali le dele za katerimi stoji avtor pričujočega dela sam. To je predvsem logika, ki skrbi za prikazovanje omrežnih podatkov, logika za organizacijo izvajanja delovnega procesa – sheme analize in logika uporabniškega vmesnika v splošnem, ki predstavlja osnovo za interaktivnost. Tu naj omenimo, da pristopa za organizacijo delovnega procesa s shemami analize ali s t.i. vizualnim programiranjem, kot ga bomo opisali v nadaljevanju, nismo zasledili pri nobenem drugem orodju za analizo omrežij. Podoben pristop uporabljajo na primer pri orodju za podatkovno rudarjenje *Orange* [96]. Uvajamo pa tudi koncept spreminjanja parametrov algoritmov v realnem času, ki ga prav tako nismo zasledili pri nobenem drugem orodju za analizo omrežij. Ostalih funkcionalnosti se v besedilu dotaknemo le toliko, kolikor je potrebno.

V poglavju opišemo nekaj popularnih orodij za analizo omrežij (6.1.1) in izpostavimo *Pajka* (6.1.2) kot inspiracijo za razvoj naše knjižnice, ki jo skupaj s privzetim uporabniškim vmesnikom opišemo v razdelku 6.2. Nadaljujemo z opisom delov orodja, ki jih je prispeval avtor disertacije in se navezujejo na njeno temo: podlaga za interaktivnost (6.3), prikazovalnik in urejevalnik omrežij (6.4.1), algoritem za razvrščanje omrežij (6.4.2), prikazovalnik hierarhij (6.4.3) in na koncu še primer algoritma, ki učinkovito izrablja prednosti procesiranja na grafičnih procesorjih v tehnologiji *OpenCL*® (6.4.4).

6.1.1 Primeri programskih rešitev za analizo omrežij

Z razvojem programskih rešitev za analizo velikih omrežij se ukvarja precej raziskovalnih skupin. Nekatere rešitve so na voljo v odprti kodi, druge komercialno. Izmed vodilnih in celostnih okolij izpostavimo paket *NetMiner* [97], saj je eden najbolj popolnih komercialnih produktov za analizo omrežij. Na področju odprtokodnih rešitev sta popularna na primer *Tulip* [58] in *Cytoscape* [98]. Prvi je splošno namenski, odlikuje pa

ga odličen modul za prikazovanje. Drugi ima korenine v biologiji, a je kasneje prerasel v splošno namensko orodje za analizo omrežij. Temelji pretežno na vtičnikih. Obe orodji sta odprtokodni. Omenimo še en popularen odprtokoden program *Gephi* [99]. Namenjen je predvsem vizualizaciji končnih rezultatov analiz, bolj šibak pa je na področju same analize. V nasprotju z naštetimi orodji podajmo še primer bolj namenske in eksotične programske rešitve *NodeXL* [100], ki se uporablja kot vtičnik za *Microsoft Excel*. Nudi le osnovne funkcionalnosti analize omrežij, vendar zaradi gostovanja v *MS Excel*-u predstavlja uporabno in praktično rešitev v poslovnem svetu.

Izmed približno sto obstoječih, smo našteli le nekaj primerov programskih rešitev. Več informacij, predvsem glede prikazovanja omrežij lahko bralec najde v delu [2]. Obširen seznam orodij za analizo omrežij navajamo tudi v poglavju A, v razdelku A.2. Poudarek je na omrežnih datotečnih formatih, bo pa bralec našel tudi druge uporabne informacije.

6.1.2 Pajek – *orodje za analizo velikih omrežij*

Pajek [34] je uveljavljeno orodje, ki dobro pokriva analizo velikih omrežij. V primerjavi s podobnimi programi lahko rečemo, da je za delo z velikimi omrežji med najboljšimi. Vključuje ogromno optimiziranih algoritmov in avtorji jih sproti dodajajo in dopolnjujejo. Ima tudi nekaj slabosti. Od uporabnika zahteva dobro poznavanje analize omrežij in terminologijo, saj pri delu ni najbolj intuitiven in ne nudi interaktivne pomoči, kar je pogosto zaslediti v bolj naprednih programih, izpostavljenih v zgornjem razdelku 6.1.1. Pred uporabo *Pajka* se je priporočljivo seznaniti s splošnimi informacijami [34] in s knjigo [35], ki dobro povzema pristope v analizi omrežij s *Pajkom*, ki jih prikaže tudi na konkretnih primerih. Pri uporabi nam večkrat prav pridejo navodila za uporabo [101]. Program po izgledu spominja na kalkulator z velikim naborom operacij, ki jih lahko izvajamo na omrežjih in drugih sorodnih strukturah. Pri tem moramo sami skrbeti za organizacijo posameznih struktur, saj se v nasprotnem primeru hitro izgubimo v množici podatkovnih datotek. *Pajek* omogoča shranjevanje večih struktur v eno samo projektno datoteko (o shranjevanju in organizaciji omrežnih struktur pišemo v dodatku A.2), a v praksi rešitev brez dodatnih zapiskov ni zadovoljiva. Delno si lahko pomagamo z "makroji", s katerimi je mogoče "posneti" oz. "predvajati" sestavljena zaporedja operacij na omrežjih in tako vsaj do neke mere avtomatizirati postopek analize. *Pajek* vključuje preprost prikazovalnik omrežij. Pogosto je potrebno vložiti precej truda, da razmeroma preprosto sliko omrežja oblikujemo in pripravimo za izvoz

v vektorski obliki. Za oblikovanje končne slike pa nam (tudi pri drugih programih) prav pride kakšno bolj zmogljivo grafično orodje. Zaradi opisanih pomanjkljivosti je delo v *Pajku* zahtevno, a hkrati preprosto. Zahteva visoko stopnjo discipline in natančnosti. *Pajka* zasledimo predvsem v akademskih okoljih, manj pa v komercialnih oz. poslovnih. Glavna prednost *Pajka* je v podpori za velika omrežja. Kljub temu, da so v smeri prikazov in interaktivnosti mnoga orodja boljša, so v smislu obvladljive količine omrežnih podatkov vsa izpostavljena orodja inferiorna.

Podrobneje o *Pajku* pišemo, ker je dobra osnova za nadgradnjo v smislu interaktivnosti in prikazovanja omrežij. Uporabili smo ga kot izhodišče pri razvoju naše knjižnice z interaktivnim uporabniškim vmesnikom – *net.Plexor*.

6.2 *Interaktivni grafični uporabniški vmesnik in knjižnica za analizo velikih omrežji* – net.Plexor

Po vzoru *Pajka* smo zasnovali okolje *net.Plexor* – novo knjižnico za delo z velikimi omrežji in na njej zgradili interaktivni grafični uporabniški vmesnik, ki uporabnikom omogoča interaktivno in dinamično uporabo njenih funkcionalnosti. Glavni cilj je bil ohraniti prednosti *Pajka*, hkrati pa dodati nove funkcionalnosti, ki bi dopolnile *Pajkove* pomanjkljivosti. *net.Plexor* smo zasnovali za uporabo na vseh pogostejših operacijskih sistemih. Namenjen je skupini uporabnikov z zmogljivejšo strojno opremo, predvsem z zmogljivo grafično kartico, procesorjem 64-bitne arhitekture in veliko delovnega pomnilnika. Priporočljiv je tudi hiter disk.

6.2.1 *Knjižnica, osnovno ogrodje*

Knjižnico smo zasnovali v obliki skalabilnega ogrodja, kamor je po potrebi možno dodajati funkcionalnosti, kot so algoritmi, vhodno/izhodne operacije, prikazovalniki in druge operacije nad omrežnimi podatki. Na nek način gre za sistem, ki omogoča dodajanje vtičnikov, brez da bi morali bistveno posegati v obstoječo kodo. Vsak vtičnik predstavlja operacijski gradnik – osnovni blok oz. element, ki ga je možno uporabiti v načrtu – shemi analize omrežja.

Poseben poudarek smo namenili kontroli delovanja operacijskih gradnikov. Razvijalec ima možnost, da implementira predčasno ustavljanje algoritma, sprotno spreminjanje parametrov algoritma in vpogleda v delovanje algoritma v realnem času. Predvsem slednja možnost lahko uporabniku znatno skrajša pot do želenih rezultatov. Da upravičimo zgornjo trditev, smo v nadaljevanju opisali kompleksen primer operacijskega

gradnika, ki mu v realnem času lahko spreminjamo vhodne parametre in hkrati spremljamo njegov odziv. Gradnik (6.4.4) realizira hitro multipolno metodo umeščanja omrežij v ravnino ali prostor. Pri operacijskih gradnikih ne smemo pozabiti na definicijo vhodnih in izhodnih struktur in parametrov. Primer operacijskega gradnika je modul za nalaganje struktur iz datoteke, ki kot parameter prejme pot do datoteke, vrne pa naslov na interno predstavitev omrežja – objekt omrežje. Algoritem, ki prejme objekt omrežje in obrne smer povezav ter vrne objekt predelanega omrežja, je primer operacijskega gradnika, ki implementira algoritem. V osnovi se opisana operacijska gradnika ne razlikujeta, se pa razlikuje njuna programska izvedba. Pomembna lastnost ogrodja je shranjevanje vmesnih rezultatov, ki jih ustvarijo operacijski gradniki. Vmesni rezultati se ne računajo vsakič znova, pač pa po potrebi ob smiselnih spremembah vhodnih podatkov oz. parametrov. Ogrodje je zasnovano tako, da se posamezni operacijski gradniki, v kolikor podana shema analize to dopušča, lahko izvajajo vzporedno. Svojevrstna lastnost ogrodja je v podpori za avtomatizacijo operacijskih gradnikov. V načinu samodejnega zaganjanja analize se operacijski gradniki lahko samodejno odzivajo na zunanje spremembe podatkov, upoštevajo razne dogodke, ki se zgodijo v gostujočem operacijskem sistemu ali vhodno/izhodni napravi ipd. Skratka z načinom samodejnega zaganjanja je ogrodje možno uporabiti za monitoring sistemov v realnem času.

Nekaj algoritmov smo v knjižnico že vgradili; predvsem tiste, ki jih vsebuje *Pajek* in se bolj pogosto uporabljajo. Glede na dodatne možnosti kontrole, ki jih ogrodje omogoča, smo jih primerno prilagodili in dopolnili. Vključili smo tudi algoritem, ki smo ga opisali v poglavju 4 in funkcionalnosti, s katerimi ga je možno smiselno uporabiti in spremljati njegovo delovanje (razdelek 6.4.2). Vgradili smo tudi nekatere metode prikazovanja omrežij in hierarhij, ki jih opisujemo v poglavju 3.

6.2.2 *Interaktivni grafični uporabniški vmesnik*

Glavni uporabniški vmesik za delo s knjižnico je preprost. Sestavljen je iz komponent knjižnice za izdelavo uporabniških vmesnikov oz. okenskih aplikacij v jeziku *c++*, ogrodje *wxWidgets* [102]. Jezik *c++* smo izbrali zaradi hitrosti in prenosljivosti, ogrodje *wxWidgets* pa ker je dobro podprto in odprtokodno. Pri konstrukciji uporabniškega vmesnika smo si pomagali z orodjem *wxFormBuilder* [1]. Vmesnik smo zasnovali tako, da bi bil čim bolj uporabniku prijazen in pri uporabi intuitiven. Sestavlja ga več enot,

[1] http://sourceforge.net/projects/wxformbuilder

glavne pa so: delovno okolje – okno za konstrukcijo sheme (načrta) analize, kamor interaktivno postavljamo operacijske gradnike, razna komunikacijska okna na katera lahko gledamo kot na neke vrste samostojne uporabniške vmesnike posameznih operacijskih gradnikov, glavni in kontekstni meni, s katerima je v shemo možno dodajati operacijske gradnike in spreminjati nastavitve izbranih gradnikov, okno za nastavljanje skupine parametrov vseh operacijskih gradnikov, ki sestavljajo shemo analize in statusni meni. Operacijski gradniki so uporabniku dostopne elementarne operacije, s katerimi se analiza izvaja. Gre za "škatle", ki jih uporabnik lahko intraktivno dodaja v shemo analize. Implementiramo jih v knjižnici (razdelek 6.2.1), vendar če želimo podpreti še interaktivno uporabo v uporabniškem vmesniku oz. menijih, moramo dodatno implementirati še funkcionalnosti uporabniškega vmesnika, ki uporabniku omogoča poljuben način interakcije v dodeljenem lastnem oknu. Obvezno je implementirati vhodno/izhodne parametre operacijskih gradnikov, za interaktivno rabo v uporabniškem vmesniku pa jim moramo določiti še vnosne forme. Izbiramo lahko med standardnimi parametri oz. vnosnimi formami ali pa jih moramo povsem na novo impementirati. Standardni parameter je na primer realno število, njegova vnosna forma je generična, izgleda pa kot drsnik, katerega intervali se določijo v programski izvedbi operacijskega gradnika. V kolikor je predviden gradnik namenjen monitoringu sistema oz. naj se bo sposoben na spremembe podatkov odzivati v realnem času, moramo zanj implementirati tudi funkcije v ta namen.

Na slikah 6.1 vidimo dva izgleda programa med izvajanjem analiz. Na namizju so okna za izpis podatkov, ki jih vrnejo razni algoritmi, realizirana je podpora za pregled vektorjev, razbitij in skupin vozlišč oz. izborov, na voljo so kontrole za parametre algoritmov, katere je pri nekaterih operacijskih gradnikih možno spreminjati sproti v realnem času med izvajanjem analize. V zgornjem kontekstnem meniju so razni gumbi, vezani na izbran operacijski gradnik, npr. operacije v prikazanem omrežju, spreminjanje pogledov, kontrola navigacije in način izbiranja vozlišč in povezav ipd. Pomembni so tudi gumbi za zaganjanje analize, ki prikazano shemo analize izvedejo, prekinejo, ustavijo ali omogočijo samodejni način zaganjanja. Vidimo tudi okni s teksturami in grafičnimi parametri. Vezani sta na grafične lastnosti omrežja in na operacijski gradnik prikazovalnik in urejevalnik omrežja (6.4.1). Spodaj je statusna vrstica v kateri se prikazujejo različni podatki, odvisni od izbranega okna ali operacijskega gradnika v uporabniškem vmesniku. Meniji, večina oken, statusna vrstica in drugi elementi so neke vrste razširitev pogleda na izbran operacijski gradnik, lahko pa jih na delovno na-

mizje pripnemo in so stalno prikazani, neodvisno od izbranega operacijskega gradnika.

Bolj kot glavni uporabniški vmesnik so pravzaprav zanimivi uporabniški vmesniki operacijskih gradnikov, saj večina interakcije z uporabnikom poteka v njih. Od tu naprej se bomo ukvarjali predvsem z njimi, še posebej s tistim za prikaz in urejanje omrežij (6.4.1) in delo s hierarhijami (6.4.2, 6.4.3).

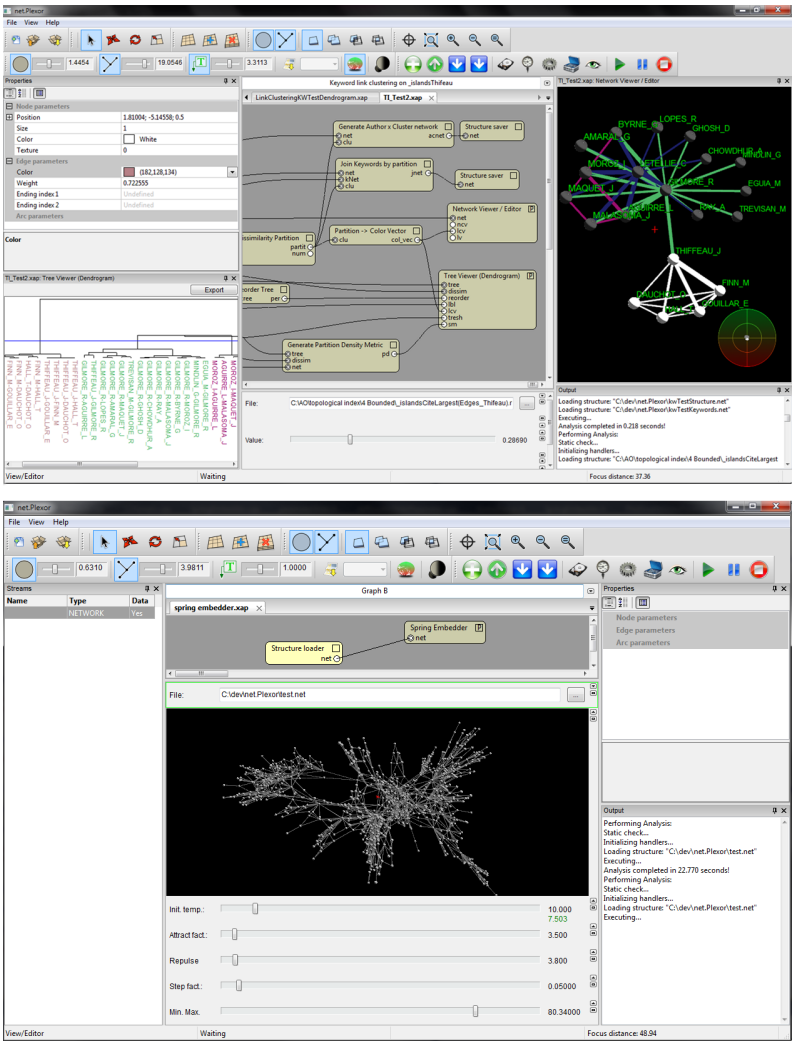
Delovno okolje za urejanje sheme analize

Jedro uporabniškega vmesnika je zasnovano tako, da omogoča razvoj načrta analize omrežij. Gre za neke vrste vizualno programiranje, kot ga zasledimo v programskem paketu za podatkovno rudarjenje *Orange* [96] ali pa na primer v programskem paketu *VisTrails* [103], ki je posebej namenjeno urejanju delovnih tokov. Za razliko od *Pajka* (opis v 6.1.2), ki je zasnovan kot kalkulator z omrežji, je v našem urejevalniku možno interaktivno definirati cel potek – shemo analize od začetka do konca v obliki omrežja operacisjkih gradnikov, ki jih med seboj povežemo s "cevmi" po katerih se pretakajo vmesni podatki oz. rezultati v interni predstavitvi – objekti, in sicer od izvornih datotek z omrežnimi podatki pa do prikazovalnikov rezultatov, gradnikov za shranjevanje, itd. Bistvena prednost pristopa je, da celotno analizo lahko enostavno ponavljamo, npr. ob spremembi kakega parametra, celo avtomatično. Pri izvajanju analize v *Pajku* moramo za vsako spremembo v neki fazi poteka analize korake v vseh naslednjih fazah ročno ponoviti. Ker posamezne operacije lahko trajajo več ur, to pomeni, da se bomo morali k računalniku vračati in dosledno izvesti vsak posamezen korak. Analizo se da sicer "posneti" v obliki makrov, vendar je zaganjanje za različne vrednosti parametrov še vedno zamudno. Naš način omogoča, da celotno analizo ponovimo že z enim samim klikom ali avtomatsko, na primer ob spremembi vhodnih podatkov. Pri tem sistem avtomatično upošteva, katerih podatkov, t.j. tistih katerih izračun ni odvisen od spremembe, ni potrebno vnovič računati. Tako prihranimo čas, še posebej pri preizkušanju z različnimi vrednostmi parametrov. Shema analize zagotavlja tudi dobro organizacijo in pregled.

Primer analize z našim algoritmom za razvrščanje povezav v omrežju (poglavje 4) je opisano v razdelku 6.4.2.

6.3 Interaktivnost v net.Plexor*-ju*

Naštejmo tri interaktivne funkcionalnosti v našem orodju. Sistem dodajanja in urejanja sheme analize je primer preproste interakcije z uporabnikom. Omogoča dodajanje,



Slika 6.1

Orodje za analizo in vizualizacijo velikih omrežij – *net.Plexor*. Zgoraj je prikazan primer analize omrežja z algoritmom za razvrščanje iz poglavja 4, implementiranim v operacijskem gradniku, ki je opisan v razdelku 6.4.2. Spodaj je prikazana analiza s hitro multipolno metodo umeščanja vozlišč omrežja v prostoru iz poglavja A.4.2 z operacijskim gradnikom, ki je opisan v razdelku 6.4.4.

premikanje in brisanje operacijskih gradnikov na delovni površini. Omogoča povezovanje operacijskih gradnikov in prikaz hitrih informacij ob lebdenju z miško nad "cevmi" oz. priključkih na operacijskih gradnikih. Interaktivno izvajanje sheme analize je naslednji primer. Operacijski gradniki lahko svoje naloge izvajajo vzporedno, pri tem pa uporabnik lahko interaktivno spreminja vhodne vrednosti pri tistih, pri katerih smo takšno interakcijo implementirali. V kolikor so vsi algoritmi zasnovani tako, da jih je možno prekiniti, je izvajanje analize možno interaktivno prekiniti ali začasno ustaviti in nadaljevati. Med izvajanjem operacijski gradniki svoje stanje interaktivno osvežujejo in posredujejo uporabniku. Tretji in najbolj očiten primer interaktivnosti pa je v zvezi z urejanjem in navigacijo po omrežju v operacijskem gradniku pregledovalnik in urejevalnik omrežja (6.4.1). Le-ta omogoča premike in obrate kamere po prostoru, spreminjanje zornega kota kamere, fiksiranje pogleda na izbrano vozlišče, prikaz ravnine, na katero je možno projicirati vozlišča. Podpira izbiranje in premike vozlišč v skupini ali posamič, v smeri koordinatnih osi, tako z ročnim premikom z miško ali z numeričnim vnosom koordinat. Možno je vrteti izbor vozlišč v prostoru okoli standardnih osi ali v frontalni ravnini uporabnika. Vozliščem je možno spreminjati teksture, barvo in velikost, povezavam je možno speminjati debelino in še kaj. Premik, rotacija in spreminjanje velikosti dela omrežja nad izbranimi vozlišči so v istem vrstnem redu prikazane na sliki 6.3.

Poglejmo si ozadje kompleksnih interaktivnih postopkov, ki smo ga razvili posebej za naš program.

6.3.1 Uporabniški vmesnik kot končni avtomat

Naivno bi uporabniški vmesnik realizirali z množico prepletajočih *if* ali pa morda *switch* stavkov, ki v danem stanju uporabniškega vmesnika poskrbijo za primeren odziv in prehod uporabniškega vmesnika v novo stanje. Že poskusi programske izvedbe preprostih interaktivnih operacij, daljših od nekaj klikov miške vodijo v razvoj težko razumljive, nepregledne in predvsem napakam podvržene kode. Dodajanje novih funkcij vmesnika, obvladovanje nesmiselnih uporabnikovih akcij in vzdrževanje pa je še težje. Mnogo boljši pristop pri izdelavi uporabniških vmesnikov dosežemo z vpeljavo primernega modela končnega avtomata, ki poskrbi za pravilno delovanje. Dela za vpeljavo nove funkcionalnosti je (na kratki rok) tako nekoliko več, vendar nam odpade večina potreb po testiranju obstoječih funkcionalnosti, ki bi zaradi nove morebiti prenehale delovati pravilno. Ker razvijamo kompleksen in skalabilen sistem z bogatim

naborom interakcij z uporabnikom, smo se odločili za sistematičen pristop. Z njim pravzaprav nič ne izgubimo. Vsak uporabniški vmesnik je ne glede na programsko izvedbo pravzaprav končni avtomat. Logiko uporabniškega vmesnika zakodiramo v obliki akcij, ki se zgodijo ob prehodih med stanji končnega avtomata in uporabniku vzbujajo občutek interaktivnosti. Prehodi se prožijo ob prevzetju znakov iz konca vrste vhodnih znakov v končni avtomat. Naboru znakov pravimo vhodna abeceda končnega avtomata, znaki pa so bodisi uporabnikovi ukazi, kot npr. kliki, pritisnjene in tipke ter notranji dogodki, ki jih porodi sam uporabniški vmesnik znotraj akcij. Vsak nov znak se zapiše na začetek vhodne vrste znakov avtomata. Model končnega avtomata na organiziran način hkrati predpisuje tudi vsa možna obnašanja sistema, saj se ob prehodih zgodijo samo in le točno predpisane akcije.

Za naše potrebe smo vpeljali še nekaj razširitev. Prilagodili in implementirali smo logiko t.i. hierarhičnega končnega avtomata [104]. Glavna ideja je v tem, da lahko vsako stanje rekurzivno vsebuje vgnezden hierarhični končni avtomat. Pojem prehoda moramo v takšnem avtomatu razširiti. V kolikor prehod za izbrani znak na vhodu avtomata v danem stanju ni definiran, je zaporedno po nivojih vse višje potrebno preveriti, če morda obstaja prehod z istim znakom iz stanja na kakem višjem nivoju. Prvi ustrezen prehod na višjem nivoju se izvede in avtomat se postavi v stanje na koncu izvedenega prehoda. Prehodov za primer krmiljenja "splošnih opravil" iz vsebovanih stanj nekega stanja s v neko drugo stanje t na istem nivoju kot s torej ne potrebujemo posebej definirati. Zadošča že, da prehod definiramo med stanjema s in t, hierarhično pa ga podedujejo vsa stanja v s vgnezdenega hierarhičnega avtomata. Prehodi so v splošnem lahko definirani med poljubnimi stanji in med poljubnimi nivoji v hierarhiji. V praksi se prehodom med različnimi nivoji izogibamo. Pri hierarhičnih končnih avtomatih smo vpeljali tudi dve dodatni vrsti akcij: vstopno in izstopno akcijo. Izvedejo se ob vstopu v stanje oz. ob izstopu iz njega. Potrebujemo jih predvsem pri prehodih med različnimi nivoji. Na primer pri prehodu iz pod-pod-stanja pps nekega stanja s na nivoju l v drugo stanje t na nivoju l se najprej izvede izstopna akcija za pod-pod-stanje pps stanja s, nato za pod-stanje ps stanja s, ki stanje pps vsebuje in nazadnje še za stanje s. Sledi izvedba akcije na prehodu med stanjema s in t, nato pa se izvede še vstopna akcija za stanje t itd. Omenimo še, da je v našem avtomatu na prehodih mogoče definirati pogoje. Če znak na koncu vhodne vrste avtomata ustreza prehodu, katerega pogoj ni izpolnjen, se ta prehod ne bo izvedel in znak bo ostal v vrsti. Če iz stanja za isti znak obstaja alternativni prehod z nižjo prednostjo, se bo, v kolikor je njegov pogoj

izpolnjen, izvedel ta itd. Prednost prehodov je določena implicitno glede na vrstni red navedbe prehodov ob definiciji hierarhičnega končnega avtomata. Če prehod za znak na koncu vrste ne obstaja v trenutnem stanju avtomata oz. v nobenem stanju na višjem nivoju, se znak vzame iz vrste, stanje avtomata pa ostane nespremenjeno. Prehodi brez določenega znaka se izvedejo takoj, pri prehodu pa se iz konca vrste ne vzame noben znak. Pred začetkom izvajanja avtomata se le-ta nahaja v začetnem stanju iz katerega pa mora biti definiran vsaj en brezpogojni prehod. Prehodi iz začetnega stanja ne smejo zahtevati sprejema znaka iz vrste. V primeru, ko se izvede prehod v stanje, v katerem je definiran vgnezden hierarhični končni avtomat, se slednji začne izvajati v svojem začetnem stanju.

Definicijo naših razredov, ki implementirajo opisan model hierarhičnega končnega avtomata podajamo razdelku A.3 v dodatku.

Primer definicije avtomata

Na sliki 6.2 je prikazan hierarhični končni avtomat, ki skrbi za interaktivno pregledovanje in urejanje omrežja z miško. Sestavljata ga dve stanji `Idle` in `Left Down`. Prvo ima vgnezden hierarhični končni avtomat s štirimi podstanji: `Begin Refresh`, `Ready to Render`, `Waiting` in `Stop Waiting`. Drugo stanje ima enajst podstanj.

Hierarhični končni avtomat s slike 6.2 in njegovo stanje `Idle` z uporabo naših razredov definiramo tako:

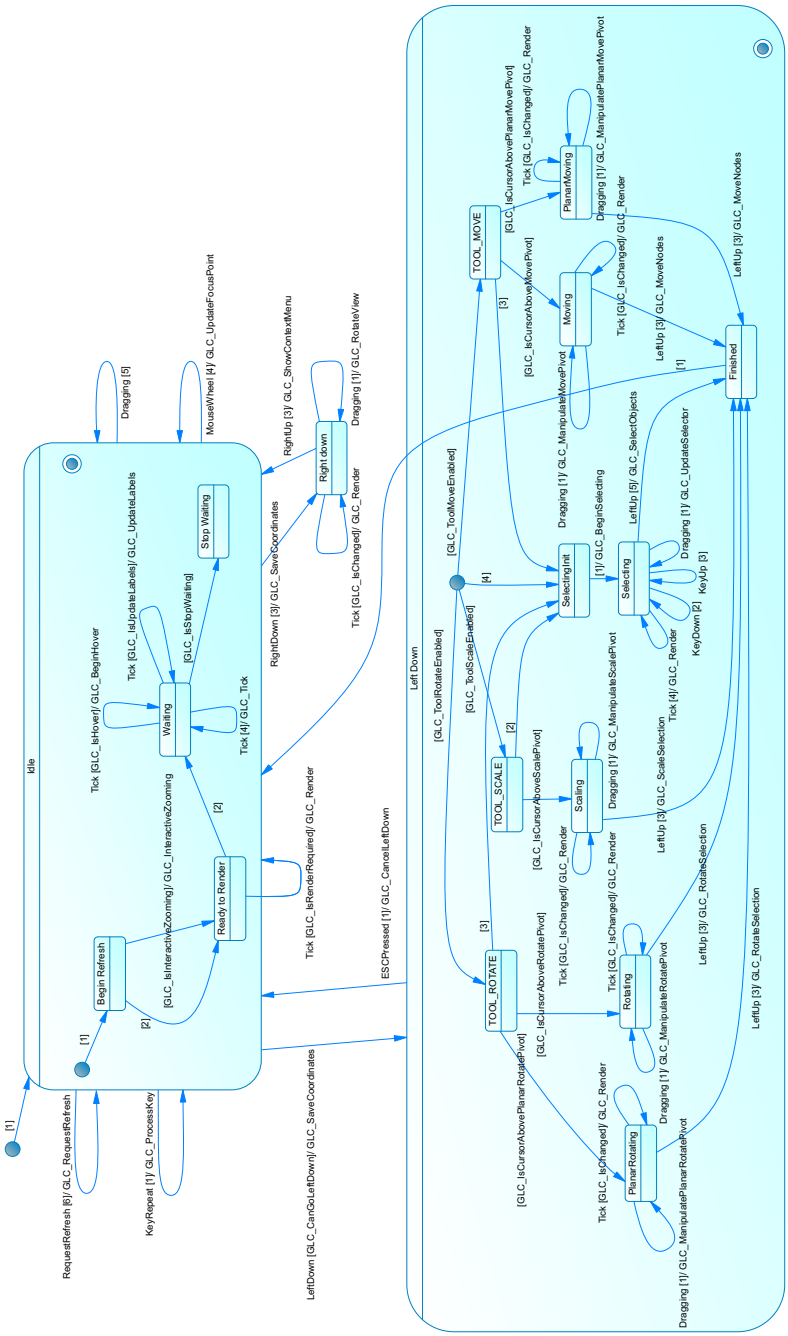
```
StateMachine *m, *n;

m = new StateMachine(NULL, NULL, NULL);
State *s = m->addNewState("Idle", &StateAction::GLC_EnterIdle, &StateAction::GLC_LeaveIdle); n = new StateMachine(s,
    NULL, NULL); s->assignStateMachine(n);
 n->addNewState("Waiting", NULL, NULL);
 n->addNewState("Stop Waiting", NULL, NULL);
 n->addNewState("Begin Refresh", NULL, NULL);
 n->addNewState("Ready to Render", NULL, NULL);

 n->addTransition(NULL, NULL, NULL, "Idle.Start", "Begin Refresh");
 n->addTransition(NULL, &Condition::GLC_IsInteractiveZooming, &Action::GLC_InteractiveZooming, "Begin Refresh", "Ready
     to Render");
 n->addTransition(NULL, NULL, NULL, "Begin Refresh", "Ready to Render");
 n->addTransition(HSTick::ID, &Condition::GLC_IsRenderRequired, &Action::GLC_Render, "Ready to Render", "Idle");
 n->addTransition(HSTick::ID, NULL, NULL, "Ready to Render", "Waiting");
 n->addTransition(NULL, &Condition::GLC_IsStopWaiting, NULL, "Waiting", "Stop Waiting");
 n->addTransition(HSTick::ID, &Condition::GLC_IsHover, &Action::GLC_BeginHover, "Waiting", "Waiting");
 n->addTransition(HSTick::ID, &Condition::GLC_IsUpdateLabels, &Action::GLC_UpdateLabels, "Waiting", "Waiting");
 n->addTransition(HSTick::ID, NULL, &Action::GLC_Tick, "Waiting", "Waiting");

m->addTransition(NULL, NULL, NULL, "Start", "Idle");

s = m->addNewState("Left down", NULL, NULL); n = new StateMachine(s, NULL, NULL); s->assignStateMachine(n);
 n->addNewState("Moving", NULL, NULL);
 n->addNewState("PlanarMoving", NULL, NULL);
 n->addNewState("Scaling", NULL, NULL);
 n->addNewState("Selecting", &StateAction::GLC_SelectionModeChange, NULL);
 n->addNewState("Rotating", NULL, NULL);
```



Slika 6.2

Hierarhični končni avtomat, ki skrbi za interaktivno pregledovanje in urejanje omrežja. Stanja so označena z zaobljenimi pravokotniki npr. stanje `Idle`. Začetna stanja so predstavljena s polnim krogcem, npr. levo zgoraj v stanju `Idle`. Končna stanja so označena z dvojnim krogcem. Povezave nakazujejo možne prehode med stanji. Oznaka na prehodu ima v splošnem obliko `<znak><[pogoj in/ali prednost]></akcija>`. Znak oz. dogodek, v kolikor je naveden, se mora nahajati na koncu vhodne vrste avtomata, da se prehod lahko izvede. V primeru več alternativnih prehodov, ki vodijo iz istega stanja, se upošteva pogoj in/ali prednost. Pogoj mora biti v vsakem primeru izpolnjen, da se prehod lahko izvede. Namesto oz. poleg pogoja je lahko podana prednost s številko. Prehodi z nižjo številko imajo prednost. V splošnem bi prehodi lahko imeli navedeno tako prednost kot pogoj, vendar imajo v našem primeru vsi prehodi z določenimi pogoji avtomatično višjo prednost, kot tisti, ki pogojev nimajo določenih. Beseda za poševnico, v kolikor je navedena, predstavlja ime akcije, ki se ob prehodu izvede. Vhodne in izhodne akcije stanj na sliki niso navedene.

```
n->addNewState("PlanarRotating", NULL, NULL);
n->addNewState("TOOL_MOVE", NULL, NULL);
n->addNewState("TOOL_SCALE", NULL, NULL);
n->addNewState("TOOL_ROTATE", NULL, NULL);
n->addNewState("Finished", NULL, NULL);

n->addTransition(HSEvent::ID, &Condition::GLC_ToolMoveEnabled, NULL, "Left down.Start", "TOOL_MOVE");
n->addTransition(HSEvent::ID, &Condition::GLC_IsCursorAboveMovePivot, NULL, "TOOL_MOVE", "Moving");
  n->addTransition(HSDragging::ID, NULL, &Action::GLC_ManipulateMovePivot, "Moving", "Moving");
  n->addTransition(HSTick::ID, &Condition::GLC_IsChanged, &Action::GLC_Render, "Moving", "Moving");
  n->addTransition(HSLeftUp::ID, NULL, &Action::GLC_MoveNodes, "Moving", "Finished");
n->addTransition(HSEvent::ID, &Condition::GLC_IsCursorAbovePlanarMovePivot, NULL, "TOOL_MOVE", "PlanarMoving");
  n->addTransition(HSDragging::ID, NULL, &Action::GLC_ManipulatePlanarMovePivot, "PlanarMoving", "PlanarMoving");
  n->addTransition(HSTick::ID, &Condition::GLC_IsChanged, &Action::GLC_Render, "PlanarMoving", "PlanarMoving");
  n->addTransition(HSLeftUp::ID, NULL, &Action::GLC_MoveNodes, "PlanarMoving", "Finished");
n->addTransition(HSEvent::ID, NULL, NULL, "TOOL_MOVE", "Selecting");
//...
```

V stanju `Idle` avtomat čaka na notranje dogodke časovnika `Tick` in v primeru, da je potrebno, poskrbi za osvežitev prikaza omrežja. Hkrati spremlja, če se kurzor miške dovolj dolgo nahaja nad kakšnim elementom omrežja (nad vozliščem ali povezavo). V kolikor se kurzor tam nekaj časa nepremično nahaja, se interaktivno prikažejo podrobne informacije o danem objektu.

V primeru, da uporabnik klikne levo tipko miške in je pogoj za prehod izpolnjen, avtomat preide v stanje `Left Down`. Stanje je namenjeno bodisi izbiranju elementov omrežja bodisi vrtenju, povečevanju in premikanju že izbranih elementov omrežja po osnovnih oseh x, y in z (slika 6.3). Poseben način urejanja je planarno vrtenje in planarno premikanje, ki omogoča vrtenje oz. premikanje dela omrežja v osi, ki je pravokotna na ravnino prikaza – poteka v globino.

V stanje `Right Down` avtomat preide ob pritisku na desno tipko miške. V tam stanju z miško krmilimo položaj kamere pogleda na prikazano omrežje.

Natančen opis delovanja avtomata na primeru

Ob zagonu programa avtomat iz začetnega stanja brezpogojno preide v stanje pripravljenosti `Idle`. V stanju `Idle` čaka na znake (dogodke) `LeftDown`, `RightDown` in druge. Predpostavimo, da uporabnik klikne z levo miškino tipko, kar povzroči, da se v vhodno vrsto avtomata doda dogodek `LeftDown`. Ne glede na podstanje stanja `Idle` v katerem se v danem trenutku avtomat nahaja, ob izpolnjenem pogoju `GLC_CanGoLeftDown` podstanje zapusti, saj implicitno, kot smo napisali v 6.3.1, iz vseh podstanj vodi povezava z znakom `LeftDown`. Pri tem se izvede akcija `GLC_SaveCoordinates`, ki v začasno spremenljivko shrani položaj miške ob kliku. V stanju `Left Down` ob enem od izpolnjenih pogojev `GLC_ToolRotateEnabled`, `GLC_ToolScaleEnabled` ali `GLC_ToolMoveEnabled` avtomat preide v podstanje `TOOL_ROTATE`, `TOOL_SCALE`

ali TOOL_MOVE. V kolikor ni izpolnjen noben od naštetih pogojev, se zaradi najmanjše prednosti brezpogojno zgodi prehod v podstanje SelectingInit.

Opisali bomo dogajanje v stanju TOOL_SCALE. V stanju TOOL_SCALE po preverjanju pogojev izhodnih prehodov v skladu s prednostjo naletimo na pogoj GLC_Is_CursorAboveScalePivot, ki določa, če se kurzor miške nahaja nad ročico za povečanje izbranega dela omrežja (slika 6.3 desno). Predpostavimo, da je pogoj izpolnjen in zato avtomat nadaljuje v stanje Scaling. Avtomat bi sicer prešel v stanje SelectingInit in uporabnika vodil v izbiranje elementov omrežja. Ob dogodkih premikov miške Dragging v stanju Scaling se na prehodu z višjo prednostjo ohranja stanje, pri vsakem prehodu pa se izvede akcija GLC_ManipulateScalePivot, ki poskrbi za spreminjanje oblike kazalnika spremembe velikosti (slika 6.3 desno). Razliko koordinat dobimo iz shranjenih koordinat iz lokalne spremenljivke, ki smo jo zapisali v GLC_SaveCoordinates in iz trenutnih koordinat, ki jih dobimo v dogodku Dragging. Ob dogodkih ure Tick, ki se porodijo v notranjem časovniku, pod pogojem GLC_IsChanged, ki je izpolnjen, če se je v GLC_ManipulateScalePivot dejansko kaj zgodilo, zgodi prehod nazaj v isto stanje, vendar se izvede tudi akcija GLC_Render, ki prikaz na zaslonu osveži: interaktivno se izriše prilagojen kazalnik velikosti in prikaže se sorazmerno povečan izbrani del omrežja. Če uporabnik spusti miškino tipko, se zgodi dogodek LeftUp, na prehodu v stanje Finished se pa izvede akcija GLC_ScaleSelection, ki operacijo na izbranem izboru elementov omrežja dokončno izvede. Če bi pred tem uporabnik pritisnil tipko *escape*, bi se ustvaril dogodek ESCPressed in zaradi definicije prehoda s tem dogodkom na višjem nivoju v stanju Left Down bi se izvedel prehod v stanje Idle. Hkrati bi se izvedla akcija GLC_CancelLeftDown, ki bi razveljavila operacijo spremembe velikosti. Iz stanja Finished vodi edini in direktni prehod v stanje Idle. Ker nima pogojev in ne potrebuje dogodkov, se neposredno takoj po prehodu v Finished brez čakanja na nov dogodek izvede. Ker nima akcije se ob prehodu tudi nič ne zgodi. S tem se interaktivna operacija zaključi in krog se lahko ponovi.

Interaktivna kontrola operacijskih gradnikov

Kljub temu, da logika v zvezi z interaktivno kontrolo operacijskih gradnikov spada tako v knjižnico, kot v uporabniški vmesnik, se konceptualno bolj nanaša na uporabniški vmesnik in jo zato opišemo tu. Kontrola operacijskih gradnikov temelji na hierarhičnem končnem avtomatu, katerih ozadje smo opisali v tem razdelku. Če pogledamo natančneje, sta za kontrolo operacijskih gradnikov pravzaprav namenjena dva hierar-

hična končna avtomata. V ozadju urejevalnika sheme analize, kamor "dodajamo" operacijske gradnike in jih med seboj povezujemo, je prvi, drugi pa skrbi za kontrolo nad potekom izvajanja operacijskih gradnikov, t.j. sheme analize. Prvi je nedvomno del uporabniškega vmesnika, drugi pa spada v okvir knjižnice, vendar se tesno navezuje tudi na uporabniški vmesnik. Oba vidika kontrole bomo v grobem opisali.

Urejevalnik sheme analize temelji na ideji vizualnega programiranja. Koncepte vizualnega programiranja na primer uporabljajo tudi v programu *Orange* [96] in *VisTrails* [103]. Pri nas gre za postopek "lepljenja" operacijskih gradnikov in povezovanje njihovih izhodov in vhodov. Primer vezja je opisan v razdelku 6.4.2. Poznamo več vrst operacijskih gradnikov, med njimi so algoritmi, urejevalniki, pregledovalniki, generatorji, nalagalniki, shranjevalniki ipd. Povezave med gradniki pa predstavljajo tokove podatkov. Poznamo več vrst podatkov, med njimi omrežje, vektorje različnih vrst, preureditev, razbitje, hierarhiji vozlišč in povezav, razred, osnovne tipe podatkov in druge.

Interaktivno povezovanje operacijskih gradnikov zagotavlja povezovanje združljivih vhodov in izhodov po vrsti. Pri tem tokovi lahko predstavljajo prenos podatkov po vrednosti ali po referenci. Pri prenosu po vrednosti, se struktura na izhodu kopira na vhod, pri prenosu po referenci pa se prenese le kazalec na strukturo. V drugem primeru prihranimo na prostoru, saj v pomnilniku obstaja le ena instanca strukture. Pri razvoju operacijskih gradnikov mora razvijalec zagotoviti, da operacijski gradnik, ki sprejema po referenci, ne spreminja vhodne strukture. Obstajajo tudi nekatere izjeme. Pri omrežjih na primer se sme slog omrežja, ki se prenaša po referenci, v naslovnem operacijskem gradniku spreminjati, ne pa tudi struktura omrežja. Pri interaktivnem povezovanju gradnikov je poskrbljeno, da ne pride do nekonsistentnih stanj. V primeru prenosa po referenci je na izhod možno priklopiti le en gradnik, ki spreminja vrednosti itd. Težavo bi preprosto rešili, če bi se vse strukture prenašale po vrednosti, a bi s tem močno povečali potrebe po pomnilniku. Potrebno je sprejeti ustrezen kompromis.

Izvajalnik operacijskih gradnikov oz. sheme analize je zasnovan interaktivno. Uporabniku omogoča zagon sheme, interaktivno prekinjanje, nadaljevanje, v kolikor določen operacijski gradnik omogoča, tudi sprotno spreminjanje parametrov in predčasno ustavljanje. V načinu samodejnega zaganjanja sheme analize se zagon izvede samodejno, brez uporabnikovega posredovanja, in sicer po krajšem pretečenem času (1s) se avto-

matično sproži na enak način, kot če bi shemo analize uporabnik vnovič ročno zagnal. Pri tem se, kot sicer, izvede le aktualni del sheme analize, t.j. tisti del, katerega vhodni podatki so bili iz nekega razloga invalidirani na primer kot posledica spremembe vhodne datoteke. Izvajanje sheme analize je realizirano v zanki, okvirno po principu razvrščanja procesov *round-robin* [105]. Vsak operacijski gradnik je pravzaprav proces, ki se mu dodeli časovno rezino, v kateri izvaja svoje delo. Osnovni razred iz katerega se izpelje implementacije funkcionalnosti operacijskih gradnikov ima v naprej predvideno obnašanje, le-ti pa se naprej specializirajo po posameznih vrstah, ki smo jih našteli v prejšnjem odstavku. Osnova vsem gradnikom je sledeča:

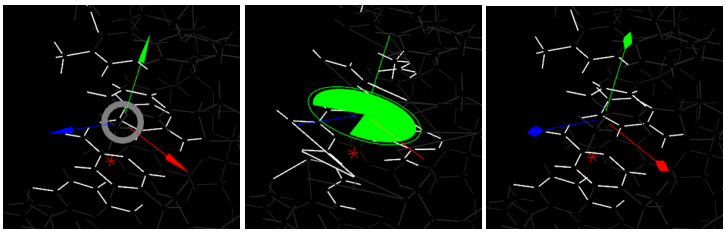
```
class PlexHandler //osnovni razred operacijskega gradnika
{
public:
   virtual bool CheckBeforeInit(); //metoda preveri, če je operacijski gradnik smiselno povezan
   virtual void Init(); //metoda proskrbi za statično inicializacija operacijskega gradnika; gradnik se pripravi na
       izvajanje svoje naloge
   virtual bool Finished(); //metoda vrne true, če je operacijski gradnik zaključil s svojim delom
   virtual bool FinishedWithError(); //metoda vrne true, če je operacijski gradnik svoje delo zaključil z napako
   virtual void PerformElementalStep(); //metoda poskrbi za izvedbo elementarnega koraka, t.j. dela naloge, ki jo mora
       opraviti operacijski gradnik v analizi; posamezen del naloge naj se nebi izvajal dlje kot 100ms
   virtual void Pause(); //metoda postavi operacijski gradnik v prekinjeno stanje izvajanja
   virtual void Resume(); //metoda po prekinitivi izvajanja operacijski gradnik pripravi za nadaljnje delo
   virtual void Terminate(); //metoda (na željo uporabnika) predčasno zaustavi izvajanje opreacijskega gradnika; s tem
       se analiza zaključi
   virtual void OnStandbyTick(); //metoda skrbi za periodične posodobitve stanja operacijskega gradnika v času
       pripravljenosti
 //...
}
```

Izvajanje se začne s klici metod `CheckBeforeInit` vseh operacijskih gradnikov v shemi analize in se nadaljuje le, če vsi gradniki izpolnjujejo pogoje za delo; so pravilno povezani, imajo nastavljene vse obvezne parametre... Sledi statična inicializacija operacijskih gradnikov s klici njihovih metod `Init`. Običajno se rezervirajo viri, ki jih gradniki potrebujejo za opravljanje svoje naloge. V neskončni zanki se nato ciklično zaporedoma s klici metode `Finished` preverja ali je posamezen operacijski gradnik svoje delo zaključil in v kolikor ga še ni, se pokliče njegova `PerformElementalStep` metoda. Metoda poskrbi za izvajanje dela, ki ga mora operacijski gradnik izvesti. V njej se hkrati izvaja tudi kontrola konsistence podatkov, npr. če so podatki že na voljo in ko so na voljo, če so združljivi. V kolikor gradnik v dani iteraciji še nima na voljo vseh podatkov, izvajanje prepusti naslednjemu gradniku. V kolikor podatki so na voljo, a niso konsistentni, na primer omrežje in preureditev na vhodu nekega gradnika sta pravkar postala na voljo, ne ustrezata pa si v velikosti, gradnik izvajanje sheme analize ustavi z napako, pokliče se njegova metoda `FinishedWithError`, uporabnik pa prejme ustrezen opis napake. Ko vsi gradniki končajo svoje delo, se izvajanje sheme

analize uspešno zaključi. V kolikor tekom izvajanja uporabnik prekine izvajanje, se po izvajanju koraka izvajajočega se gradnika vsem gradnikom pokliče metoda `Pause`, da lahko (predvsem interaktivno podprti) gradniki poskrbijo za prekinitev, izvajanje pa se prekine. Ko uporabnik sproži nadaljevanje izvajanja se vsem gradnikom pokliče metoda `Resume`, ki (pradvsem interaktivno podprtim) gradnikom omogoči nadaljevanje dela, izvajanje pa se ciklično nadaljuje. Če uporabnik kadarkoli ustavi izvajanje, se do konca izvede korak trenutno izvajajočega gradnika, nato pa se vsem gradnikom kličejo metode `Terminate`, ki operacijskim gradnikom sporočijo, da naj predčasno zaključijo z izvajanjem svojih nalog, kar povzroči tudi takojšnjo sprostitev virov, ki jih opreacijski gradniki zasedajo. Slednje bi se sicer zgodilo ob zadnjem klicu `PerformElemental-Step`, ko bi hkrati gradnik prešel v zaključeno stanje. Izvajanje se nato ustavi in ga ni mogoče nadaljevati, lahko pa ga seveda na novo začnemo. V načinu samodejnega zaganjanja je logika delovanja enaka, le da se, dokler ne pride do napake, ali pa uporabnik eksplicitno ne ustavi delovanja, po krajšem časovnem intervalu analiza vnovič samodejno zažene.

Logika znotraj metode `PerformElementalStep` običajno preveri, če so na voljo vsi podatki, da se lahko izvede elementarni korak v operacijskem gradniku, kot ga imenujemo. Gre za del naloge, ki jo mora operacijski gradnik izvesti v celotni analizi. Elementarni korak mora biti implementiran tako, da se ne izvaja predolgo. Če želimo podpreti interaktivno izvajanje, navadno priporočamo, da izvajanje elementarnega koraka ne traja več kot 100ms. V nasprotnem primeru uporabniški vmesnik lahko daje vtis neodzivnosti. Za zdaj operacijski gradniki zaporedno izvajajo elementarne korake, za interaktivnost pa mora poskrbeti razvijalec, tako da omeji maksimalno dolžino izvajanja posameznega elementarnega koraka. Poskusno smo izvajalnik realizirali z nitmi, in sicer na večkratni `round-robin` način, tako da podmnožice operacijskih gradnikov, v kolikor so njihovi vhodi neodvisni, svoje naloge lahko izvajajo paralelno. Zaradi težav z nitmi v *c++* delovanje še ni dovolj stabilno.

Po uspešnem zagonu sheme analize naj omenimo, da se izhodi vseh operacijskih gradnikov postavijo v veljavno stanje. Kar pomeni, da so vrednosti podatkov na njihovih izhodih konsistentne z vrednostmi na njihovih vhodih. Ob spremembi parametrov nekega operacijskega gradnika g ali ob spremembi povezav vhodnih tokov podatkov tega operacijskega gradnika, se v verigi operacijskih gradnikov, ki imajo vhodne podatke odvisne od izhodov gradnika g, vsem posreduje informacija, da morajo svoje izhode invalidirati. Veljavna stanja izhodov operacijskih gradnikov so interaktivno prikazana



Slika 6.3

Nekatere operacije za urejanje strukture omrežja v *net.Plexorju*. Od leve proti desni: premik, vrtenje in razširanje označenih vozlišč omrežja.

tako, da so dotični izhodi obarvani z modro. Invalidirani izhodi so obarvani belo. Ob konstrukciji sheme analize so vsi izhodi operacijskih gradnikov invalidirani. Razlog invalidiranja izhodov je v optimizaciji izvajanja. V kolikor vemo, da so vsi izhodi nekega operacijskega gradnika v veljavnem stanju, jih pri ponovnem zagonu sheme analize ta gradnik ne potrebuje vnovič izračunavati.

Metoda `OnStandbyTick` je namenjena periodičnemu preverjanju stanja operacijskega gradnika, ko je shema analize v pripravljenosti, t.j. ko se ne izvaja. Metodo smo na primer implementirali v operacijskem gradniku za nalaganje podatkov iz datoteke. Če se vhodna datoteka s podatki iz kakršnega koli razloga spremeni, v metodi `OnStandbyTick` operacijski gradnik spremembo zazna in svoje stanje invalidira.

6.4 *Operacijski gradniki v* net.Plexor-*ju*

V tem poglavju bomo izpostavili nekaj pomembnih operacijskih gradnikov, ki so povezani s pričujočo doktorsko disertacijo in jih je avtor razvil sam.

6.4.1 *Prikazovalnik in urejevalnik omrežja*

Eden izmed pomembnih delov programske opreme za analizo omrežja je modul za prikazovanje omrežij. V našem orodju se nahaja v obliki operacijskega gradnika s svojim uporabniškim vmesnikom. Izpolnjuje dve zahtevi: učinkovitost in interaktivnost, je pa tudi intuitiven za uporabo. Za pregledovanje omrežij nudi več alternativnih zunanjih/notranjih pogledov. Primer prikaza omrežja je na sliki 6.7. V operacijskem gradniku za pregledovanje je implementirana tudi funkcionalnost za urejanje omrežij. Primeri nekaterih funkcij urejanja omrežja so prikazani na sliki 6.3.

Z uporabo tipkovnice in miške ter z uporabo gumbov v menijih lahko omrežje interaktivno pregledujemo. Skozi omrežje lahko potujemo na način, ki spominja na letenje v prostoru. Uporabnik na podatke gleda skozi prvine navidezne resničnosti. V

osnovni so omrežja prikazana v treh razsežnostih. Omrežje je vedno na voljo tudi v alternativnem pogledu, in sicer v obliki ravninskega zemljevida (6.4.1), ki uporabniku pomaga pri globalni orientaciji v omrežju. Poleg tega ima na voljo še nekatere druge orientacijske sestavine, kot so koordinate položaja, oddaljenost od izhodišča, če želi, pa lahko v omrežje vstavlja pomožne ravnine. Za bolj nazorni pogled bližnjih struktur omrežja lahko vključi senčenje oddaljenih elementov omrežja (vozlišč in povezav), ki se z oddaljenostjo od uporabnika stopnjujoče zlijejo z ozadjem. Če se uporabnik v omrežju "izgubi", se lahko kadarkoli vrne v začetni položaj ali pa v enega izmed shranjenih izbran položajev, ki jih lahko dodaja med pregledovanjem. Na ta način si lahko sestavi vodene sprehode po omrežju. Trenutno lahko uporabnik na omrežje gleda skozi več "očal", ki mu omogočajo skrivanje vozlišč ali povezav omrežja. V okviru pogleda lahko povezave in/ali vozlišča začasno ojači ali ošibi (odebeli, stani). V kolikor uporabnik želi, lahko na isto omrežje gleda iz poljubno mnogo perspektiv hkrati, vsak pogled v svojem oknu. Pri tem vse opisane funkcionalnosti nastavlja za vsak pogled individualno.

Uporabnik lahko interaktivno spreminja lastnosti elementov v omrežju in ureja njegovo topografsko strukturo. Na interaktiven način lahko označuje poljubno množico vozlišč, izbrano množico vozlišč prestavlja po prostoru, povečuje po vseh oseh ali samo eni, jo vrti po poljubni osi, jo projicira na izbrano ravnino in še kaj. Vozliščem in povezavam lahko spreminja teksture, moč, barvo... Izbrano množico lahko postavi v žarišče pogleda, si ogleduje njihove opise, vključuje oz. skriva podrobnosti... Numerične parametre vozlišč in povezav lahko uporabnik vpisuje tudi numerično, in sicer tako posameznim, kot celi skupini izbranih elementov omrežja hkrati. V ozadju je možno nastaviti sliko, ki je na primer uporabna pri konstrukciji omrežja po vzorcu na sliki.

Tehnologija

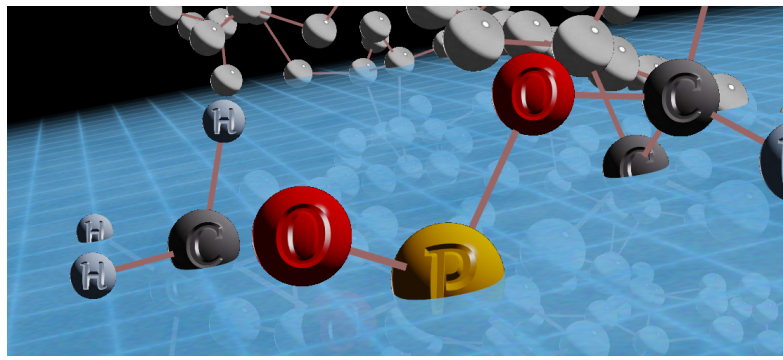
Za grafične potrebe smo izbrali knjižnico *OpenGL*® [106], verzije 3,3, ki je prva, ki standardizira geometrijski senčilnik [107]. Senčilniki (*shaders*) so upodobitveni programi, ki tečejo na grafični procesni enoti – GPE [108] na grafični kartici. *OpenGL*® predstavlja industrijski standard vmesnika oz. knjižnice za delo z grafiko za nivo višje nad gonilniki grafične kartice, ki omogoča dostop do strojnih upodobitvenih funkcionalnosti. Do neke mere implementira tudi programsko upodabljanje. Za naše potrebe je verjetno edina primerna izbira, saj jo podpira večina modernih grafičnih kartic na vseh platformah. V našem programu grafika pretežno temelji na strojnem upodabljanju, dodatno pa smo izkoristili še nekatere trike, da smo upodabljanje pospešili.

Delovanje

Ob inicializaciji omrežja v pomnilnik grafične kartice zapišemo vse potrebne podatke, ki se bodo uporabljali pri upodabljanju. To so predvsem teksture in geometrija omrežja. Zapišemo tudi senčilnike – programe za grafično kartico, ki se v gonilniku *OpenGL®* prevedejo in shranijo na grafično kartico, ki jih kasneje le-ta uporablja za izvajanje grafičnih in drugih operacij. Koordinate in velikosti vozlišč zapišemo v medpomnilnik, dostopen prek indeksov znotraj senčilnikov, saj jih potrebujemo tako pri vozliščih, kot pri povezavah. Koordinat povezav posebej ne rabimo prenašati v medpomnilnik, saj se avtomatično izračunajo iz koordinat vozlišč in pripadajočih indeksov. Ker poznamo dve vrsti povezav: usmerjene in neusmerjene, za vsako vrsto inicializiramo svoj medpomnilnik in vanj naložimo pare indeksov izvornega in ponornega vozlišča. Na ta način prihranimo precej prostora in rokovanje z omrežjem je posledično enostavnejše. Dovolj je namreč, da spremenimo le koordinate vozlišč, povezave pa naj avtomatsko obdela grafična kartica. S tem senčilniki povezav postanejo nekoliko bolj kompleksni, a ne bistveno počasnejši.

Izris vozlišč omrežja je realiziran z vnaprej pripravljenimi kvadratnimi teksturami, ki lahko dajejo vtis treh razsežnostih. Pripravimo jih tako, da v nekem 3-razsežnostnem grafičnem orodju – modelirniku upodobimo vozlišče in njegovo sliko shranimo v teksturo – rastrski slikovni format (glejte v dodatek, razdelek A.2.6) in teksture uporabimo pri upodabljanju. Pri takšni predstavitvi zadošča, da so vozlišča na grafični kartici predstavljena s posamezno točko (3 koordinate). Dejanske koordinate vozlišč se najprej pretvori v koordinate prostora kamere, teksture pa se razen spremembe velikosti, brez dodatnih transformacij nariše na zaslon v skladu s preračunanimi koordinatami. Za vozlišča tako ne potrebujemo dodatnih podatkov o geometriji in preračunavanja, ki bi znatno upočasnilo izrisovanje. Da vozlišča niso nujno kvadratna, poleg barvnih kanalov teksture vsebujejo še transparentni (alfa) kanal, ki določa robove. Za dodatno stopnjo "realizma" pa smo dodali še globinski kanal, ki določa plastičnost vozlišča, kar pride do izraza na primer pri prekrivanju z ravnino. Primer je prikazan na sliki 6.4, kjer so nekatera vozlišča presekana z ravnino. (Gre za vpeljavo tako imenovanega Z-vmesnika (*Z-buffer*).) V praksi na tak način dosežemo dovolj dobre rezultate. Pri uporabljanju ne pride do efekta poudarjene perspektive na robovih zaslona, kar ni nujno slabo, vozlišč pa ne moremo senčiti.

Izris povezav realiziramo z daljicami, ki se v geometrijskem senčilniku razširijo v



Slika 6.4

Primer prikaza v *net.Plexorju*. Vozlišča so realizirana s tremi teksturami: barvna tekstura, ki vozlišču daje izgled, transparentna tekstura (alfa), ki vozlišču določa robove in prostorska sivinska tekstura, ki določa plastičnost vozlišča. Učinek slednje vidimo pri vozliščih na preseku z ravnino, kjer vzbujajo občutek, da so vozlišča resnično kroglice in ne ploščice.

puščice t.j. piramide. Pretvorba v celoti poteka avtomatično na grafični kartici v odvisnosti od pre-definiranih indeksov vozlišč. Implementirali smo še en alternativni prikaz povezav, s t.i. "vžigalicami", saj po izgledu spominjajo nanje. Z minimalnimi dopolnitvami bi povezave lahko prikazovali tudi s teksturami.

Držali smo se načela, da se kar največ operacij glede vizualizacije in interakcije z uporabnikom prenese na GPE, saj se s tem razbremeni centralno procesno enoto – CPE, ki lahko vmes izvaja druge operacije. Zaradi visoke stopnje paralelnosti je pričakovati, da se bodo GPE še naprej hitreje razvijale kot CPE. Izbiranje vozlišč in povezav smo realizirali na GPE. Tudi nekatere enostavnejše algoritme, na primer algoritme za vzmetno umeščanje (*spring embedder*) se da realizirati na GPE, kar smo zapisali v dodadtku A.4.2. Ozko grlo je v komunikaciji med CPE in GPE, saj je vodilo med njima asinhrono in razmeroma počasno. Grafični del na GPE naj bi bil zato čim bolj neodvisen od programskega, t.j. od tistega, ki teče na CPE. Prav model senčilnikov omogoča večjo neodvisnost med procesnima enotama. Z vnaprejšnjo pripravo podatkov v pomnilniku grafične kartice t.j. geometrije, metapodatkov (npr. podatek o izbranosti vozlišča) in tekstur, ustreznih senčilnikov in drugih parametrov grafične kartice, CPE omogočimo, da celotni izris, v našem primeru omrežja, sproži že z enim samim klicem na grafično kartico. Po inicializaciji CPE želimo, da le-ta deluje čim bolj neodvisno od CPE. Komunikacijo med njima skušamo omejiti, če ne gre bolj, vsaj na bloke podatkov, ki se izmenjujejo le ob določenih akcijah uporabnika. Med izrisovanjem prometa med CPE in GPE skoraj ni. Prenašamo le ključne podatke; to so indeksi izbranih vozlišč ali njihovih tekstur, koordinate premaknjenih vozlišč, spremenjene vrednosti gradnikov

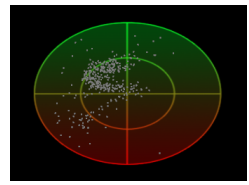
omrežja ipd.

Prikaz oznak na prikazanih strukturah

Učinkovito izrisovanje tekstovnih oznak v grafiki velikokrat predstavlja izziv, saj večina grafičnih kartic ne podpira strojnega pospeševanja za izrisovanje besedil. V nekaterih primerih je to rešeno z vmesnikom oz. gonilnikom grafične kartice, vendar v primeru verzije *OpenGL*®, ki jo uporabljamo, te podpore ni. Primeren algoritem za izrisovanje besedil smo realizirali sami. Primer rezultata je viden na zgornjem delu slike uporabniškega vmesnika: 6.1.

Ideja algoritma je v tem, da prikazuje oznake tistih objektov (vozlišč, povezav, idr.) kateri zasedajo največjo površino na zaslonu. Prikazovanje oznak vseh objektov je preveč potratno in tudi nepregledno. Potratno je, ker imamo v omrežjih običajno objekte z različini oznakami, za razliko od njihove oblike, ki jo je možno dobro parametrizirati in zato ne potrebujemo veliko vzorcev oblike. Oznaka mora biti v grafični kartici interno predstavljena s teksturo, teksture pa zasedejo precej prostora. Shranjevanja tekstur za vse oznake si zato ne moremo privoščiti. Lahko bi uporabili teksture posameznih črk in jih v fazi izrisovanja sestavljali v oznake, a se je izkazalo, da je pristop relativno neučinkovit, saj na zaslonu želimo prikazati veliko oznak. Tudi shranjevanje velikega števila majhnih tekstur je potratno. Naš algoritem v eno samo večjo teksturo shranjuje omejeno število celih oznak.

Algoritem za pripravo oznak v danem trenutku pripravi teksture za največ n (parameter – maksimalno število oznak) ustreznih oznak. Zapiše jih v grafično kartico v eno samo večjo teksturo, ustrezne odmike za posamezno oznako v teksturi pa predstavi s številkami rež, ki jih dodeli aktualnim objektom. Številke rež in skupna tekstura oznak se nato uporabijo pri izrisu. Ko se na zaslonu pojavi tendenca za izris drugih oznak, npr. po vrtenju omrežja, kjer do izraza pridejo druga vozlišča, algoritem učinkovito izračuna novo množico objektov $\mathbf{N}$, katerih oznake morajo biti vidne. To stori z uporabo senčilnika na grafični kartici, ki za vsak objekt omrežja učinkovito izračuna njegovo velikost v projekciji pogleda in vrne rezultate, algoritem pa jih nato uredi in izbere fiksno število n največjih (če jih je seveda dovolj). Nova množica $\mathbf{N}$ največjih objektov je običajno podobna stari $\mathbf{S}$, saj so premiki pogleda zvezni in postopni, kar je razlog, da algoritem lahko deluje učinkovito. Razlika množic objektov $\mathbf{N} \setminus \mathbf{S}$ predstavlja objekte za katere je potrebno na novo pripraviti teksture oznak in številke rež. Pri tem algoritem ohranja teksture oznak, ki so še aktualne, t.j. za objekte v preseku



Slika 6.5

Navigacijski zemljevid v *net.Plexorju*.

množic $\mathbf{N} \cap \mathbf{S}$. Povedano drugače; žrtvuje tiste, ki ne bodo več prikazani ($\mathbf{S} \setminus \mathbf{N}$) in jih nadomesti z novimi. Posodobi tudi številke rež objektov množice $\mathbf{S} \setminus \mathbf{N}$. Algoritem skrbi, da je na zaslonu največ n oznak, in sicer pri najvidnejših objektih.

Navigacijski zemljevid

Velik pomen pri analizi omrežja je dobra predstava uporabnika, kje v omrežju se trenutno nahaja, torej orientacija. Pristop k problemu smo delno rešili s programsko izvedbo navigacijskega zemljevida z *Riemannovo* transformacijo [109] vozlišč v prostoru na razpeto kroglo, kot jo vidimo na zemljevidu, slika 6.5. O sorodnih hiperboličnih prikazih govorimo tudi v poglavju 3.2.4.

Riemannova preslikava točk, v našem primeru vozlišč na kroglo, je uporabna metoda prikazovanja v prostoru. Na zgornji sliki je prikazan konkreten pogled na zemljevid, ki nam omogoča takšen prikaz. Točke oz. vozlišča so sivo obarvana. Preslikava deluje tako, da glede na kamero, t.j. položaj uporabnika v prostoru izračuna odmik smeri posameznega vozlišča od smeri pogleda uporabnika v intervalu od 0° (točno pred uporabnikom) do 180° (točno za uporabnikom) in kot linearno preslika v odmik od središča zemljevida. Notranji krog predstavlja kote manjše od 90°, kolobar med notranjim in zunanjim krogom pa kote več od 90°. Zunanja krožnica predstavlja 180°. Kot, ki ga točka, t.j. vozlišče glede na središče zemljevida oklepa z desno polovico vodoravne osi zemljevida narekuje smer v kateri se vozlišče nahaja. Torej točke na desni polovici zemljevida so na desni strani uporabnika, na levi pa levo od uporabnika. Podobno velja za točke spodaj in zgoraj. Osi zemljevida v bistvu predstavljata transverzalno (vodoravno) in sagitalno (sredinsko) ravnino uporabnika, točke pa lego vozlišč glede na ti dve ravnini.

Učinkovitost grafike

Tu predstavljamo le rezultate testov na obstoječih funkcionalnostih. Ko bo podprtih več lastnosti omrežja, bo hitrost upodabljanja omrežij verjetno nekoliko manjša, vendar se bo količino podrobnosti dalo poljubno prilagoditi. Meritve smo opravili empirično na dveh identičnih osebnih računalnikih, a z različnima operacijskima sistemoma. Oba računalnika sta imela procesor *Core-2-Duo* 2,5GHz, 4GB pomnilnika in *NVIDIA GeForce 9400M* grafično kartico s 16 jedri in 256MB deljenega pomnilnika. Na enem je bil nameščen *Mac OS X*, na drugem pa *Ms. Windows 7*.

Program smo primerjali s *Pajkom* [34] in *Gephi*-jem [99]. Rezultati so prikazani v obliki zmanjšanja kvalitete uporabniške izkušnje, t.j. primernosti programa za delo pri dani velikosti omrežij.

Primerjava z Gephi-*jem in* Pajkom

V vseh treh programih na obeh računalnikih smo odprli omrežja različnih velikosti in jih skušali narisati. Pri *Gephi*-ju je delo že pri relativno majhnih omrežjih postalo težavno. Pri omrežjih velikosti 2.000 vozlišč in 8.000 povezav je hitrost izrisa postala prenizka za udobno delo. Na zaslonu se je slika posodabljala komaj nekaj krat na sekundo. Avtorji programa *Gephi* sicer omenjajo, da temelji na *OpenGL*®, vendar kot kaže niso izkoristili celotnega potenciala. Bolj so se osredotočili na estetski pogled upodobitev. Povezave so v *Gephi*-ju prikazane z loki. V našem programu in v *Pajku* so predstavljene z daljicami. Hitrost izrisovanja v *Pajku* je višja, a moramo omeniti, da *Pajek* pri izrisovanju ne pozna strojnega pospeševanja. Njegov pregledovalnik omrežij omogoča nekatere interaktivne operacije nad omrežjem, vendar temelji na osnovni *Windows GDI* grafiki, ki je počasna in zastarela. Izrisi so preprosti in enostavni. V *Pajku* je predvsem možno pripraviti lepe vektorske slike za izvoz. Za razliko od *Gephi*-ja in *Pajka net.Plexor* v veliki meri izkorišča prednosti modernih grafičnih kartic, do katerih dostopamo preko standardnega vmesnika *OpenGL*®. Logiko prikazovalnika 6.4.1 smo razvili od začetka in posebej optimalno za izris velikih omrežij.

Pri omrežjih s 50.000 vozlišči in 250.000 povezavami je delo v *net.Plexor*-ju postalo ovirano, a še vedno ugodno. Močno ovirano je postalo, ko je velikost omrežij presegla 100.000 vozlišč in pol milijona povezav. Pri tem omrežju je hitrost osveževanja padla na 4 osvežitve na sekundo. Če smo opazovali samo vozlišča, je bilo delo ugodno tudi pri omrežjih z 200.000 vozlišči. Primerjavo med programi prikazujemo v tabeli 6.1.

Tabela 6.1

Subjektivna ocena estetike prikazov in hitrost izrisov, ki jo podajamo kot kvaliteto uporabniške izkušnje pri delu z različnimi reprezentativnimi omrežji v *Gephi*-ju, *Pajku* in *net.Plexorju*. Ker je v *net.Plexorju* možno prilagajati nivo podrobnosti omrežja, ki jih prikazujemo, je učinkovitost podana tudi v odvisnosti ali prikazujemo le vozlišča ali celotno omrežje. (Vse pikice pomenijo, da se v programu slika osveži vsaj 20 krat na sekundo. Nič pikic pomeni, da hitrost osveževanja pade pod eno osvežitev na sekundo. "k" in "M" pomenita tisoč oz. milijon.)

	Gephi	*Pajek*	*net.Plexor*	
Estetika prikazov	●●●●●	●●○○○	●●●●○	
#vozlišč / povezav			Le vozlišča	Vse
100/500	●●●	●●●	●●●	●●●
1k/4k	●●○	●●●	●●●	●●●
5k/10k	●○○	●●○	●●●	●●●
50k/250k	○○○	●○○	●●●	●●○
100k/500k	○○○	○○○	●●●	●●○
200k/1M	○○○	○○○	●●●	●○○
400k/2M	○○○	○○○	●●○	○○○

V smislu vizualizacije je naš pregledovalnik za dva reda hitrejši od estetsko dovršenega *Gephi*-ja. V primerjavi s *Pajkom* je za en red hitrejši.

Zavedati se moramo, da je risanje več kot recimo 10.000 vozlišč in povezav hkrati nepraktično, saj zaslon postane zasičen z elementi omrežja, zato lahko zaključimo, da je prikazovalnik dovolj hiter, da zadosti potrebam vizualizacije omrežij. Od tu naprej se moramo zateči k drugim metodam prikazov, kot je abstrakcija delov omrežja, prikaz hierarhije omrežja, o katerem pišemo v razdelku 6.4.3 ali pa aditivno risanje omrežij kateremu smo namenili poglavje 5.

6.4.2 Algoritem za razvrščanje povezav omrežja in podporni gradniki

Tu na kratko opišemo in navedemo, kako smo programsko izvedli algoritem, ki smo ga razvili in predstavili v poglavju 4.

Naša programska izvedba algoritma deluje na sledeč način. V začetku algoritma se vsako povezavo dodeli v svojo skupino. Algoritem nato vstopi v glavno zanko, kjer naključno izbere skupino povezav, pri kateri bo začel graditi verigo obiskov skupin povezav. Verigo gradi v smeri strogo padajoče najmanjše različnosti med skupinami povezav. Pri tem si pomaga tako, da poleg, v okolici verige gradi še inkrementalni povezavni podgraf različnosti med skupinami povezav. Ko naleti na par lokalno najbolj podobnih skupin povezav, ju vzame iz verige, ju združi v novo skupino in ustrezno popravi povezavni podgraf v njuni okolici. Nato z novega konca nadaljuje z gradnjo verige. Ko verigo do konca izprazni, začne iz naslednje naključno izbrane, preostale in neobiskane skupine graditi novo verigo. Spotoma še počisti spremljajoči povezavni

podgraf. Z novo verigo ga prične ponovno sestavljati. Pri združitvi para skupin algoritem skupini postavi v hierarhijo in dodatno shrani različnost med njima. Algoritem se konča z združitvijo zadnjih dveh skupin povezav.

Inkrementalni povezavni podgraf Inkrementalni povezavni podgraf realiziramo s pomočjo dveh struktur. Prva je slovar različnosti med pari skupin povezav osnovnega omrežja. Gre pravzaprav za slovar uteži na povezavah povezavnega podgrafa, ki določajo pare povezanih skupin povezav osnovnega omrežja oz. različnosti med njimi. Druga struktura je slovar množic sosednih skupin povezav osnovnega omrežja, ki nam služi za učinkovit dostop do skupini povezav sosednjih skupin povezav v povezavnem podgrafu.

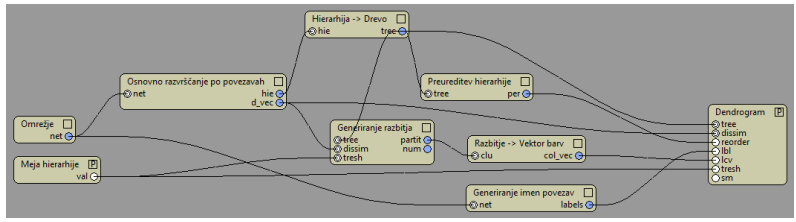
Predstavitev hierarhije Strukturo za shranjevanje hierarhije smo implementirali v obliki tabele kazalcev, ki jo poznamo pod imenom "seznam kazalcev na starše" velikosti $2 \cdot m - 1$. (m je število povezav v vhodnem omrežju.) Na začetku vsaka skupina, torej povezava dobi svoje mesto v tabeli, in sicer na položaju od 1 do m. V vsaki iteraciji algoritma se dve skupini združita v novo skupino, kateri se dodeli zaporedno številko trenutne iteracije algoritma, povečane za število povezav vhodnega omrežja m: $i + m$. Polje z indeksom $i + m$ v tabeli bo namenjeno bodočemu staršu nove skupine in predstavlja vejitev v hierarhiji. V tabeli se v polje, ki pripadata predhodnikom nove skupine zapiše indeks nove združene skupine $i + m$. V zadnji iteraciji algoritma se združita poslednji dve skupini, v njim ustrezni polji v tabeli pa se zapiše indeks korena hierarhije, ki ga običajno predstavimo z vrednostjo 0.

Struktura skupin Skupina je sestavljena iz množice povezav, ki ji pripadajo. Poleg tega v njeni strukturi hranimo še seznam kazalcev na aktivna vozlišča, v katerih je prisotna, in seznam frekvenc ključev, ki skupini pripadajo.

Struktura za aktivna vozlišča Algoritem hrani seznam aktivnih vozlišč, pri čemer ima vsako seznam skupin povezav, ki so v njem "aktivne" – se v njem prekrivajo.

Primer analize z uporabo razvrščanja povezav omrežja

Plan analize z razvrščanjem povezav v omrežju je prikazan na sliki 6.6. Še en primer je prikazan tudi na sliki 6.1 zgoraj. Operacijski gradnik *Omrežje* poskrbi za nalaganje omrežja v pomnilnik. Naloženo omrežje se v notranji predstavitvi posreduje gradniku *Osnovno razvrščanje po povezavah*, ki izvede algoritem za razvrščanje povezav z osnovno



Slika 6.6

Primer sheme analize z operacijskimi gradniki v *net.Plexor*-ju, ki realizirajo preprosto analizo s hierarhičnim razvrščanjem povezav v omrežjih, ki smo ga razvili in predstavili v poglavju 4.

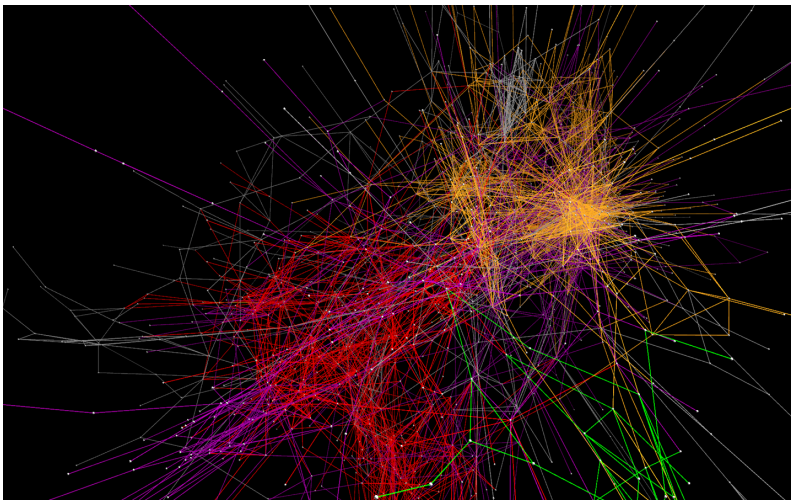
različnostjo na vhodnem omrežju in vrne hierarhijo v obliki zaporedja kazalcev na starša v novem objektu. Hkrati za vsako vejitev v hierarhiji vrne objekt vektor različnosti. Omrežje se posreduje tudi gradniku *Generiranje imen povezav*, ki za vsako povezavo omrežja ustvari ime povezave. Gre za niz znakov oblike `<ime prvega krajišča povezave>-<ime drugega krajišča povezave>`. Operacijski gradnik *Hierarhija->Drevo* poskrbi, da se hierarhija iz oblike zaporedja kazalcev na starša predela v drevesno strukturo kazalcev na potomce. *Generiranje razbitja* prejme objekt izračunane hierarhije v obliki drevesa in vektor različnosti vejišč v hierarhiji. Hkrati mu uporabnik posreduje še referenco na vrednost – mejo rezanja, ki jo ročno nastavi na primer v generičnem operacijskem gradniku za realno vrednost *Meja hierarhije*. Izhodno razbitje se prenese v gradnik *Razbitje->Vektor barv*, ki vsaki skupini v vhodnem razbitju priredi barvo iz seznama barv. Barve vrne v obliki vektorja. *Preureditev hierarhije* prejme hierarhijo v obliki drevesa in ustvari preureditev s katero je končne veje hierarhije možno preurediti tako, da se v dendrogramu ne bodo sekale. Prikazovalnik *Dendrogram* poskrbi, da se izbrana hierarhija grafično prikaže v dendrogramu. Pri tem upošteva vektor barv končnih vej v hierarhiji, različnosti, ki narekujejo višino vejitev hierarhije glede na bazo dendrograma, preureditev vej hierarhije, imena vej hierarhije in mejo rezanja, ki se prikazuje kot premica, ki dendrogram reže pri izbrani različnosti – višini.

Primer rezultata algoritma, ki smo ga pripravili z *net.Plexor*-jem je prikazan na sliki 6.7.

6.4.3 *Prikazovanje hierarhij*

Dendrogram

Eden od osnovnih prikazov hierarhije je dendrogram. V *net.Plexor*-ju smo ga implementirali v obliki operacijskega gradnika *dendrogram*, ki pričakuje 7 vhodnih tokov. Dva obvezna vhodna tokova sta hierarhija v obliki kazalcev na starša in vektor različ-



Slika 6.7

Primer slike omrežja z rezultati razvrščanja z našim algoritmom (4). Omrežje smo narisali z uporabo operacijskega gradnika za prikazovanje/urejanje omrežij. O njem je več zapisano v razdelku 6.4.1.

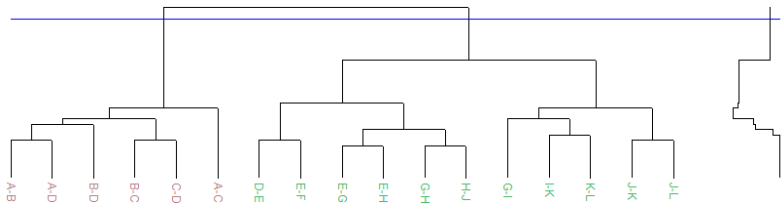
nosti, ki pripadajo posameznim vejitvam v hierarhiji. Neobvezni vhodi so preureditev listov v drevesu hierarhije, vektor oznak posameznih listov, vektor barv listov in vrednost rezanja hierarhije na skupine listov. Zadnji vhod predstavlja vektor vrednosti, ki predstavljajo odvisne vrednosti v grafu, ki se prikaže v navpični smeri desno ob dendrogramu. Uporaben je na primer za prikaz metrike kvalitete reza dendrograma na posameznem nivoju, kot je na primer gostota razbitij (*partition density*) [70]. Na sliki 6.8 je z dendrogramom prikazan primer hierarhije. Prikaz vključuje vse neobvezne vhode.

Interaktivni prikazovalnik splošnih hierarhij

V *net.Plexor*-ju smo implementirali interaktivni prikazovalnik velikih acikličnih oz. splošnih hierarhij. Hierarhijo je mogoče prikazati na več načinov, ki jih izpostavljamo v poglavju o hierarhijah 3. Prikaz je omejen na konstantno število nivojev hierarhije, saj bi v primeru prikaza celotne hierarhije hitro prišlo do velike gneče na prikazu. Hierarhija je v vsakem trenutku prikazana relativno, glede na trenuten položaj opazovanja – žarišče. Glavnino prikaza zajema vejitev trenutnega položaja v hierarhiji v žarišču (bele barve). Njej podrejene vejitve so prikazane v odtenkih sive glede na njihovo naraščajočo relativno globino v hierarhiji. V primeru, da se v okviru omejene globine doseže

Slika 6.8

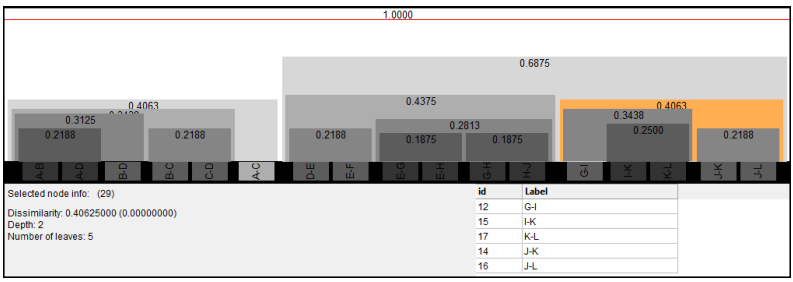
Primer izrisa dendrograma hierarhije v *net.Plexor*-ju. Modra prečna črta prikazuje položaj reza hierarhije. Listi dendrograma so v skladu z rezanjem razvrščeni v dve skupini, ki sta predstavljeni z različnima barvama. Prikaz v danem primeru podaja kvaliteto razbitja, ki bi ga dobili z rezanjem hierarhije na ustreznem nivoju, in sicer najmanjšo kvaliteto zgoraj in spodaj oz. najboljšo v sredini.



končno vozlišče oz. list hierarhije, se le-to prikaže na spodnjem robu prikaza, izven vejitve trenutnega pogleda. Za razliko od vejitev nima izpisane različnosti, kvečjemu ime, v kolikor so imena na voljo. Nad vejitvijo trenutnega položaja je prikazan en sam relativni nivo staršev trenutne vejitve oz. položaja v hierarhiji. Ker gre za aciklični graf, ima vejitev lahko več staršev.

Obvezni vhodni tok prikazovalnika je hierarhija, ki mora biti za razliko od vhodnega toka v dendrogram, vrste aciklični graf. Neobvezna vhoda sta še vektor imen listov hierarhije in vrednost rezanja, ki je na prikazu označena z vodoravno rdečo črto. Ker je prikaz omejen glede na trenutni položaj opazovanja hierarhije in z omejenim številom nivojev v globino, se lahko zgodi, da črte rezanja pri opazovanju na izbranem nivoju hierarhije ne vidimo.

Sprememba trenutnega položaja pregledovanja se izvede z dvoklikom na vejitev, katero želimo postaviti v žarišče pogleda. Z enkratnim klikom na poljubno vejitev ali list hierarhije je zagotovljeno sprotno prikazovanje ključnih informacij o vejitvi oz. listu, kot je na primer globina v hierarhiji, različnost, ki vejitvi ustreza, število in seznam pripadajočih listov ipd. Na sliki 6.9 je prikazana enaka hierarhija, kot v dendrogramu na sliki 6.8. Ena izmed vejitev je izbrana in obarvana oranžno. Na sliki spodaj levo so izpisane ključne iformacije o izbrani vejitvi. V konkretnem primeru gre za vejitev z interno zaporedno številko 29. Na desni strani so izpisane interne zaporedne številke in imena vozlišč skupine, ki jo izbrana vejitev vsebuje. Na sliki je prikazana tudi rdeča črta, ki predstavlja rez hierarhije v skupine.



Slika 6.9

Primer izrisa preproste hierarhije v interaktivnem prikazovalniku hierarhij v *net.Plexor*-ju. Trenutno izbrana vejitev prikaza – žarišče, ki v prikazanem primeru ustreza korenu hierarhije, je prikazana v beli. Podvejitve so prikazane v odtenkih sive. Pod zgornjim robom vejitev je zapisana različnost med podvejitvami. Rdeča prečna črta prikazuje položaj rezanja hierarhije. Listi hierarhije so prikazani v spodnjem delu. V levem spodnjem delu so prikazane informacije o trenutno izbrani (oranžno obarvani) vejitvi, desno spodaj pa imena listov, ki izbrani vejitvi ustrezajo.

6.4.4 Umeščanje omrežij s hitro multipolno metodo

Eden bolj zanimivih gradnikov v *net.Plexor*-ju je namenjen prostorskemu umeščanju vozlišč omrežja. Zanimiv je zato, ker ga je možno dodobra izkoristiti za dopolnitev hierarhičnih prikazov 3. Implementirali smo ga v tehnologiji *OpenCL*®, ki izkorišča prednosti računanja na hitrih grafičnih procesorjih. O testiranju in programski izvedbi smo več zapisali v dodatku (A.4), tu pa izpostavljamo operacijski gradnik, ki metodo realizira. Enostaven primer gradnika je prikazan na sliki 6.1 spodaj.

Gradnik vključuje lasten prikazovalnik omrežja, ki ga na sliki vidimo v glavnem oknu na sredini. Prikazuje omrežje, kot ga je umestil algoritem. Spodaj vidimo množico parametrov, s katerimi reguliramo obnašanje algoritma. Nastavimo lahko začetno "temperaturo" vozlišč (algoritem deluje po metodi simuliranega ohlajanja), faktorja privlačne in odbojne sile med vozlišči oz. skupinami vozlišč, velikost "časovnega" koraka v posamezni iteraciji algoritma (gre za fizikalni model) in omejitev premestitve za prehod na višji nivo v poteku algoritma. Parametri se nanašajo na hitro multipolno metodo, ki je opisana v dodatku (A.4). Po zagonu algoritem teče v neskončni zanki, dokler niso izpolnjeni pogoji za zaključek; običajno dovolj majhni premiki v omrežju, ki so odvisni od omrežja, nastavitve omejitev premestitve na višji nivo ter trenutne temperature, ki je odvisna od začetne temperature in časa izvajanja algoritma. Parame-

tre v realnem času tekom izvajanja lahko spreminjamo (glejte razdelka 6.2.1 in 6.3.1). S tem lahko vplivamo na potek algoritma, hkrati pa lahko na prikazu omrežja opazujemo spremembe. Če v začetku na primer nastavimo močno odbojno silo, se bo omrežje na začetku hitro razširilo, če pa nato odbojno silo zmanjšamo ali pa povečamo še privlačno silo, se bo algoritem odzval in omrežje se bo ponovno skrčilo in ustalilo, ko bo temperatura dovolj nizka. Trenutne vrednosti parametrov, ki se jih nastavi na začetku (na primer temperaturo) je med izvajanjem možno spremljati. Poleg začetnih vrednosti, zapisanih desno od drsnikov s črno, so trenutne vrednosti zapisane z zeleno barvo.

Vpeljava možnosti spreminjanja parametrov algoritmov v realnem času tekom izvajanja je dobrodošla novost. Da bi dobili primeren izgled omrežja, kot ga želimo na primer prikazati na sliki v članku, je dovolj da nastavimo okvirne vrednosti parametrov, zaženemo analizo in po potrebi vmes nekoliko prilagajamo vrednosti, dokler nam oblika ne ustreza. Še več, med izvajanjem je možno premikati vozlišča omrežja in tako ročno pomagati algoritmu. Kot je navada v mnogih drugih programih, algoritma ne rabimo znova in znova zaganjati z različnimi parametri.

6.4.5 net.Plexor *v prihodnosti*

Zasnovali smo temelje za učinkovit uporabniški vmesnik in podporo za uvedbo novih funkcionalnosti v obliki algoritmov oz. vtičnikov za delo z omrežji. Naše delo se bo nadaljevalo v smeri izpopolnitev in dodajanja novih interaktivnih funkcij in analitičnih algoritmov. V planu je razvoj preprostega opisnega jezika za opisovanje slogov za upodobitve omrežij. Dodali bomo podporo za prilagajanje uporabniškega vmesnika, ki je delno že implementirana. Dodali bomo nove alternativne poglede na omrežje, s katerimi bomo uporabnikom omogočili še bolj transparenten pogled in udobnejše delo ter gibanje v večjih omrežjih, velikosti milijon in več gradnikov. Ena izmed pomembnih dopolnitev programa bo tudi možnost izvajanja skript v nekem višjem jeziku. Dober primer skriptnega jezika je *Python*. S skriptno podporo bi širšemu krogu uporabnikom omogočili poganjanje lastnih avtomatskih skript. Razvoj vtičnikov je namreč težka naloga, za reševanje specifičnih problemov pa nepraktična in toga rešitev. Razvoj vtičnikov je primeren, ko imamo opravka s splošnimi problemi, za katere potrebujemo učinkovito orodje. Nekoliko manj zmogljiva in splošna, vendar vseeno močna alternativa skriptni podpori, še bolje pa razširitev, bi bila možnost animiranja (planiranja) sprememb vrednosti parametrov algoritmov. Parametre nekaterih algoritmov je že

mogoče spreminjati v realnem času izvajanja analize. Postopek bi nadgradili tako, da bi se s pomočjo načrta sprememb parametrov v času potek spreminjanja parametrov popolnoma avtomatiziral. Dolge analize bi na ta način vizualno programirali še časovno. Ideja o postopku izvira iz orodij za video obdelavo, animacijo, 3-razsežnostnih modelirnikov in drugih podobnih orodij. Izmed pomembnejših izpopolnitev navajamo še možnost razveljavljanja sprememb. Nemalokrat se zgodi, da nam sprememba v poteku analize ne uspe, kot smo si zamislili, ali pa preprosto poskušamo priti do ustrezne rešitve na več načinov. Pri tem je uporabno, če nekaj zadnjih korakov lahko preprosto razveljavimo in uberemo drugo pot. Izkaže se, da je v analizi velikih omrežij razveljavljanje akcij težka naloga. Že na primer izvedbo operacijskega gradnika, ki na vhodu po referenci (glejte 6.3.1) prejme veliko omrežje in mu z irreverzibilnim postopkom spremeni slog (npr. dodeli naključne barve vozlišč), je možno razveljaviti samo pod pogojem, da si pred izvedbo zapomnimo slog vhodnega omrežja. Pri tem je potrebno zagotoviti dovolj prostora (velika omrežja lahko zasedejo dobršen del delovnega pomnilnika). Ena od rešitev je, da zgodovino shranjujemo na disk, ali morda, da v omejenem obsegu shranjujemo le spremembe ipd. Funkcionalnost bi izboljšali tudi z vpeljavo hierarhije na nivoju sheme analize z operacijskimi gradniki. Vpeljali bi lahko generičen operacijski gradnik, ki bi mu parametrično definirali vhode in izhode, njegovo jedro pa bi sestavili iz drugih operacijskih gradnikov. Vpeljali bi torej možnost hierarhične definicije v shemi analize. Korak naprej bi bil tudi uvedba kontrolnih zank in hierarhično definiranih vrst podatkovnih tokov.

Zaključek

7

7.1 Glavni znanstveni doprinosi

Tu na kratko še enkrat povzamemo glavne znanstvene doprinose disertacije. Pri vsakem doprinosu navajamo poglavje v disertaciji, kjer smo ga razdelali in hkrati podamo povezave na naše objave v zvezi z njimi. Objave so mednarodne: ugledne znanstvene revije, monografska dela, izdana pri uglednih založnikih in mednarodne konference.

- *Obsežen scientometrični vpogled v področje topoloških indeksov*
  Za potrebe razvoja in vrednotenja v naslednji točki opisanega algoritma smo pripravili A.1 primere bibliografskih omrežij iz spletne storitve *Web of Science* na temo topoloških indeksov. Bibliografska omrežja smo v mnogih pogledih analizirali podrobneje in v poglavju 2 opisali rezultate. Navedimo nekaj primerov opravljenih analiz: analiza poti citiranj, analiza s sredicami in z otoki, analiza dvodelnih omrežij, časovna analiza, razvrščanje, analiza ključnih besed in druge. Podobne, predvsem pa ne tako poglobljene in izčrpne bibliografkse analize na področju topoloških indeksov nismo zasledili. Rezultati so bili predstavljeni na mednarodni konferenci [110] in objavljeni v članku [85].

- *Izboljšan, razširjen in posplošen algoritem za hierarhično razvrščanje povezav v omrežjih*
  V poglavju 4 smo opisali postopke, s katerimi lahko izboljšamo obstoječe algoritme za razvrščanje v omrežjih. Bistvena prednost našega pristopa je, da upošteva dodatne lastnosti omrežij kot so lastnosti vozlišč in povezav. Med drugim prispevek vsebuje opis primera obstoječega algoritma, ki ga razširimo v nov izboljšan algoritem, pripadajoče mere različnosti, lastnosti algoritma in prikaze delovanja algoritma na primerih. Prispevek je bil predstavljen na mednarodni konferenci [111]. V obliki članka [112] je v času pisanja disertacije poslan v objavo. Prejeli smo pozitivni odziv recenzentov in pripombe že vključili v disertacijo.

- *Načrt in programska izvedba interaktivnega orodja za analizo omrežij z vizualnim programiranjem, kontrolo v realnem času in prvinami navidezne resničnosti*
  Razvili smo inovativno celovito programsko rešitev za analizo velikih omrežij, ki med drugim vključuje večino postopkov, ki so opisani v pričujoči disertaciji. Bistvene novosti pri našem orodju so vpeljava vizualnega programiranja v analizo omrežij, podpora za interaktivno krmiljenje algoritmov za analizo omrežij v realnem času in pregledovanje omrežij, vključno z notranjimi pogledi oz. s

prvinami navidezne resničnosti. V orodju so vključeni interaktivni pregledovalniki velikih omrežij in hierarhij s prvinami abstrakcije. Orodje smo podrobno razdelali v poglavju 6, predstavljeno pa je bilo na dveh mednarodnih konferencah [113, 114]. V dodatku A.3 smo podali osnove hierarhičnega končnega avtomata, na katerem temelji interaktivnost našega orodja. V dodatku A.4 smo na primeru umeščanja vozlišč omrežja v prostor analizirali možnosti grafičnega pospeševanja, ki ga v orodju izkoriščamo.

- *Načrt metode za prikazovanje velikih hierarhično urejenih omrežij*
  V poglavju 3 smo opisali nekaj običajnih prikazov hierarhično urejenih podatkov. Nadalje predlagamo način, kako s prepletanjem interaktivnosti in abstrakcije prikazovati oz. urejati velike množice hierarhično urejenih podatkov, kot so na primer hierarhije velikih omrežij, ki jih vrne naš algoritem. Za razliko od običajnih pregledovalnikov in urejevalnikov hierarhij, v katerih je delo omejeno na drevesne hierarhije, predlagamo teoretično osnovo, na kateri je mogoče realizirati pregledovalnik oz. urejevalnik, ki omogoča delo s splošnimi hierarhijami oz. z acikličnimi grafi. Metoda je okvirno realizirana v našem orodju za analizo velikih omrežij *net.Plexor* 6.4.3.

- *Metoda geografskega prikaza za promet v mobilnih omrežjih*
  Razvili smo izvirno metodo za prikaz mobilnega prometa v telekomunikacijskem mobilnem omrežju in z njo prikazali mobilni promet v pol-letnem obdobju na Slonokoščeni obali. Metodo je mogoče posplošiti in uporabiti tudi na drugih področjih. Ob enem predstavlja primer prikaza s povzetki oz. z zgostitvami, ki nas zanimajo v prejšnji alineji. V okviru analiz mobilnega prometa smo metodo opisali v poglavju 5. Metodo in rezultate analiz smo na mednarodni konferenci predstavili v obliki plakata. Rezultate izsledkov nekaterih dodatnih analiz o katerih v disertaciji ne pišemo, smo na isti konferenci predstavili z drugim plakatom. Oba prispevka sta izšla tudi v zborniku konference [92, 94].

- *Pregled omrežnih datotečnih formatov*
  V dodatku A.2 smo predstavili namen in nabor omrežnih datotečnih formatov, ki se često pojavljajo v analizi omrežij, jih med seboj primerjali in opredelili nekatere podrobnosti, ki nam lahko povzročijo težave pri zbiranju omrežnih podatkov. Izpostavili smo načine pretvarjanja med omrežnimi datotečnimi formati

in navedli nekaj orodij za delo z njimi. Prispevek predstavlja nov, poenoten in izčrpen popis omrežnih datotečnih formatov, ki v takšnem obsegu do slej še ni bil opravljen. Objavljen je kot samostojno poglavje v enciklopediji [115].

7.2 Smernice za nadaljnje delo

Povzemimo še enkrat pomembnejše ideje in cilje za prihodnost, ki smo jih zapisali že v zaključkih posameznih poglavij. Zapisali smo, da bi naš algoritem za razvrščanje lahko paralelizirali. Paralelni algoritem bi tekel v obliki več vzporednih instanc, vsaka v svoji niti. S paralelizacijo bi časovno zahtevnost algoritma precej izboljšali. Algoritem bi dodatno lahko prilagodili za delo z usmerjenimi omrežji, veliko odprtega pa je tudi na področju mer različnosti. Na področju prikazov hierarhij in prikazov v splošnem je tudi odprtih veliko možnosti. Naš cilj je predvsem izpiliti predlagan postopek in ga do potankosti vgraditi v naše orodje *net.Plexor*. Aktualen je razvoj preprostega opisnega in splošnega jezika za opisovanje slogov za upodobitve omrežij, kar je hkrati zanimiva tema tudi na področju omrežnih datotečnih formatov. Precej verjetno je sicer, da se bo takšen jezik izoblikoval organsko. Pregled oz. tabele omrežnih datotečnih formatov bomo skušali ohranjati v koraku s časom. Dostopne so v obliki spletnega dodatka [116]. Ker se je izkazalo, da programov za pretvarjanje omrežnih datotečnih formatov ni veliko, bi morda bilo smiselno razviti poenoten program za pretvarjanje med pogostimi, podobnimi in združljivimi omrežnimi datotečnimi formati. Zasnovali bi ga na primer v obliki javno dostopne spletne storitve. Orodje za analizo velikih omrežij *net.Plexor* smo zasnovali tako, da je vanj enostavno vgrajevati nove funkcionalnosti oz. vtičnike. Najprej moramo odpraviti napake in izpopolnili obstoječe funkcionalnosti, nato pa bomo postopoma dodali večji in splošnejši nabor analitičnih algoritmov in funkcionalnosti. Dodali bomo alternativne poglede na omrežje, s katerimi bo delo bolj transparentno in udobno. Uporabna sposobnost orodja bi bila možnost izvajanja skript. Pomembna dograditev bi bila tudi možnost razveljavljanja sprememb ali morda vpeljava hierarhije na nivo sheme analize z operacijskimi gradniki, t.j. vpeljava generičnega operacijskega gradnika, ki bi mu uporabnik lahko parametrično definiral vhode in izhode, njegovo jedro pa bi sestavil iz drugih operacijskih gradnikov... Nenazadnje je naša najpomembnejša naloga, da orodje približamo širši skupini uporabnikov. V zvezi z analizami mobilnih omrežij obstaja verjetnost, da se tudi v prihodnje spopademo z izzivi *D4D*. Na naslednjega, ki bo potekal v letošnjem letu, smo prijavljeni. Obstaja verjetnost, da se posvetimo tudi kakšni bibliografski oz. scientometrični anali-

zi, kot smo jo izvedli na področju topoloških indeksov. Morda jo izvedemo na podlagi kake druge bibliografske baze, kot sta na primer *Google Scholar* ali *Scopus*, ali pa izvedemo primerjalno analizo rezultatov med različnimi bazami za isto temo. Možnosti v tej smeri je veliko in izkazalo se je, da je bibliografksa oz. scientometrična analiza dobrodošla.

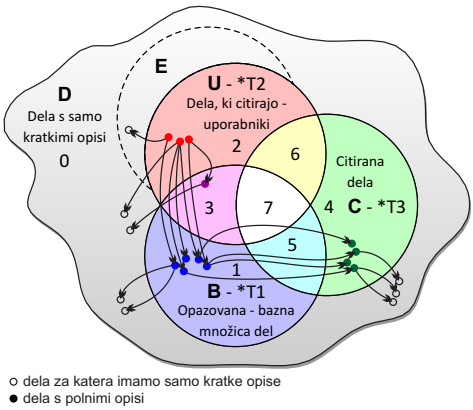
Dodatki

A

A.1 *Priprava bibliografskih omrežij in drugih konstruktov iz* Web of Science

Web of Science [16] je spletna bibliografska in akademska podatkovna storitev, ki jo nudi *Thomson Reuters*. Storitev nam omogoča dostop do bibliografskih podatkovnih baz in je namenjena interdisciplinarnemu raziskovanju in poglobljenemu raziskovanju specializiranih pod-področij znotraj akademskih ali znanstvenih strok. Indeksira večino pomembnih znanstvenih del, predvsem tistih, ki so bila objavljena po letu 1975. V tem poglavju bomo opisali postopek, kako iz *Web of Science* lahko pridobimo podatke in jih predelamo v omrežja, ki jih je možno analizirati s postopki analize omrežij.

Web of Science po izgledu spominja na spletni iskalnik in uporabljamo ga na podoben način, se pa v namenu precej razlikuje. Iskanje nam vrne povzetke oz. opise del, ki ustrezajo iskalnim kriterijem. Takšne opise je mogoče izvoziti na več različnih načinov, a je za nadaljnjo obdelavo predvsem prikladen izvoz v tekstovne datoteke. Zadetki osnovnega iskanja predstavljajo "interesno" oz. "bazno" množico opisov del, ki jo običajno označimo z **B**. Množico **B** običajno razširimo še z množico del, ki citirajo vsaj eno delo v **B** in jo imenujemo množica "uporabnikov" **U**. Ti opisi so pomembni, saj z njimi lahko bolje določimo pomembnost posameznih del v "bazni" množici. Opise del množice **U** v *Web of Science* najlažje dobimo z uporabo funkcije *citation report*, ki služi prav iskanju opisov del, ki citirajo izbrano množico del. Rezultate prav tako izvozimo v tekstovne datoteke. V splošnem množici **B** in **U** nista nujno tuji. **U** namreč lahko vsebuje dela v **B**, ki citirajo dela iz **B**. Za popoln pregled področja je priporočljivo poiskati še "samo citirana" dela iz **B**. Navadno jih označimo z množico **C**. Z njimi dobimo boljši vpogled v razvoj področja in identificiramo ključna dela, začetnike področja. Ker v *Web of Science* ne obstaja avtomatiziran postopek za iskanje samo citiranih del iz dane množice, moramo iskanje opraviti ročno. Iskalne nize lahko pripravimo ročno, smo pa razvili preprosto aplikacijo, ki iz opisov del v tekstovnih datotekah množice **B** pripravi iskalne nize. Ker so iskalni kriteriji v *Web of Science* precej redki in togi, ne moremo pripraviti striktnih iskalnih nizov. Aplikacija nam pripravi kar se da stroge iskalne nize oblike: `AU=<Priimek> <Prva črka imena>* AND PY=<Leto publikacije>`. Ker so takšni iskalni nizi preveč splošni, z njimi najdemo tudi opise del, ki niso citirana iz **B**. Množico dobljenih del moramo zato še prečistiti, kar je najlažje storiti programsko. Preostale opise izvozimo v tekstovne datoteke. Povzetek opisanih množic je grafično prikazan na sliki A.1. Množica **D** vsebuje dela, ki so citirana iz katerekoli izmed mno-



Slika A.1

Množice opisov del (**B** – "bazna" množica del, **U** – *citation report* – dela, ki citirajo **B**, t.i. "uporabniki", **C** – iz **B** citirana dela, **D** – iz **B**, **U** in **C** citirana dela, ki niso prisotna v **B**, **U** ali **C**. **E** – podmnožica **D**. Dela v **E** so citirana iz **U**, njihove opise je moč najti v *Web of Science*, a jih nimamo). Pri množicah navajamo tudi značke `*T`, ki se jih dodeli posameznim zapisom v izvornih tekstovnih datotekah na podlagi katerih *Wos2Pajek* razloči za katero množico gre.

žic **B**, **U** ali **C**, vendar zanje ne najdemo polnih opisov, le kratka imena del. Povedano drugače: *Web of Science* del iz **D** ne vsebuje ali pa jih zaradi prevelike številčnosti delno nismo izvozili. V naslednjem razdelku A.1.1 opišemo še množico **E** sorodnih del.

Kratka imena del, ki jih v disertaciji uporabljamo, izvirajo iz orodja *Wos2Pajek* [117]. Njihova struktura je sledeča: `<prvih 8 črk priimka prvega avtorja>_<prva črka imena prvega avtorja>(<leto publikacije>)<zvezek>:<prva stran>`. Z uporabo *Wos2Pajek* iz tekstovnih datotek in na podlagi oznak zapisov pripadnosti množicam **B**, **U** in **C**, ki jih dodamo v tekstovne datoteke, lahko konstruiramo nabor bibliografskih konstruktov. Prikazani so v tabeli A.1.

A.1.1 Množica sorodnih del

Zanimiva množica del **E**, kot jo označujemo, je množica sorodnih del za katere opisi v *Web of Science* obstajajo in ki so citirana iz **U**, a ne pripadajo nobeni množici izmed množic **B** ali **C**. Metoda za pridobivanje del množice **E** je enaka metodi za zbiranje množice del **C**. Namesto, da iskalne nize pripravimo na podlagi množice **B**, jih moramo pripraviti na podlagi množice **U**. Ker je množica **U** navadno bistveno večja od množice **B**, bi morali za postopek prenosa opisov del opraviti sorazmerno več dela. Kratka imena del množice **E** nam *Wos2Pajek* pripravi iz citatov v opisih del množice **U**.

Tabela A.1

Bibliografski konstrukti, ki jih ustvari *Wos2Pajek*.

Struktura	Datoteka	Opis
Cite	*Cite.net*	Usmerjeno omrežje del. Prisotna je usmerjena povezava (x, y), če in samo če je delo y citirano v delu x.
WA	*WA.net*	Dvodelno omrežje soavtorstev. Več o dvodelnih omrežjih je zapisano v razdelku 1.4.2. Vsako delo z opisom v *Web of Science* je z usmerjeno povezavo povezano z vsakim od svojih avtorjev. Če polnega opisa v *Web of Science* ni, je prisotna samo povezava na prvega avtorja. Nekatera dela nimajo avtorjev.
WK	*WK.net*	Dvodelno omrežje ključnih besed. Dela so z usmerjenimi povezavami povezana z njihovimi ključnimi besedami. Le dela, ki so v *Web of Science* imajo povezave. Ključne besede izvirajo iz naslovov, povzetkov in iz dejanskih seznamov ključnih besed v delih.
WJ	*WJ.net*	Dvodelno omrežje revij. Povezave povezujejo dela in njihove revije, kjer so bila objavljena.
Year	*Year.clu*	Razbitje del po njihovem letu publikacije.
DC	*DC.clu*	Razbitje del glede na pripadnost množicam **B**, **U**, **C** in **D** v obliki celoštevilskih vrednosti. Natančnejši opis je v razdelku A.1.2.
NP	*NP.vec*	Vektor števil strani za posamezno delo ali 0, če opis dela v *Web of Science* ni na voljo.
Clean source file	*clean.txt*	Združena datoteka opisov del iz posameznih izvornih tekstovnih datotek. Duplikati del in opisi z napakami so tu že odstranjeni.

A.1.2 Priprava razbitja po vrstah del z orodjem Wos2Pajek

Pred izvedbo orodja *Wos2Pajek* moramo zapise del v tekstovnih datotekah opremiti z značkami oblike `*T`x, kjer x označuje številko množice. To je predpogoj, da lahko z *Wos2Pajek* ločimo posamezne množice. Posledično se ustvari razbitje *DC*, ki za vsako delo določa, kateri skupini pripada. Opise množice **B** označimo z značko `*T1`, tiste iz **U** z `*T2` in tiste iz **C** z `*T3`. Skupine razbitja *DC* odražajo vrednosti x, ki smo jih podali v oznakah `*T`x. V razbitju del *DC* se opise (kratke opise) le citiranih del dodeli skupini $0 = 000_2$. To so dela za katera v vhodnih tekstovnih datotekah nimamo polnih opisov, **D**. Opise del, ki pripadajo množici **B** in hkrati ne pripadajo množicama **U** in **C** se dodeli skupini $1 = 001_2$. Dela iz množice **U**, ki hkrati ne pripadajo nobeni izmed množic **B** ali **C** se dodeli skupini $2 = 010_2$. Tista, ki pripadajo množicama **B** in **U**, ne pa tudi množici **C** se dodeli skupini $3 = 011_2$ itd. Številke skupin na desni strani enačajev so predstavljene dvojiško. Položaj posamezne enice v dvojiških številkah določa vrednost x v značkah, začenši z 1 na skrajno desnem mestu. Vse možne vrednosti – skupin razbitja *DC* se dodeli množicam, kot je prikazano tudi na sliki A.1. Zaradi duplikatov in napak v izvornih tekstovnih datotekah je velikost dobljenih množic običajno nekoliko manjša kot velikost originalnih množic opisov del v datotekah.

A.1.3 Težave z imeni avtorjev pri storitvi Web of Science *in obdelavi z* Wos2Pajek

Poleg težav z razlikovanjem soimenskih avtorjev v bibliografiji, v *Web of Science* srečamo tudi povsem enako zapisana imena različnoimenskih avtorjev. V *Web of Science* se namreč uporablja okrajšave lastnih imen avtorjev, posledično pa ista okrajšava lahko predstavlja več različnih avtorjev. V bibliografskih podatkih o topoloških indeksih o katerih pišemo v poglavju 2, se na primer avtorja *Avat Arman Taherpour* in *Arezou Taherpour* v *Web of Science* pojavljata kot `TAHERPOUR A`, *Luo-Nan Xu* in *Ling Xu* pa z `XU L`, *Shobha Joshi*, *Sheela Joshi* in *S. K. Joshi* z `JOSHI S`, itd. Gre za znano težavo v *Web of Science*. Večina lastnih imen avtorjev je v *Web of Science* okrajšanih na prvo črko, ni pa tako v vseh primerih. V slednjih redkejših primerih (in nekaterih drugih) lahko težava nastopi šele pri uporabi *Wos2Pajek*, ki od priimka upošteva največ 8 črk in izključno prvo črko lastnega imena. Na ta način se lahko vnese še nekaj dodatnih dvoumnosti. V praksi se izkaže, da je takšnih dodatnih primerov malo. V večini primerov so imena avtorjev v *Web of Science* že okrajšana na prvo črko, priimkov z več kot 8 črkami pa tudi ni zelo veliko. V tabelah 2.3 in 2.7 je *Avat Arman Taherpour* prevladujoči avtor, vendar je zaradi zgornjega zapisan v oklepajih. Isto opažanje velja za avtorja *Luo-Nan Xu* v tabelah 2.3, 2.6 in v tabeli 2.11 ter za avtorja *Zhang Heping* v tabeli 2.6, itd.

Po drugi strani v *Web of Science* najdemo primere zapisov enega avtorja na več načinov. V bibliografskih podatkih o topoloških indeksih je avtor *R. Carbó* zapisan na tri različne načine: kot *R. Carbo*, *R. Carbo-Dorca* in kot *R. Carbodorca*. V konkretnem primeru izpostavljenega avtorja gre verjetno za spremembo priimka, hkrati pa še za anomalijo v podatkih. Gre za obratni primer situacije, ki jo opisujemo v prejšnjem odstavku. Za enega avtorja obstaja več različnih opisov. V konkretnem primeru opisanega avtorja smo del podatkov, kjer avtor nastopa, ročno popravili. Avtor *R. Carbo* namreč nastopa v majhnem otoku, ki smo ga analizirali (2.4.4). Vseh podatkov je navadno bistveno preveč, da bi jih preverili ali celo popravili. Izjeme druge vrste (razen pri spremembah priimka) so redkejše, sploh, če upoštevamo dejstvo, da so na Kitajskem nekateri priimki izredno pogosti, je izjeme druge vrste težje najti. Odkrijemo jih lahko na primer na podlagi dejstva, da isti avtor velikokrat objavlja s podobnimi soavtorji. Obe težavi bi univerzalno lahko rešili na primer z vpeljavo globalne raziskovalne številke za vsakega raziskovalca.

A.2 Omrežni datotečni formati

V tem razdelku se osredotočimo na to, kako v obliki datotek na računalniški pomnilni medij oz. napravo shraniti omrežne podatke. Obstaja mnogo uveljavljenih predpisov organizacije omrežnih podatkov. Imenujemo jih omrežni datotečni formati – ODF-i. Datotečni format je predpis organizacije podatkov znotraj datoteke. ODF-i in datotečni formati v splošnem odražajo različne lastnosti, ki temeljijo na njihovi naravi uporabe kot na primer: berljivost, prenosljivost, splošnost, kompaktnost v smislu prostora, hitrost kodiranja in dekodiranja, branja in pisanja itd.

Predstavili bomo glavne delitve ODF-ov in prikazali nekaj glavnih vidikov na katere moramo biti pozorni, ko se z njimi ukvarjamo. V smislu različnih vidikov bomo primerjali nekaj uveljavljenih ODF-ov. Nekaj pomembnih ODF-ov si bomo ogledali natančneje, podali nekaj nasvetov, kako ODF-e med seboj pretvarjati in predstavili izčrpen seznam možnih pretvorb med njimi. Navedli bomo tudi nabor orodij za analizo omrežij ter tem sorodnih in njihovo podporo za uveljavljene ODF-e.

A.2.1 Vrste omrežnih datotečnih formatov

Izpostavimo dva vidika ODF-ov. Prvi sledi potrebam uporabnika pri konstrukciji omrežij (1.4.2) ter vzdrževanju in drugi zahtevam v smislu potreb po računalniških virih. Oba vidika se v precejšnji meri prepletata.

Analiza omrežja se običajno prične z zbiranjem podatkov in konstrukcijo omrežij. Preden shranimo omrežne podatke, se moramo odločiti za primeren ODF. Če obseg podatkov ni prevelik, lahko omrežje sestavimo ročno, bodisi v tekstovnem urejevalniku ali pa z uporabo programov za delo s tabelami, kot na primer *Microsoft Excel*. Če je obseg podatkov večji, vendar podatke zbiramo ročno, jih lahko organiziramo v podatkovno bazo. V ostalih primerih, ko so podatki že na voljo v digitalni obliki, se lahko zatečemo k avtomatiziranim postopkom. Podatke lahko s pretvornikom direktno pretvorimo v ustrezno obliko, lahko jih obdelamo s programi za zbiranje in konstrukcijo omrežij, ali pa se odločimo za razvoj aplikacije po meri. Primer zbiranja podatkov so: zbiranje z anketami, zbiranje podatkov na spletu, podatkovno rudarjenje, idr. Klasičen primer zbiranja bibliografskih podatkov in konstrukcijo omrežij iz primerne spletne storitve smo opisali v razdelku A.1.

V računalniškem pomenu predstavitev omrežja zavisi od velikosti omrežja, predvidene uporabe, lokacije hranjena, idr. Odvisna je od gostote omrežja, lastnosti elementov

omrežja, možnosti shranjevanja alternativnih podatkov in želenih vizualnih lastnosti omrežja itd. Na primer poraba prostora in hitrost branja oz. pisanja na medij v danem formatu sta pomembna kadar delamo z velikimi omrežji. Pogosto gre za kompromis med prijaznostjo uporabniku in potrebami po računalniških virih. Medtem, ko so optimizirani formati primernejši za branje in pisanje na pomnilne medije ter zahtevajo manj prostora in procesiranja, jih je težje urejati in so za uporabnika manj berljivi, razen seveda s posebnimi orodji.

Razlikujemo tri večje skupine ODF-ov. Tekstovni formati so usmerjeni k berljivosti in prenosljivosti. Čeprav so sami zase zaključena skupina, mednje štejemo tudi formate XML (*Extensible Markup Language*) [118]. Lastnosti formatov XML so predvsem prenosljivost, prilagodljivost in skalabilnost. V drugo skupino razvrstimo dvojiške formate. So učinkoviti, kompaktni in optimizirani za shranjevanje na računalniku. Nasprotno tekstovnim so uporabniku skoraj univerzalno neberljivi. Z njimi lahko rokujemo le posredno z uporabo posebnih orodij. Poznamo še nekaj redkih, stisnjenih formatov, ki so tehnično tudi dvojiški. Poleg navedenih bi med formate lahko šteli tudi predstavitve za shranjevanje omrežij v podatkovni bazi in morda formate za odložišče za prenos manjših količin omrežnih podatkov med aplikacijami ali med njihovimi okni. Srečujemo tudi posebne formate, določene z aplikacijskimi programskimi vmesniki spletnih podatkovnih baz. Včasih jih lahko uporabimo za črpanje omrežnih podatkov.

Tekstovni omrežni datotečni formati

Tekstovni datotečni formati so predstavljeni s tekstom, primernim za direktno branje uporabniku. Dokler je struktura formata znana, ne potrebujemo posebnih programov za delo z njimi. Vse kar potrebujemo je standardni tekstovni urejevalnik. Iz praktičnih razlogov so takšni ODF-i najbolj priljubljeni. Omenili smo podskupino formatov XML. Poleg prednosti tekstovnih formatov v splošnem, so formati XML bolj prilagodljivi za integracijo novih lastnosti. Ker sledijo standardu XML, je zanje enostavno razvijati programe. Velikokrat jih uporabljamo na mestu posrednih formatov med različnimi programi. Splošna slaba lastnost tekstovnih formatov je, da so podvrženi napakam, ki jih lahko naredimo pri ročnem urejanju. Glede procesiranja pri kodiranju in dekodiranju v računalniki pomnilnik utegnejo biti potratni in zasedejo veliko prostora.

Dvojiški omrežni datotečni formati

Glavnina programske opreme za analizo omrežij podpira delo z vsaj enim tekstovnim formatom, precej pa tudi z vsaj enim dvojiškim formatom. Dvojiški formati so običajno specifični in lastni proizvajalcu posamezne programske opreme. Cilj dvojiških formatov je navadno minimizirati potrebe po računalniških virih kot sta velikost in procesiranje shranjenih podatkov. V praksi se to izrazi v hitrih operacijah branja in pisanja. Velikost datoteke v potratnem formatu XML je lahko hitro desetkrat večja v primerjavi z datoteko v premišljeni dvojiški obliki identičnih podatkov. Nekatera orodja za analizo omrežij vpeljujejo kompromis, tako da datoteke XML berejo direktno iz stisnjenih datotek. Na račun povečanja procesiranja je tako možno reducirati velikost ODF-ov XML (in drugih ne-stisnjenih ODF-ov), dokler problem pomanjkanja prostora ni prevelik. Praktično rešitev tako pri enormnih omrežjih predstavljajo le dvojiški formati, in sicer tako za shranjevanje, ki danes lahko obsega terabajte prostora, kot tudi v smislu obsega procesiranja, ki je v primeru dvojiških formatov skoraj univerzalno manjše. Enormna omrežja po definiciji zasedajo toliko prostora, da jih v celoti ne moremo shraniti v računalnikov delovni pomnilnik. Dvojiški formati so navadno tesno povezani z obliko predstavitve podatkov v delovnem pomnilniku. Pri prenosu vsebine datoteke v delovni pomnilnik ni potrebnega veliko dodatnega procesiranja za preoblikovanje oz. razčlenjevanje podatkov, če sploh. Slaba stran dvojiških formatov je nedvomno v omejeni berljivosti za uporabnika. Brez uporabe namenskih programov za delo z njimi, si z njimi ne moremo veliko pomagati. Za delo so nepraktični in brez natančne specifikacije redko razumljivi. Težko jih je prilagajati in dopolnjevati z novimi lastnostmi. Dodajanje novih lastnosti in pri tem ohranjanje združljivosti za nazaj, lahko pri dvojiških formatih predstavlja velik izziv. Dvojiški formati navadno služijo za shranjevanje programsko specifičnih in optimiziranih vmesnih rezultatov. Vhodni in izhodni formati so navadno tekstovni.

Predstavitve omrežij v podatkovni bazi

Nekatera orodja za analizo omrežij omogočajo uvoz in izvoz podatkov preko povezave na podatkovno bazo. Slednje je posebej značilno za poslovna okolja, kjer več uporabnikov dela z istimi podatki, ki so dostopni preko skupne podatkovne baze. Tudi enormna omrežja so lahko shranjena na takšen način. Kot format lahko razumemo tudi shemo podatkovne baze, ki opisuje strukturo v njej shranjenih podatkov, s katerimi upravlja

strežnik in uporabnikom hkrati nudi dostop do njih. Primer je *Advanced Visualization and Analysis* [119]. Čeprav se sheme podatkovnih baz in podatkovne baze same po sebi precej razlikujejo, bomo o teh formatih govorili preprosto kot o formatih za podatkovne baze.

Aplikacijski programski vmesniki do spletnih storitev

Nekatere spletne storitve, kot so *Amazon*, *YouTube*, *Google Data* in druge je možno uporabiti kot vir podatkov, tudi omrežnih. Uporabnikom namreč omogočajo dostop do podatkov preko posebnih aplikacijskih programskih vmesnikov. Takšen vmesnik omogoča programski dostop do podatkov spletne storitve. Navadno je realiziran v obliki omrežne storitve – *web service*, opisane na primer z opisno shemo WSDL (*web service description language*). Iz opisa lahko razberemo format, ki se uporablja za prenos podatkov med strežnikom in uporabniško aplikacijo. Osnovni podatki so običajno predstavljeni v XML ali alternativno v JSON (objektni notaciji *JavaScript*), njihova notranja struktura pa zavisi od programske izvedbe aplikacijskega programskega vmesnika. Oba formata se pogosto uporabljata za izmenjavo podatkov v paketih, primernih za prenos preko spleta. Na podlagi opisnih shem omrežnih storitev je pogosto možno avtomatsko ustvariti kodo, ki na uporabnikovi strani podatke, ki jih strežnik pošlje, smiselno razčleni. Uporabnik tako do podatkov dostopa preprosto s klici avtomatično ustvarjenih podprogramov oz. funkcij, pri tem pa se mu ni treba ukvarjati s podrobnostmi formata, v katerem se podatki prenašajo.

A.2.2 Vidiki omrežnih datotečnih formatov

Predstavitev grafovskih podatkov v datotekah

Če želimo omrežje predstaviti v datoteki, si moramo najprej izbrati primerno predstavitev množic **V** in **L** iz definicije omrežja (1.4.2). Množica **V** je lahko navedena eksplicitno ali pa se implicitno sestavi iz specifikacije povezav – razpršena predstavitev. Povezave (množica **L**) se navadno podaja v obliki parov vozlišč. Ko se definira povezavo, se sproti definira njeni vozlišči. Težava implicitne definicije nastopi v primeru izoliranih vozlišč, saj le-ta ne pripadajo nobeni povezavi. Če vozlišča podajamo na primer z zaporednimi številkami, za njihovo definicijo zadošča že, da navedemo število vozlišč. Izoliranih vozlišč v tem primeru ne potrebujemo posebej navajati. (Takšno predstavitev lahko razumemo tudi kot eksplicitno.) Prednost eksplicitne pred implicitno navedbo je v tem, da je množico vozlišč možno uporabiti v več množicah povezav

(gre torej za več omrežij, definiranih na isti množici vozlišč). V tem primeru ne rabimo vsakič znova definirati vozlišč, kar je še posebej priročno, če imajo vozlišča vgrajene dodatne lastnosti. Vgrajevanje lastnosti je natančneje opisano v odstavku A.2.4.

Vozlišča predstavimo z unikatnimi ključi (identifikatorji). Identifikatorji so lahko umetni kot na primer zaporedna števila, lahko pa so naravni v obliki imen vozlišč ali pa vrednosti ene od njihovih lastnosti, dokler le-ta enolično določa vsako posamezno vozlišče. Običajni pristop za podajanje množice povezav je z zaporedjem parov identifikatorjev njihovih krajišč. Gre za pogost pristop pri tekstovnih formatih, še posebej pri formatih XML kjer so vozlišča predstavljena z elementi XML, ki vsebujejo identifikatorje. Če omrežje vsebuje le neusmerjene odnosno usmerjene povezave, potem vrstni red navajanja krajišč pri povezavah ni oz. je pomemben. V primeru, da omrežje vsebuje usmerjene in neusmerjene povezave, so lahko v datoteki povezave navedene v dveh ločenih delih ali pa v enem samem delu, kjer je njihova usmerjenost podana med lastnostmi. Prvi pristop je značilen za tekstovne formate in manj v formatih XML, drugi pa je pogost predvsem v formatih XML. Poleg lastnosti o usmerjenosti imajo lahko povezave lastnost, ki opredeljuje njihovo moč. V datoteki lahko povezave navajamo še na dva načina. Lahko jih podamo kot zaporedje identifikatorjev vozlišč, ki v omrežju leže na poti ali alternativno, s t.i. zvezdnimi strukturami kjer se najprej navede identifikator skupnega osrednjega vozlišča, sledi pa mu seznam identifikatorjev vozlišč sosednjih vozlišč. Prvi način je primeren za shranjevanje omrežij, ki vsebujejo veliko dolgih in ločenih poti, drugi pa za gostejša omrežja, ki vsebujejo veliko zvezd, t.j. vozlišč z veliko sosedi. Oba načina se v najslabšem primeru izrodita v standardni način. Načina nista pogosta in v formatih XML ju praktično ne zasledimo, saj prostorske omejitve v formatih XML običajno niso omejujoče. Z vsemi tremi načini se vozlišča navaja implicitno. Vsakič, ko se definira povezavo, sta, v kolikor ju še nismo srečali, implicitno definirani tudi njeni vozlišči krajišči.

Često se omrežja podaja s sosednostno matriko. Primerna je za opise relativno manjših ali pa zelo gostih omrežij. Pri gostih omrežjih je matrična predstavitev prostorsko najbolj učinkovita. Pri matriki gre v bistvu za seznam n seznamov z n vrednostmi. Vrednost na j-tem mestu v i-tem seznamu predstavlja utež povezave (usmerjene povezave) $i \to j$. Če omrežje sestoji samo iz neusmerjenih povezav, se lahko uporabi tudi trikotno matrično predstavitev, saj je celotna matrika simetrična. Nekateri ODF-i nam dovoljujejo, da izpustimo tudi diagonalo matrike, ki predstavlja zanke v omrežju. Shranjevanje z matrično predstavitvijo je mogoče v večini tekstovnih in dvojiških

ODF-ih.

Zasledimo tudi evolucijsko predstavitev omrežij, pri katerih se omrežja opisuje s tokom strukturnih dogodkov. Dogodki so dodajanje in brisanje elementov omrežja, sprememba vrednosti lastnosti elementov omrežja in drugi. Pristop je uporaben kadar želimo slediti dinamiki omrežij. Primer takšnega formata je *GraphStream* format [120]. Tok, ki vsebuje le zaporedje dodajanj vozlišč in povezav, kjer so vsi vrstni redi navajanja obeh operacij enakovredni, je tehnično gledano običajni postopek definicije elementov omrežja.

Lastnosti vozlišč in povezav omrežja

Informacije o lastnostih elementov omrežja so lahko pripeta pripadajočim elementom ali pa so shranjena na posebnih, za to namenjenih mestih, na primer v tabelah ali seznamih.

Shranjevanje razvrstitev, skupin, razbitij, preureditev in hierarhij

Razvrstitve, skupine, razbitja, preureditve in hierarhije (glejte definicije v uvodu 1.4.2, 1.4.2) lahko shranimo v obliki celoštevilskih vektorjev. Medtem, ko vrednosti posameznih skupin v razvrstitvi nimajo nujno posebnega pomena, vrednosti elementov razbitja podajajo pripadnost posameznim skupinam. V razbitju $p[v] = i$ – vozlišče v pripada skupini $\mathbf{C}_i$. V preureditvi $p[v] = i$ – vozlišče v nastopa na i-tem mestu. V hierarhiji vsak element kaže na svojega starša. Koren hierarhije je navadno zaznamovan z negativno vrednostjo ali vrednostjo 0, lahko pa tudi s kako drugo vnaprej določeno vrednostjo.

Shranjevanje časovnih omrežij

Časovna ali dinamična omrežja [3] so omrežja, ki imajo prisotno dodatno razsežnost, čas. Definirana so v razdelku 1.4.2. Njihova struktura in vrednosti lastnosti se pri njih skozi čas lahko spreminjajo. Z njimi lahko prikažemo razvoj omrežja.

Shranjevanje dvodelnih (bipartitnih) omrežij

Dvodelna omrežja [3] so poseben primer omrežij. Definiramo jih v razdelku 1.4.2. Pri shranjevanju običajno zadošča že en sam parameter, ki množico vozlišč loči na dve množici. V nekaterih omrežnih datotečnih formatih opise dvodelnih omrežij zasledimo tudi v obliki kvadratne matrike $\mathbf{A} = [a_{uv}]_{\mathbf{V}_1 \times \mathbf{V}_2}$:

$$a_{uv} = \begin{cases} w_{uv} & (u,v) \in \mathbf{L} \\ 0 & (u,v) \notin \mathbf{L} \end{cases}, \tag{A.1}$$

kjer w_{uv} predstavlja utež povezave (u,v).

Poznamo še n-delna omrežja, a se v praksi in ODF-ih redko uporabljajo. n-delno omrežje lahko določimo z uporabo dodatnih razbitij vozlišč – razbitja na dele (*mode partition*).

Shranjevanje več omrežij oz. več-relacijskih omrežij

Večkratna oz. več-relacijska omrežja [35] so omrežja, ki imajo eno množico vozlišč in različne množice povezav na njih $\mathbf{L} = (\mathbf{L}_1, \mathbf{L}_2, .., \mathbf{L}_r)$. Primeri takšnih omrežij so na primer povezave transportnega sistema v mestu (postaje, linije), semantična omrežja (besede, semantične relacije: sinonim, antonim, hiponim, meronim) itd.

Shranjevanje hipergrafov

Hipergraf [4] je posplošitev grafa. V njem lahko vsaka povezava povezuje poljubno število vozlišč. Formalna definicija je podana v razdelku 1.4.2. V nekaterih formatih je možno shraniti ne le hipergrafov, vendar tudi posplošena omrežja, ki na njih temeljijo. Omenimo, da je hipergrafe možno predstaviti z incidenčno matriko, ki je v bistvu dvodelno omrežje hiperpovezav × vozlišč. Takšne oblike sicer ne pojmujemo kot pravi ODF.

Shranjevanje komentarjev

Pri shranjevanju omrežij imajo pomembno vlogo komentarji elementov omrežja, ki jih prav tako želimo shraniti z omrežjem. V dvojiških formatih so lahko vgrajeni, a jih ne moremo brati brez namenske programske opreme. Pri formatih XML je podpora za komentarje implicitno vgrajena, zato jih podpirajo tudi vsi ODF, ki temeljijo na XML. Pojavljajo se lahko na poljubnih mestih znotraj datoteke XML. Komentarje lahko shranjujemo tudi v večini ostalih tekstovnih formatov, a se njihovo mesto in način komentiranja od formata do formata razlikuje. Medtem ko bolj kompaktni tekstovni formati ne omogočajo shranjevanja komentarjev, ali pa omogočajo le omejeno komentiranje (na primer brez prelomov vrstic) in na posebej določenih mestih, jih obsežnejši formati omogočajo na več mestih in v manj omejeni obliki.

Več struktur znotraj posamezne združene projektne datoteke

Omrežje opisujemo z njegovo grafovkso strukturo in raznimi dodatnimi podatkovnimi elementi v obliki tabel lastnosti, razbitij, skupin, hierarhij itd. Pri nekaterih formatih se vsako strukturo shranjuje v samostojno datoteko. Takšen pristop je hiter in omogoča, da imamo shranjenih več različic posameznih struktur, s čimer prihranimo nekaj prostora. Pristop je priročen pri delu na konkretni analizi, vendar pozneje brez dobre organizacije lahko hitro pozabimo, čemu je posamezna datoteka namenjena. Pri takšnih formatih mora uporabnik sam skrbeti za organizacijo struktur. Drugi formati omogočajo, da se v eno samo datoteko shrani več omrežnih struktur. Pristop je boljši, kadar želimo rezultate analize ohranjati kot celoto. Drži pa, da pri shranjevanju na takšen način računalnik potrebuje nekoliko več časa. Formati, ki temeljijo na XML so zaradi svoje drevesne strukture že vnaprej primerni za družno shranjevanje večih struktur. Na primer dodatni nivo (ovojnica) navadno zadošča, da se vsebino več datotek XML združi v eno samo datoteko. Pri tem je bistveno, da za ohranjanje združljivosti programske opreme, ki bo dodatno sposobna rokovati z združenimi datotekami, ne potrebujemo veliko dela. Alternativni in morda najboljši pristop združuje oba načina. Ohranja prednosti shranjevanja v posameznih datotekah in prednosti pri združenem načinu. Posameznim datotekam z omrežnimi strukturami se doda projektno datoteko, ki vključuje opise in relacije med strukturami. Uporabnik pri tem lahko poljubno dostopa do posameznih datotek, programska oprema za analizo omrežij pa poskrbi za red in konsistenco. Evidenca se vodi v projektni datoteki.

Grafični prikazi – pogledi in slogi

Pri risanju omrežja navadno potrebujemo dodatne informacije, t.j. grafične in druge lastnosti elementov omrežja. Gre za tako imenovane grafične prikaze oz. poglede (*layouts*) [2]. V številnih formatih lahko shranimo vsaj en grafični prikaz oz. pogled omrežja. Običajno gre za razporeditve elementov omrežja v obliki koordinat vozlišč, barv vozlišč in povezav, oblike vozlišč itd. Več grafičnih prikazov oz. pogledov ali v omejeni različici t.i. slog nasprotno lahko shranimo le v redkih formatih. Na primer *GraphML* [121] je format, ki omogoča shranjevanje različnih pogledov na isto omrežje. Slog je tehnično gledano predpis, ki v odvisnosti od parametrov na generičen način predpisuje poljubno mnogo pogledov. Lahko je podan tudi kot program. Medtem, ko noben uveljavljen format ne omogoča takšnega načina definicije sloga, je večina for-

matov XML dovolj prilagodljivih, da bi ga vanje lahko vgradili. Obstoječe programske opreme pri tem nebi bilo potrebno spreminjati.

Sestavljanje omrežja

Omrežja je možno ustvariti z združevanjem več manjših podomrežij, na katera lahko gledamo kot na osnovne gradnike – opeke. Na omrežju lahko tudi izvajamo predpisane transformacije, ki dele omrežja nadomestijo z drugimi deli (induktivne skupine grafov). Vpeljemo lahko tudi predpis za razdiranje omrežja. Informacije o sestavljanju omrežja lahko služijo v pomoč algoritmom, ki za določene probleme tako delujejo hitreje. Poskus razvoja postopka za opisovanje omrežij s sestavljanjem je nastal pod imenom *NetML* [122].

Shranjevanje metapodatkov

Metapodatki so podatki o podatkih. Namenjeni so opisu osnovnih podatkov. Pri organizaciji datotek so v pomoč uporabniku, lahko pa se jih uporabi pri optimizaciji toka analize. Primer metapodatkov je podatek o avtorju omrežnih podatkov, čas zajema, čas nastanka omrežja, vir podatkov, namen izgradnje, avtorske pravice... Metapodatke se velikokrat shranjuje kar v obliki komentarjev. Pri formatih XML in nekaterih drugih formatih najdemo zanje posebej določena mesta.

Shranjevanje splošnih lastnosti omrežij

Nekateri formati omogočajo uporabniku, da shrani splošna znanja oz. že izračunane lastnosti omrežij, ki se v datoteki nahajajo. Takšni podatki so na primer informacija o tem ali je omrežje planarno, brezlestvično, maksimalna stopnja vozlišč v omrežju, število povezanih komponent... Ideja je v shranjevanju rezultatov zahtevnih algoritmov, ki ne zasedejo veliko prostora. V nadaljevanju analize lahko takšne rezultate uporabijo drugi algoritmi. Ob prisotnosti teh informacij lahko tečejo bistveno hitreje. Pri iskanju v velikih množicah omrežij jih lahko uporabimo tudi kot kazala.

A.2.3 Vidiki nekaterih popularnih ODF-ov

V tabeli na sliki A.2 prikazujemo pregled osnovnih lastnosti (vidikov) nekaterih popularnih ODF-ov. Razlage posameznih lastnosti so sledeče: *Več struktur v posamezni datoteki* opredeljuje možnost, da se v datoteko lahko shrani več kot le eno omrežje. Pod *povezave na zunanje vire ali URL-ji* mislimo na možnost, da se v datoteko lahko vključi

Lastnosti formatov	Format	TXT, CSV	Graph Stream	GraphViz	Dynet Markup Language	Guess	Graph Exchange	Graph Modelling Language	Graph Markup Language	Graph Exchange XML	Pajek PAJ	SoNIA	Tulip	UCINET DL	Extensible Graph Markup, Modeling
Več struktur v posamezni datoteki				●	●		●	●	●	●	●		●	●	●
Povezave na zunanje vire (*URL*)					○		○		●					●	
Matrična predstavitev		●	○	○			○	○	○	●	●			●	
Podpora za strukturne dogodke			●				●				●	●			
Vektorji		●	○	○	○		○	●	●	●	●		●		●
Splošne lastnosti (atributi)		●	●	●	●	●	●	●	●	●	○	●	○		●
Razbitja, permutacije		○	○	○	●		○	●	●	●	●		●		●
Predstavitev hierarhij				●	●		●	●	●	●	●		●		●
Časovna (dinamična) omrežja			●		●		●	●	●	●	●	●	○		●
Lastnosti (atributi) v času					●		●		○	○		●	○		
Dvodelna omrežja		●	○	○	○		○	○	○	○	●	○	○	●	○
Več-relacijska omrežja					●			●		●	●			●	●
Shranjevanje relacij		○	●	○	○		○	○	○	●	●	○	○		○
Predstavitev hipergrafov				●	●			●	●	●					●
Podpora za komentarje			●	●	●		●	●	●	●	●	●			●
Podpora za poglede in stile				●	●	●	●	●			●	●	●		
Pod-komponente		○	○	●	●		●	●	●	●	●	●	●		●
Imenski prostori					●		●		●	●					●
Podpora za *Unicode*			●	●	●	●	●		●	●	●			●	●
Prijaznost uporabniku		●●●	●●	●●	●●	●●●	●●	●●●	●●	●	●●	●●	●●	●●	●●
Varčnost s prostorom		●●	●●	●	●	●●	●	●●	●	●	●●●	●●●	●●●	●●●	●

○ Je mogoče razširiti ● Da / Zadostno ●● Dobro ●●● Odlično

Slika A.2

Različni vidiki nekaterih popularnih ODF-ov.

podatke, ki se nahajajo na drugih virih – datotekah na lokalnem pomnilniku oz. na spletu. *Matrična predstavitev* določa možnost, da se omrežje v datoteki lahko predstaviti matrično. V nekaterih formatih je omrežja možno predstaviti z uporabo *strukturnih dogodkov*. Gre za inkrementalni način gradnje omrežja z dogodki sprememb. Medtem ko so *vektorji*, *razbitja*, *preureditve* in *hierarhije* samostojne strukture, so *splošne lastnosti* vezane na posamezne elemente omrežja. V nekaterih formatih jih je možno shraniti. Pri tem se je potrebno zavedati, da morajo vektorji, razbitja, preureditve in hierarhije po velikosti praviloma ustrezati omrežju. Z *lastnostmi v času* označujemo možnost shranjevanja vrednosti, ki se skozi čas lahko poljubno spreminjajo. Medtem, ko kategorija *več-relacijskih omrežij* opredeljuje formate v katerih lahko shranimo več množic povezav, *shranjevanje relacij* pomeni le, da je na eni množici povezav možno definirati lastnosti, ki podajajo, kateri relaciji posamezna povezava pripada. Lastnosti za relacije se navadno podajajo enako kot druge lastnosti povezav. *Pod-komponente* opredeljujejo možnost opisa podomrežij v izbranem formatu, in sicer z razbitjem ali kakšnim drugim specifičnim načinom.

Podpora za Unicode je dobrodošla, kadar želimo za opis lastnosti omrežij uporabiti

posebne znake. Večina tekstovnih formatov podpira predstavitev znakov UTF-8, ki je eden izmed kodiranj *Unicode*. Datoteke XML že v osnovi podpirajo *Unicode* kot tudi druga kodiranja. Novejše različice dvojiških formatov navadno omogočajo shranjevanje v *Unicode* kodiranju.

Imenski prostori opredeljujejo možnost, da se posameznim strukturam lahko določi imenski prostor v katerega naj bodo umeščene. S tem se poskrbi za selektivno združevanje struktur. Z imenskimi prostori je na primer možno ločiti ali vozlišče z enakim imenom v dveh različnih omrežjih predstavlja isto vozlišče. *Prijaznost uporabniku* predstavlja "mero" intuitivnosti oz. enostavnosti formata za razumevanje nekomu, ki se z njim še ni srečal, koliko dela bo potrebno vložiti za konstrukcijo zmerno velikega omrežja, kako mučno je branje podatkov iz datoteke, ko omrežje na primer rišemo na papir, ali recimo če lahko imena elementov omrežja takoj uporabimo za razliko od abstraktnih ključev. *Ocena prostorske učinkovitosti* podaja učinkovitost izbranega format v primerjavi z drugimi v smislu porabe prostora za enak nabor podatkov. Če "odličen" format za hrambo nabora podatkov zavzame eno enoto prostora, lahko format v kategoriji "zadostno" zavzame tudi do deset enot prostora.

A.2.4 ODF-i, ki se v praksi veliko uporabljajo

GraphML

Eden izmed najbolj popularnih formatov za izmenjevanje omrežnih podatkov v datotekah je *GraphML* [121]. Je obširen in enostaven za uporabo. Sestavljen je iz jezikovnega jedra za opis strukturnih lastnosti grafa in prilagodljivega dela za razširitve kamor se lahko vključi specifične podatke aplikacij, ki s formatom rokujejo. Glavne lastnosti, ki jih omogoča, so podpora za usmerjene, neusmerjene in mešane grafe, hipergrafe, hierarhične grafe in predstavitve grafov, povezave na zunanje vire podatkov, predvideva mesta, kamor se lahko shrani lastnosti omrežij in posebne podatke aplikacij. Za razčlenjevanje zadoščajo preprosti razčlenjevalniki.

Za razliko od večine ostalih ODF-ov v praksi *GraphML* temelji na XML in je kot tak primeren "skupni imenovalec" za vse vrste orodij za ustvarjanje, arhiviranje in procesiranje omrežnih podatkov.

Pajek *NET*

Med popularnejšimi ODF-i je format *Pajek* NET [34]. Osnovan je na tekstovni predstavitvi. Ker je *Pajek* namenjen analizi velikih omrežij, je pri njegovem privzetem for-

matu poudarek na kompaktnosti, hkrati pa je dobro berljiv. Osnovna grafovska struktura je v formatu predstavljena s seznamom povezav, s seznamom seznamov sosedov ali pa s sosednostno matriko [123], [124]. Vozlišča/povezave imajo lahko različne lastnosti kot so ime, barva, oblika/vzorec, velikost/utež, položaj/vrednost... Da bi prihranili prostor, imajo skupine zaporedno določenih vozlišč ali povezav lahko le enkrat določene lastnosti. Format NET podpira dva načina opisov časovnih omrežij. Prvi način temelji na časovnih intervalih prisotnosti v katerih so vozlišča/povezave v omrežju aktivne; drugi pa temelji na dogodkih (`prikaži povezavo`, `dodaj vozlišče`, `spremeni lastnost vozlišča`...). Format *Pajek* NET podpira tudi več-relacijska omrežja, dvodelna omrežja, predstavitev geneologij s *p*-grafi in *Petri*-jeve mreže. Vrstica, ki se v datoteki *Pajek* NET začne s %, predstavlja komentar.

UCINET DL

UCINET DL format [125] je najbolj zastopan omrežni datotečni format pri uporabnikih orodja *UCINET*. Tehnično gledano gre pravzaprav za tri formate, ki temeljijo na tekstovnih predstavitvah.

- Matrični *full matrix* format je privzet. Datoteke v tem formatu prepoznamo po končnici DL. Vsebinsko so zelo podobne matrični predstavitvi, opisani v sestavku A.2.4.
- Seznam povezav *link (edge) list* format je prikladen za shranjevanje redkih omrežij, kjer je pogosto bolj praktično navesti le pare med seboj povezanih vozlišč. V sestavku A.2.4 je dodaten opis. V tem formatu je vsaka povezava podana kot urejen par vozlišč, kateremu lahko sledi vrednost, ki določa moč povezave.
- Seznam vozlišč *node list* format ima obliko seznama vozlišč oz. povezav, ki so določene z imenom ali številko začetnega vozlišča, kateremu sledi seznam imen oz. številk sosednjih vozlišč. Format je bolj kompakten kot seznam povezav, saj začetno vozlišče tudi v primeru, da je večim povezavam začetno, nastopa le enkrat.

Sosednostna matrika (TXT, CSV)

Eden izmed najbolj osnovnih ODF je sosednostna matrika [123], [124]. Vsaka vrstica matrike je predstavljena z zaporedjem s presledki, s tabulatorji ali s katerim drugim

ločnim znakom ločenih vrednosti. V CSV – *comma separated values* datotekah je često ločni znak vejica ali podpičje. Z dodatnimi informacijami, ki jih mora poznati uporabnik, lahko v TXT ali CSV datoteke poleg sosednostne matrike shranimo tudi izmenjujoče vrstice in/ali stolpce lastnosti povezav, dodatno pa tudi naslovne vrstice oz. stolpce (*header rows/columns*) v katerih zapišemo lastnosti vozlišč. Glavna prednost pred ostalimi formati je v preprostosti in učinkovitosti pri zapisu gostih omrežij. Za shranjevanje velikih redkih omrežij format ni primeren, saj bi datoteke zasedle preveč prostora.

Seznam povezav (TXT, CSV)

Seznam povezav je najbolj preprost ODF in "razume" ga večina aplikacij za analizo omrežij. Omrežje predstavimo z zaporedjem povezav, ki so v ločenih vrsticah podane s pari njihovih krajišč v obliki indeksov ali imen. Gre za implicitno navedbo vozlišč. Na koncu posamezne vrstice lahko v seznam dodamo še lastnosti povezav. Lastnosti vozlišč lahko podamo le v primeru, ko so vozlišča podana eksplicitno v ločenem seznamu. Uporabnik mora sam poskrbeti, da poleg datoteke posreduje informacijo, da so prisotne tudi lastnosti vozlišč. Preprostost je glavna prednost seznama povezav.

Graph Modeling Language (GML)

Večina programov za delo z grafi in omrežji uporablja lastne ODF-e za shranjevanje podatkov. Poraja se vprašanje o prenosljivosti podatkov med programi. Ko je mogoče, datoteke različnih formatov pretvorimo, v nasprotnem primeru pa naloge kot so izmenjava podatkov, prikazovanje rezultatov v zunanji aplikaciji ali primerjava učinkovitosti programske opreme na istih omrežjih, lahko hitro izpade bolj kompleksno kot bi pričakovali. Format GML [126] so razvili, da bi premostili ta problem. GML podpira dodajanje poljubnih podatkov lastnosti na grafe, vozlišča in povezave. Kot tak je izredno prilagodljiv. Oponaša lahko obliko skoraj vsakega drugega tekstovnega formata.

GraphViz (DOT)

DOT [127] je pravzaprav jezik za opisovanje grafov v tekstovni obliki. Nudi preprost način opisovanja grafov ter enostaven je za branje in razčlenjevanje. Glavni predstavitvi v DOT jeziku sta namenjeni usmerjenim in neusmerjenim grafom, ki imajo lahko poljubne lastnosti vozlišč in povezav. Poljubne lastnosti se zapiše poleg identifikatorja

posameznega vozlišča. Na primer direktno v definiciji povezave $a - b$, lahko vozlišču b določimo na primer modro barvo, kar storimo tako: $a-b[barva = modra]$. Z uporabo lastnosti je možno zapisati tudi grafični prikaz. Format omogoča eno- in več-vrstične komentarje.

V DOT jeziku obstaja edinstven način zapisa povezav, podoben običajnemu sosednostnemu seznamu [123, 124], a na bolj kompakten način. Na primer povezavi $a - b$ in $b - c$ med vozlišči a, b in c lahko podamo kot en sam niz $a - b - c$. V istem nizu se hkrati implicitno izvede še definicija vozlišč.

A.2.5 Pretvarjanje različnih formatov

Uporabnik se lahko kmalu znajde v situaciji, ko ima omrežje v formatu, ki je vsaj na videz nezdružljiv s programom za program, ki bi ga želel uporabiti. Sprva lahko pogleda v uporabniško dokumentacijo programa, če morda program podpira uvoz v danem formatu. Če format ni naveden, lahko informacije poišče na spletnih straneh programov, na forumih, preko spletnega iskalnika ali pa na primer s spremljanjem e-pošte na primernem e-poštnem seznamu. Za dani program morda obstaja primeren vtičnik ali skripta za izvedbo uvoza. Veliko programov omogoča alternativne funkcionalnosti uvoza, podporo za vtičnike ali programski vmesnik za poganjanje skript. Če rešitve ne najdemo, lahko poizkusimo s pretvarjanjem v programu znan format. Če obstaja, uporabimo ustrezen pretvornik ali če podatkov ni preveč poskusimo ročno, sicer programsko. Obstaja omejen nabor orodij za pretvarjanje, ki so prikazana na slikah A.4 in A.3 levo. Označena so s C. Uporabimo lahko tudi kak drug program za analizo omrežij, ki omogoča uvoz v danem in izvoz v želenem formatu. Dano datoteko odpremo ali uvozimo in jo shranimo ali izvozimo v želenem formatu. Kot je prikazano na sliki A.6, lahko za pretvorbo potrebujemo več korakov. V praksi je vsekakor najbolj zanesljiva, vendar žal tudi najbolj zahtevna in časovno neugodna ročna pretvorba ali s programom po meri.

Večina formatov temelji na tekstovni obliki in hitro lahko najdemo podobnosti med njimi. V nekaterih primerih je razlika med dvema tekstovnima formatoma le v znaku, ki ločuje podatke in/ali "ukazne vrstice", ki ločujejo različne podatkovne dele datoteke (seznam vozlišč, seznam povezav). Tako podobne formate lahko preprosto pretvorimo v tekstovnem urejevalniku. Pri bolj kompleksnih tekstovnih formatih, vključno s formati XML, je pred izdelavo pretvornika po meri, dobro pomisliti, če si lahko pomagamo s kakšnim naprednejšim tekstovnim urejevalnikom, na primer s podporo za

regularne izraze. Obstajajo tudi namenski urejevalniki XML. Če se že odločimo, da sami napišemo pretvornik, je najprej smiselno poiskati morebitno knjižnico za delo vsaj z vhodnim ali izhodnim formatom. Če tudi tu nimamo sreče, nam ostane še odločitev o izbiri programskega jezika. Primer priročnega, visokega jezika je *Python*, saj je v njem mogoče hitro razviti manj obsežne programe. Že v osnovi podpira delo z nizi in datotekami XML in hitro ga lahko namestimo. Pri delu z velikimi količinami podatkov se pokaže njegova šibka plat. Za delovanje potrebuje precej delovnega pomnilnika in v primerjavi z nižjimi programskimi jeziki je precej počasen. Če se odločimo za učinkovitejšo rešitev, je smiselno razmišljati o *Javi*, *C#* ali celo jeziku *c++*.

A.2.6 Datotečni formati za grafično predstavitev omrežij

Ploskovni slikovni formati

Orodja za analizo omrežij, še posebej vizualno orientirana, uporabniku omogočajo, da lahko izvaža slike omrežij. Slike često smatramo kot končni rezultat analize omrežij. Skušamo jih pripraviti tako, da bi bile čim bolj informativne (3.1), je pa rekonstrukcija omrežij iz slik, predvsem avtomatično zelo težak problem. Razlikujemo dve glavni skupini formatov slik. Prva predstavlja skupino rastrskih slikovnih formatov v katerem so shranjene slike sestavljene iz matrike slikovnih elementov, t.i. pikslov. Drugo skupino predstavljajo vektorski slikovni formati. V njih je slika zgrajena iz vnaprej določenih grafičnih elementov in s parametri, ki predpisujejo njihove osnovne transformacije. Takšni elementi so krog, daljica, poligon, zlepek, tudi besedilo, transformacije za premik, rotacijo, strig... Medtem ko so rastrski formati primerni za shranjevanje kompleksnih, razpršenih, zamegljenih upodobitev, ki so po izgledu bližje organskim oblikam, fotografijam, so vektorski formati primernejši za shranjevanje ostrih, jasnih in bolj podrobnih upodobitev. Bistvena razlika med rastrskimi in vektorskimi slikami je v ločljivosti. Pri rastrskih je fiksna, pri vektorskih poljubna. Primeri rastrskih slikovnih formatov so *portable network graphics* – PNG, *Joint Photographic Experts Group* – JPEG, *bitmap* – BMP, *tagged image file format* – TIFF in drugi. Primeri vektorskih formatov so *scalable vector graphics* – SVG, *encapsulated postscript* – EPS, format *postscript* – PS, idr. Poznamo tudi kombinirane slikovne formate. V osnovi so vektorski, vendar se vanje lahko vgradi rastrske slike v obliki teksturnih grafičnih elementov. Primer takšnega formata je že omenjeni SVG, ki preko povezav na zunanje datoteke z rastrskimi slikami, le-te vključi v vektorsko sliko. Kombinirani formati, ki lahko rastrske in vek-

torske elemente združujeta v eno samo datoteko sta *portable document format* – PDF, dokumenti *CorelDraw* – CDR in dokumenti *Adobe Illustrator* – AI.

Formati za shranjevanje prostorskih modelov

Nekatera orodja za analizo omrežij omogočajo izvoz omrežij v obliki prostorskih modelov. Primer omrežij, ki jih je smiselno predstaviti v treh razsežnostih, so na primer molekule. V primerjavi s ploskovnimi upodobitvami, prostorski modeli omrežij skoraj univerzalno temeljijo na vektorski grafiki. Predstavniki prostorskih formatov so prostorska različica formata *scalable vector graphics* – SVG, *virtual reality markup language* – VRML, njegov naslednik X3D, *AutoCAD* – DXF, *3D Studio* – 3DS in drugi. V kemiji in biologiji sta popularna *Kinemages* – KIN in *Molfile* – MDL.

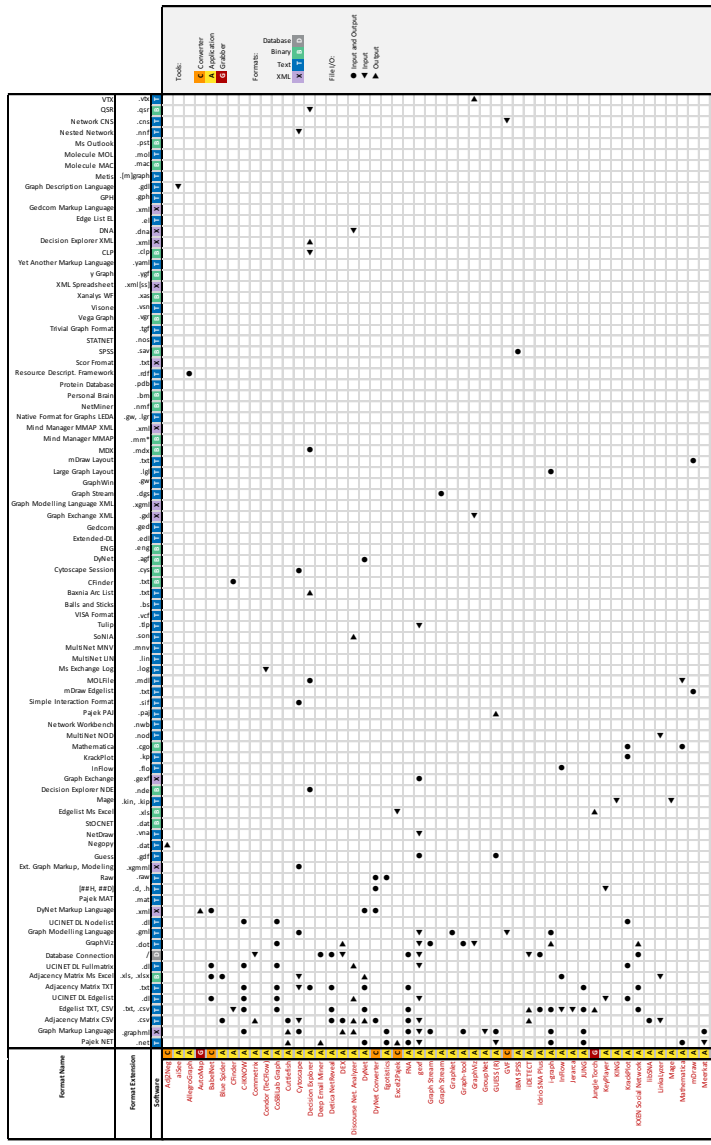
A.2.7 ODF-i in združljivi programi za analizo omrežij

Pretvarjanje med formati

Tu skušamo odgovoriti na vprašanje, če je z uporabo znanih orodij možno izbrani format pretvoriti v drugega? Na sliki A.6 je podana matrika podmnožice 56 od 86 omrežnih datotečnih formatov s slik A.4 in A.3. Z orodji na slikah A.3 in A.4 je vseh 56 formatov možno pretvoriti iz enega v drugega. Izmed 30 ostalih je dva možno medsebojno pretvarjati, sta pa v privatni lasti: *m Draw* in *Mind Manager*. Zadnjih 26 ni možno pretvoriti iz oz. v noben drug format. Iz matrike na sliki A.6 lahko vidimo, da večino formatov lahko v poljuben format pretvorimo že v dveh korakih. Razen pri redkih izjemah zadoščajo trije koraki. Če privzamemo, da imamo na voljo vsa orodja, je skoraj vse formate možno pretvarjati med seboj, vendar se moramo zavedati, da so nekatere pretvorbe nepopolne. Nekateri formati so splošnejši od drugih in pri pretvorbi splošnega v nek namenski format lahko izgubimo podatke, saj jih namenski format morda ne podpira. Na primer pretvorba iz formata *Gedcom* ali iz *Molfile* v format *Pajek* NET je vsaj kar se grafovske strukture tiče popolna. Obratno ne drži, saj ni vsako omrežje geneologija ali molekula. Opisana težava je eden izmed razlogov, zakaj splošen in poenoten pretvornik med formati ne obstaja.

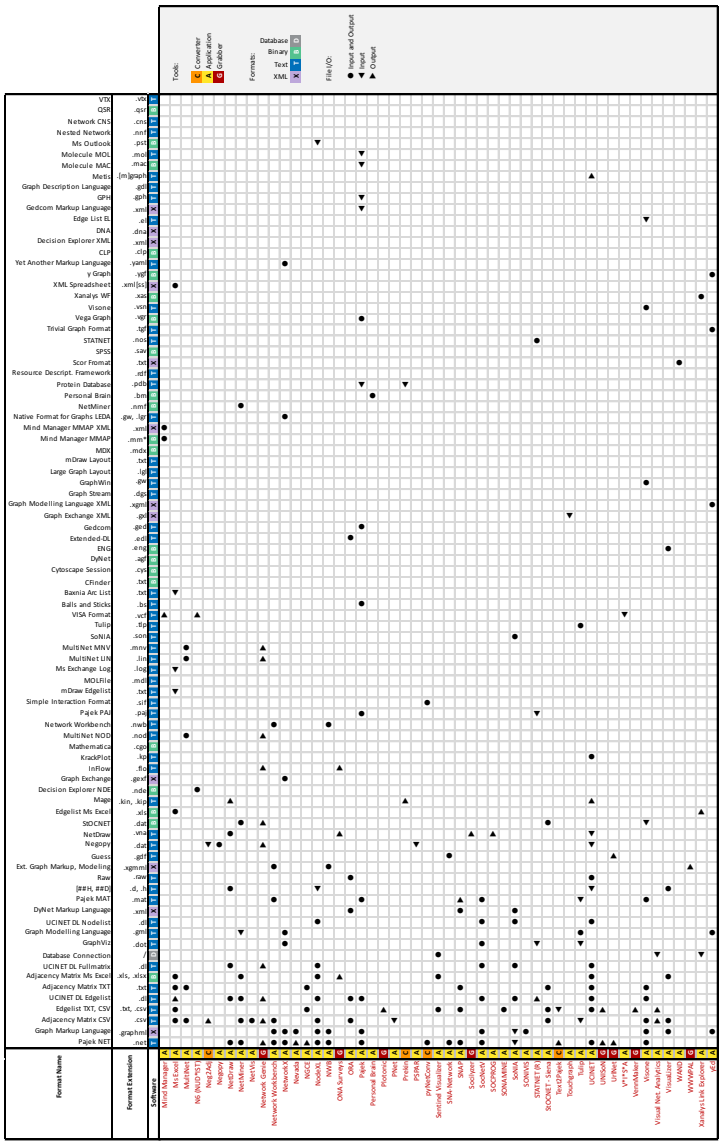
A.2.8 Opombe

Pri zbiranju podatkov o lastnostih ODF-ov smo skušali biti čim bolj natančni, vendar napake niso povsem izključene. Tudi v prihodnje se bomo trudili, da bi tabele ostale v koraku s časom. Dostopne so v obliki spletnega dodatka, ki je dostopen na [116].



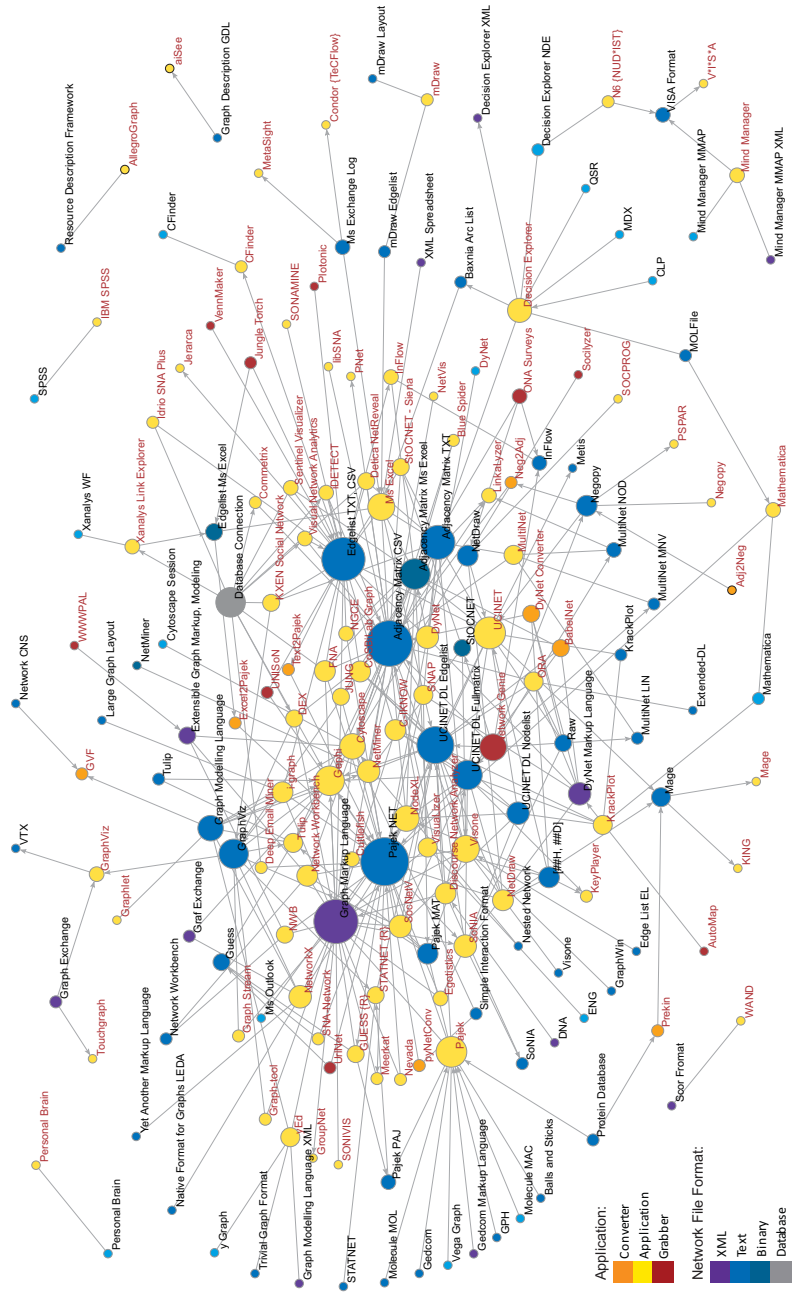
Slika A.3

Matrika ODF-ov in združljivih orodij za analizo omrežij (prvi del, glejte tudi sliko A.4).



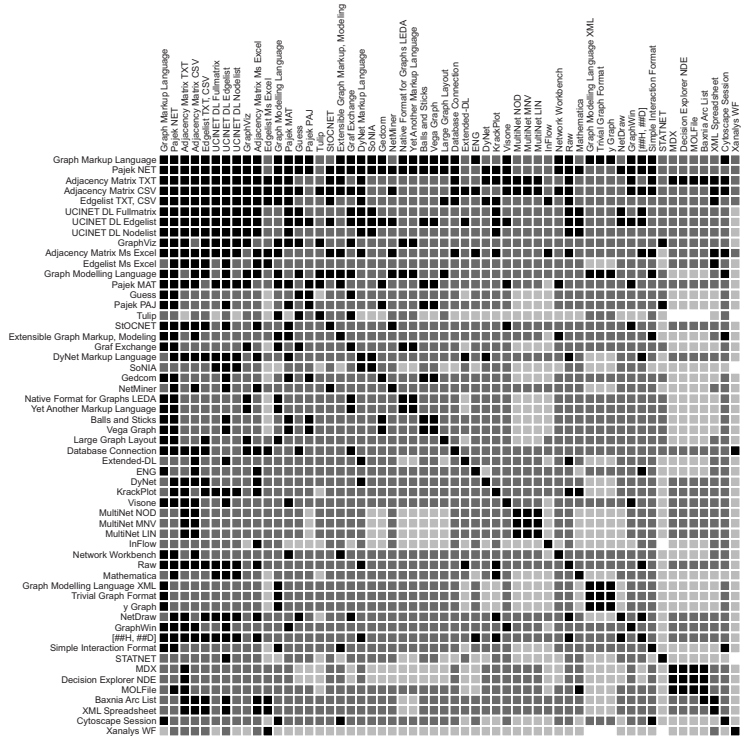
Slika A.4

Matrika ODF-ov in združljivih orodij za analizo omrežij (drugi del, glejte tudi sliko A.3).



Slika A.5

Dvodelno omrežje ODF-ov in orodij za analizo omrežij. Pomen barve vozlišč je enak kot na sliki A.2. Povezave, ki zapuščajo vozlišča formatov in vstopajo v vozlišča programov predstavljajo vhodne formate. Tiste povezave, ki zapuščajo vozlišča programov in vstopajo v vozlišča formatov, pa predstavljajo izhodne formate. Velikost vozlišč je sorazmerna številu povezav, ki vanj vstopajo in izstopajo. (Podoben prikaz je predhodno razvil *Mark Round* [128].)



Slika A.6

Matrika omrežnih datotečnih formatov, ki jih je z uporabo omenjenih orodij možno med seboj pretvarjati. Število korakov pretvorbe formata (v vrstici) v format (v stolpcu) je zaznamovano z barvo: en korak – črna (razen polj na diagonali matrike), dva koraka – temno siva, trije koraki – svetlo siva in štiri ali več korakov – bela.

A.2.9 Omrežni datotečni formati v prihodnosti

Predstavljeni formati imajo veliko lastnosti in njihov namen se od formata do formata razlikuje. Univerzalno orodje za pretvorbo med formati ne obstaja in ne more obstajati, vendar bi bilo smiselno razviti orodje, ki vsaj do neke mere omogoča pretvarjanje med pogostimi in podobnimi ODF-i. Zasnovali bi ga lahko na primer kot spletno storitev.

V prihodnosti lahko pričakujemo, da se bodo v razvoju omrežnih datotečnih formatov pojavili standardi, morda celo univerzalni jezik za opisovanje predstavitev omrežij, ki bo vgrajen v nek splošni ODF. Ker računalniki postajajo zmogljivejši, pomanjkanje prostora za shranjevanje podatkov pa je čedalje manjši problem, je verjetno, da bo standard temeljil na osnovi XML.

V zadnjih časih se razvijajo tudi grafični jeziki, ki jih uporabljamo za programsko opisovanje grafičnih prikazov. Primer grafičnega jezika [129] vsebujejo *Data-Driven Documents* – D3 [52], s katerim lahko na deklarativen način programiramo podatkovno gnane prikaze. Grafične jezike lahko uporabimo tudi za definicijo slogov omrežij in sorodnih podatkov. Za zdaj ne poznamo ODF, ki bi omogočal definicijo slogov programsko oz. s splošnim jezikom, zasledili pa smo nekaj približkov. Poleg grafičnih jezikov je pričakovati razvoj novih grafičnih elementov v povezavi s prikazi omrežij.

A.3 Definicija hierarhičnega končnega avtomata

V tem razdelku navajamo definicijo hierarhičnega končnega avtomata, ki smo ga razvili za potrebe uporabniškega vmesnika knjižnice *net.Plexor* za delo z velikimi omrežji (poglavje 6). Definirali bomo razrede `StateMachine`, `State` in `Transition`, ki določajo funkcionalnosti našega hierarhičnega končnega avtomata. Podali bomo definicije podpornih struktur in razredov, ki omogočajo nadzor nad delovanjem ter prikazali dva primera dogodkov, ki predstavljata vhodna simbola abecede avtomata. V kolikor se bralec želi seznaniti le z vsebino tega razdelka, mu pred nadaljevanjem priporočamo branje razdelka o interaktivnosti uporabniškega vmesnika v *net.Plexor*-ju, 6.3.

Najprej opišimo pojem konteksta v katerem avtomat lahko izvaja svoje prehode. Kontekst bomo večkrat omenili v opisih razredov in metod, saj predstavlja pomembno prilagoditev programske izvedbe hierarhičnega končnega avtomata, ni pa bistvenega pomena za razumevanje ideje hierarhičnega končnega avtomata. V razdelku 6.3.1 poglavja 6 ga zato ne omenjamo. Avtomat smo zasnovali tako, da ga lahko uporablja več uporabniških instanc oz. vmesnikov, v našem primeru oken. Uporabnik lahko

na primer pregleduje in ureja več omrežij hkrati, vsako v svojem oknu. Gre za več instanc oken z isto funkcionalnostjo, t.i. sorodnih oken. Sorodna okna se "obnašajo" na povsem enak način, njihova vsebina pa ni nujno (in le redko je) enaka. Dovolj je, da definiramo eno instanco hierarhičnega končnega avtomata, ki skrbi za interaktivnost sorodnih oken, vendar se mora takšen avtomat odzivati glede na trenutno vsebino posameznega okna. Strogo gledano avtomat določajo samo stanja in prehodi, vhodna vrsta znakov (dogodkov) ter trenutno aktivno stanje; spremenljivke, ki jih uporabljajo pogojne metode, akcije prehodov, vstopne in izstopne akcije stanj in nekatere druge podporne funkcije pa so del konteksta, ki ga vsebuje posamezno okno in se ga avtomatu poda pri izvajanju. Tehnično je avtomat šablona, ki predpisuje možne spremembe v okviru konteksta, stanje uporabniškega vmesnika posameznega sorodnega okna pa določa kontekst.

Naš hierarhični končni avtomat je na najvišjem nivoju definiran z razredom `StateMachine`. Njegova stanja so razrednega tipa `State`. Na globljem nivoju ima posamezno stanje lahko rekurzivno vgnezden nov hierarhični končni avtomat `StateMachine`, itd.:

```
class StateMachine //StateMachine definira herarhični končni avtomat
{
public:
 StateMachine(State *superState, StateActionMethod entryAction, StateActionMethod leaveAction);
 //konstruktor: superState je nadstanje, ki bo vsebovalo hierarhični končni avtomat te instance oz. NULL če gre za
     najvišji nivo; entryAction je kazalec na metodo, ki implementira vstopno akcijo; leaveAction na enak način kot
     entryAction implementira izstopno akcijo

 State *addNewState(string name, StateActionMethod entryAction, StateActionMethod leaveAction);
 //doda stanje z imenom name v instanco; entryAction je kazalec na metodo, ki implementira akcijo, ki se izvede ob
     prehodu v dodano stanje; leaveAction ima enak pomen kot entryAction, le da se izvede tik pred prehodom iz
     dodanega stanja

 void addTransition(HSEventId eventId, ConditionMethod condition, ActionMethod action, int sourceIndex, State *target);
 //doda prehod, ki za izvedbo potrebuje dogodek eventId ali NULL, če dogodek ni obvezen; condition je kazalec na
     pogojno metodo, ki vrne true, če je pogoj za prehod izpolnjen, če je NULL, pogoja ni; action je kazalec na
     metodo, ki se izvede ob prehodu; sourceIndex je zaporedna številka stanja iz katerega vodi prehod; target pa je
     kazalec na ponorno stanje
 //prednost prehodov je določena implicitno z vrstnim redom klicev addTransition na instanci; poznejši klic ima nižjo
     prednostjo

 void addTransition(HSEventId eventId, ConditionMethod condition, ActionMethod action, string source, string target);
 //alternativa prejšnje metode z razliko, da začetno in končno stanje navajamo z imenoma source -- izvor in target --
     ponor; začetno podstanje instance na najvišjem od vgnezdenih nivojev je dosegljivo z imenom ``Start''; če so
     StateMachine-i gnezdeni v podstanjih, do njih dostopamo z imenom v obliki niza: ``<ime podstnaja>.Start''; v
     praksi definicije prehodov se uporablja samo ta metoda

 State *getState(int stateIndex);
 //na podlagi zaporedne številke podstanja instance vrne kazalec na podstanje

 State *getState(string name);
 //alternativa zgornje metode; vrne kazalec na podstanje instance z imenom name; v praksi se pogosteje uporablja ta
     metoda

 unordered_map<string, State*> *getStateNamesTable();
 //vrne preslikavo med kazalci podstanj instance in njihovih imen

 deque<PendingEvent*> *getPendingEventsQueue();
```

```
 //vrne kazalec na vstopno vrsto prejetih znakov (dogodkov) končnega avtomata

 void initialize();
 //po definiciji vseh stanj in prehodov avtomata kličemo to metodo; s klicem avtomat izvede vsa stanja od začetnega,
     da pride do prvega stanja, ki za prehod zahteva dogodek; tako avtomat preide v stanje čakanja

 void addPendingEvent(TransitionContext *owner, HSEvent *event);
 //doda dogodek event v kontekstu owner na začetek vrste čakajočih dogodkov avtomata; če se dogodek doda med
     izvajanjem katere akcije in se s tem omogoči nadaljevanje izvajanja avtomata, bo ta izvajal prehode, dokler vrsta
     dogodkov ne bo prazna

 void executePendingEvents();
 //izvede vse možne prehode na podlagi trenutnega aktivnega stanja v instanci, dogodkov na koncu vrste avtomata v
     okviru posameznemu dogodku določenega konteksta in izpolnjenosti pogojev; če pri izvajanju ni nobenega ustreznega
     prehoda (niti v stanjih na višjih nivojih (starših) instance), se dogodek, ki je na koncu vrste, razveljavi in
     izvajanje se nadaljuje v istem stanju

 void execute(TransitionContext *owner, HSEvent *event);
 //enako kot zgornja metoda, le da pred tem v kontekstu owner doda dogodek event na začetek vrste dogodkov

 void setReporter(StateReporter *reporter);
 //nastavi kazalec reporter na poročevalca stanja avtomata; poročevalec se uporablja npr. za izpis (ali kaj drugega) v
     katero stanje je avtomat pravkar vstopil

 string getCurentStateName();
 //vrne ime stanja na najnižjem nivoju (najgloblje) v katerem se avtomat trenutno nahaja
};

class State //State definira stanje v hierarhičnem končnem avtomatu
{
public:
 State(State *superState, StateMachine *parentMachine, string name, StateActionMethod entryAction, StateActionMethod
     leaveAction);
  //konstruktor: razreda ne instanciramo eksplicitno, instancira ga StateMachine::addNewState(...)

 void assignStateMachine(StateMachine *stateMachine);
 //stanju vgnezdimo hierarhični končni avtomat stateMachine, ki realizira vsa podstanja instance stanja

 machineStatus process(TransitionContext *conditionOwner, HSEvent *event, State* &nextState);
 //metodo uporablja razred StateMachine; poskrbi za izvajanje prehodov

 void addTransition(Transition *transition);
 //metodo uporablja razred StateMachine; poskrbi za dodajanje prehoda

 StateMachine *getStateMachine();
 //vrne hierarhični končni avtomat StateMachine, ki ga instanca stanja vsebuje

 StateMachine *getParentMachine();
 //vrne hierarhični končni avtomat StateMachine, ki vsebuje instanco tega stanja kot svoje podstanje

 State *getSubState(int stateIndex);
 //vrne podstanje instance z indeksom stateIndex

 string getName();
 //vrne ime stanja instance
};

enum machineStatus { CONTINUE, WAITING_EVENT };
//status v katerem se nahaja avtomat, CONTINUE sporoča, da izvajanje še ni zaključeno, WAITING_EVENT pomeni, da se bo
    izvajanje nadaljevalo po klicu npr. executePendingEvents(), če bo v vhodni vrsti prisoten kak dogodek

class Transition //Transition definira interno predstavitev prehoda
{
public:
 Transition(HSEventId requestedEventId, ConditionMethod condition, ActionMethod action, State *target);
 conditionResult conditionAction(TransitionContext *owner, HSEvent *event);
 State *getTarget() { return target; }
};

class StateReporter //StateReporter definira poročevalca stanja avtomata
{
public:
 virtual void stateChanged(State *currentState) { }
```

```
   //virtualna metoda za poročanje v katero stanje je avtomat vstopil; metodo v poljubnem izpeljanem razredu razširimo
       in jo uporabimo npr. za poročanje stanja uporabniškega vmesnika
};

//tipi, ki jih uporabljamo za definicijo metod, katerih parametri so kazalci na metode pogojev in akcij
typedef bool (Condition::*ConditionMethod) (HSEvent *event); //tip za definicijo pogojne metode na prehodu
typedef void (Action::*ActionMethod) (HSEvent *event); //tip za definicijo akcije ob prehodu
typedef void (StateAction::*StateActionMethod) (); //tip za definicijo akcije ob vstopu v stanje ali izstopu iz stanja

class Condition //Condition razred določa vse pogojne metode, ki jih imamo namen uporabiti v programski izvedbi
    hierarhičnega končnega avtomata; v tem primeru navajamo primer metode (GLC_IsCursorAboveMovePivot), ki definira
    pogoj, ki v našem uporabniškem vmesniku prikazovalnika in urejevalnika omrežja vrača podatek ali se kazalec miške
    nahaja nad kazalnikom za premik izbranih elementov omrežja
{
public:
 //metode definiramo in jih razširimo v razredu, ki realizira dani kontekst, če bi se metoda klicala iz drugega
     konteksta, se delovanje ustavi, primer:
 virtual bool GLC_IsCursorAboveMovePivot(HSEvent *event) { assert(false); return false; }
 //...
}

class Action //Action razred določa vse metode akcij, ki se lahko izvedejo ob izvršenem prehodu v programski izvedbi
    hierarhičnega končnega avtomata; v tem primeru navajamo primer metode (GLC_SaveCoordinates), ki v našem
    uporabniškem vmesniku prikazovalnika in urejevalnika omrežja shrani trenutni položaj miške
{
public:
 //metode definiramo in jih razširimo v razredu, ki realizira dani kontekst, če bi se metoda klicala iz drugega
     konteksta, se delovanje ustavi, primer:
 virtual void GLC_SaveCoordinates(HSEvent *event) { assert(false); }
 //...
}

class StateAction //StateAction razred določa vse metode akcij, ki se v programski izvedbi hierarhičnega končnega
    avtomata lahko izvedejo ob vhodu v ali ob izhodu iz stanja avtomata; v tem primeru navajamo primer ene metode
    (GLC_SelectionModeChange), ki ob zapustitvi stanja, zaradi prekinitve na višjem nivoju poskrbi, da se v začasni
    spremenljivki za način izbiranja ponastavi vrednost
{
public:
 //metode definiramo in jih razširimo v razredu, ki realizira dani kontekst, če bi se metoda klicala iz drugega
     konteksta, se delovanje ustavi, primer:
 virtual void GLC_SelectionModeChange() { assert(false); }
 //...
}

class TransitionContext : public Condition, public Action, public StateAction //TransitionContext razred povezuje
    razrede Condition, Action in StateAction, da pri definiciji metod pogojev in akcij ne potrebujemo dedovati vseh
    treh razredov in da hkrati zadošča ena metoda za funkcionalnosti, ki pogoje in akcije uporabljajo;
    TransitionContext določa kateri nabor funkcionalnosti oz. podatkov naj hierarhični končni avtomat uporabi;
    identičen avtomat lahko deluje nad več različnimi konteksti, na primer, če želimo z eno instanco hierarhičnega
    avtomata krmiliti več sorodnih oken, ki delujejo na povsem enak način, vendar vsebujejo različno vsebino -- kontekst
{
};

//HSEvent je osnovni razred, ki implementira znak za vhodno vrsto avtomata (dogodek); z njegovim dedovanjem
    specificiramo nov znak (dogodek); primer dogodka v net.Plexor-ju, ki se v uporabniškem vmesniku pregledovalnika oz.
    urejevalnika omrežij vstavi v vrsto na vsako pretečeno časovno enoto internega števca za osveževanje in dinamiko:
class HSTick : public HSEvent
{
public:
 enum { ID = __LINE__ };
 HSTick() { id = ID; }
};

//primer definicije sestavljenega dogodka z dodatno lastnostjo keyCode, t.j. kodo pritisnjene tipke; dodatne lastnosti
    dogodkov je mogoče uporabiti znotraj akcij in pogojev, ki dogodek prejmejo; nova instanca spodnjega dogodka se v
    vhodno vrsto avtomata doda za vsako pretečeno časovno enoto internega števca ter za vsako tipko, ki je pritisnjena:
class HSKeyRepeat : public HSEvent
{
public:
 enum { ID = __LINE__ };
 HSKeyRepeat(long keyCode) { this->keyCode = keyCode; id = ID; }
 long keyCode;
};
```

A.4 *Hitra multipolna metoda v tehnologiji* OpenCL®

Sodobna tehnologija namenske strojne opreme v obliki grafičnih procesnih enot – GPE je v zadnjih letih na primer z uvedbo programske podpore *OpenCL*® in CUDA (*Compute Unified Device Architecture*) uporabnikom postala dostopna tudi v druge, bolj splošne namene. Za razliko od prej, ko so bile GPE namenjene zgolj grafičnim potrebam, lahko sedaj z njimi rešujemo splošne in procesorsko zahtevne, vendar predvsem paralelne probleme. Upoštevati moramo nekaj omejitev. Na arhitekturah GPE sklad ni na voljo in posledično ni mogoče izvajati rekurzije. Problemi morajo biti definirani iterativno. V tem razdelku na primeru pokažemo učinkovitost in prednosti procesiranja na GPE. Metodo smo vgradili v *net.Plexor* (poglavje 6).

V prispevku [130, 131] je opisan problem umestitve vozlišč grafa $\mathbf{G(V, E)}$ (1.4.2), kjer je $\mathbf{V}$ množica vozlišč in $\mathbf{E}$ množica povezav, v ravnino po metodi minimalne energije s fizikalnim modelom, ki povezave grafa modelira z vzmetmi, odbojnost med vozlišči ("točkastimi naboji") pa z elektrostatsko odbojno silo. Metoda problem umestitve razdeli na več nivojev abstrakcije grafa, tako da v začetnih iteracijah pride do močne interakcije med oddaljenimi območji v grafu, postopoma med bližjimi deli grafa in šele v zadnjih iteracijah med sosednjimi vozlišči. Brez abstrakcije grafa bi za enak učinek potrebovali bistveno več iteracij. Gre za t.i. hitro multipolno metodo (razdelek A.4.2). Podobno metodo uporabljajo tudi avtorji [132], ki temelji na sprektralnem razvrščanju. Velik poudarek razdelave problema je v minimizaciji časovne zahtevnosti algoritma in paralelizaciji za izvedbo na GPE.

A.4.1 *Naivni algoritem*

Naivni umestitveni algoritem v posamezni iteraciji izračuna odbojno silo med pari trenutno razporejenih vozlišč in privlačno silo vozlišč (krajišč) povezav ter dušeno premakne vozlišča v smeri rezultante sil na posamezno vozlišče. Časovna zahtevnost je $O(|\mathbf{V}|^2 + |\mathbf{E}|)$. Algoritem teče toliko časa, dokler premiki vozlišč med zaporednima iteracijama niso dovolj majhni, kar je možno preveriti v času $O(|\mathbf{V}|)$ oz. neko maksimalno število iteracij. Skupni čas naivnega algoritma je torej $O(k \cdot (|\mathbf{V}|^2 + |\mathbf{E}|))$, kjer je k število iteracij, ki pa je konstantno in odvisno predvsem od izbire dopustne meje velikosti vsote premikov med posameznima iteracijama, začetne (navadno naključne) razporeditve vozlišč $\mathbf{G}$ in nekaterih drugih parametrov.

A.4.2 Izboljšan algoritem

V boljšem umestitvenem algoritmu *Godiyal*-a in soavtorjev [131] skušajo člen zahtevnosti $O(|\mathbf{V}|^2)$ zmanjšati. Namesto računanja odbojne sile med vsemi pari vozlišč izkoristijo načelo aproksimacije odbojne sile na vozlišče z odbojnostjo oddaljenih multipolnih skupin (skupin večih vozlišč, blizu skupaj in oddaljenih od vozlišča na katerega odbojna sila deluje) – hitri multipolni pristop. V poteku algoritma vozlišča razporejamo na primer v štiriško ali drevo *K-D*. Sile na vozlišča, ki padejo izven posameznih vej drevesa aproksimiramo z odbojnostjo na skupni "naboj" zbran v središču posamezne veje drevesa, ki ga predstavljajo vozlišča, ki so v teh vejah. S tem se zahtevnost računanja odbojne sile zmanjša na $O(|\mathbf{V}|\log|\mathbf{V}|)$, skupna zahtevnost algoritma pa na $O(k\cdot(|\mathbf{V}|\log|\mathbf{V}|+|\mathbf{E}|))$. Pri tem moramo upoštevati, da se čas izračuna še nekoliko poveča, saj moramo v iteracijah izračunati tudi ustrezna štiriška ali drevesa *K-D*. Izračun drevesa *K-D*, kot ga uporabljajo avtorji metode [131], vzame $O(|\mathbf{V}|\log|\mathbf{V}|)$ časa.

Aproksimacija sile na posamezno vozlišče, ko je to izven določene veje drevesa, se izračuna z uporabo izreka o multipolni ekspanziji [131] z razvojem v *Laurentovo* vrsto in upoštevanjem prvih nekaj njenih členov. Ko vozlišče leži znotraj določene veje drevesa, se sila izračuna ekzaktno kot vsota prispevkov vseh njegovih sosedov znotraj iste veje drevesa.

Izboljšava algoritma gre na račun vpeljave hierarhične piramide vozlišč grafa $\mathbf{G}$, ki porodi hierarhijo grafov $\mathbf{G}_i(\mathbf{V}_i, \mathbf{E}_i) : \mathbf{V} = \mathbf{V}_0 \supset \mathbf{V}_1 \supset .. \supset \mathbf{V}_s \supset \emptyset$. Za vsak par $\mathbf{V}_i \supset \mathbf{V}_{i+1}$ velja, da je $\mathbf{V}_{i+1}$ maksimalna neodvisna množica množice $\mathbf{V}_i$. Hierarhijo je možno izračunati z učinkovitim 2-aproksimacijskim algoritmom v času $O(|\mathbf{V}|\log|\mathbf{V}|)$. Ko hierarhijo izračunamo, je ideja, da se algoritem za umestitev vozlišč zažene na naključno razporejeni množici vozlišč $\mathbf{V}_i$ in rezultat uporabi za začetno umestitev za izvajanje umestitvenega algoritma na "razpihnjeni" množici $\mathbf{V}_{i-1}$ in tako naprej po hierarhiji navzgor. Vsako vozlišče višje v hierarhiji (iz $\mathbf{G}_i$) "vsebuje" skupine vozlišč naslednjega nivoja ($\mathbf{G}_{i+1}$), ki ob prestopu nivoja podedujejo prostorske koordinate v. Pri tem se koordinate še nekoliko prilagodijo glede na povezanost s sosedi v $\mathbf{G}_{i+1}$. Gre za t.i. interpolacijo [131].

Ker *OpenCL*® ne dopušča rekurzije in sklada, moramo strukture oblikovati tako, da rekurzija ni potrebna. Drevo *K-D* je zasnovano tako, da bo obhod po njegovi strukturi na GPE izvedljiv brez rekurzije. Drevo predstavimo kot 1-razsežno tabelo vozlišč. Vsako vozlišče vsebuje podatek (indeks v tabeli) o njegovem potomcu in nasledniku v

drevesu v smislu obhoda v globino. Algoritem za obhod deluje iterativno tako, da za naslednje vozlišče izbere (začenši s korenom, t.j. prvim elementom tabele) vozlišče v tabeli na indeksu potomca, če je naveden, sicer izbere vozliče z indeksom naslednika. Če tudi ta ni naveden, je obhod končan.

A.4.3 Posplošeni algoritem v prostoru

Opisan algoritem je namenjen umestitvam vozlišča grafov v ravnino. Za potrebe programskega orodja *net.Plexor* (6.4.4) smo model posplošili na tri razsežnosti. Posplošitev drevesa *K-D* je enostavna. V 2-razsežnem drevesu *K-D* se s poglabljanjem izmenjujeta dve osi delitve prostora, v 3-razsežnem pa tri. Algoritem za gradnjo drevesa zato vključuje novo vrsto razreza po globini in ustrezne meje vej, ki so sedaj 3-razsežne.

Posplošitev razvoja po izreku multipolne ekspanzije v tri razsežnosti je nekoliko bolj zahtevena in ima slabost, da bi morali v primerjavi z 2-razsežno različico za primerljivo natančnost upoštevati več členov razvoja v *Laurentovo* vrsto (17 namesto 4). Po razmisleku, vključujoč manj stroge zahteve za potrebe vizualizacije omrežij v našem programu *net.Plexor*, za razliko od avtorjev izvirne metode [131], ki izhajajo iz sorodnih del za natančne *N-Body* simulacije, smo ubrali pristop, da za skupine nabojev posameznih vej v drevesu *K-D* vzamemo približek enakovrednega točkastega naboja v njihovem težišču. Skupno silo na vozlišče izven veje drevesa *K-D* aproksimiramo z odbojno silo vozlišča do enakovrednega točkastega naboja veje. Pretvorbi v *Laurentovo* vrsto se tako popolnoma izognemo. Izračun odbojne sile postane enostaven in enak izračunu sile med dvema vozliščema. Skladno s tem smo prilagodili strukturo, ki opisuje drevo *K-D*. Dodali smo spremenljivke za vsoto koordinat vozlišč x, y in z, katerim prištevamo koordinate vozlišč med izgradnjo drevesa *K-D*. Pri prehodu skozi drevo delimo vrednosti teh spremenljivk s številom vozlišč v obiskani veji in tako določimo težišče "naboja".

A.4.4 Programska izvedba

Izhajali smo iz psevdokode algoritma, ki je zapisana v prispevku avtorjev [131] in pomanjkljive programske izvedbe. V algoritmu smo upoštevali zgornji dve spremembi, zadevo priredili in jo vgradili v orodje *net.Plexor*. Ker avtorji podajajo le posamezne funkcije algoritma, ne pa tudi kode za inicializacijo in uporabo *OpenCL®*, smo si pomagali s priročnikom za *OpenCL®* [133] in ustrezno specifikacijo [134]. Odpravili smo nekaj napak ter optimizirali kodo, da čim bolj izkorišča v *OpenCL®* vgrajene funkcije.

Tabela A.2

Pohitritve z uporabo GPE za različne probleme pri enakih začetnih pogojih.

primer	$A(383, 410)$		$B(1000, 1773)$		$C(29067, 29290)$	
arhitektura	CPE	CPE/GPE	CPE	CPE/GPE	CPE	CPE/GPE
meritve časov [s]	6,5	5,9	35,3	19,6	43892,7	5790,4
	6,5	5,9	35,4	19,5	43168,2	5572,2
	6,5	5,9	35,7	19,7		5983,6
	6,4	5,9	35,4	19,4		
povprečje [s]	6,5	5,9	35,5	19,6	43530,5	5782,1
pohitritev z GPE	1,1x		1,8x		7,5x	

Glede na časovno zahtevnost posameznih delov algoritma smo na GPE implementirali le najzahtevnejši del algoritma, t.j. sprehod po drevesu *K-D* in izračun odbojnih sil na vsako izmed vozlišč. (Avtorji navajajo, da se v *OpenCL*® da učinkovito implementirati tudi izgradnjo drevesa *K-D*, vendar se prednost pred CPE pokaže šele pri grafih z več kot 50000 vozlišč. Pri tem razlika narašča počasi.)

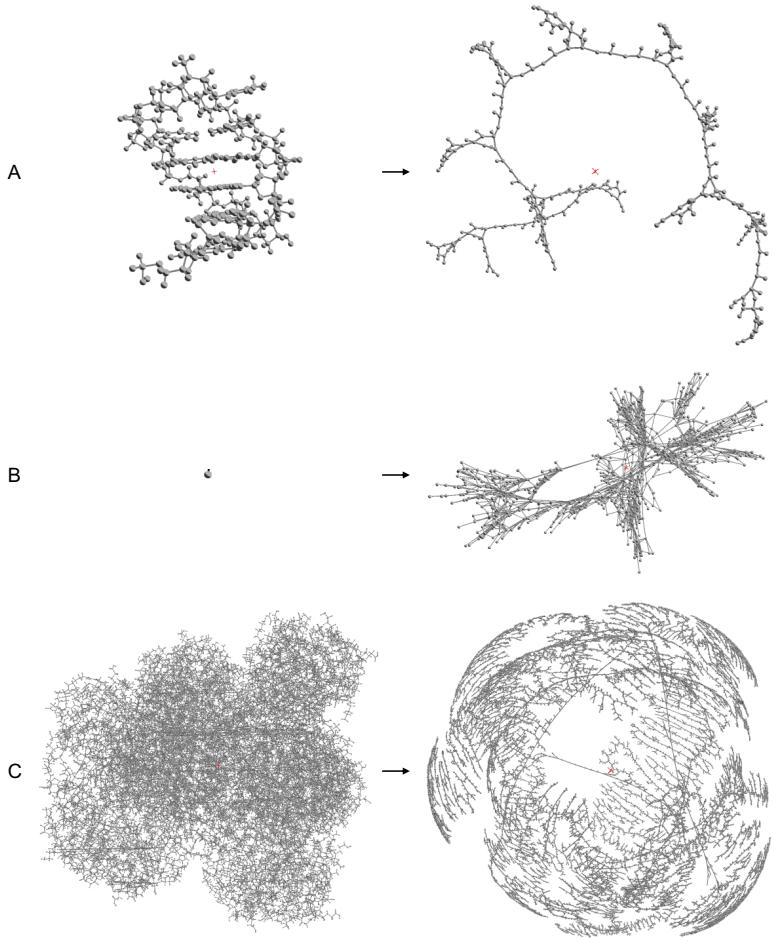
Ker je razhroščevanje na GPE težavno, smo se programske izvedbe lotili postopno. Vzpostavili smo prenose podatkov iz glavnega pomnilnika v GPE in prenos nazaj in postopoma uvajali funkcionalnost iz kode *c++* za CPE v *OpenCL*® ter sproti preverjali delovanje.

A.4.5 Analiza časovne zahtevnosti

Na treh različnih grafih $A(383, 410)$, $B(1000, 1773)$, $C(29067, 29290)$ (slika A.7) smo pri enakih parametrih in začetnih položajih vozlišč na CPE in na kombinirani arhitekturi CPE/GPE primerjali čase izvajanja algoritma. Časi so podani v tabeli A.2. V predzadnji vrstici je podano povprečje časov, v zadnji pa pohitritev GPE v primerjavi s časom na CPE.

V A.3 so podani rezultati, ko algoritem poganjamo pri istih parametrih, a z naključno postavitvijo začetnih položajev. Pohitritve so primerljive, odstopanja od povprečja pa relativno velike, predvsem v primeru C. Razlog je v strukturi grafa C, ki sestoji iz zelo dolgih verig, ki se, razvite na veliki razdalji, šibko odbijajo. Umestitev zato počasi konvergira v energetski minimum in čas je močno odvisen od začetne postavitve vozlišč. Razlike nastopijo zaradi različnega konstantnega faktorja k, ki smo ga omenili pri zahtevnosti algoritma (A.4.2).

Pri kombinirani arhitekturi na GPE teče le najbolj kompleksni del algoritma, t.j. računanje odbojne sile med vozlišči, vsi ostali deli algoritma delujejo na CPE v obeh



Slika A.7

Slike grafov *A*, *B* in *C* pred in slike grafov *A*, *B* in *C* po uporabi hitrega multipolnega algoritma za umeščanje vozlišč.

Tabela A.3

Pohitritve z uporabo GPE za različne probleme pri različnih začetnih položajih vozlišč.

primer	$A(383, 410)$		$B(1000, 1773)$		$C(29067, 29290)$	
arhitektura	CPE	CPE/GPE	CPE	CPE/GPE	CPE	CPE/GPE
meritve časov [s]	6,4	5,9	35,4	21,5	45376,2	2200,5
	7,3	6,7	40,3	19,9	40239,1	6701,0
	6,4	6,7	45,1	18,2		5790,4
	7,4	5,9	46,2	21,7		
povprečje [s]	6,9	6,3	41,7	20,3	42807,7	4897,3
pohitritev z GPE	1,1x		2,1x		8,7x	

primerih. Med računanjem odbojne sile CPE čaka na GPE. Tako smo naredili, da je primerjava rezultatov "poštena", čeprav nas nič ne obvezuje, da med izvajanjem na GPE CPE miruje. Paralelno bi lahko nekaj izračunov reševali tudi na CPE in s tem še nekoliko pohitrili izvajanje. Kakor koli, razlika med hitrostjo delovanja ene in druge programske izvedbe algoritma je v prid tisti z uporabo CPE/GPE občutna.

Pojasnimo zakaj algoritem na primeru A na CPE/GPE teče relativno počasi v primerjavi z različico CPE. Algoritem smo testirali v okolju *net.Plexor* (6), ki med procesiranjem interaktivno prikazuje vmesne rezultate algoritma – sliko trenutne razporeditve vozlišč. Ker izrisovanje vzame nekaj časa, hkrati pa je hitrost osveževanja omejena s časovnikom, ki proži konstantno število iteracij algoritma v enotah izvajanja (opisano v 6.3.1), v primeru A pohitritev ne pride do izraza. Primer je enostavno premajhen. Pri njem postane t.i. režijski čas, ki se nujno porabi za inicializacijo vmesnikov za komunikacijo med CPE in GPE, za prenos vmesnih rezultatov in za druga opravila primerljiv s časom izvajanja dejanskega algoritma. Pojav lahko opišemo z *Ahmdal*-ovim zakonom [135]. Pričakovati je, da bi na še manjših problemih CPE samostojno problem rešil hitreje.

LITERATURA

[1] Y.-Y. Ahn, J. P. Bagrow in S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, Junij 2010. ISSN 0028-0836. doi: 10.1038/nature09182.

[2] V. Batagelj. *Encyclopedia of Complexity and System Science*, poglavje Complex Networks section: Visualization of Large Networks, strani 1253–1268. Springer Verlag, 2009.

[3] V. Batagelj. *Encyclopedia of Complexity and System Science*, poglavje Social Network Analysis: Lerge-Scale, strani 8245–8265. Springer Verlag, 2009.

[4] C. Berge. *Hypergraphs, Volume 45: Combinatorics of Finite Sets (North-Holland Mathematical Library)*. North Holland, 1. izdaja, Avgust 1989. ISBN 0444874895. URL www.amazon.com/gp/product/0444874895.

[5] W. R. Mueller, K. Szymanski, J. V. Knop in N. Trinajstić. Molecular topological index. *Journal of Chemical Information and Computer Science*, 30: 160–163, 1990.

[6] H. Wiener. Structural determination of paraffin boiling points. *Journal of the American Chemical Society*, 1:17–20, 1947.

[7] A. T. Balaban, I. Motoc, D. Bonchev in O. Mekenya. *Topological indices for structure-activity correlations*, zvezek 114, strani 21–55. Springer, Berlin, 1983. doi: 10.1007/BFb0111212.

[8] A. T. Balaban. *Chemical Applications of Graph Theory*. Academic Press, 1976.

[9] R. Todeschini in V. Consonni. *Molecular Descriptors for Chemoinformatics*. Weinheim: Wiley-VCH, 2. izdaja, 2009.

[10] J. Devillers in A. T. Balaban. *Topological Indices and Related Descriptors in QSAR and QSPR*. Gordon and Breach, 1999.

[11] M. Karelson. *Molecular Descriptors in QSAR/QSPR*. Wiley-Interscience, 2000.

[12] H. Hosoya. Topological index. a newly proposed quantity characterizing the topological nature of structural isomers of saturated hydrocarbons. *Bulletin of the Chemical Society of Japan*, 44:2332–2339, 1971.

[13] M. Randić. Characterization of molecular branching. *Journal of the American Chemical Society*, 97: 6609–6615, 1975.

[14] A. T. Balaban. Highly discriminating distance-based topological index. *Chemical Physics Letters*, 89: 399–404, Julij 1982.

[15] R. Todeschini in E. Collina R. Cazar. The chemical meaning of topological indices. *Chemometrics and Intelligent Laboratory Systems*, 15:51–59, 1992.

[16] Thomson Reuters. Web of science. URL http://scientific.thomson.com/products/wos.

[17] J. M. Modak in G. Madras. Scientometric analysis of chemical engineering publications. *Current Science*, 94:1265–1272, Maj 2008.

[18] P. Willett. A bibliometric analysis of the journal of molecular graphics and modelling. *Journal of Molecular Graphics and Modelling*, 26:602–606, 2007.

[19] P. Willett. A bibliometric study of quantitative structure-activity relationships and qsar & combinatorial science. *Qsar & Combinatorial Science - QSAR COMB SCI*, 28:1231–1236, 2009.

[20] N. M. Builova in A. I. Osipov. Scientometric analysis of publications in the area of nanoenergy based on the materials of the peer-reviewed journal of viniti ras physics of nanoobjects and nanotechnology. *Scientific and Technical Information Processing*, 39:215–219, Oktober 2012.

[21] K. W. Boyack, K. Börner in R. Klavans. Mapping the structure and evolution of chemistry research. strani 112–123, 2007.

[22] M. R. Davarpanah in S. Aslekia. A scientometric analysis of international lis journals: Productivity and characteristics. *Scientometrics*, 77:21–39, 2008.

[23] N. Kejžar, S. Korenjak-Černe in V. Batagelj. Network analysis of works on clustering and classification from web of science. strani 525–536. Springer, 2010.

[24] A. Ahmed, V. Batagelj, X. Fu, S. H. Hong, D. Merrick in A. Mrvar. Visualisation and analysis of the internet movie database. strani 17–24, 2007.

[25] M. E. J. Newman. Scientific collaboration networks: I. network construction and fundamental results. *Physical Reviews E*, 64:016131, Julij 2001.

[26] J. Moody. The structure of a social science collaboration network: Disciplinary cohesion from 1963 to 1999. *American Sociological Review*, 69:213–238, 2004.

[27] V. Batagelj in M. Cerinšek. On bibliographic networks. *Scientometrics*, 96:845–864, Januar 2013. doi: 10.1007/s11192-012-0940-1.

[28] J. Bodlaj in V. Batagelj. Network analysis of publications on tolological indices from web of science – supplement, 2014. URL http://zvonka.fmf.uni-lj.si/netbook/doku.php?id=paper:visla:dodatek.

[29] L. Li, D. Alderson, R. Tanaka, J. C. Doyle in W. Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications. *cond-mat/0501169, Internet Math*, 2:431–523, 2007.

[30] E. O. Laumann, P. V. Marsden in D. Prensky. *The boundary specification problem in network analysis*, strani 18–34. Sage Publications, 1983.

[31] G. Kossinets. Effects of missing data in social networks. *Social Networks*, 28:247–268, Februar 2003.

[32] N. Marschall. Methodological problems with transformation and size reduction of data sets in network analysis, 2006.

[33] J. Scott. *Social Network Analysis: A Handbook*. Sage Publications, 2. izdaja, 2000.

[34] V. Batagelj in A. Mrvar. Pajek - program for large network analysis. *Connections*, 21:47–57, 1998. URL http://pajek.imfm.si.

[35] W. de Nooy, A. Mrvar in V. Batagelj. *Exploratory Social Network Analysis with Pajek*. Cambridge University Press, 2. izdaja, 2012.

[36] V. Batagelj. Efficient algorithms for citation network analysis, September 2003. URL http://arxiv.org/abs/cs.DL/0309023.

[37] N. P. Hummon in P. Doreian. Computational methods for social network analysis. *Social Networks*, 12: 273–288, 1990.

[38] M. Zaveršnik in V. Batagelj. Islands. Maj 2004.

[39] J. Kelley. Critical path planning and scheduling: Mathematical basis. *Operations Research*, 9:296–320, Maj – Junij 1961.

[40] V. Batagelj in M. Zaveršnik. *Fast algorithms for determining (generalized) core groups in social networks*, zvezek 5, strani 129–145. Springer, 2011.

[41] P. Doreian, V. Batagelj in A. Ferligoj. *Generalized Blockmodeling*. Cambridge University Press, 2005.

[42] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010. ISSN 0370-1573. doi: 10.1016/j.physrep.2009.11.002.

[43] H. Liu. Montylingua: An end-to-end natural language processor with common sense, 2004. URL http://web.media.mit.edu/~hugo/montylingua.

[44] G. Salton in C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, 24:513–523, 1988.

[45] C. Ware. *Information Visualization: Perception for Design*. Elsevier Inc., 2013.

[46] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Pr, 2. izdaja, 2001.

[47] S. S. Stevens. On the psychophysical law. *Psychological Review*, 64(3):153–181, Maj 1957.

[48] N. Elmqvist in J.-D. Fekete. Hierarchical aggregation for information visualization: Overview, techniques and design guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16:439–454, Maj – Junij 2010.

[49] G. G. Robertson, J. D. Mackinlay in S. K. Card. Cone trees: Animated 3d visualizations of hierarchical information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, strani 189–194, New York, NY, USA, 1991. ACM. ISBN 0-89791-383-3. doi: 10.1145/108844.108883.

[50] J. Heer, M. Bostock in V. Ogievetsky. A tour through the visualization zoo. http://homes.cs.washington.edu/~jheer/files/zoo.

[51] UC Berkeley Visualization Lab. Data visualization for the web. http://flare.prefuse.org, 2010.

[52] M. Bostock in J. Heer. Data-driven documents. http://d3js.org, 2013.

[53] IBM. Many eyes. http://many-eyes.com.

[54] E. M. Reingold in J. S. Tilford. Tidier drawing of trees. *IEEE Transactions on Software Engineering*, 7(1): 223–228, 1981.

[55] B. Johnson in B. Shneiderman. Treemaps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. 2nd International Visualization Conference 1991. IEEE*, strani 284–291, 1991.

[56] J .Lamping, R. Rao in P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, strani 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1. doi: 10.1145/223904.223956.

[57] J. W. Anderson. *Hyperbolic Geometry*, poglavje The Poincaré Disc Model, strani 95–104. New York: Springer-Verlag, 1999.

[58] D. Auber. *Tulip : A huge graph visualisation framework*. P. Mutzel in M. Junger, 2003. URL http://hal.archives-ouvertes.fr/hal-00307626.

[59] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, September 2006. ISSN 1077-2626. doi: 10.1109/TVCG.2006.147.

[60] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto in D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658, 2004.

[61] T. S. Evans in R. Lambiotte. Line graphs, link partitions in overlapping communities. *Physical Review E*, 80:016105, Jul 2009. doi: 10.1103/PhysRevE.80.016105.

[62] J. Yang in J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, strani 3:1–3:8, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1546-3. doi: 10.1145/2350190.2350193.

[63] J. Xie, S. Kelley in B. K. Szymanski. Overlapping community detection in networks: the state of the art and comparative study. *ACM Computing Surveys*, 45 (4):43, Avgust 2013.

[64] G. Palla, I. Derenyi, I. Farkas in T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[65] S. Fortunato in A. Lancichinetti. Community detection algorithms: A comparative analysis: Invited presentation, extended abstract. In *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '09, strani 27:1–27:2, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering). ISBN 978-963-9799-70-7. URL http://dl.acm.org/citation.cfm?id=1698822.1698858.

[66] J. Baumes, M. K. Goldberg, M. S. Krishnamoorthy, M. Magdon-Ismail in N. Preston. Finding communities by clustering a graph into overlapping subgraphs. In N. Guimaraes in P. T. Isaías, urednika, *IADIS AC*, strani 97–104. IADIS, 2005. ISBN 972-99353-6-X. URL http://dblp.uni-trier.de/db/conf/iadis/ac2005-1.html#BaumesGKMP05.

[67] S. Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10):103018, 2010. URL http://stacks.iop.org/1367-2630/12/i=10/a=103018.

[68] J. Xie in B. K. Szymanski. Towards linear time overlapping community detection in social networks. *CoRR*, abs/1202.2465, 2012.

[69] P. De Meo, E. Ferrara, G. Fiumara in A. Provetti. Generalized louvain method for community detection in large networks. 2011.

[70] V. D. Blondel, J.-L. Guillaume, R. Lambiotte in E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P10008, Julij 2008.

[71] A. Ferligoj in V. Batagelj. Clustering with relational constraint. *Psychometrika*, 47(4):413–426, 1982.

[72] V. Batagelj, A. Ferligoj in A. Mrvar. Hierarchical clustering in large networks. http://perso.uclouvain.be/vincent.blondel/workshops/2008/files/batagelj.pdf, 2008. International workshop on Detection and visualization of communities in large complex networks.

[73] A. Ferligoj in V. Batagelj. Some types of clustering with relational constraints. *Psychometrika*, 48(4): 541–552, 1983.

[74] G.-J. Kim, K.-Y. Whang, M.-S. Kim, H.-S. Lim, K.-H. Lee in B. S. Lee. Incremental clustering crawler for community-limited search. strani 438–445, Avgust 2009.

[75] L. Fu, D. Sun in L. R. Rilett. Heuristic shortest path algorithms for transportation applications: State of the art. *Computers & Operations Research*, 33: 3324–3343, 2006.

[76] F. Murtagh. *Multidimensional Clustering Algorithms*, chapter Synoptic Clustering, pages 59–88. Physica-Verlag, 1985.

[77] M. Bruynooghe. Méthodes nouvelles en classification automatique des données taxinomiques nombreuses. *Statistique et Analyse des Données*, (3):24–42, 1977.

[78] P. Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

[79] C. Herrera in P. J. Zufiria. Generating scale-free networks with adjustable clustering coefficient via random walks. *IEEE Network Science Workshop*, 0:167–172, 2011. URL http://doi.ieeecomputersociety.org/10.1109/NSW.2011.6004642.

[80] A.-L. Barabási. Network science, poglavje 4: The scale-free property, 2012. URL http://barabasilab.neu.edu/networksciencebook/downlPDF.html.

[81] A. A. Hagberg, D. A. Schult in P. J. Swart. Exploring network structure, dynamics in function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, strani 11–15, Pasadena, CA USA, Avgust 2008.

[82] V. Batagelj in M. Zaveršnik. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification*, 5(2): 129–145, 2011. doi: 10.1007/s11634-010-0079-y.

[83] J. Leskovec in A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, Junij 2014.

[84] C. Fellbaum, urednik. *WordNet: an electronic lexical database*. MIT Press, 1998.

[85] J. Bodlaj in V. Batagelj. Network analysis of publications on topological indices from the web of science. *Molecular Informatics*, 33(8):514–535, 2014. ISSN 1868-1751. doi: 10.1002/minf.201400014.

[86] G. Tibély, L. Kovanen, M. Karsai, K. Kaski, J. Kertész in J. Saramáki. Communities and beyond: Mesoscopic analysis of a large social network with complementary methods. *Phys. Rev. E*, 83:056125, Maj 2011. doi: 10.1103/PhysRevE.83.056125.

[87] M. K. Goldberg, M. Hayvanovych in M. Magdon-ismail. Measuring similarity between sets of overlapping clusters. In *Proceedings of the 2010 IEEE Second International Conference on Social Computing*, SOCIALCOM '10, strani 303–308, 2010.

[88] V. Batagelj in A. Ferligoj. Clustering relational data. *Data analysis*, strani 3–15, 2000.

[89] M. Štajdohar, M. Mramor, B. Zupan in J. Demšar. Fragviz: visualization of fragmented networks. *BMC Bioinformatics*, 11(1):475, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-475.

[90] B. J. T. Morgan in A. P. G. Ray. Non-uniqueness and inversions in cluster analysis. *Applied Statistics*, 44 (1):117–134, 1995.

[91] A. Fernández in S. Gómez. Solving non-uniqueness in agglomerative hierarchical clustering using multidendrograms. *Journal of Classification*, 25(1):43–65, Junij 2008. ISSN 0176-4268. doi: 10.1007/s00357-008-9004-x.

[92] J. Bodlaj, M. Cerinšek, in V. Batagelj. Visualization of traffic. In V. Blondel, N. de Cordes, A. Decuyper, P. Deville, J. Raguenez in Z. Smoreda, uredniki, *Analysis of mobile phone datasets for the development of Ivory Coast*, strani 480–495, 2013. URL www.netmob.org.

[93] V. D. Blondel, M. Esch, C. Chan, F. Cleroty, P. Deville, E. Huens, F. Morloty, Z. Smore-day in C. Ziemlickiy. Data for developement: The d4d challenge on mobile phone data, September 2012. arXiv:1210.0137v1.

[94] M. Cerinšek, J. Bodlaj in V. Batagelj. Symbolic clustering of users and antennae. In V. Blondel, N. de Cordes, A. Decuyper, P. Deville, J. Raguenez in Z. Smoreda, uredniki, *Analysis of mobile phone datasets for the development of Ivory Coast*, strani 211–226, 2013. URL www.netmob.org.

[95] N. Kejžar, S. Korenjak-Černe in V. Batagelj. Clustering of distributions: A case of patent citations. *Journal of Classification*, 28:156–183, 2011. doi: 10.1007/s00357-011-9084-x.

[96] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik in B. Zupan. Orange: Data mining toolbox in python. *Journal of Machine Learning Research*, 14:2349–2353, 2013. URL http://jmlr.org/papers/v14/demsar13a.html.

[97] Netminer. URL www.netminer.com/index.php.

[98] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski in T. Ideker. Osgi alliance cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res*, 13(11):2498–2504, 2003.

[99] M. Bastian, S. Heymann in M. Jacomy. Gephi: An open source software for exploring and manipulating networks. 2009. URL www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154.

[100] NodeXL. URL http://nodexl.codeplex.com.

[101] A. Mrvar in V. Batagelj. *Pajek and Pajek-XXL: Programs for Analysis and Visualization of Very Large Networks, Reference Manual*, Marec 2014.

[102] J. Smart, K. Hock in S. Csomor. *Cross-platform GUI programming with wxWidgets*. Bruce Perens' Open Source series. Prentice Hall, 2006. ISBN 9780131473812. URL http://books.google.co.uk/books?id=CyMsvtgnq0QC.

[103] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva in H. T. Vo. Vistrails: Enabling interactive multiple-view visualizations. *IEEE Visualization*, strani 135–142, 2005.

[104] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, Junij 1987. ISSN 0167-6423. doi: 10.1016/0167-6423(87)90035-9.

[105] A. Silberschatz, P. B. Galvin in G. Gagne. *Operating System Concepts*, poglavje Process Scheduling; Round Robin Scheduling, stran 194. John Wiley & Sons (Asia), 8. izdaja, 2010. ISBN 978-0-470-23399-3.

[106] OpenGL® library. URL www.opengl.org.

[107] J. Kessenich. The OpenGL® shading language, Julij 2009.

[108] R. S. Jr. Wright, N. Haemel, G. Sellers in B. Lipchak. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley Professional, 5. izdaja, 2010. ISBN 0321712617, 9780321712615.

[109] J. Brown in R. Churchill. *Complex Variables and Applications*. New York: McGraw-Hill, 1989. ISBN 0-07-010905-2.

[110] J. Bodlaj in V. Batagelj. Network analysis of publications on topological indices from web of science. In *Abstracts of the Workshop COMPUTERS IN SCIENTIFIC DISCOVERY 6*, sran 32, Portorož (Slovenia), 2012.

[111] J. Bodlaj in V. Batagelj. Hierarchical link clustering of networks. In *1st European Conference on Social Networks, Abstract book*, stran 29, Barcelona (Spain), 2014.

[112] J. Bodlaj in V. Batagelj. Hierarchical link clustering algorithm in networks. *Unpublished*.

[113] J. Bodlaj, V. Batagelj in A. Orbanić. OpenGL® graphical user interface for net.plexor – interactive large network analysis library. In *Proceedings of the 20th International Electrotechnical and Computer Science Conference*, strani 139–142, Portorož (Slovenija), 2011.

[114] J. Bodlaj, V. Batagelj in A. Orbanić. OpenGL® graphical user interface for net.plexor – interactive large network analysis library. In *Abstracts of the 7th Slovenian international conference on graph theory*, stran 148, Bled (Slovenia), 2011.

[115] J. Bodlaj. *Encyclopedia of Social Network Analysis and Mining*, zvezek 2, poglavje Network Data File Formats, strani 1076–1091. Springer, 2014.

[116] Dodatek. URL http://zvonka.fmf.uni-lj.si/netbook/doku.php?id=pub:esnam.

[117] V. Batagelj. Wos2pajek, 2008. URL http://pajek.imfm.si/doku.php?id=wos2pajek.

[118] Extensible markup language (xml). URL www.w3.org/TR/REC-xml.

[119] Tom sawyer perspectives. URL www.tomsawyer.com/home/index.php.

[120] The DGS file format specification. URL http://graphstream-project.org/doc/Advanced-Concepts/The-DGS-File-Format.

[121] The GraphML file format, . URL http://graphml.graphdrawing.org.

[122] V. Batagelj in A. Mrvar. Towards netml network markup language. In *International Social Network Conference*, London, Julij 1995. URL http://vlado.fmf.uni-lj.si/pub/networks/netml/snetml.pdf.

[123] C. Stein, T. H. Cormen, C. E. Leiserson in R. L. Rivest. *Introduction to Algorithms*. PHI Learning, 3. izdaja, 2010.

[124] K. Mehlhorn in P. Sanders. *Data Structures and Algorithms, The Basic Toolbox*. Springer, 2008.

[125] S. P. Borgatti, M. G. Everett in L. C. Freeman. *UCINET 6 for Windows, Software for Social Network Analysis*, 2002.

[126] Gml: A portable graph file format. Tehnično poročilo. URL www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf.

[127] GraphViz, the dot language, . URL www.graphviz.org/content/dot-language.

[128] M. Round. URL MDRound@qinetiq.com.

[129] J. Heer in M. Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, 2010. URL http://dblp.uni-trier.de/db/journals/tvcg/tvcg16.html#HeerB10.

[130] A. Godiyal, J. Hoberock, M. Garland in J. C. Hart. Rapid multipole graph drawing on the gpu. In I. G. Tollis in M. Patrignani, urednika, *Graph Drawing*, zvezek 5417 v *Lecture Notes in Computer Science*, strani 90–101. Springer, 2008. ISBN 978-3-642-00218-2. URL http://dblp.uni-trier.de/db/conf/gd/gd2008.html#GodiyalHGH08.

[131] A. Godiyal, J. Hoberock, M. Garlandy in J. C. Hart. Rapid multipole graph drawing on the gpu, 2009.

[132] Y. Frishman in A. Tal. Multi-level graph layout on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 13:1310–1319, 2007.

[133] R. Tsuchiyama, T. Nakamura, T. Iizuka, A. Asahara in S. Miki. *The OpenCL Programming Book*. Fixstars Corporation, 2010.

[134] Khronos OpenCL Working Group. The OpenCL specification, 2008.

[135] D. P. Rodgers. *Improvements in multiprocessor system design*, zvezek 13, strani 225—231. ACM SIGARCH Computer Architecture News archive (New York, NY, USA: ACM), Junij 1985. doi: 10.1145/327070.327215. ISBN 0-8186-0634-7, ISSN 0163-5964.