UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

Renata Vaderna

Algoritmi i jezik za podršku automatskom raspoređivanju elemenata dijagrama

DOKTORSKA DISERTACIJA

Novi Sad, (2018)

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска публикација
Тип записа, **ТЗ**:		Текстуални штампани документ
Врста рада, **ВР**:		Докторска дисертација
Аутор, **АУ**:		Рената Вадерна
Ментор, **МН**:		Др Игор Дејановић
Наслов рада, **НР**:		Алгоритми и језик за подршку аутоматском распоређивању елемената дијаграма
Језик публикације, **ЈП**:		Српски
Језик извода, **ЈИ**:		Српски / енглески
Земља публиковања, **ЗП**:		Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		2018
Издавач, **ИЗ**:		Факултет техничких наука
Место и адреса, **МА**:		Нови Сад, Трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		13/168/182/93/0/0
Научна област, **НО**:		Електротехничко и рачунарско инжењерство
Научна дисциплина, **НД**:		Примењене рачунарске науке и информатика
Предметна одредница/Кqучне речи, **ПО**:		Теорија графова, цртање графова, језици специфични за домен
УДК		
Чува се, **ЧУ**:		Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, 21000 Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		У склопу докторске дисертације извршено је истраживање везано за аутоматско распоређивање елемената дијаграма. Кроз анализу постојећих решења уочен је простор за побољшања, посебно по питању разноврсности доступних алгоритама и помоћи кориснику при избору најпогоднијег од њих. У оквиру истраживања проучаван, имплементиран и у појединим случајевима унапређен је широк спектар алгоритама за цртање и анализу графова. Дефинисан је поступак аутоматског избора одговарајућег алгоритма за распоређивање елемената графова на основу њихових особина. Додатно, осмишљен је језик специфичан за домен који корисницима графичких едитора пружа помоћ у избору алгоритма за распоређивање, а програмерима брже писање кода за позив жељеног алгоритма.
Датум прихватања теме, **ДП**:		13.7.2017.
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	Др Бранко Перишић, редовни професор
	Члан:	Др Владан Девецић, редовни професор
	Члан:	Др Драган Машуловић, редовни професор
	Члан:	Др Гордана Милосављевић, ванредни професор
	Члан, ментор:	Др Игор Дејановић, ванредни професор

Потпис ментора

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Образац **Q2.HA.06-05**- Издање 1

Accession number, **ANO**:		
Identification number, **INO**:		
Document type, **DT**:		Monographic publication
Type of record, **TR**:		Textual material
Contents code, **CC**:		PhD thesis
Author, **AU**:		Renata Vaderna
Mentor, **MN**:		PhD Igor Dejanović
Title, **TI**:		Algorithms and a language for the support of automatically laying out diagram elements
Language of text, **LT**:		Serbian
Language of abstract, **LA**:		Serbian / English
Country of publication, **CP**:		Serbia
Locality of publication, **LP**:		Vojvodina
Publication year, **PY**:		2018
Publisher, **PB**:		Faculty of Technical Sciences
Publication place, **PP**:		Novi Sad, Trg Dositeja Obradovića 6
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		13/168/182/93/0/0
Scientific field, **SF**:		Electrical engineering and computing
Scientific discipline, **SD**:		Applied computer science and informatics
Subject/Key words, **S/KW**:		Graph theory, graph drawing, domain-specific languages
UC		
Holding data, **HD**:		Library of Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad
Note, **N**:		
Abstract, **AB**:		This thesis presents a research aimed towards the problem of automatically laying out elements of a diagram. The analysis of existing solutions showed that there is some room for improvement, especially regarding variety of available algorithms. Also, none of the solutions offer possibility of automatically choosing an appropriate graph layout algorithm. Within the research, a large number of different algorithms for graph drawing and analysis were studied, implemented, and, in some cases, enhanced. A method for automatically choosing the best available layout algorithm based on properties of a graph was defined. Additionally, a domain-specific language for specifying a graph's layout was designed.
Accepted by the Scientific Board on, **ASB**:		13.7.2017.
Defended on, **DE**:		
Defended Board, **DB**:	President:	Branko Perišić, Ph.D., Full Professor
	Member:	Vladan Devežić, Ph.D., Fulll Professor
	Member:	Dragan Mašulović, Ph.D., Full Professor
	Member:	Gordana Milosavljević, Ph. D., Associate Professor
	Member, Mentor:	Igor Dejanović, Ph. D., Associate Professor

Menthor's sign

Obrazac **Q2.HA.06-05**- Izdanje 1

Sadržaj

1 Uvod **1**
- 1.1 Predmet i zadaci istraživanja . . . 2
- 1.2 Motivacija istraživanja . . . 3
- 1.3 Ciljevi istraživanja . . . 4
- 1.4 Primenjene metode . . . 5
- 1.5 Struktura projektnog rešenja . . . 6
- 1.6 Struktura disertacije . . . 6

2 Osnovni pojmovi **9**
- 2.1 Uvod u teoriju grafova . . . 9
- 2.2 Tipovi grafova i njihove osobine . . . 12
  - 2.2.1 Pravilni grafovi, potpuni grafovi i konture . . . 12
  - 2.2.2 Stabla . . . 13
  - 2.2.3 Planarni grafovi . . . 14
- 2.3 Algoritamska kompleksnost . . . 15
- 2.4 Algoritmi za analizu grafova . . . 16
- 2.5 Estetski kriterijumi za crtanje grafova . . . 17
  - 2.5.1 Presecanje grana . . . 17
  - 2.5.2 Minimalni uglovi . . . 18
  - 2.5.3 Prelomi . . . 18
  - 2.5.4 Tok . . . 18
  - 2.5.5 Ortogonalnost . . . 19
  - 2.5.6 Simetrija . . . 20
  - 2.5.7 Ostali kriterijumi . . . 20
- 2.6 Pregled klasa algoritama za automatsko raspoređivanje . . . 21
  - 2.6.1 Algoritmi za crtanje stabala . . . 21
  - 2.6.2 Hijerarhijski algoritmi za raspoređivanje . . . 24
  - 2.6.3 Algoritmi za kružno raspoređivanje . . . 25
  - 2.6.4 Algoritmi za simetrično crtanje grafova . . . 26
  - 2.6.5 Planarni pravolinijski algoritmi . . . 28
  - 2.6.6 Silom usmereni algoritmi za raspoređivanje . . . 30
  - 2.6.7 Ostale klase algoritama za raspoređivanje . . . 31
- 2.7 Jezici specifični za domen . . . 32

3 Pregled postojećih rešenja **35**
- 3.1 Biblioteke za vizualizaciju grafova za programski jezik Java . . . 35
  - 3.1.1 Pregled postojećih biblioteka . . . 35
  - 3.1.2 Analiza i poređenje implementacija algoritama . . . 38
  - 3.1.3 Pozivanje od strane zasebnih grafičkih editora . . . 41

3.2 Jezici specifični za domen opisa grafova 44

4 Implementacija algoritama za analizu grafova 47
4.1 Algoritmi za obilazak grafova 48
4.2 Povezanost grafova 51
4.2.1 Ispitivanje jednostruke, dvostruke i trostruke povezanosti grafova 51
4.2.2 Nalaženje artikulacionih čvorova grafova i dvostruko povezanih komponenti 51
4.2.3 Konstrukcija BC-stabala 52
4.2.4 Podela grafova na trostruko povezane komponente 53
4.2.5 Kontrukcija SPQR-stabala 56
4.3 Algoritmi za ispitivanje cikličnosti grafova 58
4.3.1 Cikličnost neusmerenih grafova 58
4.3.2 Cikličnost usmerenih grafova 59
4.4 Nalaženje uređenja grafa 60
4.5 Algoritmi za ispitivanje planarnosti i planarno utapanje 62
4.5.1 De Frajse - Osona de Mendez - Rozenštil (de Fraysseix - Ossona de Mendez - Rosenstiehl) algoritam 63
4.5.2 Bojer-Mirvold (Boyer-Myrvold) algoritam 65
4.5.3 Nalaženje planarnih lica na osnovu planarnog utapanja 68
4.5.4 Planarna augmentacija 69
4.6 Simetrija grafova 72
4.6.1 Makejev algoritam za kanoničko označavanje grafa i nalaženje grupe automorfizama 74
4.7 Provera ostalih osobina 77

5 Implementacija algoritama za automatsko raspoređivanje 79
5.1 Odabir algoritama za implementaciju 79
5.2 Kružno raspoređivanje sa minimizacijom broja preseka grana 80
5.2.1 Nalaženje poretka čvorova radi smanjenja broja preseka grana . . 80
5.2.2 Nalaženje poluprečnika kruga i pozicije čvorova 81
5.2.3 Primer 83
5.3 Crtanje grafova upotrebom Tutovog algoritma za utapanje planarnih tripovezanih grafova 84
5.3.1 Tutov algoritam 84
5.3.2 Raspoređivanje upotrebom Tutovog algoritma 86
5.4 Linearni algoritam za konveksno crtanje grafova 87
5.4.1 Provera konveksnosti 87
5.4.2 Nalaženje proširivih ciklusa lica 91
5.4.3 Konveksno crtanje 94
5.5 Ortogonalno crtanje grafova 97
5.6 Simetrično crtanje grafova 99
5.6.1 Algoritam Kara i Kokaja 100
5.6.2 Implementacija simetričnog raspoređivanja 101
5.7 Ostali algoritmi 102
5.8 Pregled naučnih i stručnih doprinosa na polju crtanja grafova 104

6 Automatski izbor algoritma za raspoređivanje 105
6.1 Implementacija automatskog raspoređivanja 106

7 Jezik specifičan za domen opisa crteža grafa **111**
7.1 Razvojne faze jezika . . . 111
7.1.1 Odluka . . . 111
7.1.2 Analiza domena . . . 112
7.1.3 Dizajn jezika . . . 112
7.2 Integracija jezika sa *GRAD* bibliotekom . . . 121
7.3 Primeri . . . 123

8 Grafički editor **127**

9 Integracija rešenja sa postojećim grafičkim editorima **133**
9.1 Pregled API-ja za crtanje grafova . . . 133
9.2 Primeri koda . . . 135

10 Primena rešenja **137**
10.1 Diskusija o mogućim primenama . . . 137
10.2 Primer korišćenja u okviru Kroki alata . . . 138
10.3 Integracija sa popularnim alatima . . . 141

11 Evaluacija **147**
11.1 Evaluaciona procedura . . . 147
11.2 Učesnici studije . . . 149

12 Zaključak **155**

Rezime

U ovoj disertaciji predstavljeno je istraživanje usmereno ka problemu automatskog raspoređivanja elemenata grafova, obuhvatajući izučavanje i implementaciju raznih algoritama ovog tipa, te osmišljavanje i realizaciju načina za pojednostavljenu selekciju, konfiguraciju i primenu odgovarajućeg algoritma.

Algoritmi za automatsko raspoređivanje elemenata grafova imaju za cilj kreiranje crteža grafova koji se odlikuju lepotom i čitljivošću. Postoji veliki broj spomenutih algoritama, uključujući one koji su osmišljeni tako da daju najbolji mogući rezultat ukoliko se primene nad grafovima koji poseduju određene osobine, i one znatno opštije, koji teže generisanju kvalitetnih crteža bez obzira na tip i veličinu grafa.

Izbor i konfiguracija odgovarajućeg algoritma bez poznavanja teorijskih osnova zahteva dosta eksperimentisanja, što je inspirisalo cilj automatizacije datog postupka.

Predložena su dva načina za rešavanje ovog problema: prvi obuhvata učešće korisnika, ali na način koji ne zahteva dobro poznavanje oblasti, a drugi predstavlja potpunu automatizaciju izbora. Prva opcija implementirana je osmišljavanjem jezika specifičnog za domen opisa prostornog raspoređivanja elemenata grafa, dok je druga realizovana analizom grafa radi provere da li poseduje osobine koji pojedini algoritmi za automatsko raspoređivanje zahtevaju.

Prethodno pomenuti načini za automatski izbor i primenu algoritama ne bi bili mogući bez postojanja implementacija niza algoritama za analizu i crtanje grafova. Kako već postoji određen broj biblioteka za programski jezik *Java*, koje se, između ostalog, bave i crtanjem grafova, one su analizirane sa ciljem pronalaska nekih od potrebnih implementacija. Uočena je potreba za implementacijom većeg broja algoritama za analizu grafova, kao i da su određene klase algoritama za automatsko raspoređivanje slabo zastupljene, ako ih uopšte ima. Takođe, primećeno je i da postojeće biblioteke mahom čvrsto povezuju raspoređivanje elemenata sa komponentama za vizualizaciju, tako da je njihova integracija sa posebno razvijenim grafičkim editorima kompleksnija nego što bi mogla biti. Navedeni problemi inspirisali su razvoj nove bliboteke za crtanje i analizu grafova za programski jezik *Java* nazvanu *GRAD*.

GRAD biblioteka uključuje implementacije bar jednog predstavnika najvažnijih klasa algoritama za crtanje grafova, kao i neophodnih algoritama za njihovu analizu. Biblioteka obuhvata jednostavni grafički editor za isprobavanje i upoznavanje sa raznim algoritmima, ali i spomenuta dva načina za pojednostavljeni izbor i konfiguraciju odgovarajućeg. Takođe, omogućena je veoma jednostavna integracija bilo koje od funkcionalnosti *GRAD* biblioteke sa postojećim grafičkim editorima.

Mogućnost primene rešenja je izuzetno velika. Naime, algoritmi za crtanje grafova, kao i njihova automatska selekcija, mogu se iskoristiti unutar bilo kog grafičkog editora ukoliko je pisan na programskom jeziku *Java* ili podržava spregu sa njim.

Summary

In this thesis a research aimed at solving the problem of automatically laying out elements of graphs is presented. It includes analysis and implementation of various graph drawing algorithms, as well as possible ways of simplifying selection and configuration of the appropriate algorithm.

Algorithms for automatically laying out graph elements strive to create aesthetically pleasing and understandable drawings of graphs. There is large number of such algorithms, including those which focus on a specific type of graphs, and more general ones, which should produce nice drawings of graphs, regardless of their type and size. It can be quite difficult to choose the most suitable algorithm without being knowledgeable in graph and graph drawing theory. This fact inspired the idea of automating this selection.

Two methods of solving this problem are proposed: one which enables users without much theoretical knowledge intuitive specification of desirable properties of the resulting drawing, and complete automatisation of the selection and execution of the algorithm. The first alternative was implemented by designing a domain-specific language for description of a graph's layout, while the other one relies on analysis of the graph, which determines if it has properties which specific layout algorithms require.

Implementation of the mentioned ways of automatically choosing and executing algorithms would not be achievable without implementations of numerous algorithms for graph analysis and drawing. There already exist a few libraries for the Java programming language which offer such implementations. However, it can be noted that none of them offer a significant number of graph analysis algorithms, as well as that they only implement graph drawing algorithms belonging to a small number of different classes. Furthermore, the existing libraries strongly couple the layout feature with visualization components, making their integration with already existing graphical editors overly complex. The mentioned problems inspired development of a new graph analysis and drawing library for the Java programming language called *GRAD*.

The *GRAD* library provides an implementation of at least one member of each of the most important classes of layout algorithm and the necessary analysis algorithms. Moreover, the library includes a simple graphical editor for familiarization with graph drawing algorithms and the two mentioned methods of automatically selecting and configuring the most appropriate one. On top of that, it can easily be integrated with any graphical editor.

There are many practical usages for the results of the research described in this dissertation. Graph drawing algorithms and their automatic selection can be used inside any graphical editor as long as it was written in Java, or a different language which can be integrated with Java.

Poglavlje 1

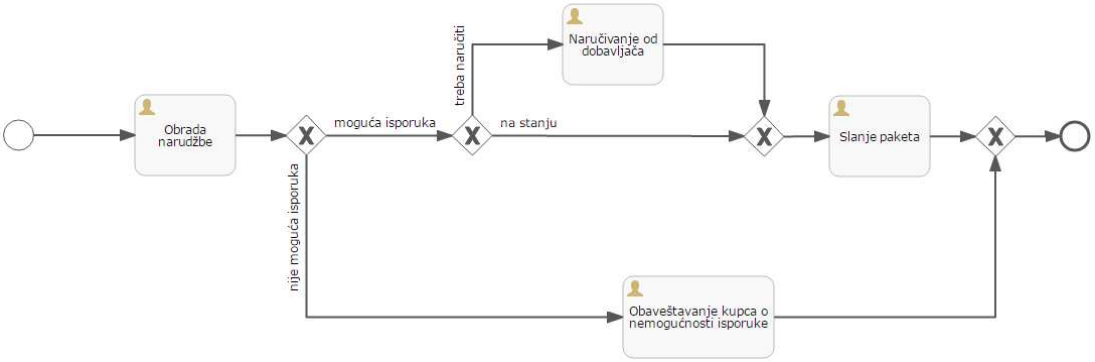
Uvod

Mnogi problemi realnog sveta mogu biti opisani pomoću dijagrama koji se sastoje od povezanih skupova tačaka - grafova. Jedan od najstarijih poznatih problema ovog tipa je problem kenigsberških mostova. Naime, grad Kenigsberg u tadašnjoj Prusiji (današnji ruski Kalinjingrad) izgrađen je na ušću reke Pregel i sastoji se iz dva velika ostrva povezana međusobno, i sa kopnom putem sedam mostova. Građani Kenigsberga pokušavali su da odgovore na pitanje da li je moguće preći preko svih sedam mostova, tako da se nijedan ne pređe dva ili više puta. Problem je 1736. godine rešio čuveni švajcarski matematičar Leonard Ojler (Leonhard Euler), pri čemu je postavio temelje teorije grafova [1]. Drugi klasični primer je problem putujućeg prodavca, koji su u 19. veku formulisali irski matematičar Vilijam Hamilton (William Rowan Hamilton) i Britanac Tomas Kirkman (Thomas Penyngton Kirkman), takođe matematičar. Pitanje koje se postavlja jeste kako na osnovu liste gradova i udaljenosti između njih može da se nađe najkraći put koji obilazi svaki grad tačno jednom i vraća se u polazni. Sa druge strane, mnogobrojni su primeri korisnosti pomenute predstave i u modernom svetu i u nauci. Samo neki od njih uključuju analizu povezanosti korisnika na društvenim mrežama, proučavanje putanja migracije neke vrste u biologiji, projektovanje digitalnih kola visokog stepena integracije, a u poslednjih nekoliko godina grafovi postaju sve popularniji kod oblasti u ekspanziji kao što je analiza podataka [2].

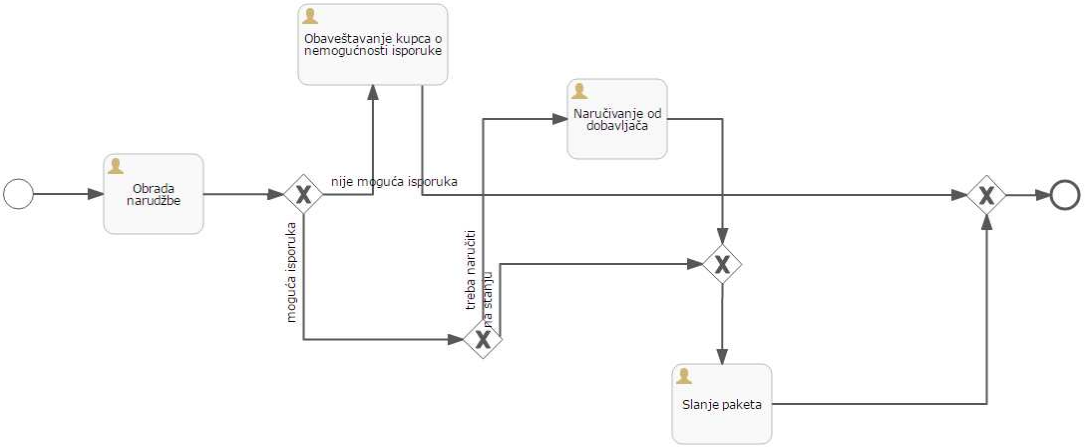
Grafovi su strukture podataka pogodne za kreiranje grafičke reprezentacije, čime se olakšava razumevanje njihovih osobina [3]. Za određene primene, poput analize društvenih mreža i kartografije, vizualizacija strukturnih informacija u formi grafova je od velike važnosti. Uzimajući široku upotrebnu vrednost u obzir, nije iznenađujuće da se crtanje grafova pojavilo kao posebna oblast matematike i računarske nauke. Ova oblast spaja metode geometrijske teorije grafova i vizualizacije informacija kako bi opisala kreiranje dvodimenzionalnih predstava grafova [4]. Događaji poput Internacionalnog simpozijuma o crtanju grafova i vizualizaciji mreža [5], koji se organizuje svake godine počevši od 1992., svedoče o tome da se ta oblast i dalje razvija.

Svaki dijagram koji se sastoji iz povezanih elemenata može se smatrati grafom - dijagram klasa, aktivnosti, slučajeva korišćenja, poslovnih procesa itd. Jednom grafu može odgovarati veliki broj crteža i ovi pojmovi se nikako ne smeju poistovetiti. Teorijski gledano, bitno je samo koji je čvor grafa, odnosno, element dijagrama sa kojim povezan, dok u praksi njihov raspored u okviru crteža direktno utiče na čitljivost i mogućnost razumevanja datog grafa [6]. Naime, dijagrami se često kreiraju ne samo uz poštovanje neke formalne notacije, nego i uz težnju pridržavanja nekoj sekundarnoj. Sekundarna notacija se definiše kao skup vizuelnih propisa koji nisu delovi formalne [7]. U grafičkom kontekstu, koristi se za povećavanje preglednosti formalne notacije,

definišući obeležja poput boja i rasporeda elemenata. Studija opisana u [8] bavi se uticajem sekundarne notacije na modele poslovnih procesa, izdvajajući upravo raspored elemenata kao najznačajniji faktor, koji određuje koliko efikasno kako početnici, tako i eksperti mogu analizirati model. Opravdanost ove tvrdnje se može naslutiti poređenjem modela sa slika 1.1 i 1.2. Iako obe slike prikazuju isti model, raspored elemenata čini prvu znatno jasnijom.



Slika 1.1: Primer dobrog rasporeda elemenata modela jednog poslovnog procesa



Slika 1.2: Primer lošeg rasporeda elemenata modela jednog poslovnog procesa

Ručno raspoređivanje čvorova grafa, tako da crtež bude pregledan, ne predstavlja poseban izazov, ali se ne može reći isto i za automatizaciju ovog postupka. Proces kreiranja crteža grafa, znajući samo iz kojih čvorova i grana se sastoji, naziva se automatsko raspoređivanje.

1.1 Predmet i zadaci istraživanja

Od sredine prošlog veka do danas pojavio se veoma velik broj raznovrsnih algoritama za crtanje grafova, koji se razlikuju ne samo po kompleksnosti i efikasnosti, nego i po izgledu crteža koje proizvode. Naime, različiti algoritmi fokusiraju se na različite principe dizajna, odnosno, estetske kriterijume. Na primer, neki rezultuju crtežima kod kojih su čvorovi kružno raspoređeni, drugi vizualizacijom grafa gde su svi uglovi pravi, a treći hijerarhijski uređenim elementima. Dodatno, određeni algoritmi za automatsko

raspoređivanje nisu predviđeni za primenu nad proizvoljnim grafovima, nego samo nad onima koji poseduju tražene osobine. Međutim, crteži kreirani njihovom upotrebom u tim specijalnim slučajevima neretko bivaju daleko bolji od onih koji bi se dobili ukoliko bi se izvršio algoritam šireg domena primene. Dakle, može se primetiti da obeležja grafa direktno utiču na mogućnost i optimalnost primene određenih algoritama. Ova činjenica predstavlja osnovu automatizacije procesa odabira odgovarajućeg algoritma, što je jedan od zadataka disertacije.

Ocena jasnoće ili lepote nekog crteža grafa nosi sa sobom i izvesnu dozu subjektivnosti, te kompletna automatizacija postupka raspoređivanja nije nužno i najbolje rešenje. Sa druge strane, ručni izbor algoritma koji bi kreirao crtež čiji je izgled u skladu sa željama korisnika zahteva ili poznavanje oblasti crtanja grafova ili podrobno eksperimentisanje. Nalaženje kompromisa između dve spomenute krajnosti (pune automatizacije i potpuno samostalnog korisničkog izbora algoritma) inspirisalo je zadatak kreiranja jezika specifičnog za domen, putem kojeg bi korisnik deskriptivno izrazio svoje preference.

Svakako, dva navedena zadatka ne bi mogla biti uspešno realizovana bez postojanja dobrih i efikasnih implementacija raznih algoritama za crtanje i analizu grafova. Dakle, istraživanje je usmereno na:

- Proučavanje algoritama za analizu i crtanje grafova, uz pregled postojećih rešenja i identifikaciju i implementaciju algoritama bez pouzdane postojeće implementacije na ciljnom programskom jeziku, Javi.
- Automatski izbor i konfiguraciju algoritma za crtanje na osnovu osobina grafa.
- Dizajniranje i implementaciju jezika specifičnog za domen specifikacije poželjnih osobina crteža.
- Mogućnost upotrebe rešenja unutar alata za modelovanje i biblioteka za vizualizaciju podataka.

1.2 Motivacija istraživanja

Motivacija za razvoj biblioteke za analizu i crtanje grafova izrodila se iz potrebe za raspoređivanjem elemenata u okviru već postojećeg grafičkog editora. Nije veliki problem naći alate i biblioteke, kako za programski jezik Java, tako i za druge jezike posvećene vizualizaciji grafova i dijagrama druge vrste. Samo nekoliko takvih primera su projekti [9]–[13]. Međutim, njihov glavni cilj nije u potpunosti u skladu sa gorenavedenim problemom. Naime, ako bi se posmatrale funkcionalnosti na koje spomenuti projekti najviše stavljaju akcenat, mogle bi se izdvojiti sledeće:

- Generisanje statičkih slika u raznovrsnim formatima na osnovu opisa grafa.
- Generisanje adaptabilnih prikaza grafa unutar posebno napravljenih komponenti za vizualizaciju.
- Podrška za kreiranje potpuno funkcionalnih grafičkih editora, uključujući i specifikaciju gradivnih elemenata dijagrama.
- Omogućavanje istraživanja i analize grafova i mreža po njihovoj vizualizaciji.

Dakle, integracija sa posebno razvijenim grafičkim editorima nije među njihovim prioritetima. Treba napomenuti da u sklopu vizualizacije obično postoji i podrška za automatsko raspoređivanje. Međutim, njeno korišćenje u kontekstu postojećeg editora je teže nego što bi se očekivalo, što se može videti iz primera koda koji će u disertaciji biti prikazani i prokomentarisani. Štaviše, u nekim slučajevima je čak i skoro neizvodljivo.

Prirodno je postaviti pitanje zašto bi se uopšte i gradili zasebni grafički editori bez korišćenja nekog od ovih rešenja, te tako i došlo u situaciju da se biblioteke moraju koristiti na način na koji njihova upotreba nije predviđena. Razloga ima više, a pre svih se ističu želje autora za više slobode pri dizajnu editora, kao i njegove eventualne specifičnosti, koje bi zahtevale implementaciju mnoštva dodatnih funkcionalnosti. Takođe, treba imati u vidu već napravljene grafičke editore, kao i činjenicu da pojedini alati za modelovanje ostavljaju mogućnost uključivanja dodatnih algoritma za automatsko raspoređivanje u vidu priključka. Priključak je softverska komponenta koja proširuje postojeću aplikaciju novom funkcionalnošću.

Osim poteškoća pri integraciji, primećeno je i da postoji dosta prostora za poboljšanje po pitanju samih implementacija algoritama za raspoređivanje. Iako se mogu naći slobodno dostupne kvalitetne implementacije, uočava se da postoji problem kada je o raznovrsnosti reč. Skup algoritama koji se najčešće pojavljuju ograničen je, pa bi njegovo proširenje implementacijama drugih algoritama bilo poželjno. Klase algoritama za crtanje grafova poput ortogonalne i simetrične, koje imaju značajnu praktičnu primenu, nemaju nijednog svog predstavnika u analiziranim postojećim alatima i bibliotekama.

Takođe, ne može se očekivati od svakog korisnika proizvoljnog grafičkog editora poznavanje algoritama za automatsko raspoređivanje, niti razumevanje uticaja njihovih parametara na konačan izgled dijagrama. Upravo je ovo polazna motivacija za osmišljavanje načina pomoći korisniku pri izboru algoritma, zavisnosti od dijagrama čije elemente želi da rasporedi.

Inspiracija za razvoj jezika putem čijih pravila bi se na desktriptivan način mogle izneti želje o konačnom izgledu crteža proistekla je iz činjenice da je skoro nemoguć zadatak definisati metriku kojom bi se rangirali razni estetski kriterijumi. Naime, rađene su studije kojima se ispituje koliko koja estetika utiče na jasnoću crteža pojedinih tipova grafova, ali su takva istraživanja još u ranim fazama. Dodatno, pojmovi lepote i razumljivosti pre svega su subjektivni, dok za različite tipove dijagrama postoje različite konvencije crtanja. Na primer, jednu električnu šemu i dijagram aktivnosti, u čijoj pozadini je isti apstraktni graf, domenski eksperati sigurno ne bi nacrtali na isti način. Korisniku se jezikom u potpunosti prepušta specifikacija izgleda koji mu odgovara u kontekstu vizualizacije određenih informacija. Treba naglasiti da jezik nema za cilj usresređivanje na pojedine uže oblasti, što bi omogućilo i uključivanje semantike. Međutim, jezik zasnovan na estetskim kriterijumima postavlja temelje kasnije razvoja uže specijalizovanih jezika.

1.3 Ciljevi istraživanja

Primarni cilj istraživanja jeste definisanje pravila za automatizovani izbor i primenu pogodnog algoritma za raspoređivanje elemenata grafa, odnosno, odgovarajuće kombinacije više njih, na osnovu:

- osobina datog grafa, odnosno
- želja korisnika koje bi se izražavale putem posebno razvijenog jezika.

Automatizacija postupka raspoređivanja elemenata grafa na navedeni način podrazumeva postojanje potrebnih agloritama za analizu i crtanje grafova. Dakle, implementacija spomenuth algoritama bila je temelj istraživanja. Osim implementacije, cilj istraživanja jeste i unapređenje ili preciznije definisanje izdvojenih algoritama. Naime, veliki broj radova koji prezentuju određeni algoritam iz teorije grafova ne ulazi u detalje realizacije određenih koraka ili pretpostavljaju da su kao ulaz dostupne informacije o grafu čije otkrivanje nije trivijalno. Efikasna implementacija nedostajućih elemenata stoga je izdvojena kao dodatni cilj.

Izbor određenih algoritama za analizu grafova radi implementacije diktiran je neophodnošću primene unutar algoritama za crtanje, odnosno, u okviru postupka za automatizovani izbor algoritma. Sa druge strane, sami algoritmi za crtanje grafova birani su analizom postojećih rešenja i detekcijom klasa algoritama ovog tipa koje su slabo ili nimalo zastupljene. Time se može navesti još jedan cilj istraživanja, a to je implementacija bar jednog predstavnika svake od najbitnijih grupa. Na slici 1.3 prikazani su ciljevi i odnos među njima.

Slika 1.3: Ciljevi i njihov međusobni odnos

Konačno, implementirani algoritmi, automatizovani izbor i jezik objedinjeni su u biblioteku za programski jezik *Java*, dizajniranu sa ciljem lakoće upotrebe unutar bilo kojeg alata za modelovanje.

1.4 Primenjene metode

Postojeći algoritmi za crtanje grafova analizirani su na osnovu računarske efikasnosti (pre svega vremena potrebnog za izvršavanje prilikom primene nad grafovima sa bar 1000 čvorova) i stepena pridržavanja raznih estetskih kriterijuma. Algoritmi, odnosno njihove implementacije, koje su se najbolje pokazale u poređenju sa istim ili sličnim algoritmima drugih biblioteka, izdvojeni su radi uključivanja u ciljnu biblioteku. Posmatrane su samo biblioteke za programski jezik *Java* nerestriktivnih licenci, koje dozvoljavaju upotrebu bilo kojih elemenata razmatranog softverskog rešenja u okviru drugih. Izdvajanje klasa algoritama za raspoređivanje elemenata grafova čiji predstavnici su implementirani u disertaciji izvršeno je izučavanjem popularnih klasa i nalaženjem onih koje imaju znatnu praktičnu primenu, ali ih pomenute biblioteke ne

podržavaju.

Potpuna automatizacija postupka izbora, konfiguracije i primene odgovarajućeg algoritma postignuta je detaljnom analizom algoritama za crtanje i uočavanjem osobina koje graf treba da poseduje da bi dati algoritam dao najbolje rezultate. Za svaku osobinu od značaja implementiran je efikasan algoritam koji proverava njeno prisustvo.

Dizajn i implementacija jezika specifičnog za domen uključuje pre svega izbor meta-jezika (jezika za definiciju jezika). Zbog jednostavnosti i velikog broja pogodnih funkcionalnosti, koristio se *textX* [14]. Ovaj meta-jezik omogućava specifikaciju jezika na programskom jeziku *Python*, ali alati poput *Jython*-a čine integraciju programskog koda pisanog u ovom jeziku sa *Java* kodom jednostavnim zadatkom.

Verifikacija rešenja, odnosno kreirane biblioteke, sprovedena je kroz spoj sa *Kroki* alatom za skiciranje poslovnih aplikacija [15]. Naime, izvršena je korisnička studija gde su se dijagrami generisani upotrebom upravo spomenutog spoja poredili sa dijagramima sa automatski raspoređenim elementima nastalim upotrebom popularnih komercijalnih alata, te alata otvorenog koda. Takođe, radi demonstracije korisnosti biblioteke, ona je upotrebljena u okviru popularnog *Sirius* [16] alata za kreiranje grafičkih editora.

1.5 Struktura projektnog rešenja

Rešenje koje je proisteklo iz postavljenih ciljeva disertacije predstavlja programsku biblioteku otvorenog koda za programski jezik *Java* koja se sastoji iz nekoliko celina:

1. Projekta koji sadrži:
   - definicije osnovnih komponenti kao što su graf i njegovi čvorovi i grane,
   - implementacije algoritama iz teorije grafova i crtanja grafova,
   - komponente zadužene za pozivanje željenog algoritma i vraćanje rezultata u pogodnom formatu.
   - implementaciju automatskog izbora i primene algoritma na osnovu osobina grafa.
   - implementaciju izbora algoritma na osnovu opisa u skladu sa definisanim jezikom.
2. Definicije interpretera jezika specifičnog za domen raspoređivanja elemenata datog dijagrama.
3. Jednostavnog grafičkog editora razvijenog sa ciljem podrške upoznavanju sa različitim algoritmima. Editor omogućava kreiranje elementarnih dijagrama i primenu kako automatskog raspoređivanja, tako i bilo kojeg drugog algoritma implementiranog u okviru prvog projekta. Svi grafovi u disertaciji nacrtani su upravo pomoću ovog editora.

Biblioteka je nazvana *GRAD* (*Graph Analysis and Drawing library*) i licencirana je pod *MIT* nerestriktivnom licencom [17]. Izvorni kod bilioteke dostupan je na [18].

1.6 Struktura disertacije

Nastavak disertacije organizovan je na sledeći način:

- U drugom poglavlju detaljnije su opisani osnovni pojmovi, nužni za razumevanje kasnijih sekcija. Pregled započinje kraćim uvodom u teoriju grafova, praćenim prikazom različitih klasa algoritama za raspoređivanje i estetskih kriterijuma, čije poštovanje je njihov primarni cilj. Konačno, objašnjava se šta su tačno jezici specifični za domen i zašto se koriste u tekućem kontekstu.

- Treće poglavlje predstavlja postojeća rešenja, kako za projekte koji se bave implementacijom određenih koncepata teorije grafova, uz akcenat na algoritme za raspoređivanje, tako i za jezike specifične za domen koji se na neki način bave grafovima.

- U četvrtom i petom poglavlju navodi se koji algoritmi za analizu i crtanje grafova su implementirani, uz ulaženje u detalje njihovog funkcionisanja i same implementacije. Posebno su naglašena eventualna poboljšanja i osmišljene realizacije nedorečenih koraka.

- Šesto poglavlje bavi se automatskim izborom algoritama za crtanje grafova na osnovu njihovih osobina, dok se u sedmom opisuje dizajn i implementacija jezika specifičnog za domen.

- Osmo poglavlje daje prikaz grafičkog editora za isprobavanje raznih algoritama i eksperimentisanje sa njima.

- U devetom poglavlju predstavljen je aplikativni interfejs biblioteke i objašnjeno na koji način se njene funkcionalnosti mogu koristiti u okviru postojećih grafičkih editora.

- Deseto poglavlje sačinjeno je iz diskusije o mogućim primenama rešenja i dva konkretna primera. Jedan od njih jeste integracija sa danas sve popularnijim *Sirius* alatom za kreiranje grafičkih editora.

- U jedanaestom poglavlju opisana je sprovedena korisnička studija i prikazani su njeni rezultati.

- Konačno, dvanaesto poglavlje zaključuje disertaciju, sumirajući postignuto i navodeći pravce daljeg razvoja.

Poglavlje 2

Osnovni pojmovi

U ovom poglavlju biće uvedeni osnovni pojmovi, bitni za razumevanje ostatka disertacije. Pre svega, radi se o bazičnim definicijama teorije grafova, algoritmima za analizu grafova i automatsko raspoređivanjanje njihovih elemenata, kao i pojmu jezika specifičnih za domen.

2.1 Uvod u teoriju grafova

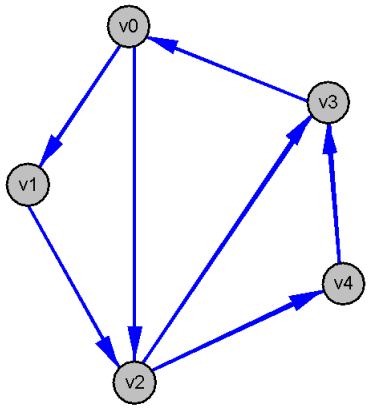
Formalno, graf u oznaci $G(V, E)$ jeste uređeni par koji se sastoji iz konačnog skupa čvorova V i grana E, tj. parova (u, v) čvorova [19]. Ukoliko neka grana povezuje jedan čvor sa samim sobom, naziva se petljom. Graf kod koga su sve grane jednostrano orijentisane naziva se orijentisanim ili usmerenim, dok u suprotnom kažemo da je neorijentisani ili neusmereni. Dva čvora grafa spojena granom nazivaju se susednim čvorovima, dok se ukupan broj suseda datog čvora naziva njegovim stepenom. Za orijentisani graf, grana (u, v) koja počinje u čvoru u a završava se u v jeste izlazna za prvi, a ulazna za drugi čvor. Takođe, može se spomenuti i termin multigrafa, koji se odnosi na graf kod kojega se između dva čvora nalazi više od dve grane različite orijentacije [20]. Konačno, graf koji nije multigraf i nema petlje naziva se jednostavnim.

Grafovi su pogodni za kreiranje grafičke reprezentacije i predstavljaju se crtežom na sledeći način:

- Čvorovi (elementi skupa V) se reprezentuju međusobno različitim tačkama u ravni.
- Svaka grana grafa (u, v) predstavlja se linijom koja povezuje čvorove u i v i ne prolazi kroz druge čvorove grafa.
- Ako je graf orijentisani, i svaka grana je orijentisana, što se na slici predstavlja strelicom koja se dodaje liniji.

Primer crteža orijentisanog grafa prikazan je na slici 2.1. Utapanje (*embedding*) grafa definiše se kao jedan njegov određeni crtež.

Graf može da bude predstavljen i kvadratnom matricom čiji red odgovara broju njegovih čvorova. Element a_{ij} ovog grafa definiše se kao broj grana koje polaze iz čvora v_i i završavaju se u čvoru v_j u slučaju orijentisanih grafova, odnosno ukupnom broju neusmerenih grana između ta dva čvora u slučaju neorijentisanih. Ovakva matrica se naziva matricom susedstva (*adjacency matrix*) i označava se sa A. Ukoliko je graf neorijentisan, ova matrica je simetrična. Ako graf nije multigraf, vrednosti elemenata matrice mogu biti samo 0 ili 1. Na primer, matrica susedstva za orijentisani graf sa slike 1 je:



Slika 2.1: Primer jednog crteža orijentisanog grafa

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

Neka su dati grafovi $G = (V, E)$ i $G' = (V', E')$. Neka je dalje $V' \subset V$ i $E' \subset E$ takav da E' sadrži sve grane između čvorova iz skupa V' koje se javljaju u grafu G. Graf G' nazivamo podgrafom grafa G obrazovanim nad skupom čvorova V'. Ukoliko za graf G' važi samo da je $V' \subset V$, dok skup E' ne sadrži sve grane grafa između izdvojenih čvorova, graf G' nazivamo delimičinim ili parcijalnim podgrafom grafa G.

Put ili putanja je niz čvorova $v_1, v_2, ...v_k$ zajedno sa granama $(v_1, v_2), ..., (v_{k-1}, v_k)$. Put može više puta prolaziti istom granom ili kroz isti čvor. Sa druge strane, elementarni put je put koji kroz svaki čvor grafa prolazi najviše jednom. Ukoliko se put završava u istom čvoru u kom i počinje, on se naziva kružnim ili zatvorenim.

Kod analize puteva od velikog značaja je da li je graf orijentisan ili ne, odnosno, da li su grane uređeni ili neuređeni parovi čvorova. U nekim slučajevima, orijentacija grane nije posebno bitna i može biti proizvoljna. Niz grana čija orijentacija nas ne interesuje, takav da bi grane nakon pogodnog orijentisanja obrazovale put, naziva se lancem. Lanac koji se završava u istom čvoru u kom počinje jeste ciklus. U neorijentisanom grafu, put i kružni put sa jedne strane, i lanac i ciklus sa druge predstavljaju istovetne pojmove. Grana grafa koja nije deo ciklusa, ali povezuje dva njegova čvora naziva se akordom.

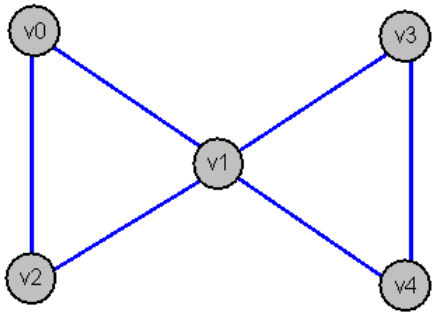
Jedna od najbitnijih osobina grafova, od koje često zavisi mogućnost primene algoritama nad njima, a definiše se preko upravo uvedenih pojmova jeste povezanost. Povezanost se različito definiše za neorijentisani i orijentisani graf, što nije iznenađujuće jer se ni putanje ne posmatraju na istovetan način. Naime, neorijentisani graf je povezan ukoliko se bilo koja dva njegova čvora mogu povezati putem (lancem). U suprotnom, graf je nepovezan. U drugom slučaju, u okviru grafa mogu se izdvojiti maksimalni povezani podgrafi, koje nazivamo njegovim komponentama povezanosti. Svaki čvor nepovezanog grafa pripada tačno jednoj komponenti povezanosti. Isto važi i za grane.

Za orijentisane grafove uvodi se više vrsta povezanosti. Tako za graf kod koga je svaki uređeni par čvorova v_i, v_j povezan putem iz v_i u v_j kažemo da je jako povezan. Po ovoj definiciji, jasno je da mora postojati i put iz v_j u v_i. Ako je svaki par čvorova povezan putem u jednom smeru, povezanost je jednostrana. Ukoliko je pak povezan

neorijentisani graf dobijen od datog grafa zamenom orijentisanih grana odgovarajućim neorijentisanim, onda je on slabo povezan. Naravno, svaki jako povezani graf je i slabo povezan.

Pojam povezanosti može se znatno uopštiti, te se uvodi pojam k-povezanosti. Kažemo da je graf k-povezan ukoliko ne postoji skup od $k-1$ čvorova čije uklanjanje bi graf pretvorilo u nepovezani [21]. Dakle, povezani graf je 1-povezani, dvostruko povezani graf je 2-povezani itd.

Dakle, graf je 2-povezan ili bipovezan ukoliko ostaje povezan nakon uklanjanja bilo kog čvora. Ukoliko pak odstranjivanje određenog čvora razruši povezanost, odnosno, dovede do pojave više komponenti povezanosti, graf nije bipovezan. Takvi čvorovi nazivaju se artikulacionim ili presečnim. Dakle, može se reći da je povezani graf bipovezan ukoliko ne sadrži nijedan artikulacioni čvor [21]. Na primer, graf sa slike 2.2 je povezan, ali nije bipovezan, a čvor v_1 je artikulacioni.



Slika 2.2: Leptir graf koji je povezan, ali nije bipovezan

Maksimalni bipovezani podgraf grafa naziva se bipovezana komponenta ili blok. Dakle, graf koji nije bipovezan može se rastaviti na više blokova. Indentifikacija artikulacionih čvorova grafa i bipovezanih komponenti uključena je u ispitivanje mnogih drugih obeležja grafa pošto je često lakše ispititati sve ovakve komponente, nego graf u celini [22].

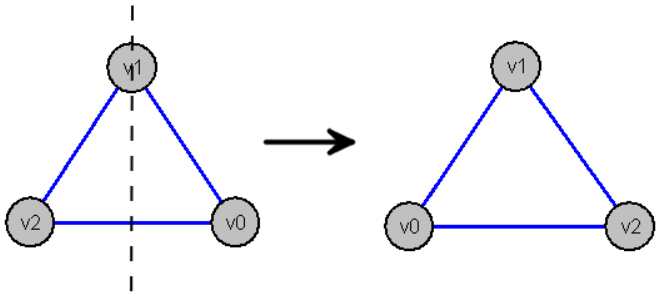
Graf je tripovezan ukoliko uklanjanje bilo kog para čvorova ne dovede do narušavanja osobine povezanosti. Slično kao i u slučaju bipovezanosti, mogu se uočiti tripovezane komponente nekog grafa kao maksimalni tripovezani podgrafi.

Osim povezanosti, za graf se može definisati i osobina obojivosti. Naime, graf G je k-obojiv ako se svakom njegovom čvoru može dodeliti jedna od k boja, tako da nijedna grana grafa ne bude povezana sa dva čvora istog skupa.

Dodatno, dva grafa se mogu nalaziti u odnosu izomorfizma. Naime, dva grafa su izomorfna ukoliko postoji jednoznačno preslikavanje njihovih skupova čvorova koje održava osobinu susednosti čvorova. Analitički uslov izomorfnosti grafova uključuje matrice susedstva dva grafa, te permutacionu matricu P. Permutaciona matrica je kvadratna matrica koja u svakoj vrsti i svakoj koloni ima tačno jedan element koji je jednak jedinici, dok su ostali elementi nule. Množenje neke matrice sa P sa desne strane dovodi do permutovanja kolona polazne matrice, dok množenje sa P^{-1} s leva dovodi do permutovanja vrsta. Imajući navedeno u vidu, uslov izomorfnosti se predstavlja formulom $A_1 = P^{-1}A_2P$.

Specijalni slučaj izomorfizma je automorfizam grafa, koji je usko povezan sa njegovim simetrijama. Automorfizam je mapiranje čvorova grafa nazad na njih same, takvo da je rezultujući graf izomorfan početnom grafu [23]. Jedan od primera prikazan je na slici 2.3. Ako se graf sa leve strane obrne oko označene linije, dobija se desni graf.

Možemo primetiti da se čvor v_1 preslikava u samog sebe, čvor v_0 u v_2, a v_2 u v_0. Pošto se matrica povezanosti nije promenila, zaključuje se da je reč o automorfizmu, odnosno, da je označena linija jedna linija simetrija datog grafa.



Slika 2.3: Primer automorfizma grafa

Takođe, može se napomenuti da graf može da bude težinski. To znači da se svakoj grani grafa pridružuju jedan ili više realnih brojeva. Njima se mogu predstaviti dodatne informacije, kao što su rastojanja ili cene.

Konačno, može se definisati i pojam mreže. Naime, mreža je orijentisani graf kod koga se svakoj njegovoj grani pridružuje kapacitet.

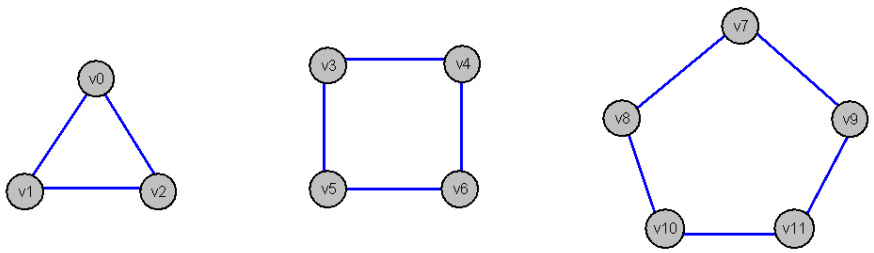
2.2 Tipovi grafova i njihove osobine

Pojedini posebni tipovi grafova neretko imaju veoma značajnu ulogu u raznim problemima, ili ih određeni algorotmi tretiraju na drugačije načine. U tu klasu spadaju pravilni, potpuni i planarni grafovi i, pre svih, stabla.

2.2.1 Pravilni grafovi, potpuni grafovi i konture

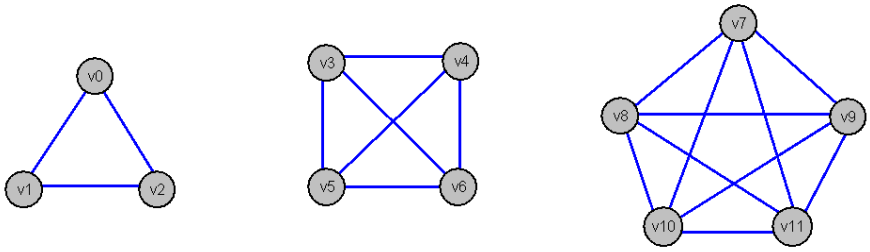
Neka su $d_1, d_2, ..., d_n$ stepeni čvorova $v_1, v_2, ..., v_n$ u neorijentisanom grafu G. Ukoliko za G važi da je $d_1 = d_2 = ... = d_n = r$, onda je on regularan ili pravilan. Pošto u svakom grafu važi da je $d_1 + d_2 + ... + d_n = 2m$, gde je m ukupan broj grana u grafu, može se primetiti da regularan graf stepena r ima $m = \frac{1}{2}nr$ grana. Analizom ove jednačine može se zaključiti da ne postoji regularni graf za svaku kombinaciju vrednosti n tj. broja čvorova i vrednost r tj. stepena. Naime, bar jedna od njih mora biti parna.

Konture (prstene) dalje možemo definisati kao povezane grafove stepena 2. Pošto ovakva vrednost stepena zadovoljava gorenavedeni uslov, primećuje se da za svaki broj čvorova postoji odgovarajuća kontura. Konture sa 3, 4 i 5 čvorova prikazane su na slici 2.4.



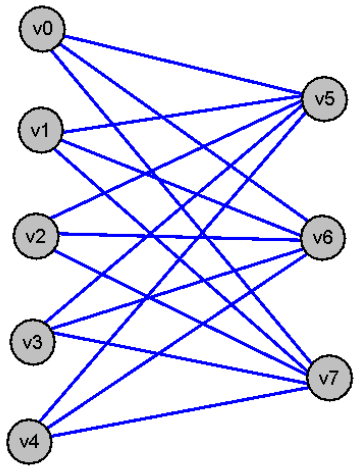
Slika 2.4: Konture sa 3, 4 i 5 čvorova

Pored kontura, izdvaja se još jedna vrsta regularnih grafova koja se često spominje u različitim oblastima teorije grafova. Reč je o potpunin grafovima, ili regularnim grafovima sa n čvorova i $n-1$ veza. Odnosno, potpuni grafovi su grafovi kod kojih je svaki par čvorova spojen granom. Potpun graf sa n čvorova označava se sa K_n. Potpuni grafovi K_3, K_4 i K_5 prikazani su na slici 2.5.



Slika 2.5: Potpuni grafovi K_3, K_4 i K_5

Bigraf je graf čiji je skup čvorova podeljen na dva skupa bez zajedničkih elemenata U i V, a svaka grana povezuje jedan čvor iz U sa jednim čvorom iz V. Odnosno, bigraf je 2-obojiv graf. Ovakav graf često se označava kao $G = (U, V, E)$. Ako skupovi U i V imaju isti broj elemenata, bigraf je izbalansirani. Ukoliko između svakog čvora skupa U i svakog čvora skupa V postoji grana, onda je bigraf potpun. Ako je m broj elemenata skupa U, a n broj elemenata skupa V, potpuni bigraf se obeležava sa $K_{n,m}$. Na slici 2.6 prikazan je potpuni bigraf $K_{5,3}$.

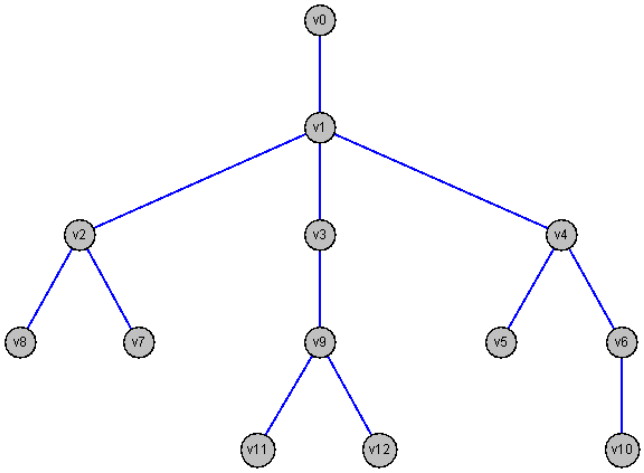


Slika 2.6: Bigraf $K_{5,3}$

2.2.2 Stabla

Verovatno najpoznatiji tip grafova, koji odlikuje veoma široka primena, jeste stablo. Stablo se može definisati kao povezani graf sa n čvorova i $n-1$ grana. Uklanjanjem bilo koje grane iz stabla rezultujući graf ne bi bio povezan. Drugim rečima, stabla su samo 1-povezana. Stablo je graf koji ne sadrži nijedan prost elementarni ciklus. Odnosno, ne sadrži nijednu konturu. Ukoliko bi se stablu dodala proizvoljna nova grana, dobio bi se graf koji ima tačno jednu konturu. Prema definiciji povezanosti grafa, u stablu postoji

put između bilo koja dva čvora, pa tako i između ona dva između kojih bi se uspostavila veza. Dakle, novododata grana sa granama uočenog puta obrazovala bi konturu. Jedan primer stabla, na kojem se može uočiti prisustvo svih navedenih osobina prikazan je na slici 2.7.



Slika 2.7: Primer stabla

Ukoliko se jedan čvor stabla posebno izdvoji i proglasi korenom, takvo stablo se naziva ukorenjenim. Za ovakav tip stabala uvode se sledeći termini:

- Čvor u je dete čvora v ukoliko je direktno povezan sa njim i nalazi se dalje od korena. Sa druge strane, čvor v je onda roditelj čvora u.
- Potomci čvora v su svi čvorovi do kojih se može stići prelaženjem iz roditeljskog čvora u dete, počinjući kretanje u čvoru v. Slično, preci čvora v su svi čvorovi do kojih se dolazi prelaženjem u roditeljske čvorove. Na primer, na stablu sa slike 2.7 potomci čvora v_3 su v_9, v_{11} i v_{12}, a preci v_1 i v_0.
- Čvorovi s istim roditeljem su braća (*siblings*).
- Čvorovi koji nemaju decu nazivaju se listovima.

Binarno stablo je korensko stablo kod koga svaki čvor ima najviše dva deteta, koja se nazivaju levim i desnim [24]. Graf koji se sastoji iz više odvojenih stabala, odnosno, čije su komponente povezanosti stabla, naziva se šuma.

Za proizvoljni povezani, neorijentisani graf može se uočiti razapinjuće stablo, koje se definiše kao njegov povezani podgraf koji sadrži sve čvorove datog grafa i ne sadrži nijednu konturu. Drugim rečima, radi se o stablu koje sadrži sve čvorove datog grafa i podskup svih grana.

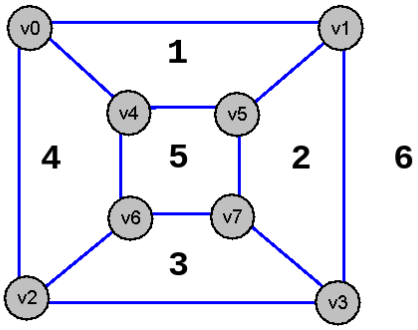
Takođe, za stablo T može se uočiti i podstablo ukorenjeno u nekom njegovom čvoru v u oznaci $T[v]$. $T[v]$ se sastoji iz čvora v i svih njegovih potomaka i grana između njih prisutnih u originalnom grafu.

2.2.3 Planarni grafovi

Planarni grafovi predstavljaju još jednu veoma interesantnu klasu grafova. Planarni ili ravni grafovi su grafovi koji se mogu nacrtati u ravni bez preseka grana. Preciznije,

grane ne smeju imati zajedničkih tačaka, osim, eventualno, čvorova koji su im zajedničke krajnje tačke.

Ako se planarni graf nacrta na objašnjeni način, on deli ravan na više konačnih zatvorenih oblasti i jednu beskonačnu oblast koja se naziva lica. Upravo se ovim oblastima bavi jedna od najstarijih teorema iz teorije grafova - Ojlerova teorema za planarne grafove. Naime, ona tvrdi da povezani planaran graf sa n čvorova i m grana deli ravan na ukupno $f = m - n + 2$ oblasti tj. lica. Primer jednog planarnog grafa sa 8 čvorova, 12 veza i obeleženim licima prikazan je na slici 2.8.



Slika 2.8: Primer planarnog grafa sa obeleženim licima

Jedan od najstarijih načina za proveru planarnosti grafa obuhvata traženje posebnih neplanarnih podgrafa. Formulisan je teoremom Pontrjagina i Kuratovskog (Pontryagin-Kuratowski) koja tvrdi da je graf planaran ako i samo ako ne sadrži kao delimični podgraf ni pentagraf ni bitrigraf ni njihovu potpodelu. Pentragraf je potpuni graf sa 5 čvorova K_5, a bitrigrag je bigraf sa po 3 čvora u odvojenim skupovima čvorova tj. graf $K_{3,3}$. Konačno, graf je potpodela datog grafa ako je dobijen od polaznog dodavanjem novih čvorova na njegove grane. Jasno je da graf koji nije planaran dodavanjem novih čvorova i grana to neće postati. Teoremu je prvi dokazao Potrjagin, ali postupak nije publikovao, te se često naziva samo teorema Kuratovskog. On je svoj dokaz objavio u [25].

Ukoliko graf nije planaran, može se postaviti pitanje sa koliko minimalno preseka grana bi on mogao biti nacrtan. Upravo je ovo nešto na šta se fokusiraju mnogi algoritmi za automatsko crtanje grafova.

2.3 Algoritamska kompleksnost

Kako je u teoriji grafova efikasnost algoritama od izuzetne važnosti, u tekućoj sekciji biće definisan pojmovi poput algoritamske kompleksnosti i NP-teških problema.

Formalno, algoritamska kompleksnost se definiše kao numerička funkcija $T(n)$, odnosno, vreme potrebno za izvršavanje datog algoritma nad ulazom dužine n. Jasno je, međutim, da to vreme ne zavisi samo od algoritma, nego i same implementacije, računara, odnosno, njegovog procesora, diska, memorije, programskog jezika i kompajlera. Prema tome, ova mera se aproksimira kao broj potrebnih elementarnih koraka, pod uslovom da je vreme koraka konstantno. Cilj njenog uvođenja jeste klasifikacija algoritama prema njihovim performansama. Za izražavanje kompleksnosti izvršavanja algoritma uvodi se O notacija [26]:

- $O(1)$ znači da algoritam ima konstantno vreme izvršavanja. Odnosno, za svaku veličinu ulaza, vreme je uvek isto.
- $O(n)$ znači da algoritam ima linearno vreme izvršavanja, odnosno, da je pomenuto vreme direktno proporcionalno veličini ulaza.
- $O(log\ n)$ označava logaritamsko vreme izvršavanja, što znači da je ono proporcionalno logaritmu veličine ulaza.
- $O(n^2)$ predstavlja kvadratno vreme izvršavanja, proporcionalno kvadratu veličine ulaza.
- $O(n^k)$ generalno označava polinomijalno vreme, što znači da je proporcionalno k-tom stepenu veličine ulaza.

U nastavku spominjaće se i termin NP-teških problema. Problem se svrstava u NP klasu ukoliko se u polinomijalnom vremenu može rešiti pomoću nedeterminističke Tjuringove mašine [27]. Ili, manje formalno, NP je skup problema odlučivanja (problema gde su mogući odgovori "da" i "ne") kod kojih instance sa odgovorom "da" imaju dokaz koji se može proveriti determinističkim izračunavanjima u polinomijalnom vremenu. Problem je NP težak ukoliko se algoritam za njegovo rešavanje može prevesti u algoritam za rešavanje nekog NP problema. Odnosno, NP-težak poblem je težak bar kao NP problem [28].

2.4 Algoritmi za analizu grafova

Prisustvo prethodno spomenutih osobina grafova, poput njihove planarnosti, cikličnosti ili povezanosti u velikoj meri utiče na mogućnost primene raznih algoritama za crtanje grafova. Takođe, sama implementacija tih algoritama zahteva primenu mnogih algoritama za analizu. Naime, kompleksniji algoritmi za crtanje neretko uključuju rastavljanje grafa na komponente, njihov obilazak, pronalaženje ciklusa ili najkraće putanje između dva čvora. Mnogi algoritmi za crtanje grafova traže prisustvo određenih karakteristika, bilo da bi uopšte mogli biti primenjeni, bilo da bi primena bila primerena i efikasna. Na primer, pojedini algoritmi pomenutog tipa posvećeni su isključivo planarnim 2 ili 3-povezanim grafovima, stablima, grafovima koji poseduju netrivijalne simetrije i sl.

Dakle, algoritmi za analizu grafova predstavljaju neizostavni sastavni deo mnogih drugih funkcionalnosti. Problem koji se rešava često ne deluje posebno kompleksno. Na primer, provera da li je neki graf 3-povezan mogla bi se izvršiti tako što bi se svaki par čvorova datog grafa uklonio, te se ispitalo da li je graf i dalje povezan. Međutim, to bi značilo da bi se za neki veći graf provera povezanosti morala vršiti stotine hiljada puta, što nikako nije optimalno rešenje. Kako određena provera ili pretprocesiranje ne bi znatno usporili generisanje crteža, primena efikasnih algoritama za analizu grafova je od ključne važnosti.

U osnovi drugih algoritama veoma često se nalaze postupci za obilazak grafa i nalaženje putanje između određenih čvorova. Najpoznatiji algoritmi ove kategorije uključuju pretragu najpre u dubinu, najpre u širinu, te Dijkstrinu najkraću putanju. Pretraga najpre u dubinu je posebno značajna, a njenim izvršavanjem prikupljaju se mnoge dodatne informacijeod velike upotrebne vrednosti.

Povezanost grafova je drugi problem na koji se mora obratiti pažnja. Uz provere da li je graf 1, 2 i 3-povezan, rastavljanje na komponente je jedan od najčešćih koraka

drugih algoritama. Naime, dodatne provere često je lakše izvršiti nad 2 ili 3-povezanim komponentama, nego nad čitavim grafom. Najčešće implementacije ovih postupaka uključuju pretragu najpre u dubinu. Naravno, što je veći stepen koji se ispituje, to je i sam algoritam kompleksniji.

Kao što je već spomenuto, za crtanje grafova neretko je planarnost jedna od najbitnijih osobina. Osim čiste provere da li graf ima ili nema planarni crtež, pojedini algoritmi za ispitivanje planarnosti mogu da odrede koji čvorovi treba da se nađu na spoljašnjem licu i u kom redosledu grane treba da izlaze iz čvorova da se ne bi presecale. Danas je poznat veliki broj algoritama koji se bave ovim problemima, poput algoritma Bojer-Mirvold (Boyer-Myrvold).

Dalje, određivanje automorfizama grafa, koji su, kao što je spomenuto, usko povezani sa njegovim simetrijama, još je jedan zadatak koji se može pojaviti u okviru algoritma za crtanje. Poput prethodno spomenutih ispitivanja, i ovo bi trebalo biti dovoljno efikasno da ne usporava postupak čiji je deo. Trenutno najpoznatiji algoritam koji se ovim bavi jeste MakKejev (McKay's).

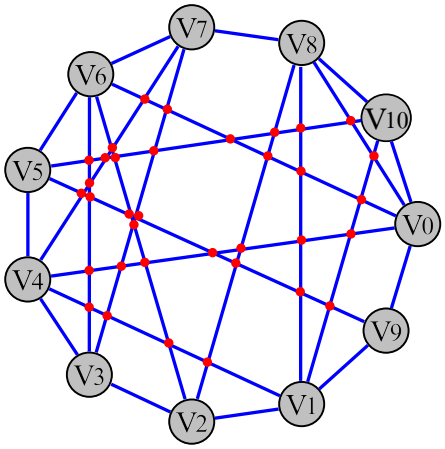
Ovim je dat samo kratak pregled važnosti algoritama pomenutog tipa, kao i problema koji se njima rešavaju. U 5. poglavlju biće predstavljeni algoritmi za analizu grafova koji su sastavni deo bilo kompleksnijih algoritama koji se implementiraju, bilo provere da li se neki algoritam za crtanje uopšte može primeniti i da li je njegova primena dobro rešenje u datoj situaciji.

2.5 Estetski kriterijumi za crtanje grafova

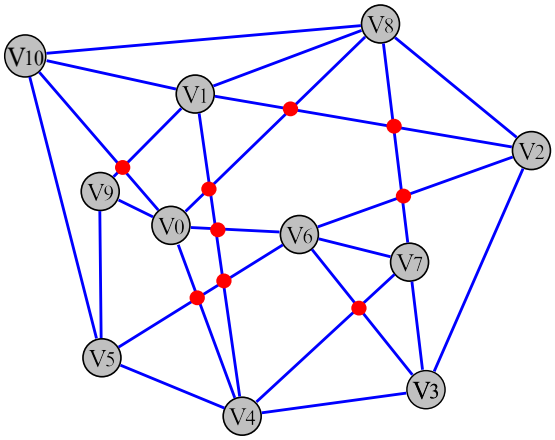
Autori algoritama za automatsko raspoređivanje elemenata grafova, odnosno, za crtanje grafova često stavljaju poseban naglasak na određene estetske kriterijume, tvrdeći da se tako dobijaju crteži koji se korisnicima posebno sviđaju. Drugim rečima, smatraju da su crteži na kojima se uočavaju određene estetske karakteristike lakši za čitanje i shvatanje informacija koje se vizualizuju pomoću datih grafova. Veliki broj ovakvih kriterijuma je predložen tokom godina, poput minimizacije broja preseka grana, ujednačenog toka ili simetrije. Počevši od sredine devedesetih godina prošlog veka, sprovedeno je nekoliko studija kojima se ispituje validnost tvrdnji autora i proverava koji estetski kriterijumi najviše pomažu prilikom uočavanja određenih osobina grafa [29]–[32]. Međutim, ova oblast je još u ranim fazama razvoja i studije su većinom bile usmerene ka određenim tipovima grafova. Takođe, bilo je i pokušaja definisanja metrike kojom bi se merile jasnoće crteža [33], ali postoji puno prostora za dalji razvoj ovog pravca istraživanja. Pošto još nije utvrđeno koji estetski kriterijumi u opštem slučaju najviše doprinose povećanju čitljivosti dijagrama, u pregledu koji sledi oni neće biti poređani po značaju.

2.5.1 Presecanje grana

Minimizacija broja preseka grana je estetski kriterijum koji mnogi smatraju jednim od najbitnijih, ako ne i najbitnijim od svih. Lakoća čitanja dvodimenzionalnog crteža grafa dovodi se u direktnu vezu sa ovim brojem, oko čega se slaže nekolicina studija, kao što su [30] i [31]. Imajući u vidu da je glavna informacija koju apstraktni graf nosi indikator da li su dva čvora povezana nekom granom, jasno je da smanjivanje broja preseka značajno povećava čitljivost crteža [34]. Slike 2.9 i 2.10 prikazuju dva crteža istog grafa, gde prvi ima daleko više preseka grana nego drugi. Preseci su na crtežu posebno naznačeni crvenim tačkama. Estetski kriterijum koji je opisan daje veliku prednost drugom crtežu, koji je, bez sumnje, daleko čitljiviji.



Slika 2.9: Crtež grafa sa velikim brojem preseka grana



Slika 2.10: Crtež grafa sa manjim brojem preseka grana

2.5.2 Minimalni uglovi

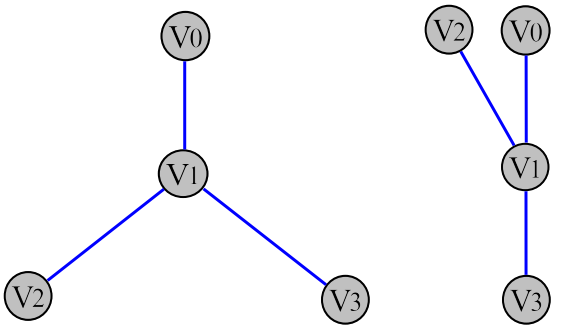
Ovaj estetski kriterijum tvrdi da minimalni ugao između grana koje izlaze iz nekog čvora treba da bude maksimizovan [35], [36]. U [33] objašnjava se da se najbolji mogući rezultat dobija kada svi čvorovi grafa imaju identične uglove između svih grana koje u njih ulaze ili izlaze iz njih. Na slici 2.11 prikazana su dva crteža istog apstraktnog grafa, gde su kod prvog maksimizovani uglovi u centralnom čvoru, dok je kod drugog minimalni ugao dosta manji.

2.5.3 Prelomi

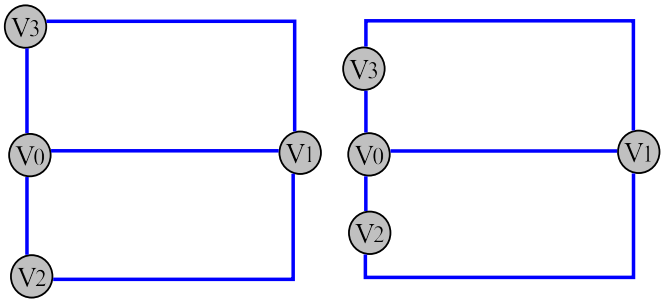
Ovaj kriterijum predložen je u [37] i izdvaja minimizaciju broja preloma grana kao jedan od bitnijih faktora za ukupnu razumljivost grafa. Pogotovo u oblasti veoma velikih integrisanih kola (VLSI), arhitektonskom dizajnu itd. Slika 2.12 prikazuje dva crteža istog apstraktnog grafa. Na prvoj grane imaju minimalan mogući broj preloma, dok druga prikazuje crtež sa prelomom više nego što je neophodno.

2.5.4 Tok

Kriterijum koji zagovara ujednačeni tok usmerenih grana grafova na crtežu. Generalno govoreći, smer grana treba da bude konzistentan [33], a najčešće se radi o smeru od gore

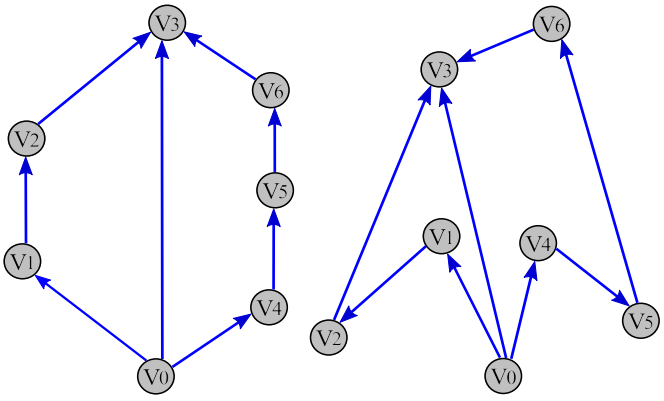


Slika 2.11: Dva crteža istog grafa sa različitim vrednostima minimalnog ugla među granama



Slika 2.12: Dva crteža istog apstraktnog grafa, gde jedan ima minimalan broj preloma grana, a drugi nema optimalno utapanje

prema dole ili obrnuto. Primer istog grafa nacrtanog na dva načina, gde jedan poštuje kriterijum koji se opisuje, a drugi ne prikazan je na slici 2.13. Ujednačen tok je pogotovo bitan kod grafova kojima su reprezentovane hijerarhijski uređene informacije, kao i grafova koji uvode fizičke ili apstraktne tokove. Na primer, poštovanje ovog kriterijuma može povećati preglednosti modela poslovnih procesa.



Slika 2.13: Dva crteža istog apstraktnog grafa, gde jedan ima konzistentan tok, a drugi ne

2.5.5 Ortogonalnost

Ortogonalni estetski kriterijum tvrdi da se čvorovi i grane grafa mogu postaviti na ortogonalnu koordinatnu mrežu, čime celokupan crtež dobija na preglednosti [37], [38].

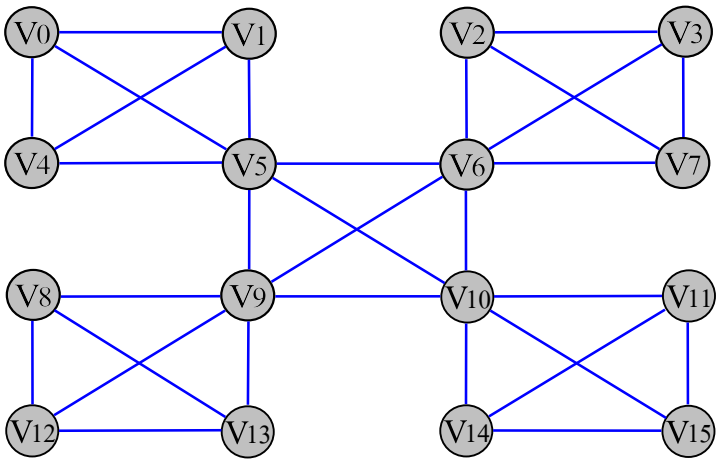
Koncept ortogonalnosti može se predstaviti kroz dva uslova [33]:

- grane i njihovi segmenti prate linije koje se postavljaju na zamišljeni Dekartov koordinatni sistem,
- čvorovi i prelomne tačke grana koliko god je to moguće treba da koriste Dekartov koordinatni sistem.

Drugim rečima, segmenti grana ne bi trebalo puno da odstupaju od pravog ugla, a čvorovi grafa i presečne tačke grana treba da budu fiksirane na preseke zamišljene koordinatne mreže. Tako se mreža maksimalno koristi.

2.5.6 Simetrija

Simetrija je estetski kriterijum, kojim se, kao što se navodi u [39], jasno uočavaju struktura i obeležja grafa. Svaki crtež grafa ima trivijalnu simetriju, te se može reći da ovaj kriterijum zahteva da crtež datog grafa ima netrivijalnu simetriju. Ili, ambicioznije, što je moguće više takvih simetrija. Na primer, oba crteža grafa na slikama 2.14 i 2.15 imaju bar jednu netrivijalnu simetriju, ali ih crtež na slici 2.14 ima daleko više. Ovaj crtež takođe ima pet preseka grana, dok je crtež sa slike 2.15 planaran. Većina ljudi bi rekla da im se više sviđa prvi crtež, što demonstrira važnost ovog kriterijuma [40].

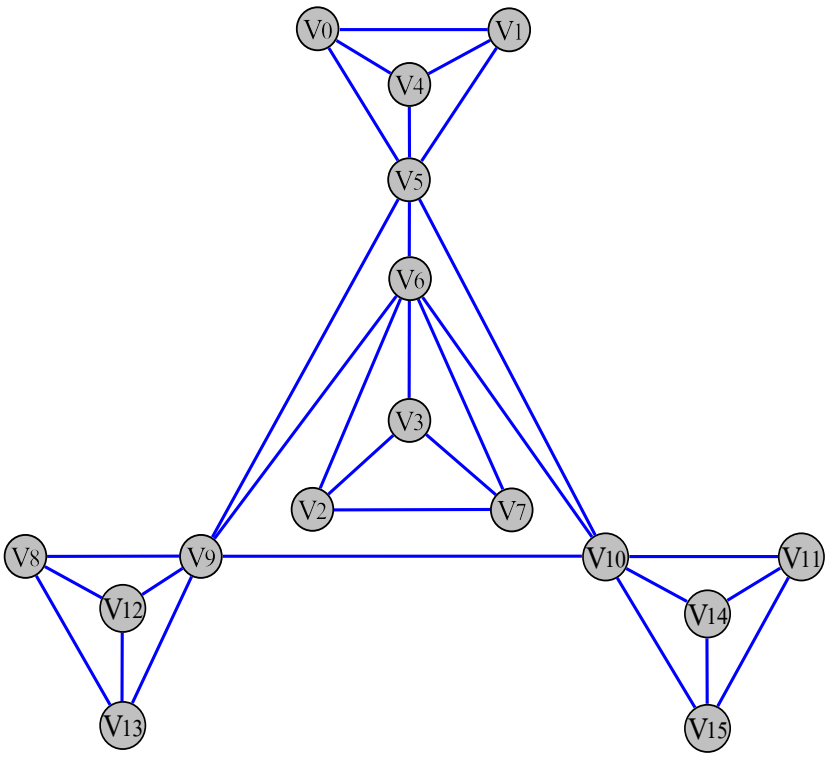


Slika 2.14: Crtež grafa sa 8 simetrija i 5 preseka grana

2.5.7 Ostali kriterijumi

Pored prethodno opisanih kriterijuma, autori različitih algoritama za raspoređivanje predložili su još nekolicinu, kao što su:

- Raspodela čvorova: čvorovi treba da budu ravnomerno raspoređeni unutar ograđenog prostora.
- Dužina grana: grane ne treba da budu ni previše dugačke, ni previše kratke.
- Varijacija dužina grana: dužine grana treba da budu ujednačene.



Slika 2.15: Planarni graf sa jednom netrivijalnom simetrijom

2.6 Pregled klasa algoritama za automatsko raspoređivanje

U nastavku će biti dat pregled najpopularnijih klasa za automatsko raspoređivanja elemenata grafova. Za svaku od njih biće navedena osnovna ideja postupka crtanja grafa, kao i istaknuti konkretni algoritmi koji im pripadaju.

2.6.1 Algoritmi za crtanje stabala

Crtanje stabala je jedna od najviše izučavanih oblasti crtanja grafova. Ovo nije iznenađujuće pošto automatsko generisanje crteža stabala nalazi mnoštvo praktičnih primena. Naime, stablo čiji čvorovi predstavljaju entitete, a grane veze između njih, često se koristi za modelovanje hijerarhijskih informacija. Crtanje stabala koristi se u softverskom inženjerstvu (hijerarhije klasa unutar objektno-orijentisanih dijagrama), sistemima za podršku odlučivanju (stabla aktivnosti), biologiji (evoluciona stabla) itd. [41]. Naravno, algoritmi za crtanje stabala primenjuju se isključivo nad grafovima koji su stabla.

Algoritmi za crtanje stabala teže da zadovolje više estetskih kriterijuma. Neki od njih su prethodno već opisani, dok drugi imaju smisla samo u kontekstu grafova ovog tipa. U ovu drugu grupu spadaju:

- Fiksiranje crteža na mrežu uz fokusiranje na minimizaciji površine koja ga sadrži u celosti. Pod površinom se misli na broj tačaka na koordinatnoj mreži unutar pravougaonika u kom je crtež.

- Odnos dužine i širine pravougaonika iz gornjeg kriterijuma. U idealnom slučaju, dužina i širina bi trebalo da su jednake tj. odnos treba da bude jednak 1.

- Ako dva podstabla $T[u]$ i $T[v]$ nemaju zejedničkih čvorova, pravougaonici koji ih sadrže ne bi trebalo da se preklapaju [42].

- Rastojanje između korena stabla i njegovih listova trebalo bi da bude što manje.

Opštiji estetski kriterijumi, koji su prema mnogim autorima poželjni u okviru crteža stabala prvenstveno se odnose na grane. Konkretno, smatra se da je crtež stabla lepši i pregledniji ukoliko ima malu vrednost za prosečnu, ukupnu i maksimalnu dužinu grana. Takođe, poželjno je i da dužine grana budu jednake [29], [43]. Maksimizovanje minimalnih uglova između grana koje izlaze iz istog čvora, kao i simetrija dva su dodatna kriterijuma koja se često spominju kada je reč o crtanju stabala. Stabla su uvek planarna, tako da se teži da se na crtežu ne pojave preseci grana, dok se neki algoritmi fokusiraju i na ortogonalnost.

Postoje različiti pristupi za crtanje stabala, od kojih su neki dizajnirani samo za primenu nad binarnim stablima, dok se ostali mogu primeniti i nad potpuno proizvoljnim grafovima ovog tipa. Najpopularniji pristupi uključuju sledeće:

- pristup baziran na nivoima,

- H-V (horizontalno-vertikalni) pristup,

- pristup baziran na putanjama,

- kružni pristup,

- pristup baziran na razdvajanju.

Pristup baziran na nivoima može da se primeniti na proizvoljna stabla i karakterističan je po tome što su čvorovi na podjednakoj udaljenosti od korena horizontalno poravnati. Ovaj pristup se odlikuje simetrijom, ali je ukupna površina crteža previše velika, dok je širina u slučaju izbalansiranog stabla sa puno čvorova daleko veća od visine. Razvijen je znatan broj algoritama za crtanje stabala u skladu sa pristupom baziranom na nivoima. Verovatno najuticajniji algoritam ove grupe je algoritam za uredno crtanje stabala Rajngolda i Tilforda (Reingold-Tilford) publikovan 1981. godine [44]. Originalni algoritam dizajniran je za binarna stabla, tako da zadovolji 4 estetska kriterijuma, koja u dobroj meri opisuju težnje tekućeg pristupa. Reč je o sledećem:

1. čvorovi na istom nivoa stabla trebalo bi da leže na pravoj liniji, dok bi prave linije koje definišu nivoe trebalo da budu paralelne,

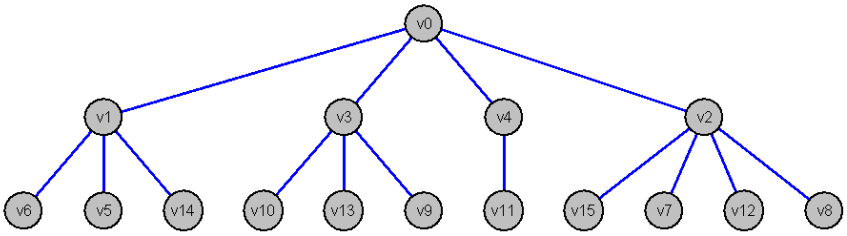
2. levo dete treba da bude postavljeno sa leve strane svog roditelja, a desno sa desne,

3. roditelj treba da bude centriran iznad svoje dece,

4. stablo i njegov odraz treba da proizvedu crteže koji su refleksije jedan drugog, dok bi svako podstablo trebalo biti nacrtano na isti način bez obzira na mesto u stablu gde se nalazi.

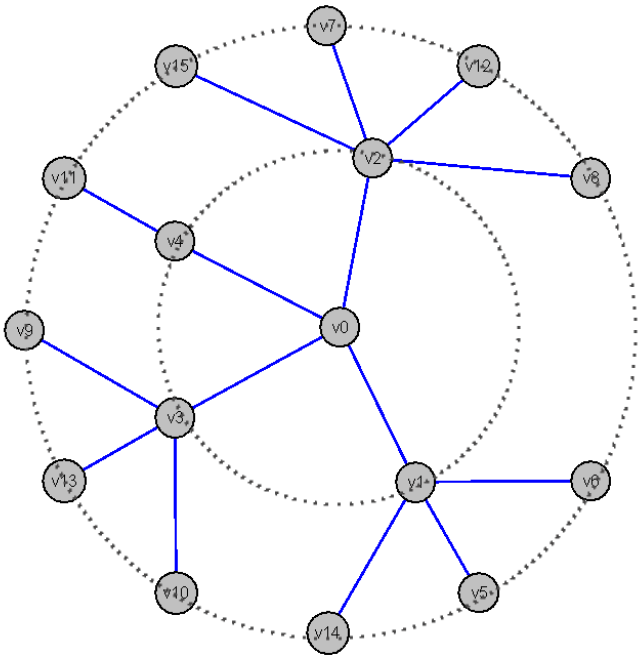
Prva tri kriterijuma formulisana su prvi put u [45], dok je četvrti originalni doprinos.

Algoritam je kasnije proširen tako da se može primeniti i za generalna stabla [46], a dizajnirani su i drugi algoritmi bazirani na nivoima, poput[47]–[49]. Primer crteža stabla prema ovom pristupu prikazan je na slici 2.16.



Slika 2.16: Crtež stabla u skladu sa pristupom baziranom na nivoima

Ako se koordinate Dekartovog koordinatnog sistema mapiraju na polarne, pristup baziran na nivoima proizvodi radijalne crteže stabala, kod kojih se čvorovi smeštaju na koncentrične krugove rastućeg poluprečnika. Za kreiranje ovakvih crteža stabala takođe je osmišljen veliki broj algoritama, a neki od njih su [50], [51]. Radijalni crtež stabla sa gornje slike prikazan je na slici 2.17.



Slika 2.17: Radijalni crtež stabla

Horizonalno-vertikalni pristup kao cilj ima kreiranje ortogonalnih pravolinijskih crteža grafova. Crteži stabala inspirisani ovim pristupom su planarni sa čvorovima postavljenim na koordinatnu mrežu i sa granama koje se sastoje samo iz horizontalnih i vertikalnih segmenata. Koren stabla postavlja se u gornji levi ugao, dok se svaki drugi čvor postavlja ili direktno desno od svog roditelja, ili direktno ispod njega.

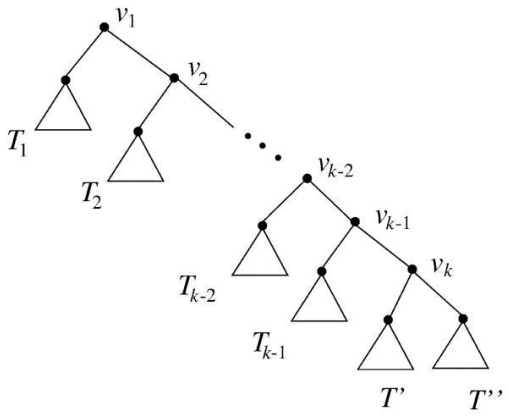
Postoje različiti algoritmi za crtanje stabala bazirani na ovom pristupu, od kojih je većina njih dizajnirana za rad sa binarnim stablima, poput [52], [53]. Neki od njih se, međutim, mogu proširiti i primeniti nad proizvoljnim stablima.

Pristup baziran na putanjama koristi se za crtanje binarnih stabala. Osnovna ideja ovih ovog pristupa zasnovana je na rekurzivnoj konstrukciji crteža stabla na sledeći način:

- Za svako podstablo ukorenjeno u čvoru v, fiksira se konstanta A, takva da ako je

$n(v) \leq A$, crteži podstabala ukorenjenih u deci čvora v se postavljaju jedan pored drugog.

- U suprotnom se graf deli na podstabla $T_1, T_2, ..., T_{k-1}, T_{k-1}, T', T''$, kao na slici 2.18.



Slika 2.18: Crtež binarnog stabla za $n(v) > A$ [42]

Dizajnirano je nekoliko algoritama prema ovom pristupu, kao što su [42], [54], [55].

Kružni pristup podrazumeva postavljanje dece na kružnicu sa centrom u svom roditelju. Razvijen je znatan broj algoritama koji se pridržavaju ovog pristupa, poput [56]–[60]. Često se koriste za crtanje velikih stabala, a dobijeni crteži se nazivaju i balonima. Jedan primer prikazan je na slici 2.19.

Pristup baziran na razdvajanju može se primeniti i u slučaju binarnih i u slučaju opštih stabala. Stablo se kreira rekurzivnom primenom sledećih akcija:

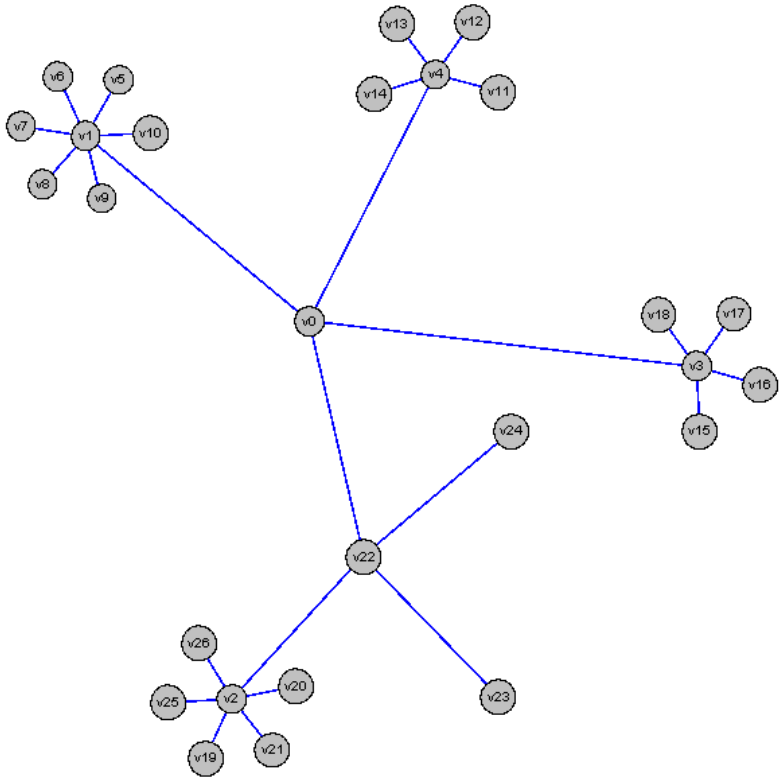
- Naći razdeljujuću granu (čvor) stabla T stepena d. Razdeljujuća grana (čvor) deli T na d parcijalnih stabala ako se ukloni. Svako stablo sadrži razdeljujuću granu (čvor) [61].
- Podeliti stablo na d parcijalnih stabala uklanjanjem razdeljujuće grane ili čvora.
- Za svako parcijalno stablo odrediti pogodnu vrednost odnosa širine i visine.
- Nacrtati sva parcijalna stabla poštujući određeni odnos.
- Složiti crtež uklapanjem svih parcijalnih stabala i dodati uklonjene grane i čvorove.

Osmišljeni su i razni algoritmi u skladu sa ovim pristupom. Neki od njih su [61]–[63].

2.6.2 Hijerarhijski algoritmi za raspoređivanje

Algoritmi za crtanje stabala su, kao što je spomenuto, pogodni za crtanje određenih hijerarhija. Međutim, nisu retki dijagrami koji su hijerarhije, ili slični njima, ali nisu stabla. Ipak, trebalo bi da budu nacrtani tako da se jasno mogu uočiti odnosi između entiteta.

Da bi se formalno definisao pojam hijerarhije, prvo je potrebno uvesti nivovske grafove. Nivovski graf $G = (V, E, \lambda)$ je usmereni aciklični graf sa mapiranjem $\lambda : V \leftarrow 1, 2, ..., k, k \geq 1$ koje particionira skup čvorova V na $V = V_1 \cup V_2 \cup ... \cup V_k, V_k =$



Slika 2.19: Crtež kružnog stabla

$\lambda^{-1}(j), V_i \cap V_j = \emptyset, i \neq j$, tako da važi $\lambda(v) = \lambda(u) + 1$ za svaku granu $(u, v) \in E$. Hijerarhija je nivovski graf za koji za svako $v \in V_j, j > 1$ postoji bar jedna grana (w, v) takva da je $w \in V_{j-1}$ [64].

Hijerarhijski algortitmi daju najbolje rezultate ako se primene na digrafe koji su hijerarhije, ali se najpopularniji algoritmi ove klase mogu primeniti i na skoro hijerarhije, kao i na neusmerene grafove, pri čemu se grane automatski orijentišu. Daleko najpopularniji algoritam je Sugijamin okvir(Sugiyama framework) [65], koji kreira crtež grafa težeći pridržavanju sledećih estetskih kriterijuma:

- jedinstveni tok,
- minimizacija dužina veza,
- ravnomerna raspodela čvorova,
- minimizacija preseka grana,
- kreiranje crteža sa pravolinijskim granama.

Svakako najbitniji od kriterijuma je jedinstveni tok. Naime, grane bi trebalo da teku ili od gore prema dole, ili od levo ka desno, prema potrebi.

2.6.3 Algoritmi za kružno raspoređivanje

Kružni crtež grafa je njegova vizualizacija koja poseduje sledeće karakteristike:

- graf je podeljen u klastere,

- čvorovi svakog klastera se kružno raspoređuju (stavljaju na obod kruga),
- svaka grana je jedna prava linija.

.

Ova grupa algoritama svoju upotrebu nalazi u situacijama kada je potreban klasterizovani prikaz informacija. Podela na klastere može da iskaže osobine grafa kao što je bipovezanost. Kako bi crteži bili što pregledniji, poželjno je da broj preseka grana unutar jednog klastera bude što je moguće manji [66]. Danas postoji više algoritama koji se bave ovom problematikom, mada je njihov broj znatno manji u odnosu na broj pripadnika popularnijim klasama. Neke tehnike za kružno crtanje grafova smeštaju sve čvorove na jedinstvenu kružnicu, recimo [67], [68], dok druge proizvode crteže koji se sastoje iz više krugova. U [69] opisuje se više tehnika za kružno crtanje, uključujući i predstavnika drugospomenute grupe, inspirisanog algoritmima za radijalno crtanje stabala. Čvorovi se u klastere grupišu na osnovu bipovezanosti, odnosno, jedan klaster predstavlja jednu bipovezanu komponentu, pri čemu se teži da artikulacioni čvorovi budu raspoređeni tako da se artikulacioni čvorovi jednog mogu povezati sa artikulacionim čvorovima drugog klastera na estetsko zadovoljavajući način.

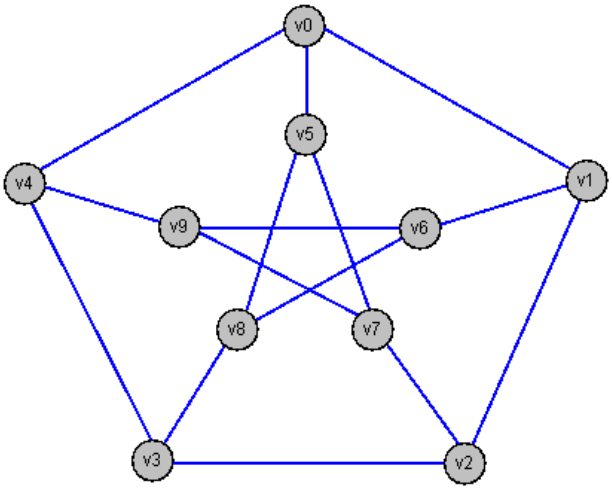
Takođe, u pojedinim situacijama korisnicima bi moglo biti interesantno da sami definišu sadržaj klastera, odnosno, kako će čvorovi biti grupisani. Ovako nešto se može očekivati ako se, na primer, crta graf koji predstavlja mrežu računara. Naime, korisnici bi možda želeli da grupišu računare unutar jednog departmana, sprata ili po nekom drugom kriterijumu. Ovakvo crtanje grafa, težeći minimizaciji broja preseka grana, kompleksnije je nego kada se raspoređivanje u klastere vrši prema bipovezanosti, te Siks (Janet M. Six) i Tolis (Ioannis G. Tollis) predlažu kombinovanje kružnog algoritma sa silom usmerenom tehnikom [69] u tim situacijama.

2.6.4 Algoritmi za simetrično crtanje grafova

Algoritmi za simetrično crtanje grafova kao glavni cilj imaju kreiranje crteža grafa sa netrivijalnom simetrijom, imajući u vidu da svaki crtež ima trivijalnu simetriju tj. mapiranje čvorova na same sebe. Ili, ambicioznije, sa najvećim mogućim brojem simetrija. Tvorci algoritama ove klase izdvajaju simetriju kao najvažniji estetski kriterijum, tvrdeći da ona jasno otkriva strukturu i obeležja grafa. Na primer, grafovi u udžbenicima o teoriji grafova se veoma često predstavljaju simetričnim crtežima.

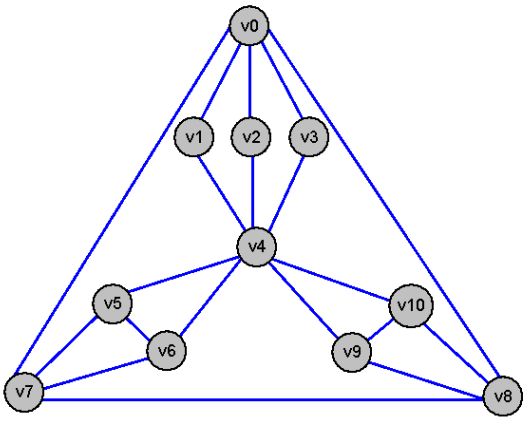
Simetrije crteža nekog grafa G povezane su sa njegovim automorfizmima. Međutim, ne može se svaki automorfizam predstaviti simetrijom nekog crteža grafa G. Na primer, poznati Petersonov graf, prikazan na slici 2.20, ima čak 120 automorfizama, ali samo 10 od njih može biti prikazano na crtežu. Dakle, algoritmi za simetrično crtanje grafova bave se i izdvajanjem podskupa automorfizama grafa koji se mogu predstaviti simetrijama nekog crteža [39]. Takvi automorfizmi nazivaju se geometrijskim.

Detekcija geometrijskih automorfizama nije jednostavan zadatak. Naime, pokazano je da je problem NP-težak [70]. Stoga su razvijeni različiti heuristički pristupi, od kojih je najčešći multidimenzionalno skaliranje ili silom usmerena metoda. Osnovna ideja ovog pristupa zasniva se na projektovanju crteža visokih dimenzija na crtež malih dimenzija (obično 2 ili 3). Teži se kreiranju crteža maksimalne simetrije u prostoru visokih dimenzija, te se u ovu svrhu definiše funkcija razdaljine između čvorova. Rastojanja između čvorova u visokodimenzionalnom prostoru trebalo bi da budu takva da se projekcijom crteža viših dimenzija na crtež manjih očuvaju maksimalno moguće. Ovakve metode često se posmatraju kao sistemi sila, odakle alternativni naziv.



Slika 2.20: Petersonov graf

Zbog kompleksnosti pomenutog problema, razvijeni su algoritmi za simetrično crtanje grafova koji se fokusiraju isključivo na planarne grafove, postižući linearno vreme izvršavanja. Upravo ovaj problem adresiraju Hong (Seok-Hee Hong) i Ejds (Peter D. Eades) u seriji radova koji se bave simetričnim crtanjem nepovezanih, te jednostruko, dvostruko i trostruko povezanih grafova [71]–[74]. Ovi algoritmi nalaze grupu automorfizama maksimalne veličine i utapanje grafa, koje potom treba i nacrtati na osnovu ove informacije. Tripovezani grafovi mogu se nacrtati pomoću čuvenog baricentričnog algoritma Tuta (Tutte's embedding)[75], ili modernijeg i kompleksnijeg algoritma, čiji su tvorci takođe Hong i Ejds uz Brendana Makeja (Brendan D. McKay) [76]. Bipovezani grafovi mogu se augmentacijom (dodavanjem određenih grana i čvorova) pretvoriti u određene tripovezane, dok nešto slično po deljenju na bipovezane komponente može biti primenjeno i za samo povezane grafove. Primer simetričnog crteža bipovezanog grafa prikazan je na slici 2.21.



Slika 2.21: Simetrični crtež bipovezanog grafa

Naravno, razvijeni su i algoritmi koji se bave simetričnim crtanjem grafova u opštem slučaju, ali njihovi rezultati teško mogu da dostignu prethodno spomenute prilikom primene nad planarnim. Tu spada, na primer, algoritam Kara (Hamish Carr) i Kokaja (William Kocay), koji zahteva jedino da graf ima netrivijalnu simetriju i da mu se

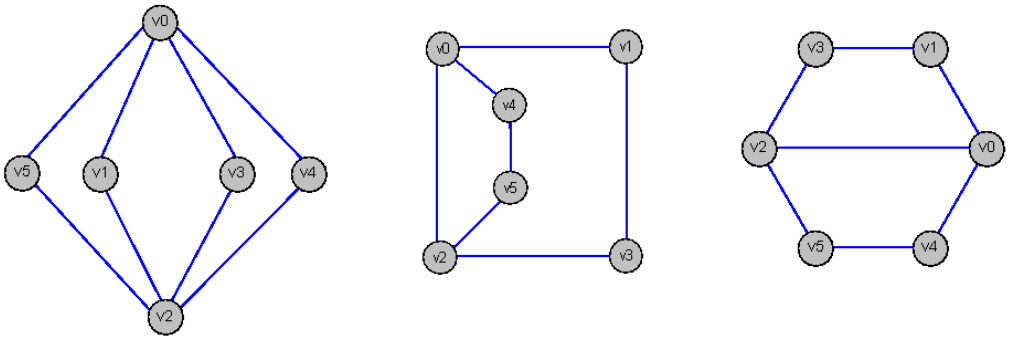
prosledi neki automorfizam datog grafa [77].

2.6.5 Planarni pravolinijski algoritmi

Planarni pravolinijski algoritmi bazirani su na činjenici da, ako se graf može nacrtati bez presecanja grana, pri čemu one mogu biti proizvoljnog oblika, onda taj graf može biti nacrtan na isti način i ako se grane sastoje samo iz pravolinijskih segmenata. Ovu tvrdnju dokazalo je više naučnika još u prvoj polovini 20. veka [78]. Sam problem crtanja planarnih grafova na ovaj način proizišao je iz crtanja električnih mreža, pre svega VLSI kola.

U okviru ove klase algoritama za crtanje grafova može se uočiti više podgrupa, kao što su planarno konveksni, ortogonalni, pravougaoni, polilinijski, te algoritmi koji čvorove postavljaju u temena koordinatne mreže.

Konveksni crtež nekog planarnog grafa je njegova reprezentacija, takva da su granice njegovih lica konveksni poligoni. Pored ovog uslova, crteži moraju zadovoljiti i prethodno spomenute, odnosno, da se sve grane sastoje isključivo iz pravih linija i da ukupan broj njihovih preseka bude 0 [79]. Nemaju svi planarni grafovi konveksni crtež. Tut (William T. Tutte) je pokazao da svaki tripovezani graf ima ovakav crtež i njegov već spomenuti algoritam [75] generiše crteže koji ispunjavaju uslov konveksnosti lica. Graf, međutim, ne mora biti tripovezan da bi imao konveksan crtež. U [79] predstavljen je algoritam koji proverava da li dati bipovezani graf ima pomenuti crtež i, ako ima, nalazi ga. Nekoliko primera konveksnih crteža prikazano je na slici 2.22.



Slika 2.22: Konveksni crteži bipovezanog grafa

Konveksno crtanje grafova, iako proučavano još u vreme kada je oblast crtanja grafova bila u svom začetku, i danas je izuzetno aktuelno. Naime, izučavaju se različite klase grafova i karakteritike koje konveksni grafovi poseduju [80]–[82].

Planarni ortogonalni i polilinijski algoritmi usresređeni su na veličinu uglova između grana. Odnosno, na međusobnu udaljenost grana koje izlaze iz istog čvora grafa. Može se primetiti da što je manji ugao između grana istog čvora, to je veća šansa da će delovati kao da su one preklopljene. Naravno, uglovi između grana čvora visokog stepena ne mogu parirati onima između grana čvora manjeg stepena. Cilj je maksimizovati najmanji ugao.

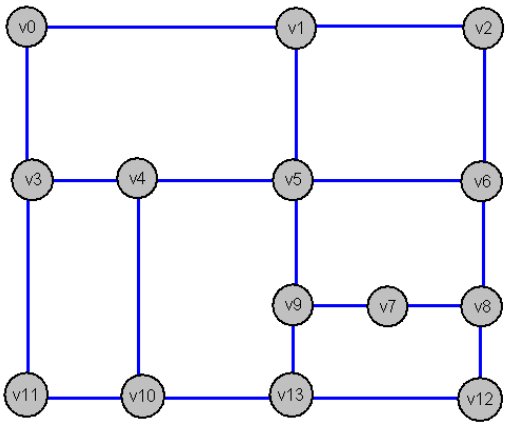
Ortogonalni algoritmi proizvode crteže kod kojih se grane mogu sastojati samo iz horizontalnih i vertikalnih segmenata. U tom slučaju, najmanji ugao između susednih grana je 90°, te su crteži poprilično lepi. Međutim, nije teško zaključiti da tada nijedan čvor ne sme imati stepen veći od 4, inače se algoritam koji pripada ovoj klasi ne može primeniti[83].

Prvi pristupi za kreiranje ortogonalnih crteža kojima se postiže linearno vreme izvršavanja uz zadovoljavanje postavljenih estetskih kriterijuma (minimizacija broja preloma grana i mala ukupna površina koju crtež obuhvata) koriste vidljivu reprezentaciju. Vidljiva reprezentacija grafa $G = (V, E)$ Γ mapira svaki čvor $v \in V$ na horizontalni segment čvora $\Gamma(v)$ i svaku granu $(u, v) \in E$ na vertikalni segment grane $\Gamma(u, v)$ takve da svaki segment $\Gamma(u, v)$ ima krajeve koji leže na horizontalnim segmentima $\Gamma(u)$ i $\Gamma(v)$ i ne seče se niti preklapa sa bilo kojim drugim segmentom. Vidljivu reprezentaciju uveli su Oten (R. Otten) i fan Vijk (J.G. van Wijk) [84] 1978. godine, da bi osam godine kasnije Tamasija (Roberto Tamassia) i Tolis pokazali da graf ima vidljivu reprezentaciju ako i samo ako je planaran, u kom slučaju se ona može naći u linearnom vremenu. Algoritam za ortogonalno crtanje grafa Otena i Fan Vijka bio je i inspiracija za algoritam koji modifikuje u manjoj meri njegovu ideju, dajući nešto kompaktnije crteže [85]. Vidljive reprezentacije takođe su i dalje aktuelna oblast istraživanja, te se tako može spomenuti dodatno poboljšanje poslednjespomenutog algoritma u vidu smanjenja širine vidljive reprezentacije iz 2005. godine [86], kreiranja vizuelne reprezentacije 4-povezanih grafova skoro optimalne visine iz 2009. [87], te izučavanja mogućnosti kreiranja ove reprezentacije i za grafove koji nisu planarni. Na primer, Brandenburg je 2014. predložio algoritam koji se odnosi na 1-planarne grafove (sa maksimalno jednim presekom za svaku granu) [88].

Drugi pristup zastupljen u ortogonalnom crtanju grafova obuhvata kreiranje mrežnih tokova. Mrežni tok je mreža sa dva specijalna čvora - polazišnim i odredišnim (izvorom i ponorom). Polazišni čvor nema ulazne, a odredišni izlazne grane. Mrežni tokovi mogu se koristiti za opis problema transportovanja nekog produkta od polazišta do odredišta, pri čemu kroz jednu granu ne sme proći više produkata nego što dozvoljava kapacitet. Problem mrežnog toka dalje se definiše kao sprovođenje maksimalne količine produkata od izvora do odredišta uz poštovanje ograničenja kapaciteta. Takođe, za svaki čvor, koji nije ni izvor ni ponor, ulazni broj jedinica toka jednak je izlaznom. Algoritmi za ortogonalno crtanje grafova koji koriste mrežne tokove računaju ortogonalni oblik grafa pomoću njih. Naime, u takvoj mreži produkti su zapravo uglovi između susednih grana. Svaka jedinica toka povezana je sa pravim uglom u ortogonalnom obliku. Drugi korak ovih algoritama predstavlja dodelu celobrojnih dužina segmentima granama ortogonalnog oblika. Ovu tehniku 1987. godine iskoristio je za crtanje grafova koji ne sadrže čvor stepena većeg od 4 Tamasia [37]. Algoritam teži minimizaciji broja preloma grana, gde bi optimalno rešenje predstavljalo ortogonalni crtež kod kojega se sve grane sastoje iz samo jednog segmenta. Međutim, problem minimizacije broja preloma grana je NP-kompletan [89] i ne postoji algoritam sa linearnim vremenom izvršavanja. Ipak, u novije vreme rađeno je na poboljšanju performansi originalnog pristupa, kao u radu [90] iz 2012. godine.

Specifičniji tip ortogonalnog crtanja grafova jeste kreiranje pravougaonih crteža. Ovaj tip algoritama dodaje još i uslov da svako lice mora biti nacrtano kao pravougaonik. Veoma su korisni u oblastima dizajna integrisanih kola, kao i arhitekturi za kreiranje šematske reprezentacije postavljanja funkcionalnih blokova, odnosno, rasporeda soba jednog stana ili kuće [91]. Jedan primer ovakvog crteža prikazan je na slici 2.23. Pravougaoni crtež, ukoliko ga graf ima, može se naći u linearnom vremenu pomoću algoritma [92].

Za crtanje grafova koji ne ispunjavaju pomenuti restriktivni uslov sa fokusom na uglove između grana koriste se polilinijski algoritmi. Poput ortogonalnih, i ovi algoritmi teže da smanje koliko god je to moguće broj preloma grana, kao i da kreiraju kompaktnije crteže. Danas postoji znatan broj ovih algoritama, poput [36], [93]–[95].



Slika 2.23: Pravougaoni crtež grafa

2.6.6 Silom usmereni algoritmi za raspoređivanje

Silom usmereni algoritmi za automatsko raspoređivanje predstavljaju najfleksibilniji tip algoritama za nalaženje rasporeda čvorova svih jednostavnih neorijentisanih grafova. Dakle, nije potrebno da grafovi budu planarni, bipovezani, tripovezani, stabla itd. Crteži kreirani korišćenjem ovih algoritama često zadovoljavaju više estetskih kriterijuma do određene mere, bivajući simetrični, sa granama slične dužine i malim brojem preseka (ili nijednim) u slučaju planarnih grafova [96]. Osnovna ideja silom usmerenih algoritama obuhvata dodelu sila granama i čvorovima grafa i simulaciju njihovog kretanja ili minimizaciju energije njihovog rasporeda [97].

Prvim algoritmom iz ove grupe smatra se već spomenuti Tutov metod baziran na baricentričnoj reprezentaciji. Tut tvrdi i dokazuje da ako se fiksira lice planarnog tripovezanog grafa u ravni, pozicije ostalih čvorova, takve da na crtežu nema preseka grana, a sva lica budu konveksna, mogu se naći rešavanjem sistema linearnih jednačina. Ovaj metod može se smatrati silom usmerenim, pri čemu se na svaku granu postavljaju privlačne sile nalik na oprugu. Sila je proporcionalna rastojanju između čvorova koje grana spaja, dok sama opruga ima idealnu dužinu jednaku nuli [43].

Veliki napredak u ovoj oblasti predstavlja Ejdsova metoda opruga [98] koja je inspirisala dalje napretke i usavršavanja. Ovaj algoritam, kao i takođe poznati pristup Fruhterman i Rajngold (Fruchterman-Reingold)[99] oslanja se na sile opruga, nalik na Hukov zakon. Naime, Ejds predlaže zamenu svih čvorova čeličnim prstenjem i svih grana oprugama, kako bi se dobio mehanički sistem. Čvorovi se postave u neki početni položaj i puste, te sile opruga povezanih sa prstenjem prevode sistem u stanje minimalne energije. U odnosu na Hukov zakon, Ejds modifikuje određene formule sa ciljem dobijanja lepšeg rasporeda čvorova. Konkretno, fokusira se na simetriju i jednaku dužinu grana, dok Fruhterman i Rajngold vode računa i o ujednačenom rasporedu čvorova. Naime, oni posmatraju čvorove grafa kao svemirska tela i računaju privlačne i odbojne sile između njih. Oni uvode i pojam temperature, koja se smanjuje kako raspored postaje bolji. Upotreba temperature je specijalan slučaj opštije tehnike simuliranog kaljenja, koja se koristi u pojedinim modernijim silom usmerenim algoritmima za raspoređivanje. Simulirano kaljenje potiče iz metalurgije i predstavlja fizički proces hlađenja istopljenog materijala.

Kamada (Tomihisa Kamada) i Kavai (Satoru Kawai) sa druge strane predlažu definisanje funkcije koja mapira rasporede elemenata grafa na pozitivan realni broj koji

predstavlja njihove energije, te nalaženje njegovog minimuma. Sa estetske tačke gledišta, Kamada i Kavai najlepšim crtežom grafa smatraju onaj kod kojega su geometrijske udaljenosti između parova čvorova jednake onim koje se izračunavaju algoritmom najkraće putanje između svih parova. Dobijeni raspored čvorova uglavnom je poprilično intuitivan i crteži se mogu smatrati lepim, ali je sam algoritam relativno neefikasan - upravo zbog računanja rastojanja između parova.

Svi spomenuti algoritmi dizajnirani su za primenu nad relativno malim grafovima. Naime, i Ejds, i Fruhterman i Rajngold svoje ideje testirali su nad grafovima sa svega 30 do 40 čvorova. Algoritmi se, naravno, mogu primeniti i nad znatno većim grafovima, ali im dobra performansa prilikom raspoređivanja grafova sa više hiljada čvorova nije postavljena kao jedan od bitnijih ciljeva. Osim vremena izvršavanja, postoji još jedan problem kod osnovnih silom usmerenim algoritama koji se uočava već prilikom primene nad grafovima sa nekoliko stotina čvorova. Naime, fizički modeli uglavnom imaju puno lokalnih minimuma i često se dešava da algoritam završi u njima. Međutim, ovo ne umanjuje značaj pomenutih algoritama, koji su doveli do velikog interesovanja mnogih drugih naučnika za silom usmereno raspoređivanje. Broj pripadnika ovoj klasi od devedesetih godina prošlog veka do danas mnogostruko je uvećan.

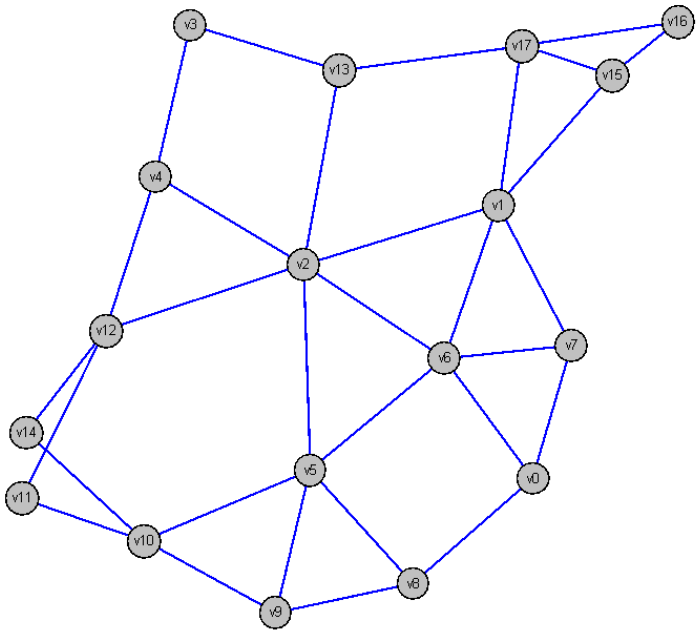
Pojedini autori usredsredili su se na poboljšanja poznatih metoda, dok su drugi nudili potpuno originalna rešenja. U vezi s poboljšanjem metoda opruga, može se izdvojiti algoritam Dejvidsona i Harela (Davidson-Harel) iz 1996. godine koji koristi spomenutu tehniku simuliranog kaljenja [100]. Takođe, veliki napredak je postignut na polju crtanja velikih grafova, pa tako danas postoje algoritmi dizajnirani za primenu nad gafovima sa desetinama, pa čak i stotinama hiljada čvorova. Hadani (Ronny Hadany) i Harel (David Harel) prvi su osmislili silom usmereni algoritam za crtanje većih grafova [101], testirajući ga nad grafovima do 1000 čvorova, da bi veličina grafova kojima su tvorci isprobavali svoje algoritme vremenom bivala sve veća. Neki primeri algoritama dizajniranih za primenu nad velikim grafovima su [102]–[105].

Imajući sve navedeno u vidu, nije iznenađujuća popularnost algoritama ovog tipa. Mogu se primeniti nad bilo kojim grafom, a rezultujući crteži obično zadovoljavaju više estetskih kriterijuma, bar do određene mere. Na primer, planaran graf možda neće biti nacrtan bez ijednog preseka grana, ali će taj broj biti poprilično nizak. Na slici 2.24 prikazan je crtež grafa kreiran pomoću algoritma Fruhtermana i Rajngolda. Graf je planaran, dok se na crtežu uočava jedan presek grana, što nije uopšte loše, uzimajući u obzir da algoritam nije specijalno dizajniran za planarne grafove. Štaviše, primena nekog novijeg algoritma, koji unapređuje osnovnu ideju spomenutih autora, dovela bi i do crteža bez ijednog preseka.

2.6.7 Ostale klase algoritama za raspoređivanje

Uz prethodno detaljnije opisane klase algoritama za automatsko raspoređivanje, može se izdvojiti još nekoliko koje imaju bitnu praktičnu primenu i više konkretnih algoritama koji im pripadaju. Tu se može svrstati crtanje po blizini i slojevito crtanje grafova.

Graf bliskosti može se neformalno definisati kao pravolinijski crtež konstruisan od skupa tačaka P u nekom metričkom sistemu povezivanjem parova tačaka koje su dovoljno blizu jedna drugoj. Iz jednog skupa P može proizaći više grafova bliskosti, u zavisnosti od same definicije bliskosti. Odnosno, kada su dve tačke dovoljno blizu. Grafovi bliskosti primenjuju se za opis oblika nekog skupa tačaka u oblastima kao što su kompjuterska grafika, prepoznavanje šablona, numerička analiza, geografski informacioni sistemi itd. U kontekstu crtanja grafova, postavlja se problem nalaženja pravolinijskog crteža grafa takvog da je on graf bliskosti. Takvi crteži bliskosti imaju



Slika 2.24: Crtež grafa kreiran primenom algoritma Fruhtermana i Rajngolda

nekolicinu zanimljivih osobina. Na primer, na njih ne utiče promena razmere, a povezani čvorovi se crtaju bliže jedan drugom u odnosu na nepovezane, dok se čvorovi koje ne povezuje data grana ne crtaju blizu nje.

Izdvajaju se dva načina definisanja bliskosti: na osnovu koncepta regiona bliskosti i na osnovu globalne mere bliskosti. Kod prvog su dva ili više čvorova bliska ako i samo ako neki pogodno definisani region koji opisuje njihovo susedstvo sadrži još najviše k čvorova, za neko dato $k \geq 0$. Globalna bliskost proizvodi crteže kod kojih je ukupan zbir dužina grana na crtežu minimalan [106].

Slojeviti crtež grafa predstavlja crtež na kojem čvorovi leže na nekim geometrijskim slojevima, koji mogu biti linije, krugovi ili neke druge krive. Razdvajanje čvorova na posebne slojeve može biti dobar način za naglašavanje određenih strukturnih obeležja grafa. Prilikom slojevitog crtanja orijentisanih grafova postavlja se zahtev ujednačenog toka, dok o ovome ne treba voditi računa kada je ulazni graf neorijentisan. U teoriji, slojevi mogu biti bilo kakvi oblici, ali je sa praktične tačke gledišta poželjno da se radi ili o paralelnim pravim linijama ili koncentričnim kružnicama. Algoritmi se fokusiraju na planarne grafove i nalaženje crteža bez preseka grana.

Crteži istog grafa se u velikoj meri razlikuju u zavisnosti od broja slojeva. Za crtež koji ima dva ili tri sloja može se tvrditi da je lepši od onog koji ima samo jedan, te svi čvorovi leže na istoj liniji ili kružnici. Ako je k broj slojeva, a b maksimalni broj preloma grana grafa, dati graf se ne može nacrtati na željeni način za bilo koje vrednosti ovih varijabli. Shodno tome, oblast slojevitog crtanja grafova bavi se i nalaženjm njihovih najvećih vrednosti, kao i ispitivanju da li za dato k i b dati planarni graf može biti nacrtan bez preseka grana, na k slojeva i sa maksimalno b preloma grana.

2.7 Jezici specifični za domen

Jezici specifični za domen (*Domain-Specific Languages*) su računarski jezici specijalizovani za određeni domen problema [107]. Drugim rečima, reč je o jezicima koji

koriste koncepte i notacije prilagođene određenom domenu, za razliku od programskih jezika opšte namene. Upotreba koncepata problemskog domena omogućava domenskim ekspertima pod određenim uslovima samostalno korišćenje datog jezika specifičnog za domen (JSD). Takođe, time se izbegava i ručno mapiranje na koncepte ciljne implementacione platforme, kako je to posao kompajlera ili generatora koda [108]. Korišćenjem koncepata domena problema smanjuje se i broj linija koda kojima se postiže određeni cilj, što ima pozitivan uticaj na brzinu razvoja i održavanje [109]. Dodatno, specifikacija pisana jezikom specifičnim za domen predstavlja istovemeno dizajn, implementaciju i dokumentaciju sistema [110].

Jezici specifični za domen mogu se klasifikovati na razne načine. Martin Favler (Martin Fowler) to radi na osnovu načina njihove konstrukcije [111] uvodeći pojmove:

- Eksternih jezika specifičnih za domen, koje karakteriše činjenica da se pišu od početka, bez oslonca na postojeće jezike. Njihov razvoj uključuje definiciju sintakse i kreiranje ili kompajlera, koji prevodi programe pisane na datom jeziku na druge programske jezike, ili interpretera.

- Internih jezika specifičnih za domen, koji nastaju proširivanjem već postojećih programskih jezika. Ovakvi jezici su manje fleksibilni, ali je njihov razvoj generalno jednostavniji jer već postoje integrisana razvojna okruženja za jezik na koji se oslanjaju.

Jezici specifični za domen mogu imati različite notacije, odnosno, konkretne sintakse. Najzastupljenije notacije su tekstualne i grafičke, ali postoje i druge alternative, poput tabela/formi ili stabala. Svaka od njih ima određene prednosti i mane, te nije neobično da se u konkretnoj situaciji podrži i više. Svaki jezik može imati tekstualnu sintaksu, ali to nije njena jedina prednost. Naime, istraživanja pokazuju da programeri preferiraju tekstualnu sintaksu, zbog bliskosti alata za rad sa tekstualnim jezicima (dostupnost editora, sistema za kontrolu verzija poput *Gita* itd.). Grafička sintaksa je, sa druge strane lakša za upotrebu domenskim ekspertima, ukoliko je prirodna za njihov domen. Dobro dizajnirana grafička sintaksa omogućava intuitivan razvoj modela, odnosno, programa u datom jeziku specifičnom za domen [112]. Posebna poteškoća povezana sa grafičkim sintaksama jeste i potreba za razvojem grafičkih editora u cilju njihove podrške.

Poglavlje 3

Pregled postojećih rešenja

U ovom poglavlju biće dat prikaz postojećih biblioteka za crtanje i analizu grafova za programski jezik *Java*, uz akcenat na pregledu i oceni kvaliteta algoritama koje implementiraju, kao i mogućnost njihovog pozivanja od strane posebno razvijenih grafičkih editora. Kako biblioteke kao glavni cilj postavljaju vizualizaciju grafova, odnosno, generisanje alata za njihovo crtanje i analizu, primena algoritama van komponenti koje su kreirane samom bibliotekom nije uvek jednostavna. Takođe, biće i reči o jezicima specifičnim za domen koji se bave grafovima, uz detaljniji opis onih koji se bar do neke mere dotiču i zadavanja načina raspoređivanja njihovih elemenata.

3.1 Biblioteke za vizualizaciju grafova za programski jezik Java

Postoji veći broj biblioteka za analizu i vizualizaciju grafova za programski jezik *Java*. U ovoj sekciji biće detaljnije prikazane najbolje biblioteke ovog tipa otvorenog koda, kako fokus nije stavljen na komercijalna rešenja, kao što je *yFiles* [113], niti na besplatne alate za crtanje i analizu grafova čiji kod nije javno dostupan. Naime, od posebnog interesa su projekti koji dozvoljavaju upotrebu implementiranih funkcionalnosti u drugim projektima bez postavljanja bilo kakvog ograničenja.

Pregled biblioteka pre svega će biti usredsređen na skup algoritama za crtanje grafova koje poseduju, uz isticanje kompleksnijih algoritama za njihovu analizu. Algoritmi za raspoređivanje elemenata grafova biće testirani radi određivanja njihove računarske efikasnosti i poređenja implementacija istih ili sličnih algoritama različitih biblioteka. Takođe, biće reči i o mogućnosti i složenosti korišćenja pojedinačnih algoritama implementiranih u okviru izdvojenih biblioteka unutar posebno razvijenih grafičkih editora.

3.1.1 Pregled postojećih biblioteka

JUNG — the Java Universal Network/Graph Framework [11] je biblioteka otvorenog koda koja pruža svojim korisnicima mogućnost modelovanja, analize i vizualizacije podataka koji se mogu predstaviti grafom ili mrežom. U celosti je napisana na programskom jeziku *Java* i podržava različite tipove grafova, kao što su orijentisani i neorijentisani, multigrafovi i hipergrafovi. Hipergrafovi su grafovi kod kojih jedna grana može povezivati proizvoljan broj čvorova. Biblioteka je licencirana BSD licencom, koja nameće minimalne restrikcije na redistribuciju softvera na koji se odnosi [114].

Tekuća verzija *JUNG* bilioteke poseduje implementacije većeg broja algoritama iz teorije grafova, ali se bavi i oblastima analiza podataka i društvenih mreža. Naime, korisnici ovog alata mogu iskoristiti njegove rutine za klasterovanje, računanje rastojanja u mrežama, računanje mera važnosti, kao što je popularni *PageRank* algoritam [115]. Upravo spomenute funkcionalnosti nisu od većeg interesa za problem crtanja grafova i automatizacije ovog postupka, te neće biti detaljnije opisane. Sa druge strane, *JUNG* sadrži i implementacije algoritama koji jesu veoma interesantni u pomenutom kontekstu. Tu spadaju Dijkstrina najkraća putanja i rastavljanje grafova na bipovezane komponente, ali i veći broj implementacija algoritama za raspoređivanje elemenata grafova, od kojih su neki veoma kompleksni.

Od raspoloživih algoritama za crtanje grafova pre svih se mogu izdvojiti algoritmi za crtanje stabala i silom usmereni. Pored njih *JUNG* biblioteka poseduje i jedan kružni, ali on je poprilično bazičan, jer ne vodi računa o minimizaciji broja preseka grana niti omogućava kreiranje klastera. Sa druge strane, algoritmi za crtanje stabala su vredni pomena, uključujući algoritam za crtanje stabala baziran na nivoima, radijalnu metodu i metodu balona. Algoritmi mogu biti primenjeni i nad šumama, odnosno, više nepovezanih stabala. Svako stablo se zasebno crta, te i ona sama bivaju raspoređena u okviru crteža. Omogućena je konfiguracija pojedinih parametara algoritama, kao što je razmak između nivoa i poluprečnik kružnice na koju se postavljaju čvorovi kod metode balona.

Silom usmereni algoritmi koje implementira *JUNG* okvir jesu metoda opruga Ejdsa, algoritmi Kamada-Kavai i Fruhterman-Rajngold, kao i Mejerova (Bernd Meyer) samoorganizujuća metoda [116], kraće nazvana ISOM. Kao i u slučaju algoritama za crtanje stabala, i implementacije silom usmerenih dozvoljavaju podešavanje nekolicine parametara, povezanih sa privlačnim i odbojnim silama i brojem iteracija. Svi parametri imaju definisane podrazumevane vrednosti koje daju zadovoljavajuće rezultate u većini slučajeva, te ručno konfigurisanje nije neophodno. Naravno, optimalno postavljeni parametri za dati graf proizvešće lepše crteže nego u slučajevima kada se algoritmi pozovu bez promene podrazumevanih vrednosti.

Povrh spomenutih algoritama, *JUNG* biblioteka veoma veliki akcenat stavlja na vizualizaciju podataka, pružajući razvojni okvir za kreiranje alata za interaktivnu analizu mreža i grafova. Kako ni ova funkcionalnost nije glavni fokus istraživanja, koje teži unapređenju procesa raspoređivanja elemenata postojećih alata za vizualizaciju, pomenuto neće biti detaljnije opisano.

JGraphX [12] je *Java Swing* biblioteka za vizualizaciju grafova. Projekat je inicijalno nazvan *JGraph*, ali je kompletan kod ponovo napisan u verziji 6, te je i preimenovan kako bi se dodatno naznačila promena. Poput *JUNG* biblioteke, i *JGraphX* se distribuira pod nerestriktivnom licencom.

JGraphX omogućava kreiranje interaktivnih editora za vizualizaciju grafova. U sklopu ove funkcionalnosti implementiran je i određeni broj algoritama za anlizu grafova, poput njihovog obilaska, kreiranja razapinjućih stabala i Dijkstrine najkraće putanje. Bitnije, ova biblioteka poseduje i raznolike, većinom poprilično kompleksne i konfigurabilne implementacije algoritama za crtanje grafova. Radi se o nekoliko silom usmerenih algoritama, kao i jednom algoritmu za crtanje stabala i jednom hijerarhijskom, koji ne samo što raspoređuje čvorove, nego se bavi i rutiranjem grana.

Konkretno, *JGraphX* sadrži takozvani kompaktni algoritam za crtanje stabala, koji predstavlja poboljšanje standardnog baziranog na nivoima, težeći da rezultujuće crteže učini što je moguće kompaktnijim. To je, naime, u skladu sa prethodno spomenutim estetskim kriterijumom, čije prisustvo se smatra poželjnim, pogotovo kada je o stablima reč. Treba napomenuti da je algoritam veoma podesiv, dopuštajući postavljanje

vrednosti niza parametara koji u velikoj meri utiču na kvalitet krajnjeg rasporeda elemenata. Radi se o rastojanju između čvorova kako različitih, tako i istih nivoa, ali i orijentaciji stabla. Dakle, stablo se može nacrtati ne samo standarno, vertikalno, nego i horizontalno. Odnosno, tako da se čvorovi raspoređuju s leva-na-desno, a ne od gore prema dole. Hijerarhijski algoritam takođe se odlikuje visokim stepenom podesivosti različitih obeležja. Slično kao i kod algoritma za crtanje stabala, radi se o različitim rastojanjima (čvorova na istim i različitim slojevima, između samih čvorova i nepovezanih hijerarhija), ali i orijentaciji crteža. Naime, i ovaj algoritam je u stanju da čvorove rasporedi kako u horizontalnom, tako i u i vertikalnom smeru, ali ide i korak dalje te određuje da li će tok ići od gore prema dole, od dole prema gore, s leva-na-desno ili zdesna-na-levo.

Dalje, *JGraphX* osim veoma interesantnog hijerarhijskog algoritma i algoritma koji se fokusira na stabla, poseduje i implementacije dva silom usmerena, koji se ne mogu okarakterisati kao osnovne verzije algoritama date klase. Radi se o:

- Brzi organski algoritam, koji je sličan metodi opruga, ali modifikuje formule sa ciljem poboljšanja efikasnosti računanja privlačnih i odbojnih sila, kao i prevazilaženja problema lokalnog minimuma. Algoritam, međutim, najbolje rezulzate daje pri primeni nad manjim grafovima regularne strukture. Prilikom primene nad odgovarajućim grafovima, algoritam važi za jedan od bržih silom usmerenih algoritama.

- Organski algoritam, koji je najkompleksnija implementacija u biblioteci. Baziran je na metodi simuliranog kaljenja Dejvidsona i Harela. Implementacija se odlikuje izuzetno velikom konfigurabilnošću. Naime, moguće je specifikovati da li i u kojoj meri će biti optimizovani preseci grana i udaljenosti među čvorovima, da li se teži jednakoj distribucija čvorova, površina u kojoj će oni biti pozicionirani itd. Može se primetiti da su ovi parametri u direktnoj vezi sa estetskim kriterijumima, te da je njihova pravilna konfiguracija može dovesti do kreiranja crteža u skladu sa žljama i očekivanjima korisnika. Dodatno, algoritam poseduje još jednu dobru osobinu koju treba spomenuti. Radi se o činjenici da će pri svakom izvršavanju za iste vrednosti parametara biti dobijeni isti crteži. Dakle, ako, na primer, pri prvom pozivu nije bilo preseka grana, neće ni u sledećem. Ovo ne važi za veliku većinu algoritama ove klase, implementiranih bilo u *JGrapX* biblioteci, bilo u nekoj drugoj, a spomenutoj.

Svi algoritmi mogu biti primenjeni nad više nepovezanih komponenti jednog grafa, koje će i same biti raspoređene na odgovarajući način. Takođe, podesivi parametri imaju pažljivo izabrane podrazumevane vrednosti, te će dobijeni rezultati biti dobri čak i bez ikakve konfiguracije. Biblioteka se bavi i razdvajanjem preklapajućih grana, definišući poseban algoritam koji se može pozvati po završetku prvoizabranog.

Prefuse je softverski okvir za kreiranje dinamičke vizualizacije strukturiranih i nestrukturiranih podataka. Najveći akcenat stavlja na jednostavno, brzo i kvalitetno dizajniranje aplikacije za vizualizaciju. Štaviše, nudi i mogućnost animacije prikazanih dijagrama. *Prefuse* je takođe licenciran BSD licencom. Okvir je spojen sa bibliotekom, koja, između ostalog, poseduje više implementacija algoritama za crtanje grafova, te će upravo ovaj njegov aspekt biti detaljnije opisan.

Algoritmi pomenute namene uključuju kružni i silom usmereni, kao i više algoritama za crtanje stabala. Silom usmereni algoritam implementiran je tako da omogući korisnicima definisanje simulatora sila koji je u njegovoj osnovi. Time se zapravo omogućava kreiranje proizvoljnih silom usmerenih metoda. Ovo, naravno, nije

obavezno, te korisnici ne moraju, ukoliko ne žele, bilo šta konfigurisati. Kao i prethodno opisane biblioteke, i *prefuse* postavlja podrazumevane vrednosti parametrima, ukoliko se eksplicitno ne navedu. Ponašanje koje se dobija bez podešavanja zasniva se na međusobnom odbijanju, pošto se grane predstavljaju opruge. Silom usemereni algoritam se može izvršavati kao animacija, što obuhvata njegovo ponavljanje veći broj puta, ili na klasičan način, samo jednom. Animacija nije od interesa za tekući pregled.

Algoritmi za crtanje stabala koje *prefuse* nudi svojim korisnicima obuhvataju radijalni, balon i nivovski metod. Poslednjespomenuta implementacija bazirana je na radu koji unapređuje performanse Vokerovog (Walker's) [46] algoritma, tako da se izvršava u linearnom vremenu [48]. Ova implementacija takođe dozvoljava postavljanje orijentacije stabla.

Može se napomenuti da *prefuse* poseduje i bazični kružni algoritam, kao i algoritam koji čvorove postavlja na jednu horizontalnu ili vertikalnu liniju, te algoritme za vizualizaciju dijagrama koji se ne uklapaju u pojam grafa.

Verovatno najzastupljeniji alat za vizualizaciju grafova u današnje vreme jeste *Graphviz* [10]. On omogućava generisanje crteža dijagrama na osnovu opisa u *DOT* [117] jeziku, koji će kasnije biti detaljnije prikazan. Krajnji rezutat je slika u jednom od nekoliko podržanih formata. Sama po sebi, ovakva funkcionalnost nema primenu u okviru već postojećih editora, te je mnogo interesantnije razmotriti mogućnost direktnog pozivanja nekog algoritma *Graphviz* biblioteke. Ona je, međutim, pisana na programskom jeziku *C*, te je upotreba željenog algoritma iz *Jave* nešto kompleksnija. Kako je kod izvršavanja algoritama za prikaz većih grafova računarska efikasnost veoma bitna, treba izbegavati bilo kakvo dodatno usporenje. Tvorci *Smetana* projekta [118] uviđaju više mogućih problema kod direktnog pozivanja *Graphviz* funkcija, te rade na realizaciji prevođenja kompletnog koda pomenutog alata na programski jezik Java. Međutim, ima puno još prostora za rad, jer je podržan samo mali podskup *Graphviz* funkcionalnosti.

3.1.2 Analiza i poređenje implementacija algoritama

Iz prikaza biblioteka može se primetiti da su najzastupljenije implementacije silom usmerenih i algoritama za crtanje stabala. Ova činjenica nije iznenađujuća, pošto je silom usmerena najfleksibilnija klasa algoritama za raspoređivanje elemenata grafova, a stabla tip grafova koji se veoma često pojavljuju u praksi. U ovoj sekciji biće analizirane nađene implementacije, pri čemu će biti diskutovano o njihovoj računarskoj efikasnosti i estetici crteža. Implementacije sličnih algoritama različitih biblioteka će biti upoređene, s obzirom na to da je krajnji cilj izdvajanje najboljih radi uključivanja u sveobuhvatnu biblioteku za crtanje i analizu grafova. Takođe će biti i reči o klasama algoritama čiji predstavnici nisu prisutni u analiziranim rešenjima.

Algoritmi za crtanje grafova ocenjuju se na osnovu resursa koje troše, kao i pridržavanja estetskih kriterijuma. Iz tog razloga, mereno je vreme potrebno za njihovo izvršavanje nad većim grafovima. Razmatrani algoritmi nisu posebno dizajnirani za primenu nad ekstremno velikim grafovima. Kao što je ranije spomenuto, ciljni grafovi često su imali tek nekoliko desetina čvorova. Međutim, jačanje računara dovelo je do mogućnosti njihove primene nad i stotinu puta većim grafovima, te je određeno da grafovi nad kojim će se izvršiti merenje imaju upravo 1.000 čvorova. U vreme izvršavanja uključuje se i priprema za pozivanje algoritma koja bi bila nužna u slučajevima kada se koriste iz drugih alata i grafičkih editora. Dakle, u situacijama kada je prvo potrebno kreirati strukturu koju algoritam prima. Ovaj faktor ne sme biti zanemaren jer se ne može preskočiti osim ako se algoritmi ne koriste baš u okviru vizualizatora kreiranih

Algoritam	Vreme [ms]
Metoda opruga (*JUNG*)	724
Fruhterman-Rajngold (*JUNG*)	689
Kamada-Kavai (*JUNG*)	5.232
ISOM (*JUNG*)	461
Brzi organski (*JGraphX*)	9.584
Organski (*JGraphX*)	90.273
Metoda opruga (*prefuse*)	1.943
Hijerarhijski (*JGraphX*)	56.100

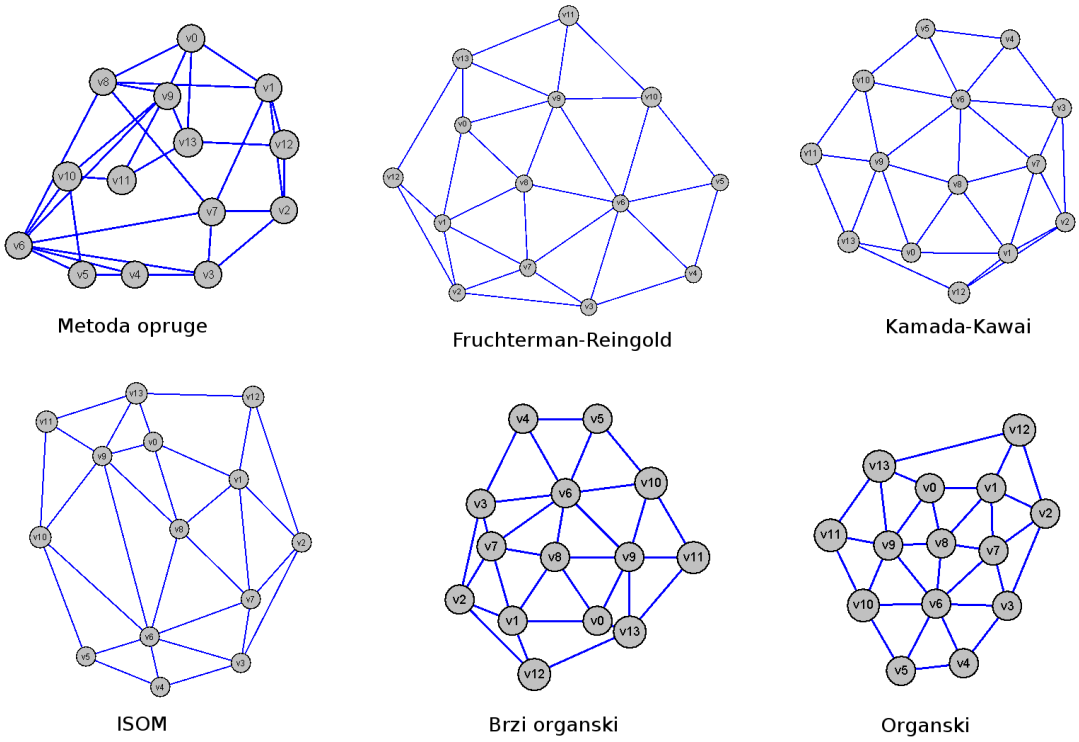
Tabela 3.1: Vreme potrebno za izvršavanje algoritama nad nasumičnim grafova sa 1.000 čvorova

pomoću date biblioteke tj. okvira.

Prva serija merenja rađena je za proizvoljne grafove koji su nasumično automatski generisani, što znači da nisu nekog određenog tipa. Dakle, primenjeni su samo oni algoritmi koji ne zahtevaju prisustvo određenih osobina. U slučaju razmatranih biblioteka, to su razni silom usmereni i hijerarhijski. Da bi se dobili precizniji rezultati, generisano je 10 grafova, te izračunato prosečno vreme izvršavanja. Time se smanjuje uticaj grafova koji eventualno posebno odgovaraju ili ne odgovaraju nekom algoritmu. Takođe, parametri algoritama nisu posebno podešavani, već su ostavljene podrazumevane vrednosti. Izvršavanje algoritma moglo bi se ubrzati ako bi se, recimo, smanjio broj iteracija, ali takva modifikacija dovela bi do manje preglednog crteža. Dobijeni rezultati prikazani su u tabeli 3.1.

U [103] predstavljeni su neki veoma efikasni silom usmereni algoritmi i izvršeno je merenje njihovih performansi. Za grafove veličine poput onih korišćenih za tekuće testiranje, navodi se da se mogu rasporediti za manje od jedne sekunde. Dobijeni rezultati pokazuju da se u rangu ove efikasnosti nalazi metoda opruga, Fruhterman-Rajngold, i ISOM *JUNG* okvira, pri čemu je poslednjespomenuti najbrži. Najdalje od ovog ideala su organski i hijerarhijski. Za hijerarhijski svakako se i ne može očekivati dobra performansa nad grafovima koji nisu hijerarhije, iako ga je nad njima moguće primeniti. Međutim, treba imati u vidu da sporiji ali kompleksniji algoritmi vrlo često daju bolje rezultate, koji su u većoj meri u skladu sa raznim estetskim kriterijumima. Na slici 3.1 prikazano je šest crteža istog grafa, dobijenih primenom analiziranih silom usmerenih metoda vidno različitog kvaliteta.

Prvi crtež kreiran je metodom opruge, koja se dobro pokazala po pitanju vremenske efikasnosti. Kao što se vidi, isto se ne može reći i za estetiku, kako je broj preseka grana velik, iako je graf planaran, dužine grana raznolike, a čvorovi nisu ravnomerno raspoređeni. Metode Fruhterman-Rajngold i Kamada-Kavai dale su veoma slične rezultate. Kod njih se već zapaža bliže pridržavanje spomenutih estetskih kriterijuma. Algoritam Kamada-Kavai proizveo je nešto kompaktniji crtež, ali je i dosta sporiji. ISOM metodom generisan je crtež grafa bez preseka grana, ali sa znatnom razlikom u njihovim dužinama. Kako se algoritam pokazao najbržim, a rezultat njegove primene solidnim sa stanovišta estetike, treba ga posebno izdvojiti kao pogodnog za rad sa velikim grafovima u situacijama kada je vreme izvršavanja značajan faktor. Brzi organski je zapravo drugi najsporiji od svih testiranih silom usmerenih, ali i znatno brži od najsporijeg, organskog. Crtež koji je nastao kao posledica njegove primene ne može se posebno pohvaliti. Znatno je kompaktniji, ali su pojedini čvorovi skoro preklopljeni. Konačno, može se spomenuti organski, koji jeste najmanje efikasan, ali



Slika 3.1: Šest crteža istog grafa

Algoritam	Vreme [ms]
Nivovsko stablo (*JUNG*)	60
Radijalno stablo (*JUNG*)	52
Kompaktno stablo (*JGraphX*)	140
Balon stablo (*prefuse*)	158
Nivovsko(čvor-veza) stablo (*prefuse*)	461
Radijalno stablo(*prefuse*)	122
Hijerarhijski (*JGraphX*)	621

Tabela 3.2: Vreme potrebno za izvršavanje algoritama nad stablima sa 1.000 čvorova

je rezultat bez sumnje najbliži postavljenim estetskim kriterijumima. Prikazane slike izabrane su kao prosek kvaliteta koji se može postići primenom datih pet metoda.

Druga serija testova rađena je nad većim stablima. Stabla su formirana obilaskom najpre u dubinu nasumično generisanih grafova. Postupak je ponovljen na istovetan način, ali su primenjeni samo algoritmi specijalizovani za ovaj tip grafova, kao i hijerarhijski. Rezultati su prikazani u tabeli 3.2. Algoritam za crtanje stabala u formi balona *JUNG* biblioteke nije razmatran, budući da je primećeno da crteži dobijeni ovom implementacijom nisu u skladu sa definicijom algoritma.

Dakle, može se zaključiti da su sve implementacije poprilično efikasne. Izvršavanje algoritama *prefuse* okvira nešto je duže u odnosu na ostale, što je pre svega posledica kompleksnog pozivanja, o čemu će u narednom poglavlju biti više reči. Isto se može primetiti i za metodu opruga ovog okvira u odnosu na implementaciju *JUNG* biblioteke. Sve testirane implementacije generišu crteže kakvi se i očekuju, imajući u vidu algoritme na koje se odnose. Može se primetiti da je kompaktno stablo *JGraphX* biblioteke veoma slično baziranom na nivoima druge dve, ali je sam algoritam optimizovan sa ciljem

davanja kompaktnijih crteža, a implementacija poseduje veći nivo konfigurabilnosti. Dakle, upravo se ona može izdvojiti kao najbolje rešenje koje pripada podgrupi algoritama za crtanje stabala baziranoj na nivoima. Radijalna implementacija *JUNG* biblioteke je nešto efikasnija od druge dostupne, dok je jedina korektna implementacija balon pristupa deo *prefuse* okvira. Takođe, može se primetiti da je hijerarhijski algoritam znatno brži nego u slučaju opštih grafova, jer su stabla hijerarhije.

Sumarno, primećeno je da postojeća rešenja obezbeđuju nekoliko dobrih silom usmerenih algoritama, od kojih su izdvojeni:

- Kamada-Kavai *JUNG* bibliteke, kao implementacija koja je prihvatljive brzine izvršavanja i solidne estetike rezultujućeg crteža,
- ISOM *JUNG* biblioteke, kao najbrži algoritam,
- organski *JGraphX* biblioteke, kao algoritam koji, iako je najsporiji, generiše najlepše crteže i odlikuje se izuzetno visokim stepenom konfigurabilnosti.

Pored silom usmerenih, dostupne su i dobre implementacije algoritama za crtanje stabala, od kojih su najinteresantnije:

- crtanje kompaktnih stabala *JGraphX* biblioteke,
- radijalni algoritam *JUNG* okvira,
- crtanje balona *prefuse* okvira.

Pored njih, izdvojen je i hijerarhijski algoritam *JGraphX* biblioteke, koji se dobro pokazuje prilikom crtanja hijerarhija i koji ne samo što raspoređuje čvorove, nego i vodi računa o tome gde treba da se nalaze prelomne tačke višelinijskih grana.

3.1.3 Pozivanje od strane zasebnih grafičkih editora

Pri opisivanju tri biblioteke otvorenog koje pružaju najveći broj funkcionalnosti povezanih sa raspoređivanjem elemenata grafova, spomenuto je da je njihov glavni cilj vizualizacija grafova i mreža, odnosno, generisanje grafičkih editora za njihovo kreiranje i analizu. Stoga nije stavljan akcenat na jednostavnost pozivanja algoritama koje obezbeđuju iz drugih projekata i editora.

Generalno, da bi se pozvao neki algoritam, potrebno je instancirati klasu grafa date biblioteke. Ovo je nešto jednostavnije u slučaju *JUNG* okvira, čiji su grafovi parametrizovani, te instance bilo kojih klasa mogu biti prosleđene kao njihovi čvorovi i grane. Za druge dve biblioteke ovo ne važi, te je prvo potrebno kreirati same elemente grafova. U nastavku su prikazana tri listinga koda, kojima se postiže pozivanje željenog algoritma za crtanje datog grafa i preuzimanje rezultata, odnosno, pozicija čvorova i eventualno prelomnih tačaka grana (ako se algoritam dotiče i tog problema). Prvi listing koda prikazuje pozivanje jednog algoritma *JUNG* okvira, drugi *JGraphX* biblioteke i treći *prefuse* okvira. Podrazumeva se da varijabla `vertices` sadrži listu svih čvorova koji se raspoređuju, a `edges` grana, kao i da se metodama `e.getOrigin()` i `e.getDestination()` dobijaju čvorovi početka, odnosno, kraja veze.

Listing 3.1: Pozivanje jednog algoritma za crtanje grafova *JUNG* okvira

```
Graph<V,E> jungGraph = new UndirectedSparseGraph<V,E>();

for (V v : vertices))
```

```
    jungGraph.addVertex(v);

for (E e : edges())
    jungGraph.addEdge(e, e.getOrigin(), e.getDestination());

TreeLayout<V,E> treeLayout =
     new TreeLayout<V, E>((Forest<V, E>) jungGraph);
//Izaziva izvrsavanje rasporedjivanja
new DefaultVisualizationModel<V, E>(treeLayout);

for (V v : vertices)
  Point2D position = treeLayout.transform(v);
```

Listing 3.2: Pozivanje jednog algoritma za crtanje grafova *JGraphX* biblioteke

```
mxGraph jGraphXGraph = new mxGraph();
mxIGraphModel model = jGraphXGraph.getModel();
model.beginUpdate();
Object parent = jGraphXGraph.getDefaultParent();
try{
  for (V v : vertices){
    Dimension size;
    if (v.getSize() == null)
      size = new Dimension(10, 10);
    else
      size = v.getSize();

    Object jgraphxVertex = jGraphXGraph.insertVertex(parent,
          null, v, 0, 0, size.getWidth(), size.getHeight());
    model.getGeometry(jgraphxVertex).setHeight(size.getHeight());
    model.getGeometry(jgraphxVertex).setWidth(size.getWidth());
    verticesMap.put(v, jgraphxVertex);
  }
  for (E e : graph.getEdges()){
    Object v1 = verticesMap.get(e.getOrigin());
    Object v2 = verticesMap.get(e.getDestination());
    Object jGraphXEdge = jGraphXGraph.insertEdge(parent,
          null, null, v1, v2);
    edgesMap.put(e, jGraphXEdge);
    }
  }
finally{
  jGraphXGraph.getModel().endUpdate();
}

mxCompactTreeLayout treeLayout = new mxCompactTreeLayout(jGraphXGraph);
Object parent = jGraphXGraph.getDefaultParent();
treeLayout.execute(parent);
mxIGraphModel model = jGraphXGraph.getModel();
for (V v : verticesMap.keySet()){
  mxGeometry geometry = model.getGeometry(verticesMap.get(v));
  Point2D position = geometry.getPoint();
}
for (E e : edgesMap.keySet()){
  mxGeometry geometry = model.getGeometry(edgesMap.get(e));
  List<Point2D> points = geometry.getPoints();
}
```

Listing 3.3: Pozivanje jednog algoritma za crtanje grafova *prefuse* okvira

```
//Kreiranje grafa zahteva kreiranje dve tabele.
//Po jednu za cvorove i grane.
Table nodeData = new Table();
Table edgeData = new Table(0,1);
nodeData.addColumn(
    prefuse.data.Graph.DEFAULT_NODE_KEY, int.class);
edgeData.addColumn(
      prefuse.data.Graph.DEFAULT_SOURCE_KEY, int.class);
  edgeData.addColumn(
      prefuse.data.Graph.DEFAULT_TARGET_KEY, int.class);
prefuse.data.Graph prefuseGraph =
    new prefuse.data.Graph(nodeData, edgeData, true);

int key = 0;
for (V v : vertices){
  Node node = prefuseGraph.addNode();
  node.setInt(prefuse.data.Graph.DEFAULT_NODE_KEY, key);
  verticesMap.put(v, node);
  nodeKeyMap.put(node, key);
  key++;
}

for (E e : edges){
  Node n1 = verticesMap.get(e.getOrigin());
  Node n2 = verticesMap.get(e.getDestination());
  prefuse.data.Edge edge = prefuseGraph.addEdge(n1, n2);
  edgesMap.put(e, edge);
}

//Kreiranje vizualizacije i akcije za rasporedjivanje
Visualization vis = new Visualization();
vis.add("graph", prefuseGraph);
RadialTreeLayout radialTreeLayouter =
    new RadialTreeLayout("graph");
ActionList layout = new ActionList();
layout.add(radialTreeLayouter);
vis.putAction("layout", layout);
//Akcija za preuzimanje pozicija cvorova.
//Dodatno napravljena klasa koja nije deo prefuse okvira.
//Parametar konstruktora je trajanje akcije.
PositionAction positionAction = new PositionAction(0);
positionAction.setVisualization(vis);
vis.putAction("layout", layout);
vis.putAction("position", positionAction);
//Takodje neophodno za rasporedjivanje
Display d = new Display(vis);
d.setSize(1000,1000);

//Izvrsavanja rasporedjivanja. Koriste se planeri, sto znaci
//da se mora sacekati da se rasporedjivanje zavrsi da bi
//se preuzele izracunate pozicije
vis.run("layout");
vis.runAfter("layout", "position");

while (true){
  try {
    Thread.sleep(5);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  if (positionAction.isFinished())
```

```
        break;
    }

    Map<Integer, Point2D> positionsMap = positionAction.getPositionsMap();

    //U klasi PositionAction, koja nasledjuje prefuse Action klasu
    TupleSet visGroup = m_vis.getVisualGroup("graph.nodes");
    Iterator<Tuple> iter = visGroup.tuples();
    while (iter.hasNext()){
      Tuple t = iter.next();
      VisualItem item = m_vis.getVisualItem("graph.nodes", t);
      positionsMap.put(item.getRow(),
          new Point2D.Double(item.getX(), item.getY()));
    }
```

Listinzi koda pokazuju zasnovanost tvrdnje kompleksnosti pozivanja algoritama za raspoređivanje analiziranih biblioteka, pri čemu je najmanji problem korišćenje *JUNG*, a najveći *prefuse* okvira. Međutim, treba napomenuti da je podešavanje veličine čvorova kod provospomenute biblioteke veoma kompleksno [119], te se relativna jednostavnost gubi ukoliko je za datu primenu od velike važnosti uzimanje dimenzija čvorova u obzir. Biblioteke se takođe ne bave rekurzivnim i preklopljenim granama, bar ne bez eksplicitnog pozivanja dodatnog algoritma, što čini *JGraphX*.

3.2 Jezici specifični za domen opisa grafova

Pored analize biblioteka otvorenog koda za crtanje i analizu grafova, u okviru istraživanja ispitivani su i jezici specifični za domen koji se na neki način bave grafovima, uz poseban akcenat na onima koji podržavaju i specifikaciju raspoređivanja elemenata grafa. Svakako najpoznatiji od takvih jezika je *DOT* [117] jezik spomenutog *Graphviz* alata.

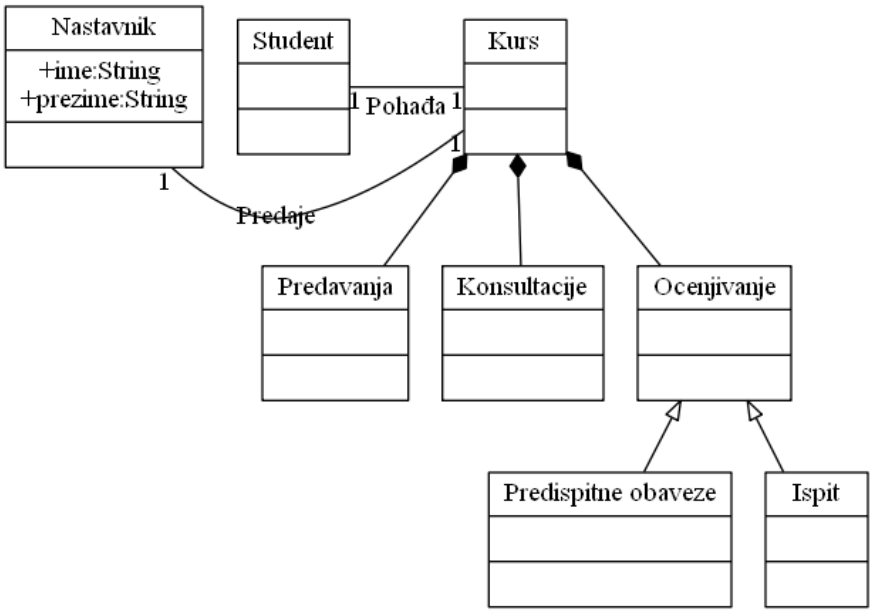
DOT je jezik za definisanje orijentisanih i neorijentisanih grafova. Opis grafa u ovom jeziku sastoji se iz navođenja čvorova koji će mu pripadati, kao i specifikacije na koji način su oni međusobno povezani. Jezik je izuzetno bogat, omogućavajući definisanje zaista impresivnog broja svojstava. Samo neki primeri uključuju boju, oblik i labelu svakog čvora i grane, boju i oblik završetka grane, margine grafa i font koji će biti korišćen. Ovako širok spektar podesivih karakteristika grafa znači da *Graphviz* i *DOT* omogućavaju vizualizaciju najrazličitijih dijagrama, od dijagrama klasa, preko dijagrama aktivnosti do poslovnih procesa. Kada je reč o raspoređivanju elemenata grafa, *DOT* jezik dozvoljava navođenje da li će preklapanja biti izbegavana, kao i poželjnu dužinu i oblik grana i međusobnu bliskost čvorova. Grane mogu biti nacrtane kao splajnovi, prave ili izlomljene linije. Jezik takođe omogućava i zadavanje grupe čvorova čiji članovi će se nalaziti na istom nivou, u slučaju primene algoritma za nivovsko crtanje stabala. Slično, podržano je i definisanje parametara drugih algoritama za raspoređivanje, kao i navođenje podgrafova koji će biti prikazani u posebnim klasterima, te orijentacije crteža. U listingu koda 3.4 dat je opis jednog grafa koji predstavlja objektni model u *DOT* jeziku. Definisana je orijentacija crteža i navedena su tri čvora koja bi trebalo da se nađu na istom nivou.

Listing 3.4: Primer specifikacije grafa u *DOT* jeziku

```
digraph objectmodel{
        node[shape=record];
        rankdir="BT";
```

```
        teacher [label = "{Nastavnik| +ime: String \n +
            prezime: String|\n}"];
        course [label = "{Kurs|\n|\n}"];
        student [label = "{Student|\n|\n}"];
        lesson [label = "{Predavanja|\n|\n}"];
        tutorial [label = "{Konsultacije|\n |\n}"];
        assessment[label = "{Ocenjivanje|\n|\n}"];
        coursework [label = "{Predispitne obaveze|\n|\n}"];
        exam [label = "{Ispit|\n|\n }"];

        {rank=same; teacher course student}

        teacher->course [label="Predaje",arrowhead="none",
        arrowtail="normal",headlabel="1",taillabel="1"];
        student->course [label="Pohadja", arrowhead="none",
        arrowtail="normal",headlabel="1",taillabel="1"];
        lesson->course [arrowhead="diamond",arrowtail="normal"];
        tutorial->course [arrowhead="diamond",arrowtail="normal"];
        assessment->course [arrowhead="diamond",arrowtail="normal"];
        coursework->assessment[arrowhead="onormal"];
        exam->assessment[arrowhead="onormal"];

}
```

Po završetku specifikacije grafa, kreiranje njegovog crteža vrši se upotrebom jednog od nekoliko alata za raspoređivanje koje podržava *Graphviz*. Svaki alat je implementacija jednog ili više algoritama za crtanje grafova. U tekućem trenutku, na raspolaganju su: hijerarhijski, slojeviti, metoda opruge, radijalni i kružni. Krajnji proizvod alata je slika u izabranom formatu (PDF, PNG i sl.). Ukoliko se upotrebi hijerarhijski alat za crtanje grafa iz gornjeg listinga koda, dobija se rezultat prikazan na slici 3.2.



Slika 3.2: Objektni model definisan i raspoređen upotrebom *Graphviz* alata i *DOT* jezika

Kada se sve uzme u obzir, jasno je da je *DOT* izuzetno bogat jezik čija je upotrebna vrednost u kombinaciji sa *Graphviz* alatom izuzetno velika. Naime, ukoliko je potrebno u okviru aplikacije koja se razvija implementirati needitabilni dijagram koji bi vizuelno prikazao određene informacije, pomenuta kombinacija je mnogima prvi izbor. Međutim, kao što je već spomenuto, vrednost alata je manja ako se posmatra iz konteksta ugradnje u postojeće editore. Dodatno, *DOT* jezik ne stavlja akcenat na obezbeđivanje načina za jednostavnu i deskriptivnu specifikaciju željenog izgleda crteža grafa. Algoritam za raspoređivanje mora biti ručno izabran, dok jezik dozvoljava samo izmenu nekih njegovih parametara, namenjenih za fino podešavanje. Dakle, može se zaključiti da samo upućeniji korisnici mogu pomenuto iskoristiti na pravi način.

DOT jezik nije, međutim, jedini koji se dotiče problema vizualizacije grafova. Naime, u [120] predložen je jezik specifičan za domen predstavljanja softverskih zavisnosti u formi grafova, nazvan *Graph*. U tom konteksu, čvorovi grafa odgovaraju softverskim elementima, a grane zavinostima između dva entiteta koja spajaju. *Graph* je interni JSD izgrađen u objektno-orijentisanom jeziku i razvojnom okruženju zvanom *Pharo* [121].

Graph dozvoljava svojim korisnicima definisanje čvorova i grana grafa, kao i njihovih obeležja poput oblika i veličine. Zadavanje načina raspoređivanja elemenata putem jezika jeste omogućeno, ali samo na nivou direktnog izbora i podešavanja parametara jednog od nekoliko dostupnih algoritama (silom usmereni, kružni, za crtanje stabala), kao u primerima u listingu 3.5.

Listing 3.5: Primer zadavanja raspoređivanja u *Graph* jeziku

```
layout force charge: -80.
layout circle radius: 250.
```

Treba napomenuti da se uzima u obzir činjenica da je nekada bolji pristup primena različitih algoritama nad različitim delovima grafa, a ne jednog nad njim u celosti. Prema tome, ovaj JSD omogućava definisanje particija, odnosno podgrafova i za svakog od njih posebnog algoritma. Međutim, zaključuje se da se ni ovaj jezik ne bavi olakšavanjem načina izbora odgovarajućeg algoritma ili algoritama za crtanje grafa korisnicima koji nemaju puno iskustva u tom domenu.

Dva opisana jezika su najbliža sprovedenom istraživanju, baveći se, bar do neke mere, crtanjem grafova, a ne samo njihovom definicijom. Veći je broj projekata koji su usmereni isključivo na probleme izgradnje i analize grafova. Jedan takav primer je opisan u [122]. Reč je o jeziku čiji je cilj pojednostavljivanje unosa podataka o grafovima u okviru *Java* programa koji se bave teorijom grafova. On dozvoljava korisnicima zadavanje čvorova grafa i grana između njih sa težinama, što se dalje može koristiti u algoritmima iz teorije grafova. Još jedan primer jezika specifičnog za domen kreiranog u svrhu podrške analize grafova predstavljen je u [123] i nazvan *Green-Marl*. Naime, omogućeno je generisanje efikasnih implementacija algoritama iz teorije grafova u nekom jeziku opšte namene kao što je *C++* na osnovu opisa većeg nivoa apstrakcije.

Kada se sve sumira, uočava se da postojeći jezici specifični za domen koji se dotiču grafova bave ili njihovom specifikacijom uz mogućnost konfigurisanja većeg ili manjeg broja obeležja. Neki za cilj imaju generisanje dijagrama koji se mogu predstaviti grafom, neki olakšavanje pisanja algoritama teorije grafova. Nijednom od njih nije prioritet osmišljavanje načina za jednostavnu specifikaciju načina na koji će elementi nekog grafa biti raspoređeni. Direktan izbor i konfiguracija algoritma od strane korisnika je jedina opcija koju neki od analiziranih jezika pružaju.

Poglavlje 4

Implementacija algoritama za analizu grafova

U ovom poglavlju biće detaljnije prikazani razni algoritmi iz teorije grafova implementirani u projektnom rešenju koji su bitni za oblast njihovog crtanja i implementaciju automatskog izbora pogodnog algoritma. Za svaki od izdvojenih algoritama biće predstavljena njegova osnovna ideja.

Samo na osnovu definicija različitih osobina grafa, bez većih problema mogli bi se osmisliti jednostavni algoritmi za proveru prisustva tih osobina. Međutim, veoma bitan faktor jeste algoritamska kompleksnost takvog rešenja, odnosno njihova efikasnost. Neka je, na primer, zadatak proveriti bipovezanost grafa, koja garantuje da se može ukloniti bilo koji čvor, a da graf ostane povezan. Trivijalno rešenje obuhvatalo bi uklanjanje jednog po jednog čvora, te proveru da li je rezultujući graf povezan. Povezanost opet garantuje da postoji putanja između svaka dva čvora, te bi se u krajnje pojednostavljenoj implementaciji mogla proveriti izdvajanjem jednog po jednog para čvorova i traženjem načina prelaska iz jednog u drugi. U slučaju iole većeg grafa, vreme izvršavanja bi bilo izuzetno veliko, što nije prihvatljivo. Iz tog razloga, od sredine prošlog veka do danas osmišljavaju se sve efikasniji algoritmi koji se bave pomenutom tematikom. Naime, teži se postizanju linearnog vremena izvršavanja, te je izbor algoritama za implementaciju bio usmeren na upravo one koji postižu pomenuti ideal.

Iako se u tekstu koji sledi neće ulaziti u sitnije implementacione detalje algoritama, treba napomenuti da su njihovi autori često veoma pažljivo birali strukture podataka koje bi optimizovale rešenje. Kako su mnogi od tih algoritama nastali pre pojave modernih programskih jezika, uputstva autora nisu praćena u potpunosti, već su iskorišćene prednosti odabranog programskog jezika novije generacije, gde je to bilo odgovarajuće. Na primer, korišćene su strukture podataka poput mapa, koje nisu bile na raspolaganju u starijim jezicima.

Algoritmi za analizu grafova nalaze se u osnovi postupaka njihovog crtanja, kao i određivanja da li dati graf uopšte može biti vizualizovan na traženi način. Pre svega, uočava se da su mnogi algoritmi za raspoređivanje elemenata grafova fokusirani isključivo na one koji poseduju osobine planarnosti, povezanosti, bipovezanosti i/ili tripovezanosti. Dalje, mnogi kompleksniji algoritmi za crtanje kao jedan od koraka uključuju postupke kao što je dekompozicija na dvostruko ili trostruko povezane komponente, te nalaženje planarnog utapanja, odnosno ciklusa lica. Takođe, kako je simetrično crtanje blisko povezano sa automorfizmima grafa, i njihovo nalaženje je jedna od potrebnih akcija. U predstojećim sekcijama će stoga biti dat pregled algoritama

upravo navedenih tipova, uz akcenat na one koji su implementirani u projektnom rešenju.

U samom korenu mnogih složenijih postupaka nalaze se algoritmi za obilazak grafova. Njima se direktno rešavaju problemi kao što je isptivanje jednostruke povezanosti i nalaženja putanje između dva čvora, te su sastavni deo provere viših stepena povezanosti, ali i pojedinih klasa algoritama za proveru planarnosti. Iz tog razloga, predstojeći pregled će biti započet upravo osvrtom na načine i značaj obilazaka grafova, kao i informacija koje se na osnovu njih dobijaju. Na primer, korišćene su strukture podataka poput mapa, koje nisu bile na raspolaganju u starijim jezicima.

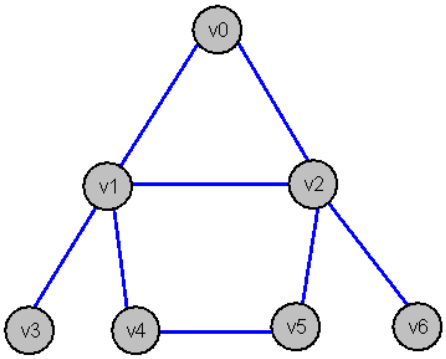
4.1 Algoritmi za obilazak grafova

Osnova mnogih drugih analiza u teoriji grafova je pretraga najpre u dubinu - DFS (*Depth-First Search*). Ovom pretragom se može ustanoviti da li su neka dva čvora povezana ili obići ceo graf radi izdvajanja informacija potrebnih za dalju analizu. Na DFS kao moćan mehanizam za rešavanje mnogih problema u teoriji grafova skrenuli su pažnju Džon Hopkroft (John E. Hopcroft) i Robert Taržan (Robert E. Tarjan) u [124], ali ga je prvi izučavao francuski matematičar Čarls Pjer Tremo (Charles Pierre Trémaux) [125].

Kod pretrage najpre u dubinu počinje se od nekog proizvoljnog čvora koji se proglasi korenom, i ide se sve dublje prolaskom kroz grane, dokle god je to moguće. Ako tekući čvor nije povezan ni sa jednim čvorom koji još nije posećen tokom pretrage, pretraga se vraća jedan ili više koraka unazad, do grane koja vodi ka novom čvoru. Postupak se obustavlja kada se posete svi čvorovi grafa. Grane kroz koje se prošlo tokom pretrage formiraju takozvano Tremoovo stablo, koje je jedan od bitnijih koncepata u teoriji grafova. Radi se o razapinjućem stablu polaznog grafa kod kog su povezani čvorovi u relaciji pretka i potomka.

Još jedan koncept koji se često koristi u raznim algoritmima jeste DFS uređenje. Uređeni niz čvorova naziva se DFS uređenjem ukoliko je mogući rezultat DFS pretrage. Odnosno, ukoliko se prilikom DFS pretrage čvorovi mogu posetiti u datom redosledu. Takođe, često se koristi i pojam DFS indeksa, koji predstavlja ništa drugo do indeks čvora u DFS uređenju.

Na primer, neka je dat graf prikazan na slici 4.1.



Slika 4.1: Graf za koga se konstruiše DFS stablo

Polazni čvor se bira proizvoljno, te neka to bude v_0, i neka se leve grane u grafu

biraju pre desnih. DFS pretraga bi funkcionisala na sledeći način:

- Iz v_0 prešlo bi se u v_1, iz njega u v_3.
- Iz v_3 se ne može ići dublje, pa bi se pretraga vratila u v_1 i prešla u v_4.
- Iz v_4 otišlo bi se u v_5, pa dalje iz njega u v_2.
- Iz v_2 izlaze tri grane, ali samo jedna vodi ka još neposećenom čvoru. Stoga bi se dalje prešlo u v_6.
- Posetom čvora v_6 konstatuje se da su se obišli svi čvorovi. Može se primetiti da se tom prilikom nisu koristile sve grane.

Dakle, konačan rezultat tj. poredak čvorova bio bi v_0, v_1, v_3, v_4, v_5, v_2, v_6. Čvorovi grafa i grane kojima se prošlo tokom DFS pretrage čine DFS Treumoovo stablo. Za ovakvo stablo se može primetiti da preci imaju manji DFS indeks od svojih potomaka. Koren DFS stabla je čvor u kojem je pretraga započeta i ima DFS indeks 1.

Svaka grana (u, v) nekog grafa može pripadati jednoj od nekoliko klasa, u koje se smešta na osnovu ishoda DFS pretrage [126]:

- Ako se tokom pretrage prošlo granom, reč je o grani stabla
- Inače:
  - Ako je v predak čvora u, (u, v) je povratna grana.
  - Akoje v potomak u, (u, v) je direktna grana.
  - Ako v nije ni predak ni potomak čvora u, (u, v) je poprečna grana.

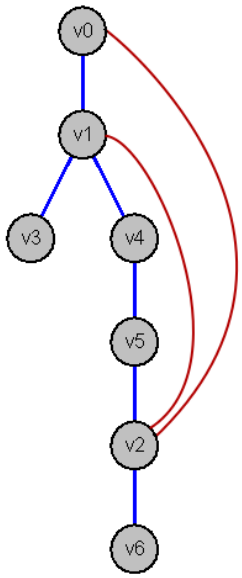
Ako je graf neusmeren, povratne i direktne grane su istovetni pojmovi, dok se poprečne grane ne mogu javiti. Na slici 4.2 prikazano je DFS stablo, kao podgraf grafa sa slike 4.1 sa plavim granama, dok se crvene grane klasifikuju kao povratne za prethodno opisanu pretragu. Pošto čvorovi v_0 i v_2, i v_0 i v_1 nisu u odnosu roditelj-direktni potomak u DFS stablu, grane između njih su povratne.

Kao što je napomenuto, ako je graf usmeren, onda se razlikuje više tipova grana koje ne pripadaju DFS stablu. Na slici 4.3 prikazan je upravo jedan orijentisani graf, čije DFS stablo sadrži grane plave boje, dok su ostale grane razvrstane prema tipu - povratne su obojene crvenom bojom, direktne zelenom, a poprečne narandžastom.

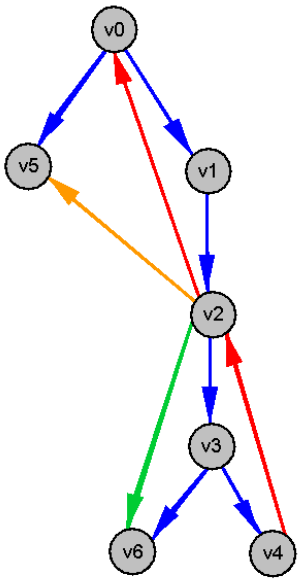
Pored pretrage najpre u dubinu, poznat je i algoritam najpre u širinu - BFS (*Breadth-First Search*). Slično kao kod DFS pretrage, počinje se od nekog proizvoljno izabranog čvora koji se proglasi korenom, ali se dalje obilaze svi njemu susedni čvorovi, da bi se tek posle prešlo na sledeći nivo. Algoritam su nezavisno jedan od drugog osmislili Mur (Edward F. Moore) i Li (C. Y. Lee), '50-ih, odnosno, početkom '60-ih godina prošlog veka.

Još jedan algoritam povezan sa obilaskom grafa široke primene jeste Dijkstrina najkraća putanja. Algoritam je nazvan po svom pronalazaču, Edsgeru Dijkstri (Edsger W. Dijkstra), koji ga je osmislio 1956. godine, a objavio tri godine kasnije u [127]. Glavni problem koji se rešava obuhvata pronalaženje najkraće putanje od izabranog čvora težinskog povezanog grafa, do proizvoljnog odredišnog. Na primer, ovaj algoritam se može primeniti za nalaženje najkraćeg puta između dve lokacije na geografskoj mapi. U tom slučaju, čitava mapa se predstavlja grafom, lokacije na njoj čvorovima, a putevi među njima granama, kojima se dodeljuje težina koja odgovara njihovoj dužini.

Algoritam uključuje nekoliko koraka, u okviru kojih postavlja određene početne vrednosti cene, da bi ih kasnije poboljšavao. Postupak obuhvata sledeće:



Slika 4.2: DFS stablo i povratne grane prema opisanoj pretrazi



Slika 4.3: DFS stablo i povratne, direktne i poprečne grane

1. Bira se početni čvor i njemu dodeljuje cena 0, a svim ostalima beskonačna.

2. Neposećeni čvor koji ima najmanju cenu bira se za tekući. Za svaki od njemu susednih čvorova proverava se da li je cena tekućeg čvora sabrana sa težinom grane koja iz tekućeg vodi ka njemu manja od njegove trenutne cene. Ako to jeste slučaj, cena se ažurira. Tekući čvor se proglašava posećenim.

3. Ako je odredišni čvor proglašen posećenim, ili svi neposećeni čvorovi imaju beskonačnu cenu (nema putanje do njih), algoritam se završava. Inače, ponavlja se drugi korak.

Ukoliko se graf nije težinski, a problem koji se rešava traži nalaženje putanje koja sadrži najmanji broj grana između dva čvora, može se primeniti Dijkstrin algoritam podrazumevajući da su sve težine jednake 1.

4.2 Povezanost grafova

Mnogi algoritmi za crtanje grafova zahtevaju da dati graf bude povezan. Često, međutim, to samo po sebi nije dovoljno jer je primena nekih od njih moguća samo ukoliko je graf 2-, ili čak 3-povezan. Dodatno, neki algoritmi drugačije obrađuju grafove većeg stepena povezanosti od, recimo, samo jednostruko povezanih. Stoga će u nastavku biti opisano kako se može proveriti prisustvo pomenutih osobina. Takođe, složeniji algoritmi za analizu i crtanje grafova neretko uključuju rastavljanje grafova na bi- ili tri- povezane komponente, te će i ovi problemi biti razrađeni.

4.2.1 Ispitivanje jednostruke, dvostruke i trostruke povezanosti grafova

Jasno je da je k-povezani graf i $k-1$ povezan. Dakle, graf koji je dvostruko povezan ispunjava i uslov jednostruke povezanosti, a trostruko povezani i jednostruke i dvostruke. Provera da li je graf samo povezan svakako je najjednostavnija od tri pomenute. Standardan način za njenu implementaciju baziran je na pretrazi najpre u dubinu. Naime, ako se prilikom obilaska grafa upotrebom DFS algoritma prođe kroz sve čvorove, graf je povezan, dok se u suprotnom može tvrditi da to nije slučaj. Nepovezani graf sastoji se iz dve ili više komponenti povezanosti.

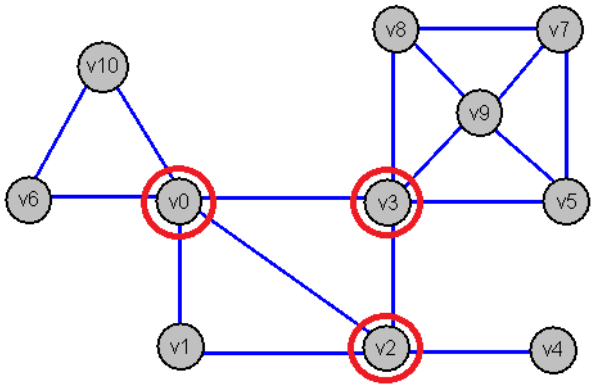
Implementacija provera bi i tripovezanosti grafa, kao i njegovo rastavljanje na 2 i 3-povezane komponente se na prvi pogled takođe ne čini kao zahtevan problem. Međutim, u ovim slučajevima izazov ne predstavlja osmišljavanje algoritma kojim bi se navedeno nekako postiglo, nego kako bi to moglo biti realizovano na što je moguće efikasniji način. Odnosno, za što je moguće manje vremena. U nastavku će detaljnije biti opisani takvi postupci.

4.2.2 Nalaženje artikulacionih čvorova grafova i dvostruko povezanih komponenti

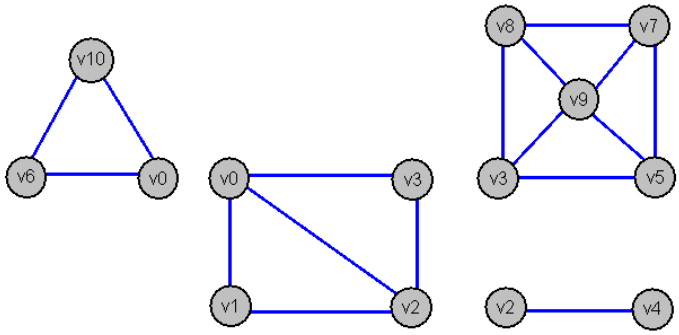
Čvor v neorijentisanog povezanog grafa G jeste presečni ili artikulacioni čvor ako i samo ako postoje dva druga čvora x i y takva da svaka putanja između njih prolazi kroz v. Samo u ovom slučaju uklanjanje čvora v prekida svaki put od x do y, što znači da narušava povezanost grafa. Imajući ovo u vidu, može se zaključiti da se nalaženje artikulacionih čvorova i bipovezanih komponenata grafa može realizovati pomoću DFS sa $O(|V|+|E|)$ operacija [128].

Posmatrajmo graf sa slike 4.4. Njegovi artikulacioni čvorovi v_0, v_3 i v_2 zaokruženi su crvenom bojom na slici. Može se primetiti da, recimo, nije nikako moguće iz čvora v_1 doći do čvora v_7 da se ne prođe kroz čvor v_3, niti do čvora v_{10}, a da se na toj putanji ne nađe čvor v_0. Ako bi se grafu dodala grana koja spaja čvorove v_8 i v_{10}, tada ovo ne bi bio slučaj i čvorovi v_0 i v_3 ne bi bili artikulacioni. Blokovi na koje se graf može rastaviti prikazani su na slici 4.5.

Algoritam koji koristi DFS i nalazi artikulacione tačke u linearnom vremenu osmislili su Hopkroft i Taržan i objavili 1973. godine [129]. Pre predstavljanja osnovne ideje algoritma, potrebno je uvesti funkciju $lowpt(v)$. Neka je $num(x)$ DFS indeks čvora x. Vrednost $lowpt(v)$ se tada može definisati kao najmanja vrednost $num(x)$, gde je x čvor



Slika 4.4: Povezani graf sa obelženim artikulacionim čvorovima koji se rastavlja na bipovezane komponente



Slika 4.5: Bipovezane komponente grafa sa prethodne slike

grafa do kojeg se može doći iz čvora v krećući se kroz nula ili više grana DFS stabla te najviše jednu povratnu granu. Hopkroft i Taržan potom uvode i dokazuju sledeće tvrđenje, koje je i osnova algoritma koji se predstavlja.

Neka je $G = (V, E)$ povezani graf sa DFS stablom T i skupom povratnih grana B. Tada je $a \in V$ artikulacioni čvor ako i samo ako postoje čvorovi $v, w \in V$ takvi da je $(a, v) \in T$, gde w nije potomak v u T i važi $lowpt(v) \geq num(a)$. Vrednosti *lowpt* računaju se tokom same pretrage, što znači da ne moraju biti određeni pre izvršavanja glavnog dela algoritma.

Algoritam obezbeđuje i korektno formiranje bipovezanih komponenti po nailasku na artikulacionu tačku. Osnovna ideja zasniva se na čuvanju grana koje se prelaze tokom pretrage najpre u dubinu u strukturi tipa steka. Naime, kada se naiđe na artikulacioni čvor, grane se skidaju sa vrha steka, dok se ne dođe do grane između čvorova v i a iz navedene teoreme.

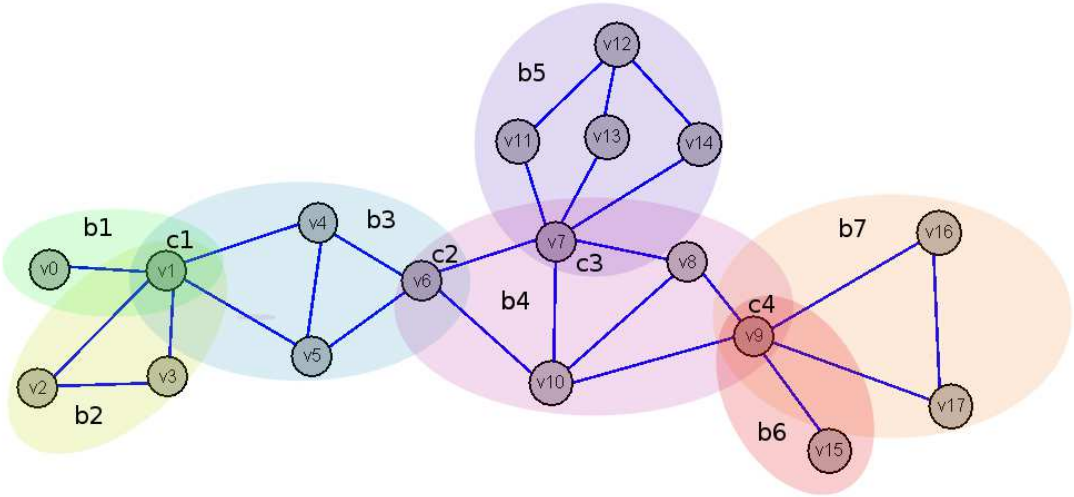
4.2.3 Konstrukcija BC-stabala

Struktura blokova, odnosno, bipovezanih komponenti grafa, kao i njegovih artikulacionih (presečnih) čvorova može biti opisana blok-presečna tačka stablom, koje se kraće obeležava kao BC (*block-cut vertex*) stablo [130]. Konstrukcija BC-stabla predstavlja jedan od koraka mnogih algoritama za analizu grafova.

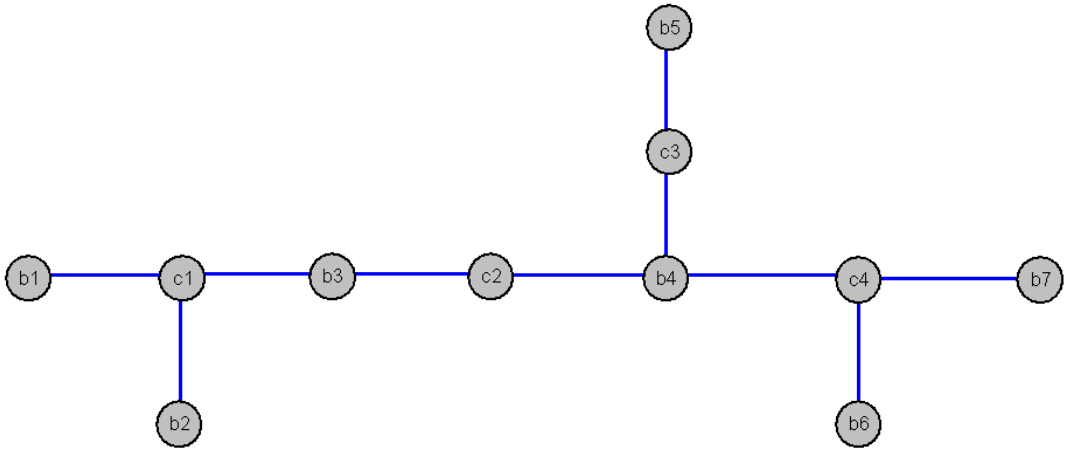
Neka je B skup blokova, a C artikulacionih čvorova grafa G koji je maksimalno 1-povezan. Graf $H = (V_1, E_1)$ je BC-stablo grafa G ako važi:

- $V_1 = B \cup C$. Odnosno, graf H poseduje jedan čvor za svaki blok i svaki artikulacioni čvor grafa G.
- Grane grafa H spajaju čvorove iz skupa C sa čvorovima iz skupa B. Čvor $c_i \in C$ je susedan čvoru b_j ako i samo ako blok b_j u G sadrži artikulacioni čvor c_i grafa G.

Ne slici 4.6 prikazan je graf sa označenim blokovima za koje će biti konstruisano BC-stablo. Artikulacioni čvorovi grafa su v_1, v_6, v_7 i v_9. Kao što se može primetiti, njih sadrži su više blokova. Odgovarajuće BC-stablo prikazano je na slici 4.7.



Slika 4.6: Graf sa označenim blokovima



Slika 4.7: BC-stablo prethodnog grafa

4.2.4 Podela grafova na trostruko povezane komponente

Pored rastavljanja povezanih grafova na bipovezane komponente, jedan od fundamentalnih problema u teoriji grafova, posebno bitan za oblast njihovog crtanja jeste dekompozicija bipovezanih grafova na tripovezane komponente. Efikasno rešenje

ovog problema, poput podele na bipovezane komponente, dali su Hopkroft i Taržan [131]. Osnovna ideja algoritma podseća na slučaj bipovezanih komponenti, ali je postupak znatno kompleksniji. Naime, ovaj algoritam smatra se jednim od najizazovnijih za razumevanje i implementaciju, te se mogu naći i radovi poput [132], u kojima se predstavljaju pojašnjenja određenih koncepata koji imaju ključnu ulogu u algoritmu.

Algoritam Hopkrofta i Taržana za nalaženje tripovezanih komponenti dodatno je poboljšan 2000. godine u [133], gde su ujedno i popravljene greške koje se mogu javiti u određenim, specifičnim situacijama. Uz pomenute ispravke, algoritam se i danas smatra optimalnim rešenjem datog problema. Pre nego što se opišu osnovni koraci algoritma, potrebno je uvesti nekoliko pojmova.

Neka je $\{a, b\} \subset V$ grafa G. Neka su grane G podeljene u klase ekvivalencije $E_1, E_2, ...E_n$ tako da su dve grane koje leže na istoj putanji koja ne sadrži nijedan od čvorova a i b u istoj klasi. Klase E_i nazivaju se razdelnim klasama grafa G prema paru a, b. Ako postoje bar dve klase ekvivalencije, a, b je razdvajajući par G, osim ako:

- postoje tačno dve razdelne klase i jedna od njih se sastoji od samo jedne grane ili,
- postoje tačno tri klase i svaka od njih se sastoji od samo jedne grane.

Bipovezani graf koji nema nijedan razdvajajući par jeste tripovezan [131].

Neka je $\{a, b\}$ razdelni par, a $E_1, E_2..., E_n$ razdelne klase. Neka je dalje $E' = \bigcup_{i=1}^{k} E_i$ i $E'' = \bigcup_{i=k+1}^{n} E_i$, pri čemu važi $|E'| \geq 2$ i $|E''| \geq 2$. Grafovi $G_1 = (V(E'), E' \cup (a, b))$ i $G_2 = (V(E''), E'' \cup (a, b))$ nazivaju se razdeljeni grafovi prema (a, b). Zamena multigrafa sa dva razdeljena grafa naziva se rastavljanje, dok se dodata grana (a, b) naziva virtuelnom.

Iz definicije se može primetiti da jedan graf na više načina može biti podeljen prema jednom razdelnom paru, ali u svakom slučaju važi da, ako je sam graf G bipovezan, onda ovu osobinu poseduje i svaki razdeljeni graf. Ukoliko se multigraf G podeli na razdeljene grafove, te se i oni sami dele na sopstvene razdeljene grafove dokle god su podele moguće, dolazi se do 3-povezanih grafova koji se nazivaju razdelnim komponentama grafa G. Razdelne komponente multigrafa mogu biti jednog od sledećih tipova:

1. skup tri multigrane između ista dva čvora, oblika $\{a, b\}, \{(a, b), (a, b), (a, b)\}$,
2. trougao - graf K_3,
3. tripovezani graf.

Ovakve komponente nisu nužno jedinstvene. Da bi se dobile jedinstvene tripovezane komponente, razdelne komponente se spajaju ukoliko je to moguće. Ako je (a, b) virtuelna grana dobijena prilikom i-te podele, to se skraćeno prikazuje kao (a, b, i). Neka su dalje grafovi $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ dve razdelne komponente koje sadrže virtuelnu granu (a, b, i). Operacija spajanja te dve komponente proizvodi graf $G = (V_1 \cup V_2, E_1 - \{(a, b, i)\} \cup E_2 - \{(a, b, i)\})$. Dakle, dve razdelne komponente se spajaju po zajedničkoj virtuelnoj grani, pri čemu se dobija graf koji sadrži uniju njihovu čvorova i grana, ali se sama virtuelna grana izostavlja.

Neka je G multigraf čije su razdelne komponente skupovi trostrukih veza $\mathcal{B}_3$, trouglova $\mathcal{T}$ i tripovezanih grafova $\mathcal{G}$. Ako se trostruke veze maksimalno spoje, dajući skup $\mathcal{B}$, trouglovi spoje u konture (poligrane), dajući skup $\mathcal{P}$, ova dva skupa uz skup $\mathcal{G}$ predstavljaju skup trostruko povezanih komponenti grafa. Ove komponente su jedinstvene.

Algoritam se ugrubo može podeliti na tri koraka:

1. Uklanjanje multigrana iz polaznog grafa G i formiranje skupa trostrukih grana preostalog grafa G'.

2. Podela grafa G' na bipovezane komponente, te njih na razdelne komponente.

3. Spajanje trostrukih grana i trouglova radi formiranja konačnog skupa tripovezanih komponenti.

Najizazovniji od njih svakako jeste nalaženje razdelnih komponenti. Da bi se taj korak mogao opisati, potrebno je definisati i neke dodatne termine koji su od velikog značaja za dati algoritam. Pre svega, neka za razapinjuće stablo T važi sledeće:

- $v \to w$ označava granu između čvorova v i w u T. Dakle, v je roditelj, a w dete.

- $v \underset{\to}{*} w$ označava da postoji putanja u T od v do w. Dakle, v je predak, a w je potomak.

- $v \hookrightarrow w$ označava povratne grane.

Neka je dalje P usemereni multigraf koji se sastoji iz 2 nepovezana skupa grana: $v \to w$ i $v \hookrightarrow w$. Neka P zadovoljava sledeće:

- Podgraf T koji sadrži grane $v \to w$ je razapinjuće stablo grafa P.

- Svaka grana koja nije u T povezuje čvor sa nekim od svojih predaka u T. Odnosno, ako je $v \hookrightarrow w$ onda $w \underset{\to}{*} v$.

Ovakav graf P se naziva palmino stablo, a $v \hookrightarrow w$ su njegove povratne grane.

Konačno, treba definisati funkcije $lowpt1(v)$ i $lowpt2(v)$. $lopwt1(v)$ je funkcija$lowpt(v)$ definisana prilikom opisa algoritam za pronalaženje bipovezanih komponenti. Dakle, vrednost $lowpt1(v)$ ima vrednost najmanjeg indeksa čvora do kojeg se može doći iz čvora v krećući se kroz nula ili više grana razapinjućeg stabla, te najviše jednu povratnu granu. Funkcija $lowpt2(v)$ je veoma slična, gde je jedina razlika sadržana u činjenici da se dobija drugi najmanji indeks.

Imajući prethodne definicije u vidu, može se predstaviti teorema, koja je osnova algoritma. Neka su a i b čvorovi grafa G takvi da je a predak b u razapinjućem stablu tj. $a < b$. $\{a, b\}$ je razdvajajući par grafa G ako i samo ako je zadovoljen jedan od sledeća tri uslova:

1. Postoje različiti čvorovi $r \neq a, b$ i $s \neq a, b$ takvi da je $b \to r$, $lowpt1(r) = a$, $lowpt2(r) \geq b$ i s nije potomak od r. $\{a, b\}$ je tada razdvajajući par tipa 1.

2. Postoji čvor $r \neq b$ takav da je $a \to r \underset{\to}{*} b$, b je prvi potomak čvora r (a, r i b leže na istoj generisanoj putanji) i $a \neq 1$, a za svako $x \hookrightarrow y$ važi:

   (a) ako je $r \leq x < b$, onda je $a \leq y$,

   (b) ako je $a < y < b$ i $b \to w \hookrightarrow x$ onda je $lowpt1(w) \geq a$,

   $\{a, b\}$ je tada razdvajajući par tipa 2.

3. (a, b) je multigrana u G takva da postoje bar četiri grane između ta dva čvora.

Algoritam Hopkrofta i Taržana uključuje izvršavanje pretrage najpre u dubinu dva puta. Prvom se pravi palmino stablo i računaju početne vrednosti bitnih funkcija, poput $lowpt1(v)$ i $lowpt2(v)$. Na osnovu tih vrednosti konstruiše se posebna matrica susedstva A. Sledeća pretraga najpre u dubinu obuhvata prolazak kroz palmino stablo P, koristeći pripremljenu matricu A. U ovoj fazi proverava se i da li su parovi čvorova razdvajajući, te ako se uspostavi da jesu, formira se razdelna komponenta. Da bi to bilo moguće, definišu se dve dodatne strukture:

- *ESTACK* - stek grana, na koji se svaka grana (v, w) stavlja neposredno pre nego što se proveri da li je v, w razdvajajući par. Kada se nađe razdvajajući par, u novu komponentu se ne stavljaju sve grane sa steka, nego uzastopne grane koje povezuju čvorove unutar određenog raspona DFS brojeva. Za razdvajajući par $\{a, b\}$, taj broj se određuje na osnovu DFS indeksa čvorova a i b, kao i broja potomaka čvorova susednih čvoru b.

- *TSTACK* - stek trojki oblika $(h, num(a), num(b))$. $\{a, b\}$ je potencijalni razdvajajući par tipa 2, a h čvor sa najvišim DFS indeksom u komponenti koja bi se odvojila ako bi se ispostavilo da par jeste razdvajajući.

Po nalaženju razdelnih komponenti, vrši se spajanje koje je prethodno opisano, čime se po okončanju algoritma dobijaju tripovezane komponente i lista razdvajajućih parova.

4.2.5 Kontrukcija SPQR-stabala

U bliskoj vezi sa tripovezanim komponentama nekog grafa je i takozvano SPQR-stablo. Naime, ono pruža pregled visokog nivoa jedinstvene dekompozicije grafa na njegove tripovezane komponente. SPQR-stabla prvi put su koristili u konačnom obliku Di Batista (Giuseppe Di Battista) i Tamasija [134] sa ciljem reprezentacije planarnog utapanja planarnog bipovezanog grafa. Od tada, SPQR-stabla postaju jedna od značajnijih struktura u oblasti teorije grafova. Njihova primena može se sresti i unutar algoritama za planarno testiranje i nalaženje planarnog utapanja, nalaženje minimalnog razapinjućeg stabla, omogućavanje primene algoritama dizajniranih za tripovezane grafove nad onima koji su samo bipovezani itd. [135]–[137]. SPQR-stabla mogu biti konstruisana u linearnom vremenu, prema algoritmu čiji su tvorci Karsten Gutvenger (Carsten Gutwenger) i Petra Mucel (Petra Mutzel) [133]. Njihovo istraživanje inspirisano je algoritmom Hopkrofta i Taržana, koji su se prvi bavili implementacijom rastavljanja grafa na tripovezane komponente. Osim što predstavljaju novi način kojim se pomenuto postiže, u ovom radu autori, kao što je spomenuto ranije, skreću pažnju na poboljšanja algoritma Hopkrofta i Taržana kojima se postiže njegova korektnost.

SPQR stabla definisana su inicijalno u kontekstu planarnih grafova, dok se ovde citira opštija definicija koja se može odnositi i na neplanarne grafove data u [138].

Neka je G bipovezani graf. Razdelni par grafa G je ili razdvajajući par, ili par susednih čvorova. Razdelna komponenta razdelnog para $\{u, v\}$ je ili grana (u, v) ili maksimalni podgraf C od G takav da $\{u, v\}$ nije njegov razdelni par. Maksimalni razdelni par $\{u, v\}$ od G u odnosu na drugi razdelni par $\{s, t\}$ je takav da za bilo koji drugi razdelni par $\{u', v'\}$ važi da su čvorovi u, v, s i t u istoj razdelnoj komponenti.

Neka se $e = (s, t)$ grana grafa G proglasi referencijalnom. SPQR-stablo T grafa G prema e jeste korensko uređeno stablo čiji čvorovi mogu biti jednog od četiri tipa: S, P, Q i R. Uređeno stablo karakteriše se činjenicom da je uređenje njegove dece specifikovano za svaki čvor koji mu pripada. Ovo je ekvivalentno utapanju stabla u ravan, gde je koren na vrhu, a deca svakog čvora se nalaze ispod njega. Ako se u takvom utapanju

odredi smer dece, recimo s leva-na -desno, dobija se njihovo uređenje. Svaki čvor μ u T ima pridružen bipovezani multigraf koji se naziva kosturom od μ. Konačno, SPQR stablo se može rekurzivno definisati na sledeći način:

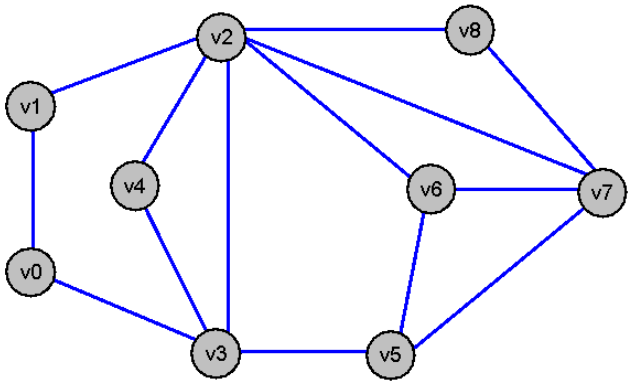
1. Trivijalni slučaj - ako se G sastoji iz tačno dve paralelne grane između s i t, T se sastoji iz tačno jednog Q-čvora čiji kostur je sam graf G.

2. Paralelni slučaj - ako razdelni par $\{s,t\}$ ima bar tri razdelne komponente $G_1, G_2, ..., G_k$, koren T stabla je P-čvor μ, čiji se kostur sastoji iz k paralelnih grana $e = e_1, ..., e_k$ između s i t.

3. Serijski slučaj - ako prva dva uslova nisu ispunjena, a par $\{s,t\}$ ima tačno dve razdelne komponente, jedna od njih je e, a druga se označava sa G'. Neka su $c_1, c_2, ..., c_{k-1}$, $k \geq 2$ artikulacione tačke G' koje dele G na blokove $G_1, G_2, ..., G_k$ u ovom poretku od s do t. Koren stabla T tada je S-čvor μ čiji kostur je ciklus $e_0, e_1, ..., e_k$, gde je $e_0 = e$, $c_0 = s$, $c_k = t$ i $e_1 = (c_{i-1}, c_i)$, za $i = 1, 2, ...k$.

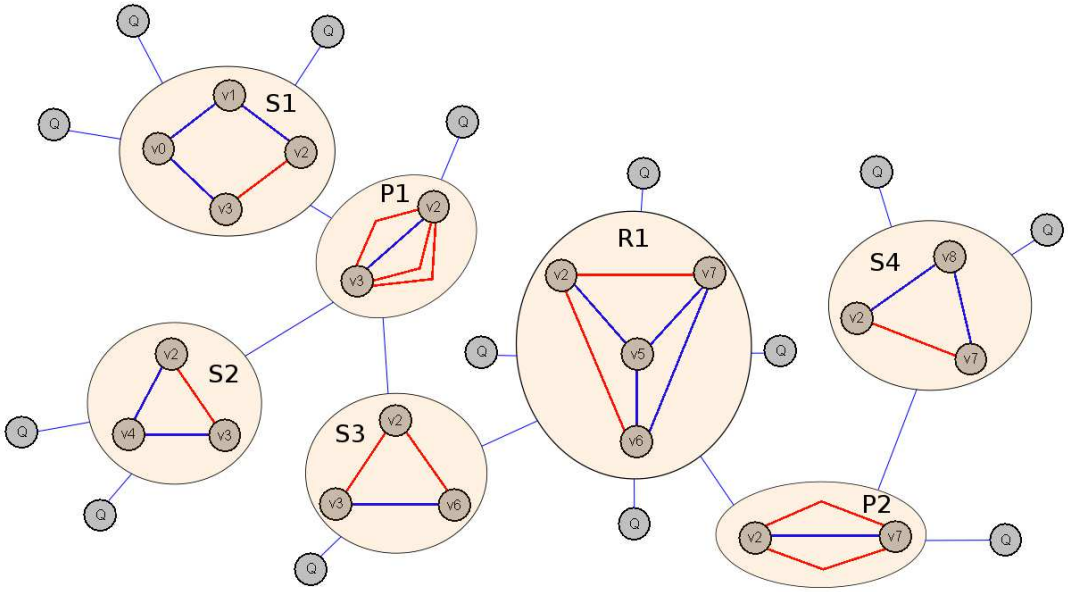
4. Rigidni slučaj - ako nijedan od prethodnih uslova nije zadovoljen, neka su $\{s_1, t_1\}, ..., \{s_k, t_k\}$ za $K \geq 1$ maksimalni razdelni parovi G prema $\{s,t\}$. Neka je za $i = 1, ..., k$ G_i unija svih razdelnih komponenti para $\{s_i, t_i\}$, osim onog koji sadrži granu e. Koren stabla T je tada R-čvor čiji kostur se dobija zamenom svakog podgrafa G_i granom $e_1 = (s_i, t_i)$.

Osim u trivijalnom slučaju, μ ima decu $\mu_1, \mu_2, ...\mu_k$ takvu da je μ_i koren SPRQ stabla grafa $G_i \cup e_i$ prema grani e_i, za $i = 1, 2, ..., k$. Krajevi grane e_i se nazivaju polovima čvora μ_i. Virtuelna grana čvora μ_i je grana e_i kostura od μ. Stablo T se kompletira dodavanjem Q-čvora koji predstavlja referentnu granu e i proglašenjem ovog čvora roditeljem μ, postajući koren. Na slici 4.8 prikazan je graf za koji je konstruisano stablo sa slike 4.9. Primer je inspirisan primerom konstrukcije SPQR-stabala iz [19].



Slika 4.8: Graf za koji se konstruiše SPQR-stablo

Gutvenger i Mucel u manjoj meri menjaju ovu definiciju SPQR-stabla, dajući svoju, ekvivalentnu polaznoj. Naime, ovi autori izostavljaju Q-čvorove i uvode pojam stvarnih grana, koje se razlikuju od virtuelnih unutar kostura grafova. Naime, grana u kosturu od μ koja je pridružena Q-čvoru u originalnoj definiciji jeste stvarna grana koja nije u vezi sa detetom od μ, dok su sve druge grane kostura virtuelne i povezane sa P-,S- ili R-čvorom. Koristeći ovu definiciju, može se pokazati da su kosturski grafovi jedinstvene tripovezane komponente grafa G. P-čvorovi odgovaraju skupovima multigrana, S-čvorovi poligonima, a R-čvorovi tripovezanim grafovima. Algoritam za



Slika 4.9: SPQR stablo grafa sa prethodne slike

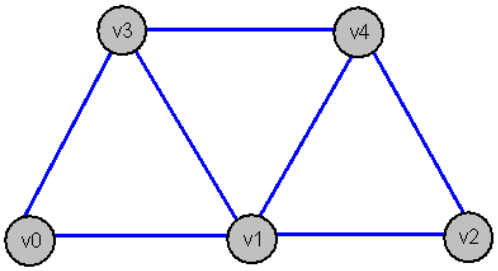
konstrukciju SPQR stabala u linearnom vremenu Gutvengera i Mucelove zasnovan je na tripovezanoj deobi Hopkrofta i Taržana, ali uz nekoliko bitnih korekcija, kojima se ne menja suštinska ideja, već modifikuju određeni uslovi i funkcije koje se koriste, te vrše ažuriranja nekolicine vrednosti koja su inicijalno previđena.

4.3 Algoritmi za ispitivanje cikličnosti grafova

Prilikom analize grafova, često je potrebno naći njihovu cikličnu strkturu. Imajući u vidu da se sama definicija ciklusa razlikuje za usmerene i neusmerene grafove, nije iznenađujuće da su razvijeni različiti algoritmi za dve spomenute grupe. U svakom slučaju, standardni način uključuje nalaženje fundamentalnog skupa elementarnih ciklusa. Fundamentalni skup elementarnih ciklusa je skup elementarnih ciklusa čijim kombinovanjem se mogu generisati svi ostali ciklusi. Ciklus je elementaran ukoliko se nijedan čvor ne ponavlja, osim prvog koji je ujedno i poslednji. Osnovna ideja fundamentalnog skupa elementarnih ciklusa može se uočiti posmatranjem primera sa slike 4.10. Prikazani graf ima šest jednostavnih ciklusa - (v_0, v_1, v_3), (v_1, v_3, v_4), (v_1, v_2, v_4), (v_0, v_1, v_3, v_4), (v_1, v_3, v_4, v_2), $(v_0, v_1, v_2, v_3, v_4)$. Međutim, nije teško uočiti da se kombinovanjem prva tri mogu dobiti preostali.

4.3.1 Cikličnost neusmerenih grafova

Za neusmerene grafove razvijen je veliki broj algoritama za nalaženje fundamentalnog skupa ciklusa, poput [139]–[141]. Za ovaj tip grafova može se primetiti da je kombinovanje ciklusa mnogo slobodnije nego u slučaju orijentisanih, budući da usmerenje grane nije od značaja. Standardni algoritmi za nalaženje fundamentalnog skupa ciklusa neusmerenih grafova bazirani su na ideji kreiranja razapinjućeg stabla, te pravljenju ciklusa za svaku granu grafa koja ne pripada stablu. Kao što je već napomenuto, svako dodavanje grane između čvorova razapinjućeg stabla uspostavlja jedan ciklus. Razni predloženi algoritmi, poput tri spomenuta u ovom pasusu,



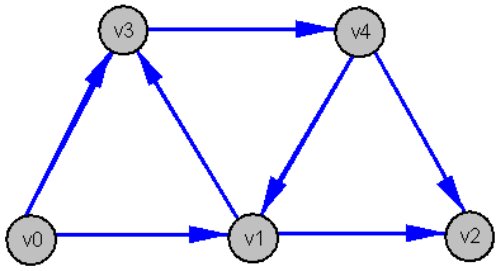
Slika 4.10: Jednostavan graf sa šest elementatnih ciklusa

produbljuju ovu osnovnu ideju, trudeći se da kalkulacija potroši što je moguće manje resursa tj. vremena i memorije.

U projektnom rešenju za implementaciju je izabran poslednji spomenuti, Patonov algoritam za nalaženje fundamentalnog skupa ciklusa. Radi se o relativno jednostavnom algoritmu u okviru kojeg se pretragom prvo u širinu prolazi kroz razapinjuće stablo, te za svaki čvor analiziraju njegovi susedi, i na kraju pamti ko im je roditelj u stablu. Ukoliko se ispostavi da je sused već prethodno posećeni čvor, konstatuje se da se naišlo na ciklus, te se određuje koji čvorovi mu pripadaju. Pošto se beleži ko je roditelj obrađenim čvorovima, kretanje kroz stablo radi formiranja detektovanog ciklusa ne predstavlja problem. Takođe, algoritam proverava i da li je neki čvor sam sebi sused, što znači da je pronađena petlja, koja je takođe ciklus.

4.3.2 Cikličnost usmerenih grafova

Rešavanje istog problema je nešto kompleksnije za usmerene grafove, ali i u ovom slučaju postoji veći broj algoritama koji mu se posvećuju. Na primer, [142]–[144]. Orijentacija grana grafa u ovom slučaju ima značajnu ulogu. Ukoliko se grane grafa sa prethodne slike orijentišu kao na slici 4.11, može se uočiti samo jedan ciklus - (v_1, v_3, v_4). Međutim, osnovna ideja je veoma slična onoj za neorijentisane grafove. Dakle, posmatraju se razapinjuće stablo i grane grafa koje mu ne pripadaju. U ovom slučaju razlikuju se tri tipa ovih grana, a samo usmerene povratne grane (koje su usmerene od potomka do pretka u stablu) formiraju cikluse.



Slika 4.11: Primer usmerenog grafa sa jednim ciklusom

Algoritam implementiran u projektnom rešenju je Džonsonov (Johnson) algoritam za nalaženje elementarnih ciklusa usmerenih grafova [144]. U osnovi, algoritam polazi

od nekog uređenja čvorova i u svakoj iteraciji određuje tekući korenski čvor s i izdvaja podgraf koji ga sadrži, kao i sve čvorove koji se u uređenju nalaze posle s. Da bi se izbeglo dupliranje ciklusa, čvor v se proglašava blokiranim kada se doda u ciklus koji počinje u s. Čvor ostaje blokiran dokle god svaka putanja od v do s preseca tekuću elementarnu putanju u čvoru koji nije s. Takođe, algoritam se stara o tome da čvor ne postane korenski za konstrukciju elementarnih putanja ukoliko nije najmanji čvor (sa najmanjim indeksom u uređenju) u bar jednom elementarnom ciklusu. Iz ovih razloga algoritam izbegava pretrage koje neće uroditi plodom, što ga čini efikasnijim od mnoštva drugih rešenja datog problema.

Nalaženje čvora pogodnog da bude tekući korenski podrazumeva nalaženje strogo povezanih komponenti podgrafa indukovanih tekućim indeksom uređenja čvorova tj. podgrafa koji sadrži sve čvorove grafa indeksa većeg ili jednakog zadatom. Korenski čvor postaje čvor sa najmanjim indeksom koji se nalazi u nekoj komponenti, dok se sama ta komponenta izdvaja kao podgraf u okviru kojeg se u tekućem koraku traže ciklusi. Počinje se od najmanjeg indeksa koji se povećava u svakoj iteraciji, čime i podgraf postaje sve manji. U svakoj narednoj iteraciji indeks se postavlja na indeks prethodnog tekućeg čvora uvećanog za jedan. Algoritam se završava ako više nema čvorova koji mogu biti korenski, ili je početni indeks dostigao ukupan broj čvorova grafa.

4.4 Nalaženje uređenja grafa

Mnogi algoritmi u oblasti teorije grafova i njihovog crtanja kao svoj prvi korak navode nalaženje određenog uređenja ulaznog grafa. Odnosno, sortiranje čvorova grafa u cilju dobijanja zahtevanog poretka. Među češće korišćenim uređenjima spadaju st-numeracija i topološko uređenje.

Lempel, Even i Kaderbaum uveli su st-numeraciju [145] 1967. godine, da bi Even i Taržan devet godina kasnije objavili algoritam za njegovo efikasno nalaženje [146]. Ova numeracija od velike je važnosti za mnoge algoritme za crtanje grafova, pre svih ortogonalne i hijerarhijske, a sastavni je deo i pojedinih algoritama za ispitivanje planarnosti grafa, poput testa korišćenjem PQ-stabala. Ako je data grana (s,t) bipovezanog grafa G sa n čvorova, ti čvorovi se mogu numerisati brojevima od 1 do n, tako da čvor s dobije broj 1, a čvor t broj n, dok je svaki drugi čvor susedan i čvoru koji ima veći broj od njega, i čvoru koji ima manji. Dobijena numeracija naziva se st-numeracijom.

Originalni algoritam za nalaženje st-numeracije imao je $O(nm)$ vreme izvršavanja, dok ga Even i Taržan skraćuju na $O(n+m)$. Broj m odgovara broju grana datog grafa. Algoritam Evena i Taržana obuhvata tri procedure:

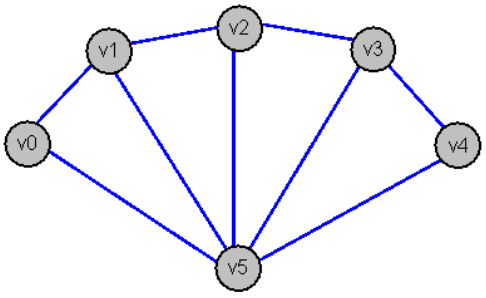
1. Pretragu najpre u dubinu i nalaženje razapinjućeg stabla T grafa G takvog da je prva grana pretrage (t,s). Dakle, t je koren stabla, a $t \to s$ jedna od njegovih grana. Tokom pretrage se računa i početna numeracija čvorova.

2. Proceduru koja nalazi jednostavne putanje između određenih čvorova. Pri inicijalnom izvršavanju nalazi se putanja od s do t koja ne sadrži granu (s,t) uz pamćenje posećenih čvorova. U svakom narednom pozivu se određuje putanja koja ne sadrži nijednu granu koja je deo neke prethodno nađene putanje od nekog zadatog posećenog čvora v do posećenog čvora w koji je povezan sa v.

3. Treća nalazi samu st-numeraciju. Koristi strukturu tipa stek, inicijalno stavljajući na njega prvo t, pa s. U svakoj iteraciji, dok se stek ne isprazni, uzima čvor sa

vrha steka, i dodeljuje mu broj koji je inicijalno 1 i koji se povećava posle svake dodele. Na stek se potom stavljaju čvorovi koji pripadaju putanji koju detektuje procedura za nalaženje putanji pozvana za čvor sa vrha steka. Čvorovi putanje se stavljaju u obrnutom poretku - od poslednjeg do prvog.

Na slici 4.12 prikazan je graf za koji se može naći st-numeracija. Ako je, na primer, $s = v_0$, a $t = v_1$, st-numeracija koja se izračunava je: $v_0 = 1, v_5 = 2, v_4 = 3, v_3 = 4, v_2 = 5, v_1 = 6$, što je u skladu sa definicijom.



Slika 4.12: Bipovezani graf za koga se nalazi st-numeracija

U vezi sa *st*-uređenjem su i pojmovi planarnog *st*-grafa i njegovog duala. Naime, *st*-graf se definiše kao orijentisani aciklični graf koji ima jedan izvor i jedan ponor. Planarni *st*-graf je *st*-graf koji ima planarno utapanje takvo da se čvorovi s i t nađu na spoljašnjem licu. Za *st*-graf G dualni planarni *st*-graf $G^* = (V^*, E^*)$ definiše se kao digraf sa sledećim osobinama:

1. V^* je skup lica grafa G, sa izuzetkom spoljašnjeg $(s, ..., t, ...s)$ koje je podeljeno na dva dela koja se dodaju u skup. Prvi deo, s^*, zahvata čvorove od s to t, a drugi deo, t^*, od t nazad do s.

2. Za svaku granu $e \in E$ postoji grana $e^* = (f, g) \in E^*$, gde je f lice levo od e, a g desno.

Topološko uređenje grafa nalazi se za orijentisani aciklični graf $G = (V, E)$ i označava sa $T(G)$. Ono predstavlja dodeljivanje celobrojnih vrednosti $T(v)$ svakom čvoru $v \in V$ tako da za svaku orijentisanu granu (u, v) važi $T(u) < T(v)$. Veličina topološkog uređenja $s(t)$ računa se kao razlika maksimalne i minimalne vrednosti dodeljenih nekom od čvorova u uređenju. Optimalno topološko uređenje $T^*(G)$ je topološko uređenje najmanje veličine [83]. Topološko sortiranje koristi se za rešavanje mnogobrojnih praktičnih problema koji se mogu predstaviti grafovima. U oblasti crtanja grafova javlja se u okviru ortogonalnih algoritama.

Algoritam za nalaženje optimalnog topološkog uređenja nekog orijentisanog acikličnog grafa obuhvata sledeće korake:

1. Nalaženje svih čvorova koji nemaju nijednu ulaznu granu i inicijalizacija broja koji se dodeljuje čvorovima, n, na 0.

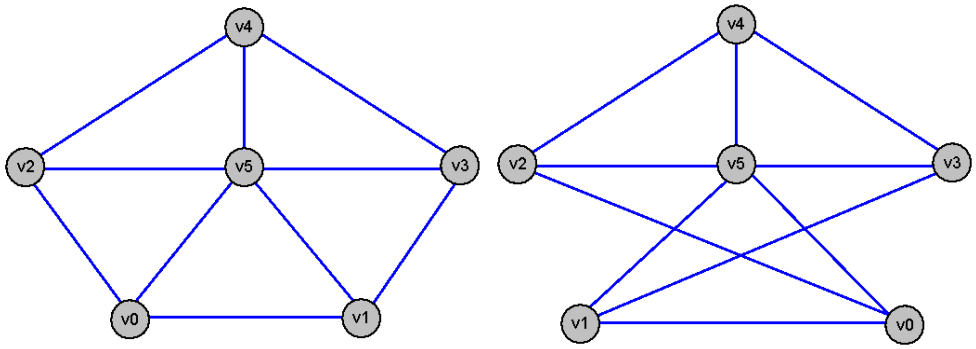
2. Uklanjanje svih identifikovanih čvorova iz grafa, što uključuje i grane koje ih spajaju sa drugim čvorovima. Broj ulaznih grana susednih uklonjenih čvorova se time smanjuje. Svakom čvoru koji se uklanja dodeljuje se tekuća vrednost broja n, koja se posle svake dodele inkrementira.

3. Ponavljanje koraka 1 i 2 za novodobijeni graf dokle god postoje čvorovi koji nemaju ulazne grane.

4.5 Algoritmi za ispitivanje planarnosti i planarno utapanje

Planarnost predstavlja osobinu od velikog značaja za crtanje grafova. Kao što je prethodno definisano, graf je planaran ukoliko ima planaran crtež, odnosno, ako se može nacrtati bez presecanja grana. Mnoštvo algoritama za crtanje razvijeno je za ovaj tip grafova, garantujući da će rezultat njihove primene biti u skladu sa spomenutim zahtevom. Naravno, njihova primena je moguća jedino ukoliko je uslov planarnosti zadovoljen. Osim čistog odgovora na pitanje da li je neki graf planaran ili ne, neki algoritmi za ispitivanje planarnosti nalaze i planarno utapanje, koje je ulaz u mnoge algoritme za crtanje grafova bez presecanja grana.

Utapanje planarnog grafa $G = (V, E)$ jeste konstruisanje lista susedstva, takvih da za neki čvor v lista $A(v)$ sadrži sve njegove susede, u rasporedu smera kazaljke na satu prema stvarnom crtežu [147]. Radi pojašnjenja veze utapanja sa planarnim crtanjem grafova, mogu se uporediti dva crteža istog apstraktnog grafa, prikazana na slici 4.13. U prvom crtežu lista susednih čvorova u smeru kazaljke na satu čvora v_5 je v_4, v_3, v_1, v_0, v_2, a u drugom v_4, v_3, v_0, v_1, v_2.



Slika 4.13: Dva crteža grafa gde je jedan planaran a drugi nije

Od sredine prošlog veka do danas razvijen je veliki broj algoritama za testiranje planarnosti. Stariji su se izvršavali u suplinearnom vremenu, da bi prvi algoritam u linearnom vremenu dali Hopkroft i Taržan 1974. godine [147]. Od tada se konstantno radi na poboljšanju efikasnosti i elegancije algoritama ovog tipa.

Rani algoritmi pomenute namene prvenstveno se mogu svrstati u grupu cikličnih algoritama za proveru planarnosti. Ideja svih algoritama ove klase zasniva se na Žordanovoj teoremi, koja tvrdi da svaka zatvorena jednostavna kriva deli ravan na dva povezana regiona, gde se dve tačke mogu povezati samo ako pripadaju istom regionu. Aciklični grafovi su planarni, a ako graf sadrži ciklus, taj ciklus u svakom planarnom crtežu uzrokuje jednostavnu zatvorenu krivu. Svaki preostali povezani deo grafa mora biti u celosti nacrtan u jednom od dva regiona tog ciklusa.

Algoritmi ove grupe dalje se mogu podeliti na algoritme dodavanja segmenata, dodavanja putanja i dodavanja grana [19]. Predstavnik prve grupe i njen začetnik je Oslander-Parter (Auslander-Parter) algoritam [148]. Ovaj algoritam zahteva $O(n^3)$ vremena. Predstavnik druge grupe je spomenuti prvi algoritam za planarno testiranje u linearnom vremenu čiji su tvorci Hopkroft i Taržan. Algoritam, međutim, karakteriše i teškoća implementacije, a u prvobitnom obliku ne vraća planarno utapanje. Predstavnik treće grupe je algoritam De Frajse - Osona de Mendez - Rozenštil (de Fraysseix - Ossona de Mendez - Rosenstiehl) [149], koji je jednostavniji od prethodno spomenutih,

pritom dajući i mogućnost nalaženja planarnog utapanja. Ovaj algoritam je izabran za implementaciju u projektnom rešenju, te će biti detaljnije objašnjen u nastavku.

Drugu grupu algoritama za testiranje planarnosti čine algoritmi zasnovani na dodavanju čvorova. Da bi se predstavila njihova osnovna ideja, može se primetiti da se, polazeći od planarnog crteža nekog grafa, može uklanjati jedan po jedan čvor, da bi se na kraju dobila sekvenca manjih planarnih crteža završno sa jednim izolovanim čvorom. Izvršavanje ovog postupka u suprotnom smeru (polazeći od jednog čvora) je glavni zadatak ovakvih algoritama. Generalno, kod algoritama ovog tipa počinje se od jednog izolovanog čovra v_1 koji čini graf G_1. U svakom koraku, i, dodaje se novi čvor v_i i posmatra se graf $G_i(V_i, E_i)$, podgraf početnog grafa G sačinjen od dotle dodatih čvorova. Ispituje se planarnost grafa G_i i ažurira struktura podataka namenjena efikasnom ispitivanju u koraku $i+1$. Raspored dodavanja čvorova nije proizvoljan i kreće se od lista ka korenu nekog stabla nastalog od početnog grafa G. U [145], predstavljen je 1967. godine prvi algoritam koji se zasniva na paradigmi dodavanja čvora i ima kvadratno vreme izvršavanja.

Značajan pomak u oblasti desio se nakon formulisanja algoritma za testiranje planarnosti upotrebom takozvanih PQ-stabala Buta (Kellogg Booth) i Luekera (George Lueker) [150] 1976. godine. Ovaj algoritam može biti implementiran tako da ima linearno vreme izvršavanja, ali se odlikuje popriličnom kompleksnošću. PQ-stablo definiše se kao struktura koja se sastoji iz P-čvorova (artikulacionih tačaka grafa), Q-čvorova (bipovezanih komponenata - blokova) i lišća - virtuelnih čvorova. Osnova ideja algoritma obuhvata primenu niza šablona na čvorove PQ-stabla. Ovakvih šablona ima preko deset i definišu delove PQ-stabla koji se traže i zamenjuju, kao i samu zamenu. Utapanje koje se na kraju dobija nije konačno planarno, nego mora biti prošireno, prema uputstvima u [151].

Sledeći veliki korak unutar ove familije predstavlja algoritam Ši-Su (Shih-Hsu), čija je konačna verzija objavljena više od dve decenije nakon algoritma Buta i Luekera. Ovo je prvi algoritam u kom se čvorovi grafa razmatraju u obrnutom DFS poretku. Algoritam zamenjuje PQ-stabla PC-stablima. PC-stabla u osnovi jesu veoma slična PQ-stablima, takođe posedujući čvorove koji se odnose na artikulacione čvorove stabla i blokove, ali nisu ukorenjena. PC-stabla verno predstavljaju parcijalno planarno utapanje na prirodniji način. Primena šablona je znatno jednostavnija.

Naredni iskorak u smanjenju kompleksnosti testiranja planarnosti, uz zadržavanje linearnog vremena izvršavanja jeste Bojer-Mirvold (Boyer-Myrvold) [152] algoritam. Objavljen je 2004. godine i još se smatra jednim od najmodernijih. Tehnički gledano, ovaj algoritam ne pripada familiji dodavanja čvorova kako se zapravo dodaju čitave grane, ali ga neki autori ipak svrstavaju u tu grupu [19], pošto je inspirisan pojedinim idejama njenih pripadnika. Ovaj algoritam je izabran za implementaciju u projektnom rešenju i biće u nastavku detaljnije opisan.

Takođe, treba spomenuti i inkrementalne algoritme za ispitivanje planarnosti. Radi se o klasi algoritama dizajniranoj za primenu u dinamičkom okuženju, gde se planarni graf G gradi postepeno dodavanjem novih čvorova i grana. Njihov osnovni cilj je brzo ustanoviti da li se dati element može dodati grafu bez gubitka planarnosti. Osmišljeno je više algoritama koji se bave navedenim problemom, kao što su [134], [153], [154].

4.5.1 De Frajse - Osona de Mendez - Rozenštil (de Fraysseix - Ossona de Mendez - Rosenstiehl) algoritam

Algoritam De Frajse - Osona de Mendez - Rozenštil predstavnik je klase algoritama za ispitivanje planarnosti putem dodavanja grana. Algoritam pojednostavljuje ideje

Oslandera i Partera, dodajući jednu po jednu granu.

Neka je dat neorijentisani graf koji ne mora biti bipovezan i neka je $G = (V, T \uplus B)$ usmereni graf koji se dobija DFS pretragom, gde je T skup grana stabla, a B skup povratnih grana. Svaka povratna grana (u, v) se vraća na pretka polaznog čvora, formirajući cilkus koji se naziva fundamentalnim ciklusom. Grana (u, v) je povratna za svaku granu svog fundamentalnog ciklusa, osim za prvu granu koja izlazi iz v.

Podelom $B = L \uplus R$ povratnih grana na dve klase, levu i desnu, dobija se LR particija, ako za svaki čvor v sa ulaznom granom stabla e i izlaznim e_1 i e_2 važi:

- Sve povratne grane e_1 koje se završavaju striktno više od $lowpt(e_2)$ pripadaju jednoj klasi.
- Sve povratne grane e_2 koje se završavaju striktno više od $lowpt(e_1)$ pripadaju drugoj.

Funkcija *lowpt* koja se spominje u definiciji ima isto značenje kao i u algoritmima Hopkrofta i Taržana - najmanji DFS indeks pretka do kojeg se dolazi preko nula ili više grana stabla i jedne povratne grane. Ranije se računala za čvor, ovde se primenjuje nad granom. Prosto, za granu se uzme niži (sa većim DFS indeksom) čvor i za njega se izračuna vrednost *lowpt*. Treba spoemnuti i funkciju $highpt(v)$, koju su takođe koristili Hopkroft i Taržan. Ona se, naime, definiše kao najveći DFS indeks pretka čvora v koji se može dostići preko povratne grane potomka čvora v.

Particionisanje povratnih grana u L i R klase odgovara orijentisanju fundamentalnih ciklusa tako da oni zatvoreni granom iz L budu orijentisani suprotno od smera, a oni u R u smeru kazaljke na satu. Odavde se izvodi sam uslov planarnosti koji se koristi u ovom algoritmu. Naime, graf je planar ako i samo ako se može formirati LR particija.

U LR particiji može se pretpostaviti da se sve povratne grane iz grane e koje se vraćaju u $lowpt(e)$ nalaze na istoj strani. Za takvu particiju se kaže da je poravnata. Svaka LR paricija se može srediti tako da bude poravnata.

Mogućnost formiranja LR particija dakle daje odgovor na pitanje da li graf ima planarno utapanje. Prirodno je u slučaju pozitivnog odgovora pokušati ga naći. Upravo se iz tog razloga nakon particioniranja sprovode sledeća dva koraka:

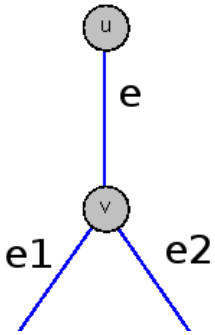
1. LR particija se proširuje tako da se pokriju izlazne grane stabla za svaki čvor v. Ako grana stabla ima povratne grane, a čvor u koejm počinje nije ni koren ni artikulacioni čvor, stavlja se na istu stranu kao i jedna od povratnih grana koja se završava u najvišoj tački. U suprotnom, strana je proizvoljna.

2. Posmatraju se grane koje izlaze iz čvora v. Neka su to e_1 i e_2 kao na slici 4.14. Ako i e_1 i e_2 imaju povratne grane, v je tačka grananja bar dva fundamentalna ciklusa. Pomenute dve grane se moraju rasporediti tako da se spreči pojava preseka grana.

   Pretpostavlja se da sve povratne grane pripadaju desnoj particiji i da je koren stabla na spoljnom licu, dok se grane raspoređuju imajući sledeće u vidu:

   - e_2 je u smeru kazaljke na satu posle e_1 ako i samo ako je $lowpt(e_1) < lowpt(e_2)$
   - e_2 je posle e_1 ako je $lowpt(e_2) = lowpt(e_1)$, a samo e_2 ima povratni čvor iznad sebe. Odnosno, ako je samo ona akord.

   Ako nijedna od particija L i R nije prazna, može se desiti da su i e_1 i e_2 akordi. U tom slučaju se odlučuje proizvoljno, pošto u bilo kom planarnom utapanju ove grane mogu biti u različitim particijama.



Slika 4.14: Grana e iz koje izlaze e_1 i e_2

Neka je $e = (v, w)$ grana stabla. Sa $L(e)$ ($R(e)$) označava se sekvenca ulaznih povratnih grana u v iz nekog potomka čvora w uređena tako da ako su $b_1 = (x_1, v)$ i $b_2 = (x_2, v)$ dve takve grane i ako je $(z, x), (x, y_1), (x, y_2)$ račvanje dva ciklusa zatvorena sa b_1 i b_2, tada b_1 dolazi pre b_2 u $L(e)$ ($R(e)$)ako i samo ako (x, y_1) dolazi pre (x, y_2) u listi susedstva čvora x, odnosno posle u slučaju $R(e)$.

Nake je za LR i čvor v $e_1^L, e_2^L, ..., e_l^L$ skup levih izlaznih čvorova stabla iz v, a $e_1^R, e_2^R, ..., e_l^R$ desnih. LR uređenje grana oko v je sledeće:

$$(u, v), L(e_l^L), e_l^L, R(e_l^L), .., L(e_l^1), e_l^1, R(e_l^1), L(e_r^1), e_r^1, R(e_r^1), ..., L(e_r^R), e_r^R, R(e_r^R).$$

Ako je v koren, (u, v) se izostavlja. LR uređenje daje planarno utapanje.

Neka su $b_1 = (u_1, v_1)$, i $b_2 = (u_2, v_2)$ dve povratne grane sa prepletenim fundamentalnim ciklusima, a (u, v), (v, w_1), (v, w_2) njihovo račvanje. Neka je sa $C(e)$ obeležen fundamentalni ciklus neke grane e. Tada b_1 i b_2 zadovoljavaju sledeće:

1. b_1 i b_2 pripadaju drugačijim klasama ako je $lowpt(w_2) < v_1$ i $lowpt(w_1) < v_2$, odnosno

2. b_1 i b_2 pripadaju istoj klasi ako postoji grana $e' = (x, y), x \in C(b_1) \cap C(b_2), u \notin c(b_1) \cap C(b_2)$ tako da je $lowpt(y) < min(v_1, v_2)$

Ako postoji par povratnih grana koji zadovoljava oba uslova, ne može postojati LR particija i graf nije planaran. Algoritam u originalnom obliku nije komplikovan za implementaciju, ali se izvršava u kvadratnom vremenu, ne u linearnom. Međutim, moguće ga je proširiti tako da se postigne linearno vreme [19].

4.5.2 Bojer-Mirvold (Boyer-Myrvold) algoritam

Algoritam Bojer-Mirvold predstavnik je modernijih pristupa za ispitivanje planarnosti. Ima linearno vreme izvršavanja i pruža mogućnost kako nalaženja planarnog utapanja ukoliko provera da pozitivan odgovor, tako i grafa Kuratovkog koji je podgraf ulaznog neplanarnog grafa.

Osnovna ideja algoritma zasniva se na konstrukciji i održavanju strukture $\tilde{G}$ - skupa bipovezanih komponenata grafa. Komponente inicijalno sadrže samo jednu granu polaznog grafa, te se proširuju i spajaju u narednim koracima algoritma, kako se koja grana utopi. Ključan zahtev koraka dodavanja grana jeste zadržavanje svih čvorova koji treba da učestvuju u budućim utapanjima grana na spoljašnjem licu nove komponente. Iz tog razloga, neke bipovezane komponente pre spajanja moraju biti obrnute.

Jedan od ključnih problema koje moraju da reše efikasni algoritmi za ispitivanje planarnosti jeste kako da dodavanje nekog dela ulaznog grafa postojećem utapanju ne zahteva njegovu veću naknadnu korekciju pre nego što proces može biti nastavljen. Algoritam koji koristi PQ-stabla podrazumeva prethodnu konstrukciju s,t - numeracije, dok se Bojer-Mirvold (kao i Ši-Su) oslanja na indekse izračunate tokom pretrage najpre u dubinu. Naime, čvorovi se obrađuju u obrnutom DFS redosledu, tako da postoji putanja između svakog čvora koji se obrađuje u datom koraku i čvora koji će poslednji biti procesiran - korena DFS stabla. Iz tog razloga neobrađeni čvorovi moraju biti na spoljnom licu bipovezanih komponenti parcijalnog utapanja $\tilde{G}$.

Među najbitnijim konceptima u datom algoritmu za ispitivanje planarnosti su spoljno aktivni čvorovi. Neka je w DFS potomak čvora v u bipovezanoj koponenti B. Čvor w je spoljno aktivan ako postoji putanja od w do DFS pretka u čvora v koji se sastoji od povratne grane i nula ili više potomaka čvora w koji nisu u B. Ukoliko je čvor spoljno aktivan, može se tvrditi da će učestvovati u budućem utapanju grana po procesiranju čvora v. Oni moraju ostati na spoljašnjem licu bipovezane komponente koja ih sadrži.

Kao što je spomenuto, korak dodavanja grana može zahtevati obrtanje bipovezane komponente. Najlakši način da se ovo uradi bio bi invertovanje liste susedstva, ili orijentacije, svih čvorova komponente. Međutim, algoritam Bojer-Mirvold sugeriše i kako bi ta operacija mogla biti efikasno implementirana. U tu svrhu, definiše se prvo koren bipovezane komponente B - r, kao čvor koji ima najmanji DFS indeks. Grana koja povezuje koren r sa svojim jedinim DFS detetom c u B naziva se korenskom. U bipovezanoj komponenti sa korenskom granom (r, c), čvor r se predstavlja virtuelnim čvorom r^c, kako bi se razlikovao od drugih kopija čvora r u $\tilde{G}$. Obrtanje bipovezane komponente se vrši invertovanjem orijentacije svog korenskog čvora.

Planarno utapanje sa konzistentnom orijentacijom čvorova se može dobiti postprocesiranjem ukoliko se čuva jedna dodatna informacija tokom utapanja. Svakoj grani se pridružuje znak, koji se inicijalizuje na +1. Kada se bipovezana komponenta obrće, samo se invertuje orijentacija liste susedstva korenskog čvora r^c kako bi odgovarala orijentaciji čvora r u komponenti sa kojom se spaja. Potom se znak korenske grane (r^c, c) menja u -1. Planarno utapanje za bilo koju bipovezanu komponentu se može dobiti u bilo kom trenutku ako se orijentacija njenog korena nametne svim čvorovima te komponente. Ako je proizvod znakova grana putanje kroz stablo od čvora do korena bipovezane komponente -1, lista susedstva tog čvora se obrće.

Nakon opisa osnovnih ideja i operacija algoritma, mogu se izložiti njegovi ključni koraci:

1. Izdvojiti tekući čvor v, počevši od čvora sa najvećim DFS indeksom do onog sa najmanjim.

2. Za svako dete c čvora v u ulaznom grafu G kreirati jednu bipovezanu komponentu koja samo nju sadrži u $\tilde{G}$.

3. Za svaku povratnu granu grafa G (v, w), gde je w potomak čvora v, sprovodi se *walkup* rutina.

4. Za svako DFS dete c čvora v u G utapaju se povratne grane između čvora v i potomaka čvora c. Odnosno, sprovodi se *walkdown* rutina.

5. Za svaku povratnu granu datog čvora v koja ga povezuje sa njegovim potomkom w, proverava se da li je grana utopljena. Ako nije, graf nije planaran i eventualno se izdvaja podgraf Kuratovskog.

6. Ako je graf planaran, određuje se kompletno planarno utapanje.

U *walkup* fazi identifikuju se čvorovi i povezane komponente bitne u kontekstu ugradnje date grane. Krećući se od v do w granama stabla, izdvajaju se artkulacioni čvorovi, pošto će oni postati tačke spajanja. Takođe, za svaki artikulacioni čvor r sa DFS detetom s koje je takođe na razmatranoj putanji, nalazi se r^s, koren bipovezane komponente u $\tilde{G}$.

U *walkdown* koraku obilazak počinje od čvora v^c i kreće se po spoljašnjem licu bipovezane komponente B kojoj čvor pripada. Spoljašnje lice se obilazi dva puta, jednom za svaku putanju kroz njega koja počinje u v^c. U oba slučaja postupak je isti. Dakle, traže se krajevi povratnih grana sa nižim DFS indeksom. Kada se naiđe na čvor koji je i koren neke bipovezane komponente, prelazi se na nju i pretraga se nastavlja. Kada se dođe do traženog čvora, spajaju se sve koponente koje su se prešle, uz moguće obrtanje. Potom se povratna grana utapa. Komponenta se obrće ako se smer obilaska po ulasku u artikulacioni čvor r menja po izlasku iz r^s. Ova faza se završava ili kada se obilazak vrati do v^c, ili kada se dođe do čvora koji nije označen kao bitan, ali je spoljno aktivan. Takav čvor se naziva stopirajućim. Da bi se obezbedilo da će sve povratne grane koje mogu biti utopljene zaista to i biti i to uz zadržavanje planarnosti, kao i da će se izabrati prave putanje po spoljašnjem licu počevši od korena r^s, algoritam uvodi sledeće dva pravila:

- Kada se naiđe na čvor w, prvo se ugradi povratna grana ka w, pa se onda pređe na njegove potomačke interno aktivne bipovezane komponente (ako postoje), pre nego što se procesiraju bitne spoljno aktivne potomačke bipovezane komponente.
- Kada se određuje putanja na spoljašnjem licu od korena r^s bipovezane koponente ka sledećem čvoru, bolje je izabrati putanju na spoljašnjem licu ka interno aktivnom čvoru ako postoji, u suprotnom izabrati putanju ka bitnom čvoru.

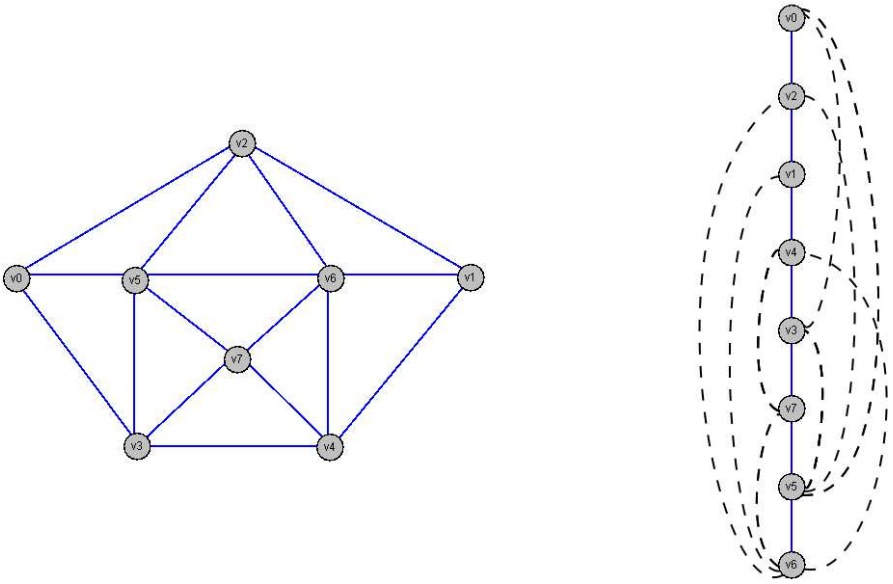
Čvorovi i bipovezane komponente su interno aktivni ako su bitni, ali nisu spoljno aktivni.

Ako obe putanje kroz spoljašnje lice vode ka stopirajućem čvoru, zaključuje se da graf nije planaran i algoritam se odmah zaustavlja. Bojer i Mirvoldova pokazuju da se ovaj korak, kao i čitav algoritam izvršavaju u linearnom vremenu ako se prate sve smernice i optimizacije koje predlažu.

U nastavku će biti dat jedan primer izvršavanja Bojer-Mirvold algoritma za planarni graf sa slike 4.15, gde je prikazano i njegovo DFS stablo ukorenjeno u čvoru v_0.

Kao što se vidi, čvor v_6 ima najmanji DFS indeks i on se prvi uzima u razmatranje, potom v_5, pa v_7, koji je prvi čvor takav da postoji povratna grana iz pretka ka njemu. U pitanju je grana (v_6, v_7), koja se u ovom koraku utapa. U skladu sa drugim korakom, do tog trenutka kreirane su dve bipovezane komponente, jedna koja sadži čvorove v_6 i v_5 i granu između njih, te druga koja sadrži čvorove v_5 i v_7 i granu između njih. Te dve komponente se spajaju i formira se nova koja uključuje i utopljenu povratnu granu. Potom se prelazi na čvor v_3 i utapa grana (v_5, v_3), uz spajanje komponenti kao i u prethodnom slučaju. Čvorovi v_5 i v_6 su spoljno aktivni, kako postoje povratne grane koje ih spajaju sa još neobrađenim čvorovima. Odnosno, pošto će u jednom od narednih koraka učestvovati u utapanju povratne grane. Dakle, prilikom ugradnje povratne grane, pazi se da data dva čvora ostanu na spoljašnjem licu. Na slici slici 4.16 data je vizualizacija opisanih, i nekoliko narednih koraka. Bipovezane komponente su zaokružene, spoljašnje aktivni čvorovi su obojeni plavom bojom, a novoutopljene grane crvenom.

Naredni čvor sa najvećim DFS indeksom je v_4, u kome se završavaju dve povratne grane - (v_7, v_4) i (v_6, v_4). Čvor v_3 je takođe spoljno aktivan, kako u njemu počinje



Slika 4.15: Jedan planarni graf i njegovo DFS stablo

povratna grana (v_3, v_2). Prema pravilima izloženim u opisu algoritma, prvo se utapa povratna grana (v_7, v_4), a potom (v_6, v_4). Nije moguće ugraditi drugu granu tako da svi čvorovi ostanu na spoljnom licu. Naime, novo lice može biti ili v_4, v_7, v_6, ili v_4, v_3, v_5, v_6. Druga opcija sadrži spoljno aktivne čvorove, tako da je utapanje povratne grane neophodno izvršiti na način da upravo to bude novi sadržaj spoljnog lica bipovezane komponente. Dakle, čvor v_7 se nalazi u unutrašnjosti nove komponente, što se može uočiti na slici 4.16. Postupak se dalje nastavlja na sličan način, dok se ne obradi i sam koren stabla.

U prikazanom primeru nije bilo potrebe za obrtanjem nekog bloka, te je jedna ovakva situacija prikazana na slici 4.17. Obrtanje se vrši kako bi oba spoljno aktivna čvora ostala na spoljašnjem licu.

4.5.3 Nalaženje planarnih lica na osnovu planarnog utapanja

Planarno utapanje dobijeno opisanim, ili nekim drugim algoritmom samo po sebi nema posebno veliki značaj pre nego što se na osnovu njega kreiraju lica. Naime, određivanje lica zahtevaju mnogi algoritmi za planarno crtanje grafova.

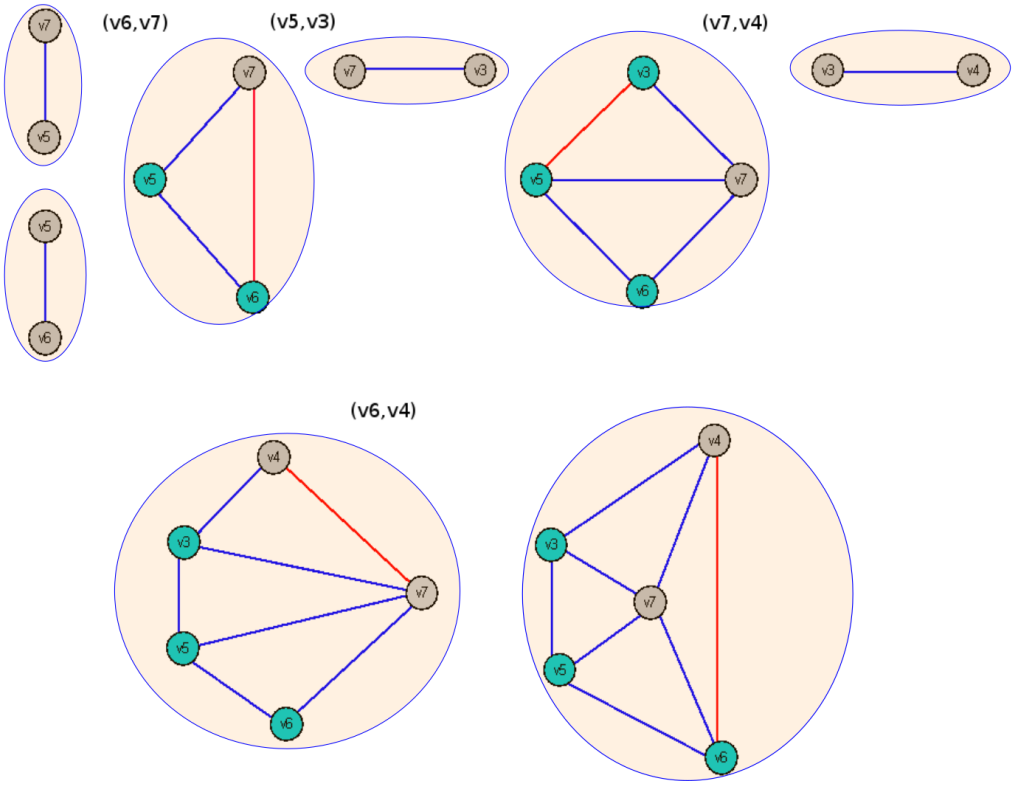
Neka se pod pojmom rotacije smatra cirkularna lista grana jednog čvora u redosledu smera kazaljke na satu. Na osnovu njih lica se kreiraju primenom sledećih nekoliko koraka [85]:

1. Izabere se proizvoljna grana (v, w), te se pređe od v do w.

2. Prođe se granom najbližoj grani (v, w) u tački w u smeru kazaljke na satu. Postupak se nastavlja za svaku narednu granu dok se ne stigne nazad u v.

3. Proces se ponavlja dok se svaka grana ne prođe u oba smera. U svakoj iteraciji počinje se od neke grane u smeru u kom se njom još nije prošlo.

4. Za usmerenu granu (v, w) njeno levo lice je ono koje sadrži (v, w), a desno ono koje sadrži (w, v).



Slika 4.16: Primer izvršavanja algoritma Bojer-Mirvold

Neka je dat graf $G = (V, E)$, gde je $V = \{v_0, v_1, v_2, v_3, v_4\}$, a $E = \{(v_0, v_1), (v_0, v_2), (v_0, v_4), (v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$. Neka je nekim algoritmom za nalaženje planarnog utapanja nađeno utapanje A:

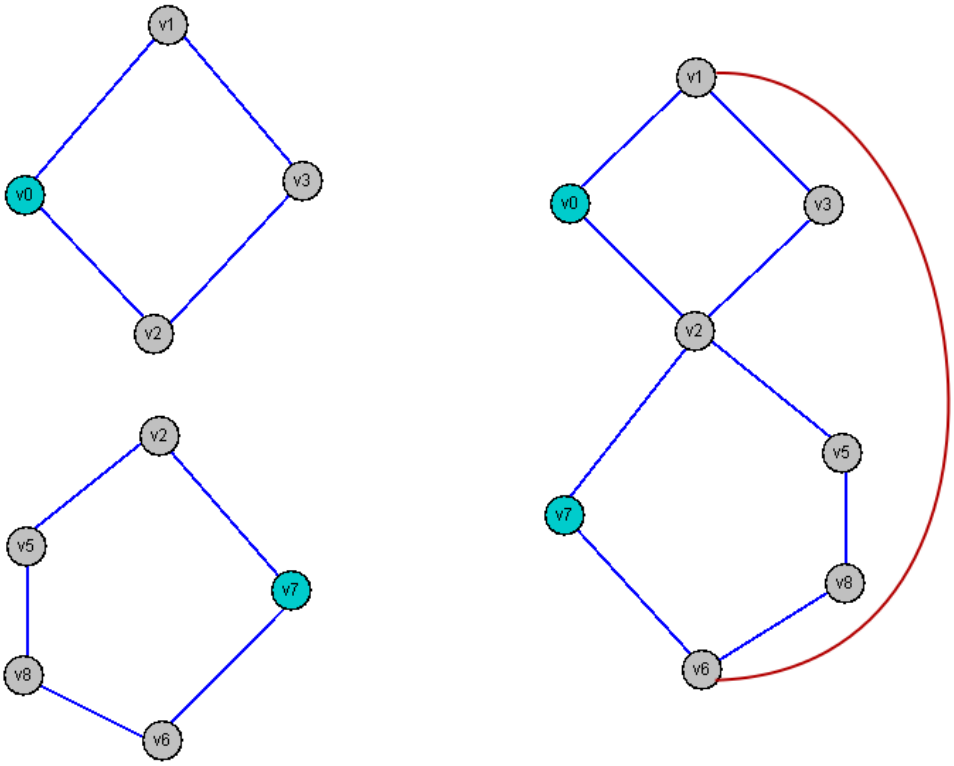
$$v_0 = (v_0, v_4), (v_0, v_2), (v_0, v_1)$$
$$v_1 = (v_1, v_4), (v_0, v_1), (v_1, v_2), (v_1, v_3)$$
$$v_2 = (v_2, v_4), (v_2, v_3), (v_1, v_2), (v_0, v_2)$$
$$v_3 = (v_3, v_4), (v_1, v_3), (v_2, v_3)$$
$$v_4 = (v_1, v_4), (v_3, v_4), (v_2, v_4), (v_0, v_4)$$

Na slici 4.18 prikazan je crtež grafa G u skladu sa ovim utapanjem.

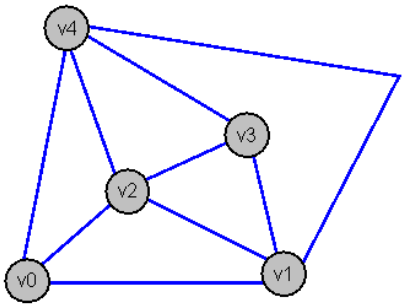
Neka se izabere grana (v_0, v_1), te se iz v_0 pređe u v_1. U v_1 grana najbliža polaznoj u smeru karaljke na satu je (v_1, v_2). Znači, dalje se prelazi u čvor v_2, u kom je sledeća grana u odnosu na (v_1, v_2) (v_0, v_2). Tu se zaokružio ciklus i dobilo se lice $(v_0, v_1), (v_1, v_2), (v_2, v_0)$, koje se jasno može uočiti i na crtežu. Za grane (v_0, v_1) i (v_1, v_2) nađeno lice je levo, za granu (v_0, v_2) desno. Dalje se bira neka druga grana i ponovlja isti postupak, sve dok se, kao što je navedeno, ne prođe svima u oba smera.

4.5.4 Planarna augmentacija

Više algoritama za crtanje planarnih grafova, osim planarnosti, kao preduslov za primenu navodi i potrebu da graf bude bipovezan. Naravno, između planarnosti i bipovezanosti ne postoji tesna veza, tako da nije teško zamisliti graf koji zadovoljava prvi, ali ne i drugi uslov. Međutim, ukoliko bi se grafu dodale grane koje bi ga učinile bipovezanim, a da ostane planaran, ispunjavao bi sve kriterijume pomenutih algoritama.



Slika 4.17: Primer obrtanja algoritma Bojer-Mirvold



Slika 4.18: Crtež grafa G u skladu sa navedenim utapanjem

Upravo je ovo zadatak algoritama za planarnu augmentaciju - dodavanje minimalnog broja grana planarnom grafu tako da postane bipovezan, a ostane planaran. Bitno je težiti da se doda što što manje grana, kako se graf ne bi previše promenio. Problem je NP-kompleksan, ali postoje aproksimacije, poput algoritma [155], koji je implementiran u projektnom rešenju i čije će glavne ideje biti predstavljene u nastavku.

Polazna tačka algoritma je konstrukcija BC-stabla ulaznog grafa G. Pojam koji je od posebne važnosti za dati algoritam su privesci pomenutog stabla. Za BC-stablo $T(G) = (B_G, C_G)$, privesci predstavljaju blokove koji sadrže samo jednu artikulacionu tačku. Ako se između dva priveska B_1 i B_2 doda nova grana e, dobija se ciklus u $G' = G \cup e$. Ova dva bloka i svi drugi na putu od B_1 do B_2 u $T(G)$ spajaju se u novi blok B'. Ako je B' i sam privezak, grana e je neprofitabilna, a u suprotnom

profitabilna. Neprofitabilne grane je bitno izbeći. Sa tim ciljem, ustanovljen je uslov za proveru profitabilnosti, kojim se tvrdi da je spojna grana čvorova b_1 i b_2 u $T(G)$ profitabilna ako i samo ako sadrži:

- dva čvora stepena bar 3 ili
- jedan b-čvor stepena bar 4.

.

Blok je blokirajući ako ima bar dva artikulaciona čvora takva da bi dodavanje grana između njih narušilo planarnost grafa. Neka su to čvorovi c_1 i l_2. U tom slučaju nije moguće dodati granu između dva priveska koja su povezana putanjom koja prolazi kroz njih. Ako graf sadrži blokirajuće blokove, ponekad se moraju dodati grane između privezaka i blokova koji nisu privesci. Takve grane nazivaju se dodatnim. Ako privezak p u originalnom grafu ne može biti spojen profitabilnom granom, ili može, ali po cenu gubitka profitabilne grane na drugom mestu koja bi postojala u optimalnom rešenju, grana dodeljena datom privesku, e_p se naziva jeftinom. U suprotnom, ako grana nije ni profitabilna ni jeftina, naziva se skupom. Jefinine spojne grane su ponekad neophodne.

Glavna ideja algoritma sastoji se u nalaženju privezaka takvih da se mogu povezati profitabilnim spojnim granama. U svakom koraku dodaju se nove grane između privezaka, uz ažuriranje grafa repreyentovanog prethodno konstruisanim BC-stablom.

Dalje, algoritam uvodi i pojmove snopa privezaka kao i labela, koji imaju značajnu ulogu u njegovoj implementaciji. Naime, ako je P skup čvorova koji su privesci u $T(G)$, $B \subset P$ je snop privezaka ako zadovoljava sledeće uslove:

- Svaki par privezaka p_1 i $p_2 \in B$ se može spojiti bez gubitka planarnosti.
- Za svaki par p_1 i $p_2 \in B$ dodavanje grane između njih vodi do novog priveska u $T(G) \cup e$.
- Svaki skup B je maksimalan prema prethodnim uslovima.

Snop privezaka sa svojim roditeljima naziva se labela. Ako je roditelj neke labele c-čvor, i labela je c-labela, u supronom je u pitanju b-labela. Veličina labele u oznaci $L(l)$ definiše se kao broj privezaka u snopu. Za dve labele l_1 i l_2, takve da je l_1 veća od l_2, kaže se da je l_1 planarna prema l_2 ako je moguće spojiti sve priveske u l_2 sa $L(l_2)$ privezaka u l_1 bez gubitka planarnosti. Kako dodavanje novih grana može izazvati neplanarnost određenih parova labela, u algoritmu se ne vrši direktno spajanje privezaka, nego se pokušavaju spojiti čitavi snopovi.

Imajući prethodno opisane pojmove i ideje koji se koriste, mogu se uočiti sledeći koraci koji sačinjavaju dati algoritam za planarnu augmentaciju:

1. Za ulazni graf G konstruiše se BC-stablo $T = T(G)$
2. Odredi se lista svih privezaka u $T(G)$, kao i lista svih labela, za koje se ujedno određuje i koji čvorovi stabla su njihovi roditelji. Uslovi koje čvor stabla treba da zadovolji da bi bio roditelj određenoj labeli su nešto kompleksniji, te neće biti navedeni, ali se mogu naći u [155].
3. Dokle god u stablu ima c-čvorova, nalazi se trenutno najveća labela, l_1, i traži se l_2 prema kojoj je l_1 planarna.
4. Ako se ne može naći labela l_2 koja zadovoljava spomenuti uslov, radi se sledeće:

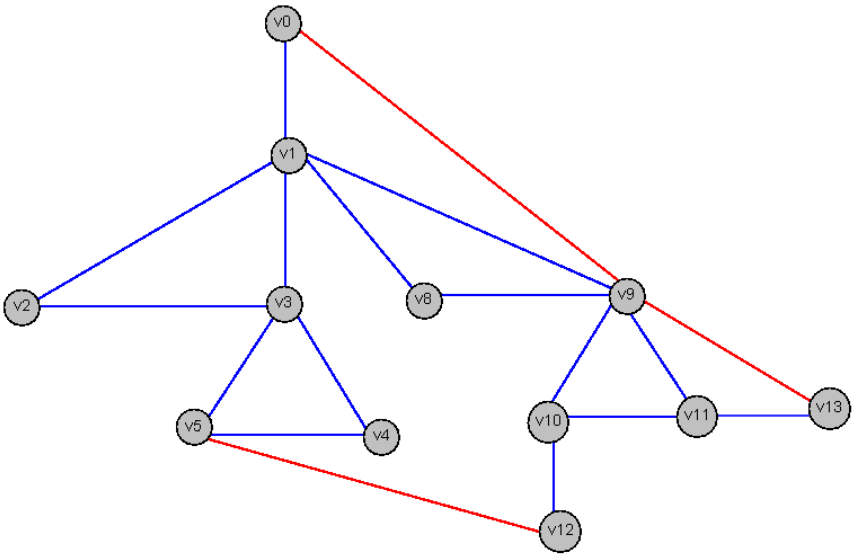
- Dodaje se $L(l_1)$ grana između prvog priveska u snopu i svih ostalih, čime se dobija novi privezak p', čija je jedina artikulaciona tačka c_1.
- Za novi privezak se nalazi roditelj, v'_p.
- Grafu G se dodaje nova grana između čvora spojenog sa v'_p koji ne pripada blokovima na putanji od p' do v'_p i čvora u p' koji je spojen sa c_1.

5. Ako se nađe labela l_2, svi privesci p_2^j labele l_2 povezuju se privescima p_1^j labele l_1 u $T(G)$ za $j = 1, 2, ...L(l_2)$. Preciznije, za svaku veznu granu (p_1^j, p_2^j) biraju se čvorovi u G koji pripadaju privescima p_1^j i p_2^j i spojeni su sa jedinim artikulacionim čvorom u p_1^j i p_2^j. Time se kreira novi blok p'.

6. Ažuriraju se liste labela i privezaka. Ako su dužine labela l_1 i l_2 iste, l_1 se uklanja iz liste labela.

Za proveru planarnosti koja je potrebna u koraku određivanja labele l_2 može se koristiti bilo koji algoritam za ispitivanje planarnosti, ali se preporučuje neki iz klase inkrementalnih.

Na slici 4.19 prikazan je primer planarne augmentacije. Plave grane su deo originalnog grafa, dok crvene predstavljaju rezultat primene opisanog algoritma. Odnosno, grane koje je početnom grafu potrebno dodati da bi postao bipovezan. Sa crteža je očigledno da dodavanje grana nije narušilo planarnost, a može se i spomenuti da bi uklanjanjem bilo koje od njih graf izgubio osobinu bipovezanosti. Autori tvrde da se u velikoj većini slučajeva dobija optimalno rešenje, a u ostalim do tri dodatne grane.



Slika 4.19: Primer planarne augmentacije

4.6 Simetrija grafova

Simetrija je jedan od estetskih kriterijuma koga pojedini autori smatraju veoma važnim. Kao što je već spomenuto, simetrija grafa G je povezana sa njegovim automorfizmima. Automorfizam se definiše kao izomorfizam na samog sebe. Izomorfizam grafova G i H definiše se kao bijekcija između čvorova data dva grafa koja čuva susednost. Odnosno,

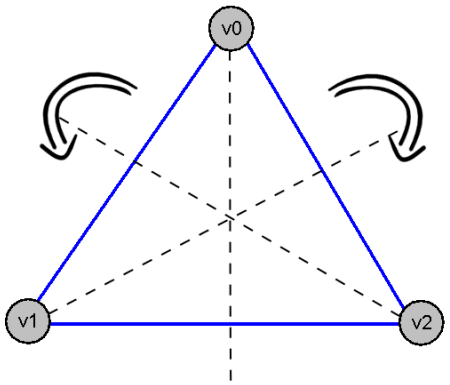
kao preslikavanje $f : V(G) \to V(H)$ takvo da su bilo koja dva čvora u i v grafa G spojena u G ako i samo ako su $f(u)$ i $f(v)$ spojeni u H. Dakle, u pitanju je bijekcija koja očuvava veze. Bijekcija koja mapira jedan graf na samog sebe je, dakle, automorfizam.

Automorfizmi grafa se veoma često predstavljaju permutacijama. Naime, automorfizam grafa G se može definisati i kao permutacija $\alpha : V(G) \to V(G)$ takva da je $(\alpha(u), \alpha(v)) \in E(G) \equiv (u, v) \in E(G)$. Automorfizmi grafova poseduju sledeće bitne osobine:

- Ako su α i β dva automorfizma grafa G, tada je i kompozicija $\alpha \circ \beta$ takođe automorfizam od G.
- Identitet je uvek automorfizam, koji fiksira svaki čvor.
- Ako je α automorfizam grafa G, tada je i α^{-1} takođe automorfizam datog grafa.

Skup svih automorfizama grafa G formira grupu koju zovemo grupom automorfizama grafa G u oznaci $aut(G)$ [39].

Na slici 4.20 prikazan je jedan jednostavan graf sa šest automorfizama: tri refleksije, dve rotacije (za 120° u smeru i suprotno od smera kazaljke na satu) i identitetom, koga imaju svi grafovi.



Slika 4.20: Graf i njegove simetrije

Dakle, graf poseduje sledeće automorfizme:

- Identitet $\epsilon = \begin{pmatrix} v_0 & v_1 & v_2 \\ v_0 & v_1 & v_2 \end{pmatrix} = (v_0)(v_1)(v_2)$
- Refleksija 1 $\alpha_1 = \begin{pmatrix} v_0 & v_1 & v_2 \\ v_0 & v_2 & v_1 \end{pmatrix} = (v_0)(v_1 v_2)$
- Refleksija 2 $\alpha_2 = \begin{pmatrix} v_0 & v_1 & v_2 \\ v_2 & v_1 & v_0 \end{pmatrix} = (v_1)(v_0 v_2$
- Refleksija 3 $\alpha_3 = \begin{pmatrix} v_0 & v_1 & v_2 \\ v_1 & v_0 & v_2 \end{pmatrix} = (v_2)(v_0 v_1)$
- Rotacija 1 $r_1 = \begin{pmatrix} v_0 & v_1 & v_2 \\ v_1 & v_2 & v_0 \end{pmatrix} = (v_0 v_1 v_2)$

- Rotacija 2 $r_2 = \begin{pmatrix} v_0 & v_1 & v_2 \\ v_2 & v_0 & v_1 \end{pmatrix} = (v_0 v_2 v_1)$

Notacija oblika $(v_0 v_1 v_2)$ naziva se cikličnom i zbog kratkoće zapisa veoma često se koristi. U ovom primeru navodi da se v_0 preslikava na v_1, v_1 na v_2, a v_2 na v_0. Odatle i naziv ciklična - svaki element se preslikava na sledeći, a poslednji na prvi. Samo, na primer, (v_0) označava da se v_0 preslikava na samog sebe. Konačno, može se spomenuti da je grupa automorfizama za graf sa slike $aut(G) = \{\epsilon, \alpha_1, \alpha_2, \alpha_3, r_1, r_2\}$.

Red automorfizma β je najmanji broj k takav da je β^k identitet. Podskup $B = \{\beta_1, \beta_2, ..., \beta_k\}$ grupe automorfizama A generiše A ako se svaki element u A može napisati kao proivzvod elemenata iz B. Grupa A se onda označava kao $< \beta_1, \beta_2, ..., \beta_k >$.

Trenutno najefikasnija implementacija testiranje izomorfnosti i nalaženja automorfizama grafa u opštem slučaju su programi *nauty* i *Traces* [156]. Algoritam koji implementiraju osmislio je Brendan Makej još 1981. godine [157]. Implementacija je realizovana na programskom jeziku *C*. Kako nisu nađene stabilne implementacije algoritma na programskom jeziku *Java*, on je implementiran u programskom rešenju i u nastavku će biti prezentovane njegove osnovne ideje.

4.6.1 Makejev algoritam za kanoničko označavanje grafa i nalaženje grupe automorfizama

Bez umanjenja opštosti, može se pretpostaviti da svi razmatrani grafovi imaju skup čvorova $[n] = \{1, 2, ...n\}$. Takođe, u nastavku će biti razmatrani samo jednostavni neorijentisani grafovi. Osnovna ideja algoritma zasniva se na izboru kanoničke reprezentacije iz svake klase izomorfizama grafova nad $[n]$, a testiranje izomorfizama se svodi na proveru jednakosti kanoničkih izomorfa. Izomorf grafa G je graf sa skupom čvorova $[n]$ izomorfan sa G. Kanonička funkcija dodeljuje svakom grafu G izomorf $C(G)$ takav da za H izomorfno sa G važi $C(H) = C(G)$. $C(G)$ je kanonički izomorf grafa G. $C(G)$ se naziva i kanoničkom oznakom.

Jedan način da se definiše kanonička funkcija je da se specificira potpuno uređenje u odnosu na $\leq$ nad grafovima nad $[n]$. $C_{\leq}(G)$ je $\leq$-najveći graf u klasi izomorfizama grafa G. Glavni problem je nalaženje odgovarajućeg uređenja na način koji nije previše računarski zahtevan.

Makej polazi od ideje korišćenja informacija koje su sadržane u samom grafu. Naime, početna tačka je sortiranje čvorova u rastućem redosledu na osnovu njihovog stepena. Ovo samo po sebi nije dovoljno za nalaženje kononičnog izomorfizma, ali se ove lokalne informacije mogu propagirati po grafu [158].

Propagacija informacija o stepenima povezana je sa pojmom uređene particije. Uređena particija π od $[n]$ definiše se kao niz $(V_1, V_2, ..., V_r)$ nepraznih podskupova skupa $[n]$ takvih da je $\{V_1, V_2, ..., V_r\}$ particija od $[n]$. Particija skupa je grupisanje njegovih elemenata u neprazne skupove, na takav način da se svaki element uključi u tačno jedan podskup polaznog skupa. Podskupovi $V_1, V_2, ..., V_r$ nazivaju se delovima particije π. Trivijalni deo sadži samo jedan element, dok se diskretna particija definiše kao particija koja se sastoji od isključivo trivijalnih delova. Jedinična particija sa druge strane ima samo jedan element, $[n]$ i označava se sa μ.

Za dve uređene particije π_1 i π_2 kaže se da je π_1 finija od π_2, ili da je π_2 grublja od π_1 ako:

- svaki deo V_i particije π_1 je sadržan u nekom delu W_k particije π_2,
- za delove V_i i V_j particije π_1, gde je $i \leq j$ i delove W_k i W_l particije π_2 za koje važi $V_i \subset W_k$ i $V_j \subset W_l$, važi da je $k \leq l$.

Čvorovi u različitim delovima particije π se mogu razlikovati jedan od drugog, kao i čvorovi različitog stepena u istim delovima. Sledeći izazov, dakle, predstavlja razlikovanje čvorova istog dela istog stepena.

Uređena particija $\pi = (V_1, V_2, ..., V_r)$ skupa $[n]$ je ujednačena uređena particija ako za svako $1 \leq i, j \leq r$, $v, w \in V_i$ važi $deg(v, V_j) = deg(w, V_j)$. Ujednačena particija τ je najgrublje profinjenje particije π ako je finija od π i ne postoji ujednačena uređena particija finija od π i striktno grublja od nje. Ovakve particije nisu jedinstvene, iako naziv sugeriše suprotno.

Neka $deg(v, V_i)$ odgovara broju veza čvora v sa čvorovima koji pripadaju V_1. Neka je dalje $\pi = (V_1, V_2, ..., V_r)$ neujednačena uređena particija. Deo V_j razbija deo V_i ukoliko postoje dva čvora $v, w \in V_i$, takvi da je $deg(v, V_j) \neq deg(w, V_j)$. Razbijanjem V_i sa V_j dobija se uređena particija $(X_1, X_2, ..., X_t)$ od V_i takva da ako $v \in X_k$ i $w \in X_l$, onda je $k < l$ ako i samo ako je $deg(v, V_j) < deg(w, V_j)$. Dakle, $(X_1, X_2, ..., X_t)$ sortira čvorove V_i prema stepenu u V_j.

Imajući prethodno u vidu, Makej uvodi proceduru ujednačenog profinjenja, koja obuhvata sledeće korake:

1. τ se inicijalizuje na uređenu particiju π koja je ulaz u proceduru.
2. Za $\tau = (V_1, V_2, ..., V_r)$ određuje se skup parova $B = \{(i, j),\ V_j \text{ razbija } V_i\}$.
3. Ako je B prazan skup, τ je konačni rezultat $R(\pi)$ i procedura se završava.
4. Inače, uzima se par (i, j) koji je najmanji prema leksikografskom uređenju. V_i se razbija sa V_j, čime se dobija $(X_1, X_2, ...X_t)$. U particiji τ se V_i zamenjuje sa $(X_1, X_2, ...X_t)$. Procedura se vraća na korak 2 sa novom particijom τ.

Po leksikografskom uređenju za parove (a, b) i (c, d) važi $(a, b) < (c, d)$ ako je $a < c$ ili $a = c$ i $b \leq d$.

Neka je dat graf sa slike 4.21. Može se uočiti da:

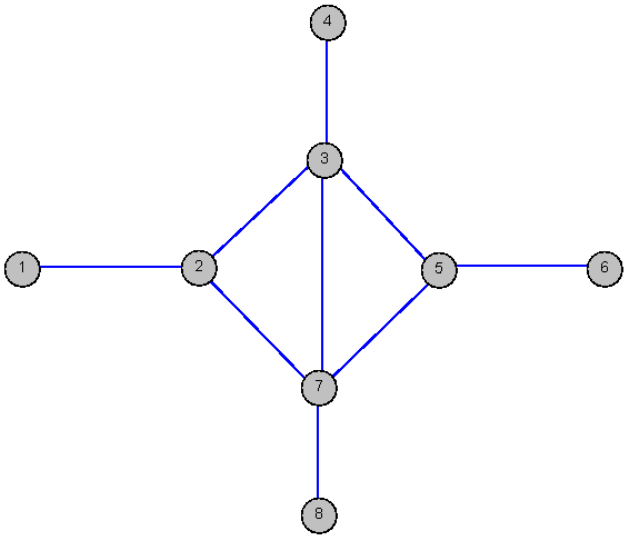
- čvorovi 1, 4, 6 i 8 imaju stepen 1,
- čvorovi 2 i 5 imaju stepen 3,
- čvorovi 3 i 7 imaju stepen 4.

Profinjenje jedinične particije vrši se na sledeći način:

π	B	V_i	V_j
(1 2 3 4 5 6 7 8)	{(1,1)}	(1 2 3 4 5 6 7 8)	(1 2 3 4 5 6 7 8)
(1 4 6 8 | 2 5 | 3 7)	{(1, 2), (1,3)}	(1, 4, 6, 8)	(2, 5)
(4 8 | 1 6 | 2 5 | 3 7)	$\emptyset$		

U prvom koraku profinjenja praktično se čvorovi dele na osnovu stepena u celom grafu. U svakom narednom koraku računa se koliki je stepen čvora jednog dela unutar nekog drugog. Neka je $V_1 = (1\ 4\ 6\ 8)$, $V_2 = (2\ 5)$ i $V_3 = (3\ 7)$. Možemo izračunati sledeće:

$deg(1, V_2) = 1$	$deg(1, V_3) = 0$	$deg(2, V_3) = 2$
$deg(4, V_2) = 0$	$deg(4, V_3) = 1$	$deg(5, V_3) = 2$
$deg(6, V_2) = 1$	$deg(6, V_3) = 0$	
$deg(8, V_2) = 0$	$deg(8, V_3) = 1$	



Slika 4.21: Ulazni graf za Makejev algoritam

$deg(1, V_2) = 1$ jer je čvor 1 povezan sa čvorom 2 iz V_2, $deg(4, V_2) = 0$ jer nije povezan ni sa jednim čvorom itd. Prema definiciji, može se primetiti da V_2 i V_3 razbijaju V_1, dok recimo V_3 ne razbija V_2. Prema leksikografskom uređenju par (1,2) je manji od (1,3), dakle sledeći korak je razbijanje V_1 sa V_2. Dalja rastavljanja nisu moguća, tako da se tu procedura profinjenja završava.

Profinjenjem jedinične particije dobijaju se početne informacije o stepenima čvorova grafa. Ovo je upravo i prvi korak Makejevog algoritma. Međutim, nakon što se dobije ujednačena particija, potrebno je na veštački način omogućiti razlikovanje čvorova. Stoga se sistematski istražuje prostor ujednačenih particija pomoću stabla pretrage.

Neka je π ujednačena uređena particija skupa $[n]$ sa netrivijalnim delom V_i, dok $u \in V_i$. Deljenje π sa u, u oznaci $\pi \perp u$, je ujednačeno profinjenje $R(\pi')$ ujednačene particije $\pi' = (V_1, V_2, ..., \{u\}, V_i/\{u\}, V_{i+1}, ..., V_r)$. Operacija deljenja je ključna za formiranje stabla pretrage, što čini sledeći korak algoritma.

Stablo pretrage $T(G)$ grafa G je korensko stablo čiji čvorovi se formiraju na sledeći način:

1. Koren stabla je profinjenje jedinične uređene particije, koja je početna tekuća particija.

2. Nalazi se prvi netrivijalni deo tekuće particije.

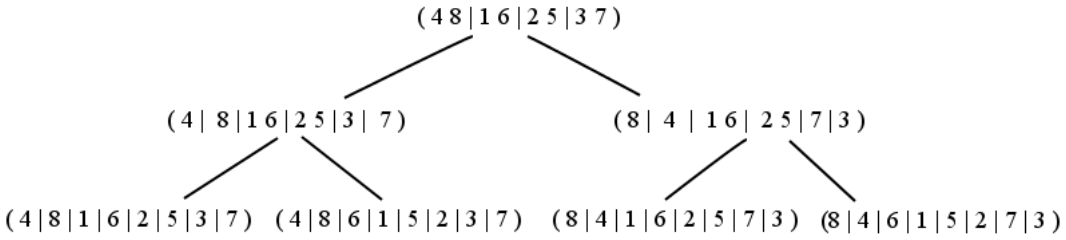
3. Formiraju se deca tekuće particije, deljenjem particije sa po jednim čvorom iz izdvojenog dela.

4. Iterativno se obrađuje svako dete vraćanjem na korak dva. Diskretne uređene particije se ne procesiraju dalje.

5. Postupak se završava kada više nema uređenih particija koje nisu diskretne. Diskretne uređene particije su terminalni čvorovi stabla pretrage.

Stablo pretrage za graf sa slike 4.21 prikazan je na slici 4.22.

Na osnovu diskretnih uređenih particija definišu se permutacije. Naime, ako je $\pi = (V_1, V_2, ..., V_n)$ diskretna uređena particija, onda je $\sigma_\pi \in \Sigma_n$ (grupi svih permutacija



Slika 4.22: Stablo pretrage za graf sa slike 4.21

nad $[n]$) permutacija takva da je $i^{\sigma_\pi} = j$ ako V_j sadrži i. Na primer, neka je $\pi = (4|8|1|6|2|5|3|7)$. σ_π mapira $4 \to 1, 8 \to 2, 1 \to 3$ itd. Za svaki terminalni čvor vezuje se izomorf G^{σ_π} grafa G. Najveći od tih izomorfa po leksikografskom uređenju je kanonički izomorfizam.

Stablo pretrage može biti poprilično veliko ukoliko graf ima veliku grupu automorfizama. U tom slučaju, broj terminalnih čvorova stabla je veći ili jednak veličini date grupe. Iz tog razloga se u algoritam uvodi i korak smanjenja stabla pretrage, u okviru kojeg se deo stabla koji nije još obrađen odbacuje nakon što se otkrije da nijedan od terminalnih čvorova koji se nalaze u tom delu neće proizvesti izomorfizam bolji od nekog već otkrivenog. Više o detaljima ovog postupka, kao i drugim optimizacijama može se naći u [158].

4.7 Provera ostalih osobina

Provera postojanja osobina poput planarnosti, povezanosti, bipovezanosti itd. grafa kao i njegovo rastavljanje na komponente ili nalaženje planarnog utapanja zahtevaju implementaciju kompleksnih algoritama u cilju postizanja dobrih performansi. Međutim, za crtanje grafova i otkrivanje pogodnog algoritma potrebne su i neke dodatne, manje složene analize, koje će biti opisane u ovoj sekciji.

Prva od osobina koje nisu ranije spomenute u tekućem poglavlju jeste 2-obojivost grafa. Odnosno, provera da li je dati graf bigraf. Za graf koji poseduje ovu osobinu važi da se njegovi čvorovi mogu podeliti u dve grupe, odnosno obojiti dvema bojama, tako da su čvorovi jedne boje povezani samo sa čvorovima druge. Provera 2-obojivosti lako se može formulisati na osnovu navedene definicije. Naime, postupak bi uključivao iteraciju kroz sve čvorove grafa uz beleženje obrađenih čvorova i bojenje. Čvor koji se trenutno obrađuje, ukoliko nije već obojen, boji se prvom od dve odabrane boje, te se analiziraju čvorovi sa kojima je povezan. Ukoliko povezani čvor nije obrađen, dodeljuje mu se druga boja, koju nema trenutno posećeni. Ukoliko jeste obrađen, proverava se da li je iste boje kao i tekući. Ako to jeste slučaj, konstatuje se da graf nije bigraf i algoritam se završava. Ako se iteracija završi bez prekida, graf poseduje razmatranu osobinu.

Druga osobina vredna pomena odnosi se na stabla, odnosno, proveru da li je neki graf uopšte stablo i, ako jeste, da li je ono binarno. Korišćenjem bilo algoritma najpre u dubinu bilo najpre u širinu, problem se može rešiti na nimalo komplikovan način. Implementacija se u manjoj meri razlikuje za orijentisane i neorijentisane grafove:

- Ukoliko je graf orijentisan, pronalazi se čvor koji ima samo izlazne grane i proglašava korenom. Ako takav čvor nije jedinstven, graf nije stablo. Potom se sprovodi jedna od spomenutih pretraga, te se ako se naiđe na već posećeni čvor, konstatuje da graf nije stablo, pošto nije acikličan. Ako se pretraga završi a da nisu posećeni svi čvorovi, graf opet nije stablo jer nije povezan. U suprotnom, može se tvrditi da jeste. Za proveru da li je reč o binarnom stablu, proverava se i da li svaki čvor ima ne više od dve izlazne grane.

- Ukoliko graf nije orijentisan, preskače se korak nalaženja korena i proveravaju se cikličnost i povezanost. Stablo je binarno ukoliko ne postoji čvor koji ima više od ukupno tri grane.

Za binarno stablo može se definisati i pojam visinske balansiranosti. Radi se o osobini koju binarno stablo poseduje ukoliko se za svaki čvor visine njegovog levog i desnog stabla ne razlikuju za više od 1. Visina stabla jeste dužina putanje od korena do najdubljeg čvora. Najjednostavniji način za ispitivanje ove osobine obuhvata rekurzivno izračunavanje visine levog i desnog postabla za svaki čvor. Počinje se od korena, te se izračunavanje visine poziva za njegovo levo i desno podstablo. Visina tekućeg postabla jeste maksimum visina podstabala ukorenjenih u njegovom levom i desnom potomku uvećan za 1. Ukoliko se u bilo kojem trenutku primeti da se dve spomenute visine previše razlikuju, konstatuje se da stablo nije balansirano.

Konačno, pojam konture neretko se spominje u raznim algoritmima. Kontura je, kao što je prethodno definisano, graf koji se sastoji iz samo jednog ciklusa. Provera da li je graf prsten verovatno je i najlakša od svih do sada analiziranih. Naime, ukoliko se polazeći od proizvoljnog čvora grafa i prelazeći iz svakog čvora u njemu susedni koji nije dotle posećen stigne opet do početnog, a da se pri tome prošlo kroz sve grane i posetili svi čvorovi tačno jednom, graf je prsten.

Poglavlje 5

Implementacija algoritama za automatsko raspoređivanje

U ovoj sekciji biće prikazani implementirani algoritmi za raspoređivanje elemenata grafova. Kao i kod algoritama za analizu, i prilikom crtanja grafova bitno je dobijanje dobrih performansi. Iako fokus nije stavljen na izuzetno velike grafove i algoritme optimizovane za rad sa njima, mogućnost crtanja grafova od nekoliko stotina do nekoliko hiljada čvorova u prihvatljivom vremenu svakako jeste poželjna. Osim toga, od velikog značaja je i estetika rezultujućeg crteža. Naime, raspoređivanje čvorova na proizvoljne pozicije zahteva veoma malo vremena, ali je kvalitet takvog crteža kranje upitan. Cilj algoritama je, dakle, zadovoljiti jedan ili više estetskih kriterijuma, ali bez eskalacije potrebnog vremena. Neki algoritmi su osmišljeni samo za rad sa manjim grafovima, ali njihova upotrebna vrednost svakako nije zanemarljiva, ukoliko takve grafove raspoređuju na zadovoljavajući način.

5.1 Odabir algoritama za implementaciju

Pregledom postojećih biblioteka otvorenog koda koje dozvoljavaju korišćenje u okviru drugih projekata uočeno je da su veoma dobro podržane klase crtanja stabala i silom usmerena. Takođe je i primećeno da je dostupna veoma dobra implementacija hijerarhijskog raspoređivanja. Međutim, pojedine klase koje imaju nezanemarljivu praktičnu primenu nisu pokrivene. Pre svih, može se izdvojiti planarno pravolinijsko crtanje, kako se mnogi autori slažu sa konstatacijom da je minimizacija, ili potpuno izbegavanje, broja preseka grana jedan od najbitnijih, ako ne i najbitniji estetski kriterijum. Dodatno, podtip ove klase, ortogonalno crtanje, možda i iznenađujuće, nije fokus nijedne od analiziranih biblioteka za programski jezik Java. Dakle, u projektnom rešenju implementirano je više pripadnika spomenute klase, počevši od relativno jednostavnog Tutovog, preko izuzetno kompleksnog algoritma za konveksno planarno crtanje, do ortogonalnog raspoređivanja uz oslonac na pojam vidljivih reprezentacija.

Još jedna klasa za koju je uočen nedostatak kvalitetnih implementacija jeste cirkularna ili kružna. Naime, nađene implementacije ne vode računa o minimizaciji broja preseka grana, što je adresirano u projektnom rešenju. Dodatno, obrađeno je i izračunavanje optimalnog poluprečnika, koje uzima u obzir veličine čvorova, preuzimajući tu odgovornost od korisnika.

Konačno, potpuno zanemarena klasa algoritma za raspoređivanje, koja optimizuje jedan od estetskih kriterijuma koji pojednini autori smatraju veoma bitnim, jeste simetrično crtanje. Algoritmi iz ove klase bave se generisanjem crteža na kojima se

uočavaju jedna ili više simetrija i često su veoma kompleksni. U datom trenutku implementiran je jedan agloritam, koji kreira crtež takav da jasno prikaže simetriju indukovanu odabranim automorfizmom, te bi dalji razvoj bio usmeren ka implementaciji naprednijih opcija.

Takođe, autori algoritama za crtanje grafova pojedine korake osmišljenih postupaka ne definišu dovoljno precizno, iako njihova realizacija nije uvek trivijalna. Prilikom implementacije takvih algoritama osmišljeni su načini rešavanja spomenutih delova, koji ne narušavaju originalnu efikasnost.

Pre ulaženja u detalje implementiranih algoritama, može se napomenuti da su neki od njih posebno dizajnirani za crtanje orijentisanih grafova, vodeći računa o ujednačenom toku. Međutim, ukoliko spomenuti estetski kriterijum nije bitan, orijentisani graf može se nacrtati i pomoću bilo kojeg algortima namenjenog neorijentisanima, ukoliko se zanemari usmerenje grana. Kako ujednačeni tok nije težnja nijednog od izdvojenih algoritama, u njihovim implementacijama ne vodi se posebno računa o tipu grafa.

5.2 Kružno raspoređivanje sa minimizacijom broja preseka grana

Kružno raspoređivanje elemenata grafa vrši njihovo postavljanje na obod jednog određenog kruga. Algoritam je originalna implementacija u okviru projektnog rešenja i definisan je tako da kao ulazni parametar prima željeno rastojanje između susednih čvorova, na osnovu kojeg se može odrediti poluprečnik kružnice na koju će čvorovi biti postavljeni. Ova kalkulacija uzima u obzir veličinu čvorova, kako bi se izbeglo njihovo preklapanje.

Kružno rasporediti elemente grafa na pomenuti način ne predstavlja poseban problem. Međutim, ako se ne vodi računa o poretku čvorova, broj preseka grana može biti veoma velik, čak i u slučaju planarnih grafova. Iz tog razloga, može se primeniti algoritam Siks-Tolis [67], koji izračunava raspored čvorova takav da broj preseka grana bude mali, ili jednak nuli. Međutim, algoritam je dizajniran za primenu nad bipovezanim grafovima (ili njihovim bipovezanim komponentama), tako da ne može uvek biti izvršen, bar ne sa garancijom dobrog rezultata.

Ceo postupak kružnog raspoređivanja obuhvata sledeće korake:

1. Nalaženje uređenja čvorova grafa kojim se smanjuje broj preseka grana primenom algoritma Siks-Tolis, ukoliko je to moguće.

2. Nalaženje poluprečnika kružnice na kojoj će čvorovi biti raspoređeni.

3. Određivanje pozicija čvorova znajući njihov redosled i poluprečnik.

5.2.1 Nalaženje poretka čvorova radi smanjenja broja preseka grana

Algoritam Siks-Tolis teži postavljanju grana prema perifieriji unutrašnjosti kruga, grupišući povezane čvorove blizu jedne drugima. Osnovna ideja algoritma sastoji se u pažljivom uklanjanju pojedinih grana i nalaženju redosleda čvorova putem pretrage najpre u dubinu za rezultujući graf. U cilju uklanjanja odgovarajućih grana, ovaj algoritam definiše pojmove prednjeg i centralnog talasnog čvora. Prednjim talasnim čvorom proglašava se sused poslednje procesiranog, a centralnim čvor koji je susedan nekom drugom već posećenom čvoru. Algoritam u prvom koraku obrađuje neki od

čvorova najnižeg stepena, te posećuje prednji i centralni talasni čvor, ako su i oni tog stepena. Ako nijedan od centralnih i prednjih talasnih čvorova nema odgovarajući stepen, algoritam bira neki drugi čvor najmanjeg stepena. Obilazak se potom nastavlja od izabranog čvora i prethodnog prednjeg i centralnog talasnog. Ovakav obilazak autori nazivaju talasnim, odakle potiče i naziv deifnisanih bitnih čvorova.

Osim pomenutih čvorova, uvodi se i pojam grane para, koja spaja dva čvora koji imaju bar jednog zajedničkog suseda. Ako međusobno povezani čvorovi v i w imaju zajedničkog suseda u, kaže se da su v i w upareni čvorom u, odnosno, da u uspostavlja granu para (v, w). Tri spomenuta čvora praktično formiraju trougao. Grane para se uklanjaju pre nego što se pređe na korak pretrage prvi u dubinu. Ako u grafu nema dovoljno grana parova, dodaju se nove grane koje se nazivaju trougaonim granama. One se takođe uklanjaju pre pretrage najpre u dubinu. Na kraju posete čvoru se on i sve grane koje ga sadrže se uklanjaju iz grafa.

Poslednji korak algoritma obuhvata pretragu najpre u dubinu kojom se traži najduža putanja u DFS stablu i čvorovi koji joj pripadaju se postavljaju na kružnicu. Preostali čvorovi se potom dodaju u utapanje. Za svaki od tih čvorova posećuju se njegovi susedi i ukoliko se ustanovi da su dva suseda jedan pored drugog na kružnici, čvor se stavlja između njih. Ako ima više takvih parova, nasumično se bira jedan. Ako, sa druge strane, nema nijednog odgovarajućeg para, čvor se ubacuje pored jednog svog suseda koji je na kružnici. Ako nema ni takvog suseda, ubacuje se na proizvoljno mesto.

Imajući navedeno u vidu, celokupan algoritam se odvija na sledeći način:

1. Čvorovi se sortiraju prema stepenu u uzlaznom redosledu.
2. Dok graf poseduje bar tri čvora:
   (a) Ako neki od prednjih talasnih čvorova ima najmanji stepen, on se proglašava tekućim. Ako to nije slučaj, proverava se da li neki od centralnih talasnih čvorova ima najmanji stepen, te se on proglašava tekućim. Ako tekući čvor nije nađen, uzima se neki od čvorova najmanjeg stepena.
   (b) Posećuje se svaki susedni čvor tekućeg čvora. Za svaka dva čvora se proverava da li između njih postoji grana para. Ako postoji, evidentira se. Ako ne postoji, dodaje se trougaona grana između tekućeg para suseda.
   (c) Ažuriraju se stepeni susednih čvorova tekućeg čvora
   (d) Tekući čvor i grane koje ga sadrže uklanjaju se.
3. Graf G se vraća u prvobitno stanje, te se uklanjaju sve trougaone grane. Takođe, uklanjaju se i grane para.
4. Sprovodi se pretraga napre u dubinu i nalazi se najduža putanja.
5. Na kružnicu se postavljaju čvorovi najduže putanje, te se ostali čvorovi ubacuju na prethodno opisani način.

Za bipovezani graf koji ima planaran crtež, ovaj algoritam ga generiše za linearno vreme O(n), a generalno nalazi crtež bilo kog bipovezanog grafa za O(m), gde je n broj čvorova, a m broj grana grafa.

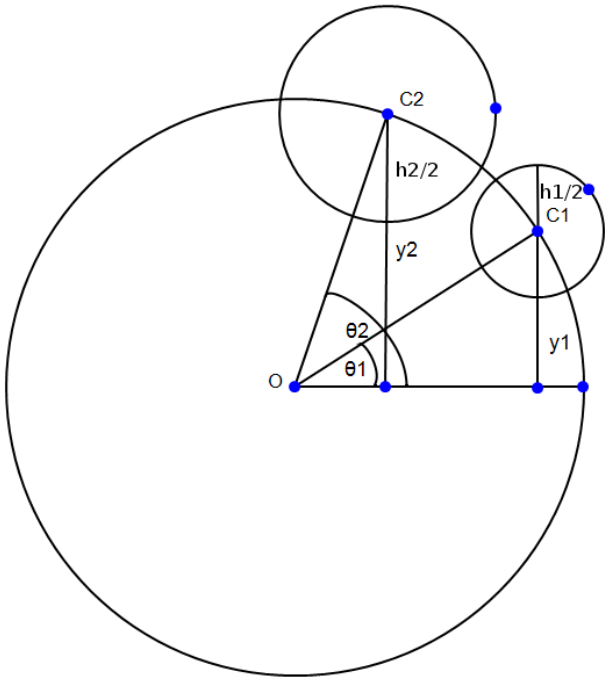
5.2.2 Nalaženje poluprečnika kruga i pozicije čvorova

Implementirano kružno raspoređivanje za cilj ima postavljanje čvorova grafa na kružnicu tako da rastojanje između njih bude u skladu sa korisnikovim željama, a da pri tome

ne dođe do preklapanja između susednih čvorova. Ulazni parametar ovog koraka, osim zadate distance, jeste i lista čvorova grafa, u rasporedu u kojem treba da budu nacrtani. Takođe se zahteva i da čvorovi budu ravnomerno raspoređeni po kružnici. Odnosno, centri čvorova treba da se nalaze na samoj kružnici, a dužina kružnog luka između dva susedna centra treba da bude jednaka za sve parove. To znači i da je ugao koji formiraju centri čvorova para i centar kruga isti za sve. Neka je to ugao θ. Bez gubitka opštosti, može se reći da je ugao između prva dva čvora i centra baš θ, prvog i trećeg čvora i centra $2 * \theta$ itd.

Bazična ideja nalaženja poluprečnika sastoji se u izračunavanju minimalne vrednosti ovog parametra tako da se za proizvoljni par susednih čvorova ne dođe ni do njihovog preseka ni po vertikalnoj ni po horizontalnoj osi, te da se ispoštuje željeno rastojanje koje korisnik zada. Dakle, rastojanje je samo jedan od faktora, a ne jedini parametar koji određuje poluprečnik. Naime, čak i ako je njegova vrednost nula, do preklapanja susednih čvorova neće doći, ali će najbliži čvorovi (par čvorova koji su u zbiru najvećih dimenzija) biti skoro spojeni. Naravno, što je to zadato rastojanje veće, to je i poluprečnik veći. Poluprečnik kojim bi se izbeglo preklapanje posebno se računa za svaki par suseda i po x i po y-osi, te se kao konačan poluprečnik za taj par uzima manja od tih vrednosti. Naime, ako su čvorovi na dovoljnom rastojanju po bilo kojoj osi, onda do preklapanja svakako ne može doći.

Neka su h_1 i θ_1 visina i ugao prvog čvora para, h_2 i θ_2 drugog, kao što je obeleženo na slici 5.1. Za i-ti čvor, vrednost ugla θ računa se kao: $\frac{2\pi}{n}(i-1)$, gde je n broj čvorova grafa. Dakle, prvi čvor ima ugao 0, drugi $\frac{2 * \pi}{n}$ itd.



Slika 5.1: Promenljive koje učestvuju u računanju poluprečnika kruga

Može se primetiti da je krajnje donja y-koordinata višeg čvora $y_2 - \frac{h_2}{2}$, a gornja nižeg čvora $y_1 + \frac{h_1}{2}$. Donja granica gornjeg čvora mora biti iznad gornje drugog da bi se izbeglo preklapanje po y-osi. Prema tome, potrebna veličina poluprečnika po y-osi

može se odrediti na sledeći način:

$$y_2 - \frac{h_2}{2} - y_1 - \frac{h_1}{2} \geq 0$$
$$y_2 - y_1 \geq \frac{h_2}{2} + \frac{h_1}{2}$$
$$y_2 = r_y \sin\theta_2, y_1 = r_y * \sin\theta_1$$
$$2r_y(\sin\theta_2 - \sin\theta_1) \geq h_2 + h_1$$
$$r_y \geq \frac{h_2 + h_1}{2(\sin\theta_2 - \sin\theta_1)}$$

Konačno, dodaje se i parametar razdaljine t, te se razlika sinusa obuhvata aposlutnom vrednošću pošto može biti negativna, čime je konačan oblik formule:

$$r_y = \frac{h_2 + h_1 + t}{2|\sin\theta_2 - \sin\theta_1|}$$

Slično se nalazi i poluprečnik po x osi, samo se koriste širine (w_1 i w_2) i x koordinate čvorova:

$$r_x = \frac{w_2 + w_1 + t}{2|\cos\theta_2 - \cos\theta_1|}$$

Poluprečnik koji se dobija za dati par je, dakle, minimum nađenih r_x i r_y. Poluprečnik kruga na kraju se nalazi kao najveći od svih poluprečnika parova. Preklapanja između čvorova raspoređenih u krug nađenog poluprečnika može biti jedino ako se preklope nesusedni čvorovi. Takva situacija može se desiti ako su čvorovi takvi da im je visina mnogo veća od širine ili obrnuto. To bi se moglo sprečiti ako bi se za poluprečnik para uzeo veći od r_y i r_x. Tada bi krugovi bili dosta veći i više bi se odstupalo od specifikovane razdaljine koju je korisnik zadao.

Poznajući poluprečnik kruga i koordinate centra (koje algoritam prima kao ulazne parametre), pozicije svih čvorova se poprilično jednostavno mogu odrediti na osnovu sledeće formule:

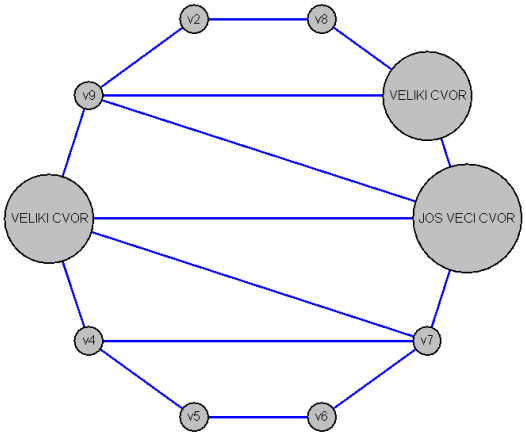
$$x_i = o_x + r\cos\theta(i-1)$$
$$y_i = o_y + r\sin\theta(i-1)$$
$$\theta = \frac{2\pi}{n}$$

x_i je x, a y_i y-koordinata i-tog čvora, a o_x i o_y koordinate centra kruga.

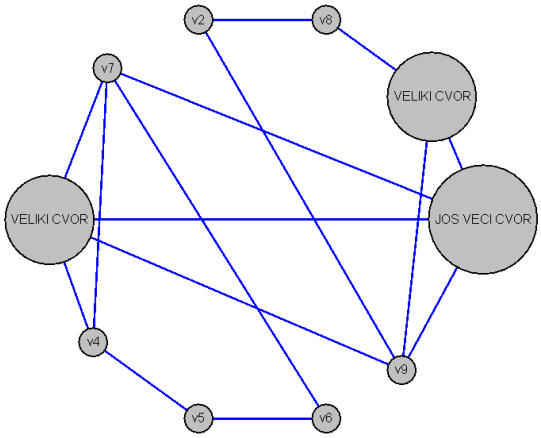
5.2.3 Primer

Na slici 5.2 prikazan je crtež bipovezanog grafa kreiran pomoću kružnog algoritma sa minimizacijom broja preseka grana. Važnost nalaženja odgovarajućeg rasporeda čvorova može se uočiti poređenjem te slike sa slikom 5.3, gde su čvorovi v_7 i v_9 zamenili mesta, čime se umesto planarnog crteža, dobio crtež sa čak 8 preseka grana. Takođe, može se primetiti da je poluprečnik kruga zaista određen uzimajući u obzir veličinu čvorova, te se veći čvorovi ne preklapaju, kao i da su čvorovi raspoređeni tako da su rastojanja između centara čvorova jednaka za svaki par suseda. Iako su u primeru čvorovi i sami

kružnog oblika, to ne mora biti slučaj. Oblik čvora nije bitan dokle god su poznate njegove dimenzije.



Slika 5.2: Kružni crtež bipovezanog grafa



Slika 5.3: Kružni crtež bipovezanog grafa sa prethodne slike, gde su zamenjene pozicije čvorova v_7 i v_9

5.3 Crtanje grafova upotrebom Tutovog algoritma za utapanje planarnih tripovezanih grafova

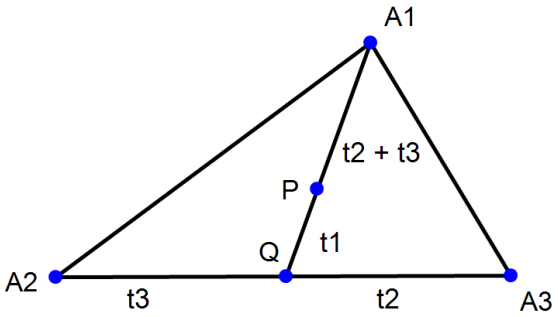
Tutov algoritam za nalaženje crteža tripovezanih planarnih grafova smatra se prvim silom usmerenim algoritmom. Nastao 1963. godine, pune dve decenije pre nego što su algoritmi tog tipa počeli dobijati na popularnosti. Algoritam proizvodi pravolinijske crteže i garantuje planarnost. Kao ulaz dobija skup čvorova koji će se nalaziti na spoljašnjem licu, te izračunava pozicije ostalih čvorova.

5.3.1 Tutov algoritam

Tut konstruiše utapanje tripovezanog grafa koje je zasnovano na baricentričnim koordinatama. Baricentrični koordinatni sistem je koordinatni sistem u kojem su lokacije tačaka simpleksa određene kao centri mase, ili baricentri, masa postavljenih

u njegova temena. Mase ne moraju biti jednake. Simpleks je generalizacija trougla u proizvoljnom broju dimenzija, pri čemu k-dimenzioni simpleks ima $k+1$ temena. Dakle, trostrana piramida je 3-simpleks. Same baricentrične koordinate su trojke brojeva (t_1, t_2, t_3) koje odgovaraju masama stavljenim u temena referentnog trougla $\triangle A_1A_2A_3$. Te mase određuju tačku P koja je geometrijski centar triju masa. Otkrio ih je nemački matematičar August Mobius (August Ferdinand Möbius) 1827. godine [159].

Nalaženje baricentričnih koordinata proizvoljne tačke P obuhvata nalaženje t_2 i t_3 iz tačke Q, koja se nalazi u preseku linija A_1P i A_2A_3, kao što je označeno na slici 5.4. Broj t_1 se zatim nalazi kao masa u A_1 koja će izbalansirati masu $t_2 + t_3$ u Q, čime P postaje geometrijski centroid masa [160].



Slika 5.4: Nalaženje baricentričnih koordinata

U cilju predstavljanja Tutovog algoritma, potrebno je prvo definisati pojam periferalnog poligona. Periferalni poligon J u grafu G je jednostavni ciklus povezanog grafa takav da za svake dve grane e_1 i $e_2 \in J$ postoji putanja u G koja počinje u e_1 i završava se u e_2, pri čemu nijedan unutrašnji čvor putanje ne pripada J [43]. Periferalni poligoni se danas nazivaju periferalnim ciklusima, ali će se u nastavku koristiti termin poligona, kako je upravo njega upotrebljavao sam Tut. On je uveo nešto drugačiju, ali ekvivalentnu definiciju pomenutog pojma. Dalje se može predstaviti nalaženje baricentričnog mapiranja:

1. Neka je J periferalni poligon tripovezanog planarnog grafa G i neka je dalje skup čvorova G u J označen sa $V(J)$ i neka sadrži $n \geq 3$ elemenata. Neka se potom čvorovi grafa G označe sa $v_1, v_2, ...v_m$, tako da prvih n čvorova pripada poligonu J.

2. Neka je Q geometrijski n-tougao u Euklidskoj ravni, a f 1-1 mapiranje čvorova J na čvorove Q. To znači da je ciklični redosled čvorova J pod f jednak redosledu čvorova n-tougla.

3. Funkcija f se može proširiti tako da obuhvati i preostale čvorove grafa G. Za svako v_i, $n < i \leq m$ definiše se $N(i)$, kao skup svih čvorova G povezanih sa njim.

4. Neka je svako $v_i \in N(i)$ m_j jedinična masa koja se postavlja u tačku $f(v_i)$. Tada je $f(v_i)$ centar mase m_j.

5. Postavlja se koordinatni sistem koji svakoj $f(v_i), 1 \leq i \leq m$ dodeljuje koordinate (v_{ix}, v_{iy}). Potom se definiše matrica $K(G) = C_{ij}$, $1 \leq (i, j) \leq m$ na sledeći način:

$$C(i,j) = \begin{cases} -(\textbf{broj grana između } v_i \textbf{ i } v_j) \\ deg(v_i) \end{cases}$$

6. Baricentrične koordinate v_{ix}, v_{iy} za $n < i \leq m$ nalaze se rešavanjem dva sistema linearnih jednačina:

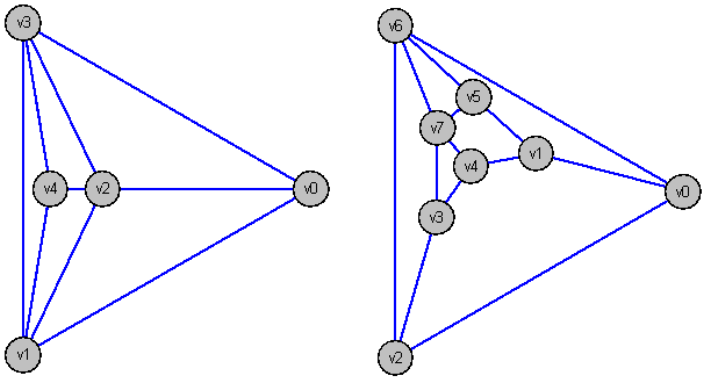
$$\sum_{j=1}^{m} c_{ij} v_{ix} = 0$$
$$\sum_{j=1}^{m} c_{ik} v_{iy} = 0$$

Za $1 \leq j \leq n$ koordinate su već poznate [161]. Jednačine se mogu rešiti primenom poznatog Kramerovog pravila.

5.3.2 Raspoređivanje upotrebom Tutovog algoritma

Tutov algoritam nalazi pozicije preostalih čvorova ako je poznat periferalni poligon i pozicije čvorova koji mu pripadaju. Dakle, potrebno je odrediti tražene ulazne parametre pre pozivanja pomenutog algoritma. Kako se Tutov algoritam bavi trostruko povezanim grafovima, nalaženje periferalnih ciklusa može se poistovetiti sa nalaženjem njegovih lica utapanja. Dakle, jedno od lica nađenih upotrebom algoritama za ispitivanje planarnosti opisanih u prethodnom poglavlju može se proslediti Tutovom algoritmu kao periferalni ciklus. Recimo, to može biti baš nađeno spoljašnje lice koje vraća algoritam Bojer-Mirvold.

Sledeći korak predstavlja određivanje koordinata čvorova periferalnog ciklusa, koje su takođe ulaz u Tutov algoritam. Radi dobijanja što lepšeg crteža, oni se smeštanjem u temena pravilnog n-tougla, gde je n broj čvorova datog periferalnog ciklusa. To praktično znači da se čvorovi kružno raspoređuju upotrebom opisanog kružnog algoritma, samo bez koraka optimizacije preseka. Poluprečnik kruga je ulazni parametar algoritma, te na taj način korisnici mogu definisati dimenzije crteža. Preostali čvorovi se po kalkulaciji pozicija spoljnih (čvorova periferalnog ciklusa) raspoređuju Tutovim postupkom, te se smeštaju unutar pomenutog n-tougla. Time se dobijaju crteži kao na slici 5.5.



Slika 5.5: Primeri grafova nacrtanih Tutovim algoritmom

Tutov algoritam za manje grafove daje poprilično dobre rezultate. Crteži dobijeni na upravo opisani način su konveksni, što znači da je svako lice konveksno. Međutim, ne preporučuje se njegova upotreba za raspoređivanje grafova koji ima preko 100 čvorova, jer rezultujući crtež neće biti pregledan. Već na prikazana dva primera vidi se da su čvorovi uglavnom zbijeni na jednom delu, što nije u skladu sa estetskim kriterijumom

ravnomerne raspodele. Takođe, problematično je izbegavanje preklapanja čvorova grafa ukoliko su njihove dimenzije nešto veće. Naime, poluprečnik kruga bi morao biti veoma velik, čime bi se dodatno naglasio problem neravnomerne raspodele. Još jedan problem Tutovog algoritma je i vreme izvršavanja, pošto rešavanje sistema linearnih jednačina uglavnom zahteva $O(n^3)$ vremena. Dodatno, i sama činjenica da je algoritam razvijen isključivo za tripovezane grafove umanjuje njegovu upotrebnu vrednost.

5.4 Linearni algoritam za konveksno crtanje grafova

Navedeni nedostaci Tutovog algoritma, pogotovo prilikom primene nad iole većim grafovima, inspirisali su druga istraživanja usmerena ka nalaženju efikasnijih algoritama za konveksno crtanje grafova. Jedan od najznačajnijih u ovoj klasi je algoritam Čibe (Norishige Chiba), Onogučija (Kazunori Onoguchi) i Nišizekija (Takao Nishizeki)[79], koji proizvodi konveksan crtež u linearnom vremenu. Ovaj algoritam značajan je ne samo zbog mogućnosti samostalne primene, nego i zbog činjenice da ga pojedini moderniji i još kompleksniji algoritmi za raspoređivanje koriste za crtanje određenih delova grafa [72]. Ovaj algoritam je najsloženiji originalno implementirani u projektnom rešenju. Naime, on uključuje veći broj prethodno opisanih algoritama za analizu (nalaženje putanja, proveru planarnosti, dekompoziciju grafa na tripovezane komponente itd.), te i sam definiše znatan broj koraka koji vode ka konveksnom crtežu. U okviru algoritma na nekoliko mesta navode se relativno složene instrukcije, koje nisu dodatno rasložene, odnosno, gde nije precizno definisano kako realizovati date korake. U ovom poglavlju biće predstavljen dati algoritam, uz posebno izdvajanje nepreciznih koraka i predloga za njihovu implementaciju.

Algoritam se sastoji iz tri dela:

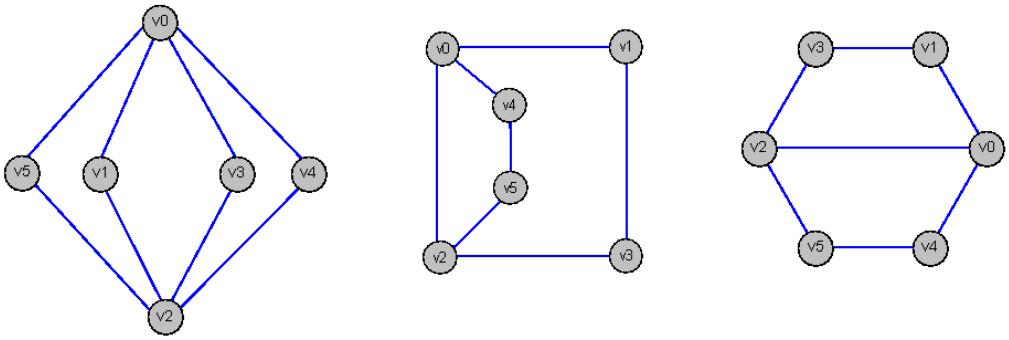
1. provere da li dati bipovezani graf ima konveksni crtež,
2. nalaženja proširivih ciklusa lica ako postoje i
3. konveksnog crtanja.

Tut je pokazao da se može konstruisati konveksni crtež svakog tripovezanog grafa. Međutim, za planarne grafove koji ne poseduju ovaj stepen povezanosti ne može se tvrditi bez posebne provere ni da je pomenuto moguće, ni da nije. Takođe, može se primetiti da za graf koji ima konveksni crtež, on ne može biti konstruisan za svako lice kao spoljno, što se može videti na slici 5.6. Prvi crtež nije konveksan i za takav graf uopšte se i ne može konstruisati crtež koji zadovoljava posmatrani uslov. Drugi crtež takođe nije konveksan, ali se drugačijim izborom ciklusa lica graf može nacrtati na traženi način, što se vidi na trećem crtežu koji prikazuje isti apstraktni graf.

5.4.1 Provera konveksnosti

Pre prelaska na korak kreiranja konveksnog crteža grafa, treba proveriti da li je to uopšte moguće. Takođe, treba i naći odgovarajuće spoljašnje lice. U tu svrhu definiše se proširivost ciklusa lica. Ciklus lica S grafa G je proširiv ukoliko ima proširivi konveksni poligon S^*. Konveksni poligonalni crtež, ili kraće samo konveksni poligon ciklusa lica proširiv je ukoliko postoji konveksni crtež grafa G koji ima taj poligon kao spoljašnji. Proširivih ciklusa lica može biti više, te je algoritam dizajniran sa ciljem da ih sve nađe.

Testiranje konveksnosti uključuje podelu grafa na trostuko povezane komponente upotrebom Hopkroft-Taržan algoritma, kao i nalaženje razdvajajućih parova prvog i



Slika 5.6: Primeri konveksnog i nekonveksnih crteža

drugog tipa i razdelnih komponenti. Ovi pojmovi kao i sam algoritam opisani su u poglavlju 5.2.4. Za testiranje konveksnosti uvodi se, međutim, nekoliko novih pojmova, te će oni biti definisani u nastavku.

Ako je $\{x, y\}$ razdvajajući par grafa G, te se on podeli na razdelne grafove prema $\{x, y\}$, pa se podele i ti grafovi itd. dokle god je podela prema $\{x, y\}$ moguća, na kraju se dobije više grafova koji ne moraju biti tripovezani. Takvi grafovi, koji se dobijaju na kraju podele, nazivaju se $\{x, y\}$-razdelnim komponentama grafa G ukoliko poseduju bar jednu nevirtuelnu granu. Razdelni par $\{x, y\}$ je važan ukoliko su x i y krajnji čvorovi neke virtuelne grane sadržane u tripovezanoj komponenti. Odnosno, $\{x, y\}$ je važan ako postoje dva grafa $G'_1 = (V_1, E'_1)$ i $G'_2 = (V_2, E'_2)$ takvi da:

- $V = V_1 \cup V_2, V_1 \cap V_2 = \{x, y\}$ i $E = E'_1 \cap E'_2, E'_1 \cap E'_2 = \emptyset, |E'_1| \geq 2, |E'_2| \geq 2$ - uslovi da je sam par $\{x, y\}$ čvorova razdvajajući.
- Neki od grafova G'_1 i G'_2 je bipovezan ili je podela grane koja povezuje dva čvora stepena 3 ili većeg.

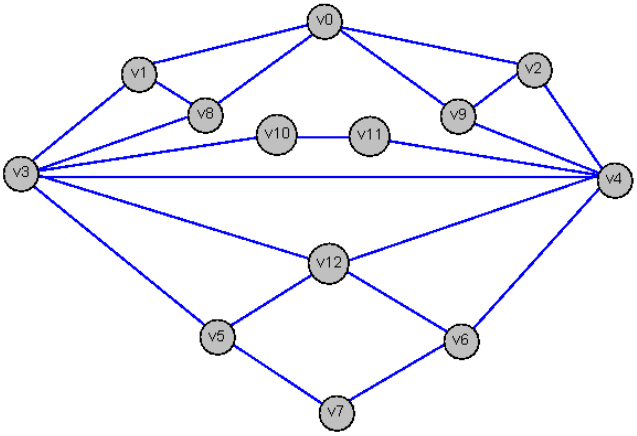
Razdvajajući par je zabranjen ukoliko je važan i za njega postoje bar 4 razdelne komponente ili tačno tri komponente takve da nijedna nije ni prsten ni skup tri grane između dva čvora - spona. Ako planarni graf sadrži bar jedan zabranjeni razdelni čvor, on ne može biti nacrtan konveksno.

Razdelni par $\{x, y\}$ je kritičan ukoliko je bitan i formira tačno tri $\{x, y\}$-razdelne komponente, pri čemu je bar jedna prsten ili spona, ili su tačno dve razdelne komponente takve da nijedna od njih nije prsten. Kod grafa koji nema kritične razdvajajuće parove zapaža se osobina da je svaki ciklus lica grafa proširiv.

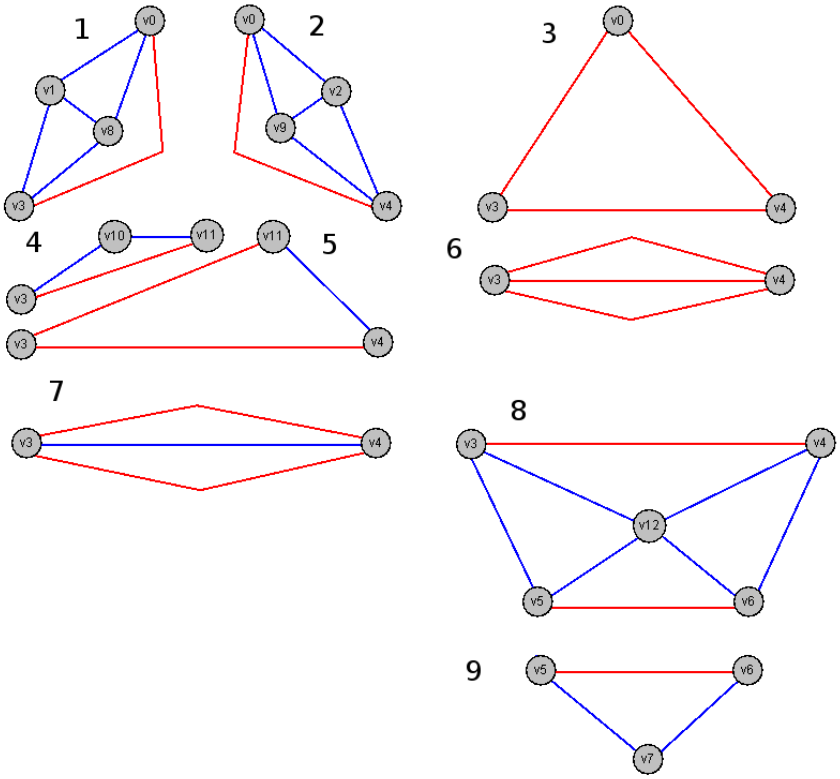
Na slici 5.7 prikazan je graf koji će biti rastavljan na razdelne komponente i za kojeg će biti navedeni svi važni razdvajajući parovi. Graf je pogodan jer sadrži više razdvajajućih parova svih tipova i preuzet je iz [79].

Razdvajajući parovi grafa, dobijeni algoritmom Hopkroft-Taržan jesu: $\{v_0, v_3\}, \{v_0, v_4\}, \{v_3, v_4\}, \{v_3, v_{11}\}, \{v_4, v_{10}\}, \{v_5, v_6\}$. Na slici 5.8 prikazane su razdelne komponente grafa. Crvenom bojom obeležene su virtuelne grane.

Razdelne komponente su redom: (1) tripovezani graf, (2) tripovezani graf, (3) trougao, (4) trougao, (5) trougao,(6) spona (7) spona, (8) tripovezani graf (9) trougao. Razdelne komponente za neki par čvorova dobijaju se spajanjem komponenti po zajedničkim virtuelnim granama, osim onoj koja spaja baš ta dva čvora. Virtuelne grane po kojima se izvrši spajanje ne pripadaju rezultujućim komponentama. Na primer, ako se formiraju razdelne komponente za par $\{v_3, v_4\}$ komponente 4 i 5 se mogu spojiti po zajedničkoj virtuelnoj grani (v_3, v_{11}), a komponente 1, 2 i 3 po granama (v_0, v_3) i (v_0, v_4)



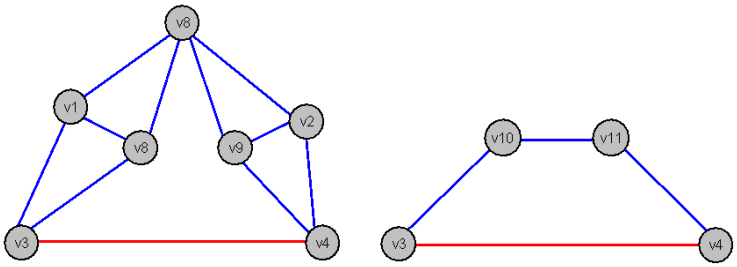
Slika 5.7: Graf koji se rastavlja na razdelne komponente



Slika 5.8: Razdelne komponente grafa

itd. Dakle, ne spajaju se nužno samo dve komponente po jednoj virtuelnoj grani, nego se ovo udružavanje vrši dokle god je moguće. Dakle, po spajanju komponenti 1 i 3 vidi se da se može izvršiti i spoj sa komponentom 2. Dve spomenute komponente mogu se videti na slici 5.9. Za par $\{v_5, v_6\}$ se, recimo, dobijaju samo 2 razdelne komponente: komponenta 9 i komponenta koja predstavlja ostatak grafa, pošto se komponenta 8 spaja sa ostalima jer nigde drugde ne figuriše virtuelna grana (v_5, v_6).

Važni razdvajajući parovi su: $\{v_0, v_3\}, \{v_0, v_4\}, \{v_3, v_4\}$ i $\{v_5, v_6\}$. Kritični parovi su $\{v_0, v_3\}$ i $\{v_0, v_4\}$, a $\{v_3, v_4\}$ je zabranjen.



Slika 5.9: Dve komponente para $\{v_3, v_4\}$

Uvedeni pojmovi koriste se za definisanje konačnog uslova konveksnosti. Neka je G bipovezani planarni graf sa spoljašnjim ciklusom lica S, a S^* strogo konveksni poligon tog lica (svaki čvor iz S je teme poligona S^*). S^* je proširiv ako i samo ako je zadovoljeno sledeće:

1. G nema zabranjenih razdvajajućih parova
2. Za svaki kritični razdvajajućih par $\{x, y\}$ postoji najviše jedna $\{x, y\}$-razdelna komponenta koja ne sadrži nijednu granu iz S. Ta komponenta je ili spona, ako $(x, y) \in E$, ili prsten u suprotnom.

Navedeni uslovi predstavljaju osnovu za ispitivanje mogućnosti konveksnog crtanja. Iz njih se može izvesti još nekoliko tvrdnji:

- Bipovezani graf koji ne sadrži ni kritični ni zabranjeni razdvajajućih par ima konveksni crtež za svaki ciklis lica grafa.
- Tripovezani graf ima konveksni crtež za bilo koji ciklus lica.
- Ako je S ciklus lica bipovezanog grafa koji zadovoljava gorenavedene uslove, onda S sadrži svaki čvor kritičnih razdvajajućih parova grafa.

Konačno, mogu se predstaviti koraci algoritma za konveksno testiranje. Detalji u vezi sa dolaženjem do njih i dokazivanjem korektnosti algoritma mogu se naći u [79]. Dakle, algoritam za konveksno testiranje obuhvata sledeće:

1. Naći sve razdvajajuće parove bipovezanog grafa upotrebom algoritma Hopkrofta i Taržana za rastavljanje grafa na tripovezane komponente. Detektovati važne, kritične i zabranjene parove. Ako graf ima zabranjeni razdvajajućih par, završiti testiranje uz poruku da graf nema konveksni crtež. Ako graf nema zabranjeni par, razlikuju se sledeći slučajevi:
   (a) Graf nema ni kritični razdvajajući par. Algoritam se završava javljajući da graf ima konveksan crtež i da su svi ciklusi lica proširivi.
   (b) Graf ima tačno jedan kritični razdvajajući par. Postoji sedam tipova takvih grafova, za svaki od kojih se precizno definiše proširivi ciklus lica. Algoritam se završava sa porukom da graf ima konveksni crtež, vraćajući i pronađeni ciklus lica.
   (c) Graf ima više kritičnih razdvajajućih parova. Prelazi se na korak 2.
2. Za svaki kritični razdvajajućih par $\{x, y\}$ primenjuje se sledeća operacija, čime se konstruiše graf G_1:

- Ako $(x, y) \in E$, obrisati granu (x, y) iz G.
- U suprotnon ako je samo jedna $\{x, y\}$ razdelna komponenta prsten, obrisati putanju $x - y$ iz G.

Potom se od grafa G_1 konstruiše graf G_2 dodavanjem novog čvora v grafu G_1 i njegovim spajanjem sa svim čvorovima kritičnih razdvajajućih parova. Zatim se proverava planarnost grafa G_2, te ako se dobije negativan rezultat, algoritam obustavlja javljajući da graf nema konveksni crtež. Odnosno, ne postoji ciklus lica koji sadrži svaki čvor kritičnih razdvajajućih parova. Inače se S nalazi kao v-ciklus planarnog grafa G_2. v-ciklus nekog grafa $G_2 = G_1 + v$ je ciklus planarnog podgrafa G_1 koji graniči lice G_1 u kojem čvor v leži.

Koraci potrebni za testiranje da li uopšte postoji konveksni crtež ne predstavljaju poseban izazov za razumevanje i implementaciju, pod uslovom da su prethodno implementirani svi neophodni algoritmi za analizu, obilazak i dekompoziciju grafova. Naime, osim algoritma za nalaženje tripovezanih komponenti i razdvajajućih parova Hopkrofta i Taržana, potrebno je i nalaženje putanje između čvorova, što se može uraditi pretragom najpre u dubinu ili eventualno Dijkstrinim algoritmom. Takođe, u testiranje mogućnosti konveksnog crtanja uključena je i provera planarnosti, za šta se može koristiti algoritam formiranja LR particija, algoritam Bojer-Mirvold ili bilo koji drugi algoritmom ove svrhe koji je na raspolaganju. Autori rada preporučuju Hopkroftov i Taržanov, ali u to vreme nisu bile uvedene neke bolje opcije, te je u projektnom rešenju izabran Bojer-Mirvold algoritam. Nalaženju proširivog ciklusa lica, međutim, potrebno je posvetiti više pažnje.

5.4.2 Nalaženje proširivih ciklusa lica

Najjednostavniji slučaj u pogledu nalaženja odgovarajućeg ciklusa lica javlja se kada nema kritičnih parova, kako su tada svi ciklusi proširivi. Algoritam se ne bavi posebnim izdvajanjem nekih od njih, već se data faza obustavlja konstatacijom da se može izabrati bilo koji. Dakle, prateći istu logiku kao kod implementacije Tutovog algoritma, nema potrebe za nalaženjem svih ciklusa lica planarnog utapanja, već se prosto može uzeti spoljašnje lice koje vraća osnovni deo algoritma Bojer-Mirvold.

Algoritam za ispitivanje konveksnosti dalje razdvaja slučajeve kada postoji samo jedan, i kada postoji više kritičnih razdvajajućih parova, baveći se nalaženjem svih proširivih ciklusa nekog grafa koji se može nacrtati konveksno. U situaciji kada se detektovao samo jedan kritični razdvajajući par, autori predstavljaju sedam tipova planarnih grafova i njihove proširive cikluse lica, koji poseduju pomenutu osobinu, te prosto navode određivanje tipa kome graf pripada i nalaženje ciklusa sa tim u vidu kao rešenje problema. Ne ulazi se u detalje realizacije ovog koraka. Ukoliko graf, pak, ima više presečnih tačaka, uvodi se procedura koja deli graf G_2 na tripovezane komponente koje se potom na precizno definisani način spajaju, kako bi se formirao graf koji pripada jednoj od tri moguće grupe. Za svaki graf u dvema od tri grupe dalje se vrši višestruko nalaženje putanja između čvorova povezanih virtuelnim granama, kao i provera da li je neki skup putanja ciklus lica određenog grafa. Sam postupak, dakle, daleko je od trivijalnog, ali treba naglasiti i da ispitivanje da li je neki ciklus ciklus lica određenog grafa nije trivijalno. Za dati graf treba prvo naći planarno utapanje, što opet obuhvata primenu nekog od algoritama za testiranje planarnosti i nalaženje lica na osnovu utapanja. Međutim, za samo crtanje grafa na konveksan način nije potrebno poznavati sve proširive cikluse lica, te ovaj korak algoritma nije sastavni deo

implementacije u okviru projektnog rešenja. Naime, osmišljen je način izdvajanja jednog ciklusa koji zadovoljava definiciju proširivog ciklusa, odnosno, poseduje sve osobine kojima Čiba, Jamanouči i Nišizeki karakterišu takav podgraf.

Kao što je već spomenuto, svaki proširivi ciklus mora zadovoljavati sledeće:

- Sadrži sve tačke kritičnih razdvajajućih parova.
- Za svaki kritični razdvajajući par $\{x, y\}$ postoji najviše jedna njegova komponenta koja nema grane u ciklusu i koja je ili spona, ako $(x, y) \in E$ ili prsten, ako taj uslov nije zadovoljen.
- Naravno, mora biti ciklus.

Može se primetiti da ako je komponenta nekog kritičnog razdvajajućeg para tripovezani graf, neke njene grane moraju ući u proširivi ciklus lica. Ako neka razdelna komponenta zadovoljava uslov nepripadanja proširivom ciklusu, dodavanje njenih grana je opciono, samo je bitno da se izbegne izostavljanje dve komponente istog para. Dalje, ako neka razdelna komponenta datog razdvajajućeg para $\{x, y\}$ ne sadrži čvorove drugih razdvajajućih parova, neka putanja od x do y trebalo bi da se nađe u proširivom ciklusu lica (osim ako ne ispunjava uslov neuključivanja, kada je opciono). Naime, kako se neke komponente grane moraju naći u ciklusu, kao i sami čvorovi kritičnih parova, ukoliko se u razmatranoj komponenti data dva čvora para ne bi spojila, ciklus bi ostao otvoren. Slično, u proširivi ciklus lica ne sme se dodati ni više putanja iste komponente između dva čvora kritičnog para, kako rezultat ne bi bio ciklus. Takođe, na toj putanji se ne smeju naći virtuelne grane koje ne postoje u početnom grafu. Pridruživanjem nađene putanje proširivom ciklusu lica obezbeđuje se i prisustvo čvorova x i y kritičnog razdvajajućeg para u ciklusu. Grane opcione komponente tipa prsten se dodaju u ciklus lica, spone ne po automatizmu.

Kada se u proširivi ciklus lica dodaju grane neke komponente koja ne sadrži druge kritične parove, u grafu koji se analizira ona se može zameniti samo granom između čvorova razdvajajućeg para, pošto unutrašnje grane te komponente ne predstavljaju faktor u nastavku. Svakako se ne može pojaviti nijedna druga komponenta koja sadrži čvorove samo jednog razdvajajućeg para i baš te grane.

Imajući u vidu način na koji se formiraju komponente razdvajajućeg para, uočava se da one mogu ili sadržati samo čvorove tog jednog razdvajajućeg para, bivajući bilo kojeg tipa, ili čvorove više parova, bivajući ili prsten, ili tripovezani graf. Složenije komponente, koje sadrže čvorove drugih razdvajajućih parova, sadržaće i njihove prethodno uočene i analizirane komponente, samo, naravno, bez virtuelnih grana. To znači da će obuhvatati nađene putanje, te da će zadovoljavati uslov da neke njihove grane moraju biti u proširivom ciklusu lica. Dalje ostaje samo nalaženje odgovora na pitanje da li i neke druge grane kompleksne komponente treba da se nađu u proširivom ciklusu. Takođe, traženi ciklus ne sme na kraju ostati nepovezan niti otvoren.

Dakle, celokupan postupak sastoji se iz sledećih koraka:

1. Za svaki kritični razdvajajući par $\{x, y\}$ proverava se da li sadrži komponentu među čijim čvorovima nema čvorova drugih razdvajajućih parova. Ukoliko to jeste slučaj, a komponenta nije spona, nalazi se putanja od x do y. Za prsten je ovo trivijalno, dok je dosta složenije u slučaju tripovezanog grafa. Putanja, naime, treba da bude takva da pripada ciklusu lica planarnog utopljenog grafa. Pošto su poznata dva čvora putanje, problem je ipak nešto lakši nego u opštem slučaju. Dakle, ne nalaze se planarno utapanje i ciklusi lica grafa, već se razmatra na koji način data dva čvora mogu biti spojena u okviru komponente, a da se pri

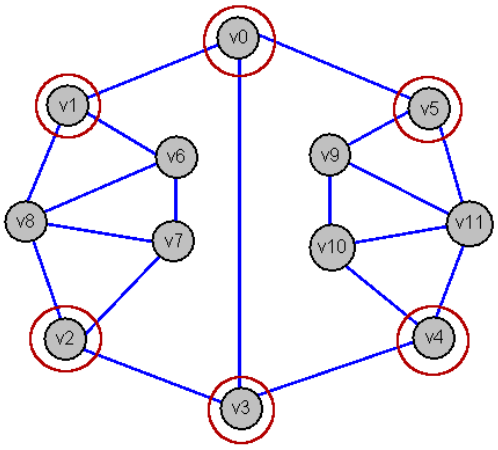
tome garantuje da neće doći do preseka grana. Ako je komponenta spona, putanja predstavlja samo jednu granu između dva čvora, tako da nije potrebna posebna obrada.

2. Ukoliko komponenta nije spona, nađene grane se dodaju na proširivi ciklus lica i uklanjaju iz polaznog grafa. U graf se umesto njih dodaje grana koja povezuje dva čvora razdvajajućeg para.

3. Graf kod koga su sve putanje koje su dodate na proširivi ciklus lica zamenjene jednom granom obilazi se sa ciljem zatvaranja ciklusa i obezbeđivanja da se čvorovi razdvajajućih parova koji eventualno nisu ranije uključeni nađu u proširivom ciklusu lica.

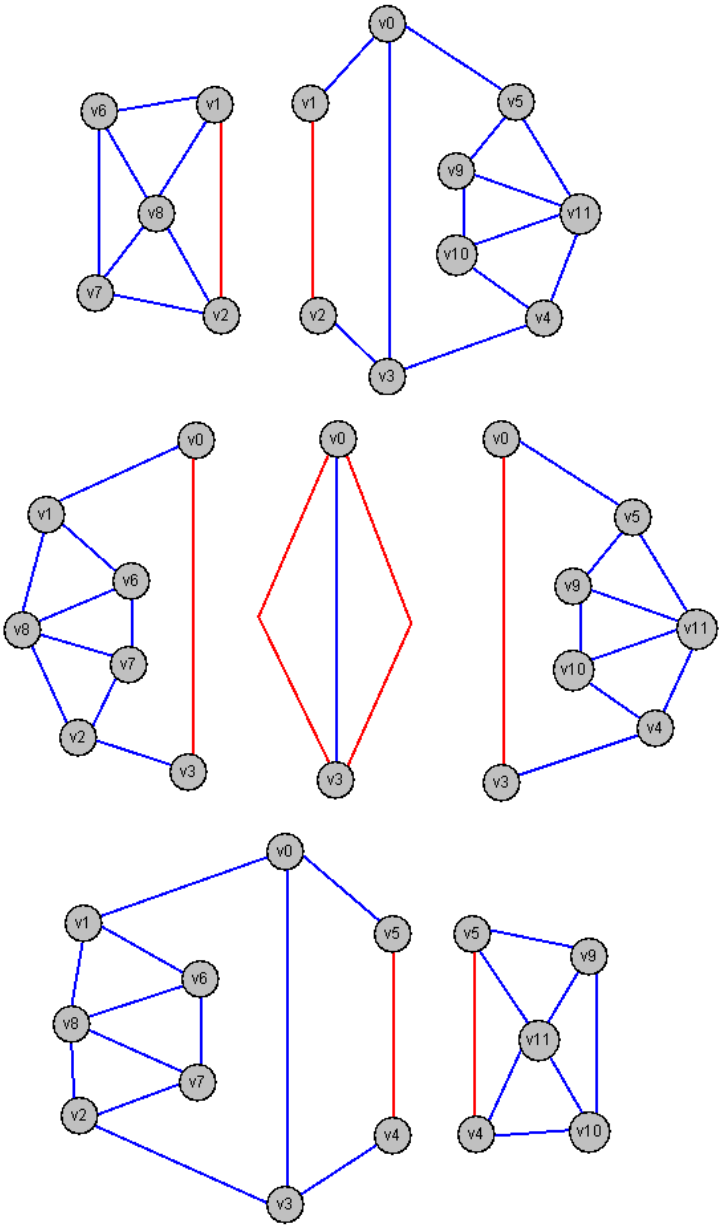
Za nalaženje putanje unutar $\{x, y\}$-komponente tipa tripovezanog grafa, koja će biti dodata na proširivi ciklus lica, koristi se pretraga najpre u dubinu, kojom se polazeći od čvora x kreće ka čvoru y, pri čemu se pri poseti svakom čvoru proverava da li bi pri tekućem rasporedu čvorova tj. ako bi tekuća putanja bila spoljašnje lice, došlo do preseka grana. Ovi uslovi inspirisani su algoritmima za ispitivanje planarnosti i proveravaju da li pri određenom rasporedu postoje putanje koje se presecaju, poredeći indekse njihovih početaka i krajeva i proveravajući da li dve putanje imaju neke zajedničke delove.

U nastavku će biti dat primer čitavog postupka. Neka je dat graf kao na slici 5.10, na kojoj su čvorovi kritičnih razdvajajućih parova zaokruženi crvenom bojom. Kritični parovi su $\{v_1, v_2\}, \{v_0, v_3\}, \{v_4, v_5\}$. Njihove razdelne komponente prikazane su na slici 5.11.



Slika 5.10: Graf sa označenim čvorovima kritičnih razdvajajućih parova

Vidi se da za $\{v_1, v_2\}$ postoji njegova komponenta koja ne sadrži čvorove drugih razdvajajućih parova (leva od dve komponente tog para) i da je ona tripovezani graf. Za nju se traži putanja koja će sadržati v_1 i v_2 i davati planaran crtež te komponente. Takvih putanja ima više. Koja tačno će se dobiti, zavisi od redosleda posete čvorovima u okviru DFS pretrage. Neka je to v_2, v_8, v_1. Isto važi i za par $\{v_4, v_5\}$, samo je desna komponenta ta koja ne sadrži druge čvorove. Neka se za nju našla putanja v_4, v_{11}, v_5. Unutrašnjost dvaju analiziranih komponenti više nije bitna, te se ti delovi grafa svode samo na grane između čvorova parova. Za treći razdvajajući par, $\{v_0, v_3\}$, vidi se da su pojedine grane dve komponente tada već dodate ciklusu, te da je uključivanje grane spone opciono. Nakon zamene, dobija se graf kao na slici 5.12.

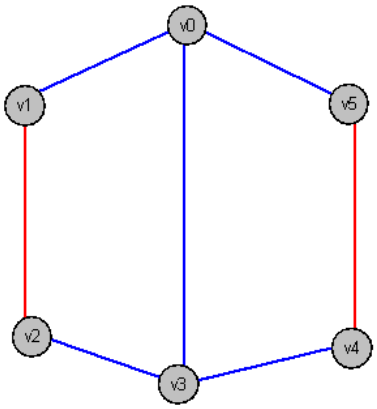


Slika 5.11: Razdelne komponente kritičnih razdvajajućih parova

Dakle, ostaje još samo da se zatvori putanja i obezbedi dodavanje svih čvorova razdvajajućih parova tj. v_0 i v_3. Šta je na ciklusu lica između v_1 i v_2 ili v_4 i v_5, nije bitno u ovom koraku. Jednostavnom pretragom, nalaze se putanje koje nedostaju, pa se na proširivi ciklus lica dodaju grane $(v_1, v_0), (v_0, v_5), (v_2, v_3), (v_3, v_4)$. Poslednji korak jeste sortiranje grana ciklusa tako da čine zatvorenu putanju polaznog grafa. Time je dobijen proširivi ciklus lica $(v_1, v_0), (v_0, v_5), (v_5, v_11), (v_11, v_4), (v_4, v_3), (v_3, v_2), (v_2, v_8), (v_8, v_1)$

5.4.3 Konveksno crtanje

Ako se ustanovi da bipovezani planarni graf G ima konveksni crtež i da je S njegov ciklus lica, neka je S^* proširivi konveksni poligon tog ciklusa. Algoritam za linearno nalaženje konveksnog crteža dalje se odvija prema sledećim koracima:



Slika 5.12: Graf po uklanjanju obrađenih komponenti

1. Za svaki čvor v stepena 2 koji nije na S, v i grane koje ga sadrže zamenjuju se jednom granom koja povezuje dva čvora povezana sa ovim čvorom. Neka je rezultujući graf G'.

2. S^* se proširuje na konveksni crtež G' pozivanjem procedure koju autori algoritma nazivaju *DRAW*.

3. Za svaki izbrisani čvor stepena 2 njegova pozicija se nalazi na pravolinijskom segmentu koji spaja njegova dva suseda. Time se algoritam završava.

Ostaje još da se prikaže procedura $DRAW(G, S, S^*)$, koja proširuje konveksni poligon S^* spoljašnjeg ciklusa lica S grafa G na konveksni crtež tog grafa, pri čemu on ne sadrži čvor stepena većeg od 2, koji nije na S.

1. Ako G ima 3 ili manje čvorova, konveksni crtež grafa je nađen i procedura se završava. Ako to nije slučaj, odnosno, graf ima bar 4 čvora, na proizvoljan način se bira čvor v poligona S^* i nalazi graf G' kao $G - v$. Dalje, graf G' se deli na blokove $B_i, 1 \leq i \leq p$. Neka su v_1 i v_{p+1} dva čvora ciklusa S povezana sa v, a $v_i, 2 \leq i \leq p$ artikulacioni čvorovi grafa G' takvi da je $v_1 \in V(B_1), v_{p+1} \in V(B_p)$, dok je $v_i = V(B_{i-1}) \cap V(B_i)$. Svaki čvor $v_i, 1 \leq i \leq p+1$ se nalazi na S.

2. Svaki blok B_i crta se konveksno primenom sledeće procedure:

   (a) Traži se konveksni poligon S_i^* spoljašnjeg ciklusa lica S_i bloka B_i. Pozicije čvorova iz $V(S_i) \cap V(S)$ već su poznate nalazeći se na S^* Dakle, potrebno je pozicionirati čvorova iz $V(S_i) - V(S)$, i to unutar trougla $\triangle v \cdot v_i \cdot v_{i+1}$ na taj način da svi čvorovi povezani sa čvorom v budu temena konveksnog poligona S_i^*, dok su drugi na pravolinijskim segmentima S_i^*.

   (b) Procedura $DRAW(B_i, S_i, S_i^*)$ se rekurzivno poziva, kako bi se S_i^* proširilo na konveksni crtež bloka B_i.

Algoritam ne precizira na koji način se određuju pozicije čvorova ciklusa lica, niti kako se tačno čvorovi postavljaju na pravolinijske segmente ili kao temena konveksnog poligona, te će u nastavku biti više reči o implementaciji ovih koraka.

Konveksni poligon S^* može se, kao i u slučaju Tutovog algoritma, odrediti kružnim raspoređivanjem, na način da se čvorovi postave na jednaka rastojanja jedan od drugog. Više pažnje treba posvetiti pozicioniranju čvorova unutar spomenutog trougla $\triangle v \cdot v_i \cdot$

v_{i+1}, tako da budu i temena konveksnog mnogougla. Osmišljeno rešenje nađeno je kao kompromis između jednostavnosti kalkulacije i dobijanja simetričnog, konveksnog mnogougla. Naime, postupak je sledeći:

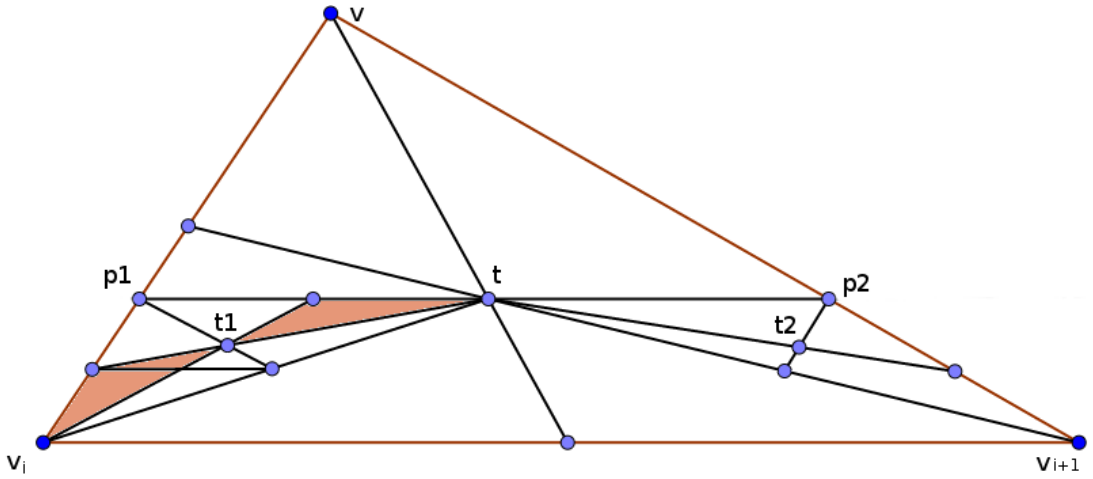
1. Počinje se od trougla $\triangle v \cdot v_i \cdot v_{i+1}$. Pozicije svih njegovih temena su poznate i pre pozivanja algoritma.

2. Nalazi se težište trougla t, te se povlači linija paralelna sa v_i, v_{i+1} kroz njega i nalaze preseci sa $[v, v_i]$ - p_1 i $[v, v_{i+1}]$ - p_2. Time se dobiju dva nova trougla - $\triangle v_i \cdot p_1 \cdot t$ i $\triangle v_{i+1} \cdot p_2 \cdot t$

3. Ukoliko nisu raspoređeni svi izdvojeni čvorovi, nalaze se težišta tekuća dva trougla. Za svakog se kreiraju dva nova trougla. Temena prvog su: težište trougla, sredina strane trougla čija se oba temena nalaze na prethodno spomenutoj paralelnoj liniji (gde je težište roditeljskog trougla), i teme trougla koje se ne nalazi na toj liniji. Temena drugog trougla dobijaju se povlačenjem nove paralelne linije kroz težište trougla. Dakle, to će biti: teme trougla koje nije na liniji paralelnoj težištu roditeljskog trougla, presek odgovarajuće strane trougla sa linijom paralelnom liniji kroz težište roditeljskog trougla povučenoj u novom težištu i to težište. Glavna ideja je da se obezbedi da temena konstantno budu jedno iznad ili ispod drugog do nekog vrha, te da sledi konstantno opadanje ili rast. Slika 5.13 ilustruje glavnu ideju ovog postupka.

4. Ukoliko broj izračunatih temena trougla nije veći ili jednak broju čvorova koje treba rasporediti, prethodni korak se ponavlja.

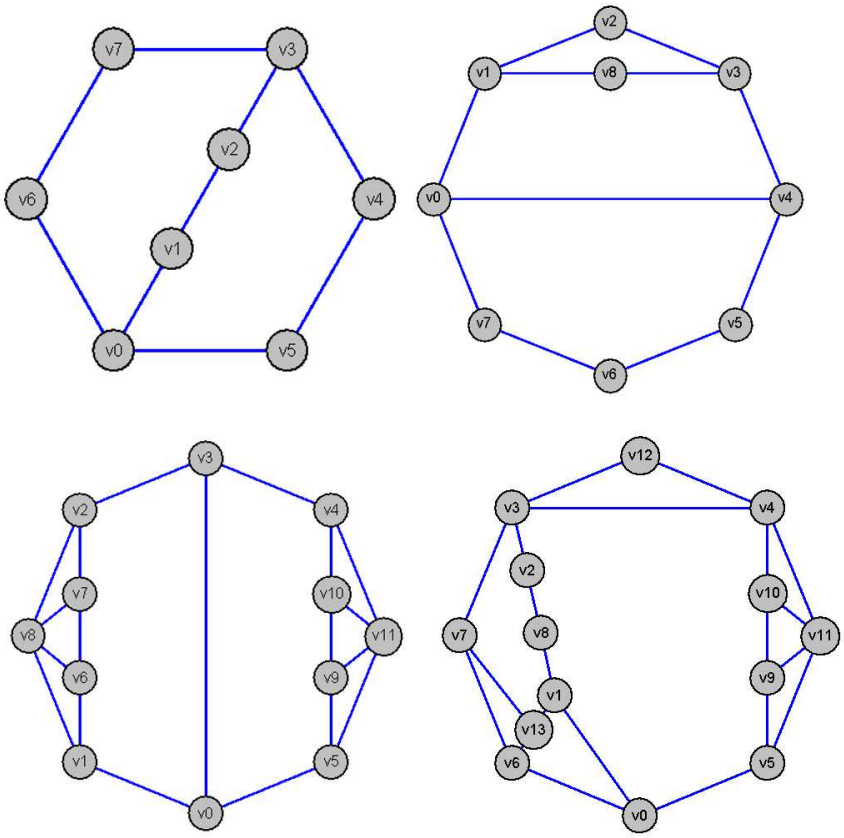
5. Po nalaženju svih težišta, dalji postupak obuhvata smeštanje čvorova na njihove pozicije, pri čemu se vodi računa o simetriji. Dakle, ako treba rasporediti 2 čvora, neće se postaviti u prvo i drugo težište, već u prvo i treće.



Slika 5.13: Raspoređivanje čvorova unutar trougla $\triangle v \cdot v_i \cdot v_{i+1}$

Postavljanje čvorova na pravolinijske segmente vrši se računanjem ukupne razdaljine između dva zadata čvora, te njenom podelom na $n+1$ jednakih delova, gde je n broj čvorova za pozicioniranje. Dalje se na svakom spoju dva segmenta postavlja po jedan čvor.

Na slici 5.14 prikazano je nekoliko primera konveksnih crteža grafova. Treba napomenuti da izgled crteža zavisi od nasumičnog izbora čvora koji se uklanja.



Slika 5.14: Primeri konveksnih crteža grafova

5.5 Ortogonalno crtanje grafova

Različiti načini za ortogonalno crtanje grafova opisani su u kraćim crtama u okviru pregleda klasa algoritama za automatsko raspoređivanje elemenata grafova. U rešenju je za implementaciju odabrano korišćenje vidljive reprezentacije planarnog grafa. Nalaženje vidljive reprezentacije prvi je korak mnogih, ne samo ortogonalnih, nego i opštijih algoritama za crtanje planarnih grafova, te implementacija njenog kalkulisanja može imati i šire primene.

Konstrukcija vidljive reprezentacije planarnog grafa $G = (V, E)$ sastoji se iz sledećih koraka:

1. Ukoliko graf nije bipovezan, primenjuje se algoritam za planarnu augmentaciju.
2. Nalazi se spoljašnje lice, recimo upotrebom algoritma Bojer-Mirvold i na njemu se bira jedna grana (s, t).
3. Nalazi se st-graf nad G, G_1, te za njega dualni planarni st-graf G^*.
4. Određuju se optimalna topološka uređenja $T_x = T(G^*)$ i $T_y = T(G_1)$.
5. Svakom čvoru $v \in V$ se nalazi pozicija na horizontalnim segmentima. Ako su f_l i f_r lica, odnosno, čvorovi dualnog grafa G^*, takvi da je f_l levo od krajnje leve izlazne grane čvora v, a f_r desno od krajnje desne izlazne istog čvora, računa se:
   (a) $\Gamma(v).y \leftarrow T_y(v)$
   (b) $\Gamma(v).xmin \leftarrow T_x(f_l)$

(c) $\Gamma(v).xmax \leftarrow T_x(f_r) - 1$

6. Za svaku granu $e = (u, v) \in E$ se nalazi pozicija na vertikalnim segmentima. Ako je f_l lice levo od e, računa se:

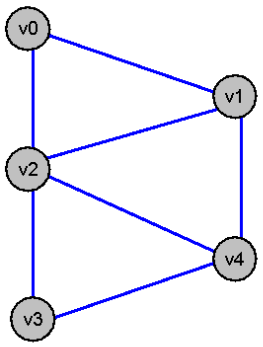
   (a) $\Gamma(e).x \leftarrow T_x(f_l)$

   (b) $\Gamma(e).ymin \leftarrow T_y(u)$

   (c) $\Gamma(e).ymax \leftarrow T_y(v)$

7. Na samom kraju se uklanjaju grane dodate planarnom augmentacijom.

Vidljiva reprezentacija, dakle, mapira svaki čvor na horizontalni, a granu na vertikalni segment. Na osnovu nje može se konstruisati crtež grafa pozicioniranjem čvorova na neku lokaciju koja pripada njemu dodeljenom segmentu. Na primer, na mesto završetka neke od grana, kako bi se do neke mere smanjio broj njihovih preloma. Na slici 5.15 prikazan je jedan bipovezani graf, a na slici 5.16 njegova vidljiva reprezentacija, gde je za s odabran čvor v_1, a za t čvor v_4. Prvom od ovih čvorova dodeljen je najniži, a drugom najviši segment.

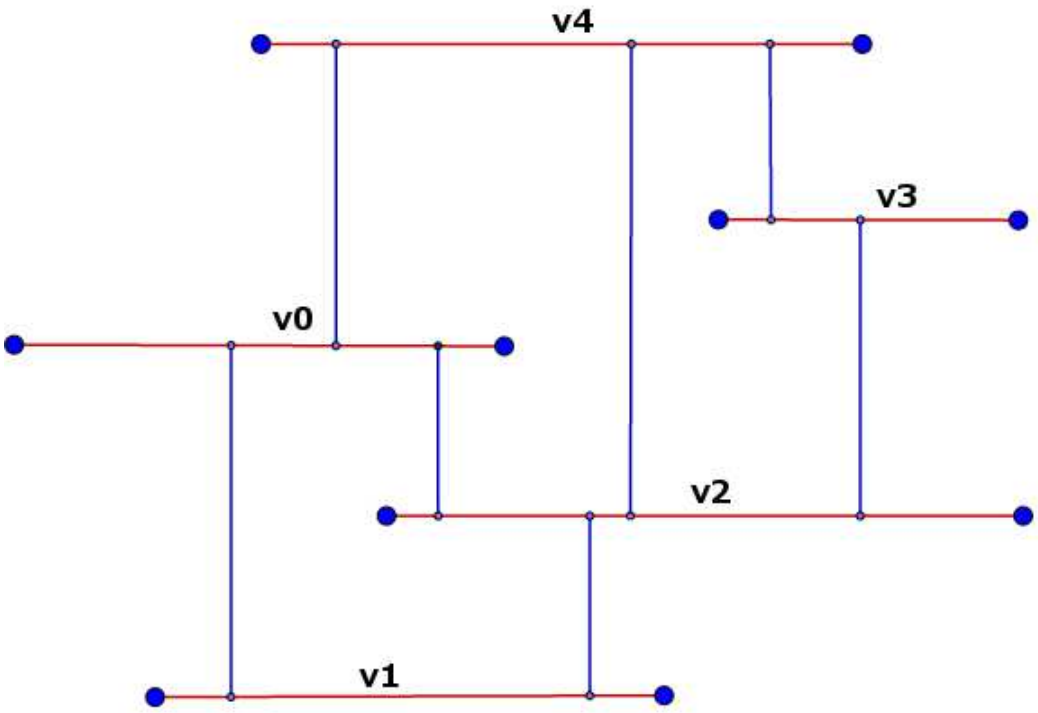


Slika 5.15: Graf koji se crta ortogonalno

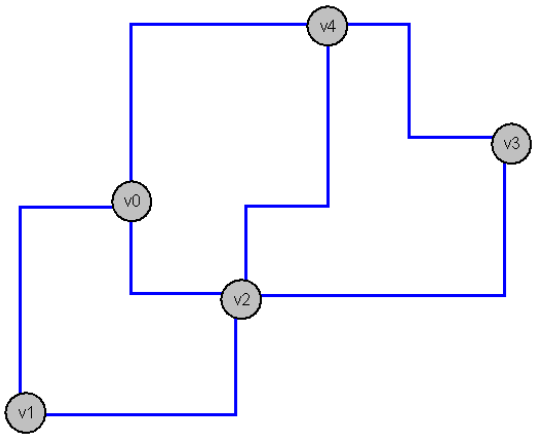
Na slici 5.17 prikazan je crtež grafa dobijen od vidljive reprezentacije bez dodatnog procesiranja. Dakle, čvorovi su postavljeni u okviru svojih horizontalnih segmenata, na mesta spoja sa nekom od grana.

Primećuje se poprilično veliki broj preloma grana koji bi se mogao znatno smanjiti ukoliko bi se čvorovi poravnali jedan sa drugim gde je to moguće. Na primer, ako bi se čvoru v_3 y-koordinata izjednačila sa ovom vrednošću iste koordinate čvora v_4, broj preloma bi se smanjio za 2. Ovo pomeranje je moguće imajući u vidu da ne postoji čvor sa istom ili bliskom vrednošću x koordinate koji se nalazi između dva pomenuta čvora na vertikalnoj osi. Upravo je ovo glavna ideja postupka kojim se formira konačni crtež grafa. Dakle, gde god je moguće poravnati čvorove po horizontalnoj ili vertikalnoj osi, a da se pritom ne poremeti planarnost ili dođe do preklapanja čvorova, to se i čini. Time se teži ispunjavanju estetskog kriterijuma minimizacije broja preloma grana. Naime, konačni izgled crteža posmatranog grafa, prikazan na slici 5.18, nakon prilagođavanja pozicija čvorova ima samo 3 preloma grana.

Tokom pomeranja čvorova vodi se računa i o njihovim dimenzijama. Recimo, da je čvor v_2 toliko širok da bi grana iz v_0 i bez pomeranja ulazila u njega bez preloma grana, njegova pozicija ne bi se korigovala. Takođe, grana koja iz v_0 vodi ka v_1 morala bi imati



Slika 5.16: Vidljiva reprezentacija

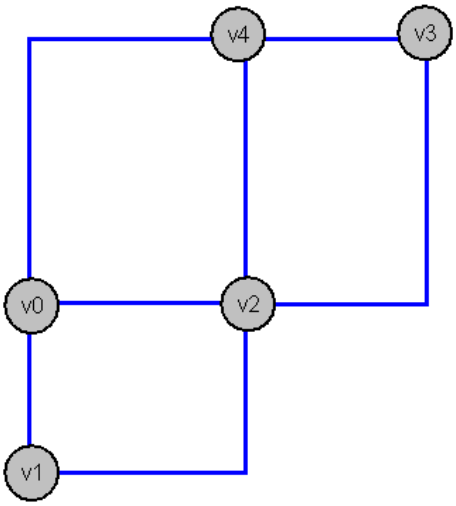


Slika 5.17: Crtež grafa nastao od vidljive reprezentacije pre dodatnog procesiranja

duži horizontalni segment kako bi se obišao široki čvor v_2, te bi tada i x-koordinata čvora v_1 bila za istu vrednost modifikovana, kako bi se izbeglo dodatno prelamanje grane.

5.6 Simetrično crtanje grafova

Simetrično crtanje grafova u projektnom rešenju implementirano je iz oslonac na algoritam koji osmislili Kar i Kokaj. Algoritam kao ulaz prima graf G koji mora imati netrivijalnu grupu automorfizama $Aut(G)$ i jednu netrivijalnu prethodno odabranu simetriju $g \in Aut(G)$ [77]. Osnovni cilj algoritma je nalaženje crteža na kojemu se uočava data simetrija. Dakle, dobijeni crtež u velikoj meri zavisi od izbora permutacije,



Slika 5.18: Konačni izgled crteža

odnosno, simetrije g.

Štaviše, nije moguće za bilo koje g konstruisati simetrični crtež, bar ne bez dodatnih informacija. Algoritam ne ulazi u detalje određivanja odgovarajuće permutacije, ali izdvaja one koje sadrže više ciklusa iste dužine kao pogodne. Kar i Kokaj određuju čvorove koji će se kružno rasporediti na spoljašnjem ciklusu, ali ne specifikuju način raspoređivanja preostalih čvorova.

5.6.1 Algoritam Kara i Kokaja

Algoritam, dakle, prima kao ulazni parametar permutaciju i pretpostavlja da sadrži bar k ciklusa $C_1, C_2, ..., C_k$ takvih da su svi dužine $m \geq 2$, gde je $k \geq 1$. Permutacija osim izdvojenih ciklusa može sadržati i druge. U cilju veće preglednosti, za čvor $v \in C_i$ sa $v + j$ se označava j-ti čvor posle v u ciklusu. Dakle, čvorovi u nekom C_i mogu se prikazati u obliku: $(v, v+1, v+2, ..., v+m-1)$. Po uvođenju ove konvencije, može se navesti lema koja se nalazi u osnovi algoritma, a čiji dokaz se može naći u [77].

Neka su $C_1, C_2, ..., C_k$ ciklusi u kojima se nalaze čvorovi $v_1, v_2, ..., v_k$ redom, pri čemu $v_i \to v_{i+1}, i = 1, 2, ..., k-1$, i $v_k \to v_1' \in C_1$, gde je $v_1' \neq v_1$. Graf G tada sadrži ciklus C dužine $\frac{km}{NZD(m,d)}$, gde je $d = v_1' - v_1$. Ukoliko je $NZD(m,d) = i > 1$, G sadrži i ciklusa sa odvojenim čvorovima, koji su svi iste dužine. Simetrija g permutuje ove cikluse u kružnom rasporedu, a g^i rotira ciklus kroz $\frac{d}{i}$ čvorova. Ovakvi ciklusi se nazivaju simetričnim. Takođe, ako su brojevi d i m međusobno prosti, postoji jedan ciklus C maksimalne moguće dužine km. Prvi deo algoritma za cilj ima nalaženje takvog simetričnog ciklusa.

Pošto su uvedeni osnovni pojmovi i ideje, mogu se predstaviti i sami koraci algoritma:

1. Konstruiše se ciklična reprezentacija permutacije g, kako bi se njeni ciklusi klasifikovali po dužini.

2. Iterira se kroz sve cikluse permutacije. Za tekući ciklus nalazi se putanja P takva da svaki čvor na njoj pripada drugom ciklusu čija je dužina jednaka dužini tekućeg.

Kreiranje putanje vrši se posebnom procedurom koju autori algoritma nazivaju *ExtendPath*.

3. Ciklus se označava posećenim, da se ne bi mogao pojaviti u budućim putanjama. Maksimalna dužina putanje koja može biti konstruisana za cikluse tekuće dužine umanjuje se upravo za tu dužinu, te se prelazi na sledeću iteraciju.

Procedura *ExtendPath* poziva se prosleđivanjem čvora koji reprezentuje tekući ciklus, u. Obično se za predstavnika ciklusa postavlja prvi čvor u njemu. Neka je dužina tekućeg ciklusa obeležena sa l. *ExtendPath* obuhvata sledeće korake:

1. Iterira se kroz sve čvorove v takve da je $v \to u$.

2. Za tekuće v nalazi se ciklus kome on pripada. Ukoliko je taj ciklus odgovarajuće dužine (iste kao ciklus kome pripada čvor u) i nije prethodno posećen, čvor v se dodaje na putanju P i označava posećenim. *ExtendPath* se rekurzivno poziva prosleđujući v, te ako taj poziv ne dovede do konstrukcije najduže putanje, čvor v se uklanja sa putanje.

3. Ukoliko se čvor v nalazi u istom ciklusu kao i u, što znači da se pretraga vratila u polazni ciklus, razlikuju se slučajevi kada je v jednako reprezentu ciklusa, i kada nije.

   - Ako je v reprezent ciklusa, proverava se da li je dužina tekuće putanje veća od do tada najduže nađene za prosleđeni čvor u. Ukoliko to jeste slučaj, ona se pamti kao trenutno najbolja putanja. U svakom slučaju, prelazi se na sledeću iteraciju, odnosno, sledeće v.
   - Ako v nije predstavnik ciklusa, računa se dužina najvećeg mogućeg ciklusa, C, l_C. Ako je l_P dužina putanje, a $g = NZD(d, l)$, onda je je $l_C = l_P \frac{l}{g}$. Ako je l_C manja od najveće prethodne izračunate l_C, prelazi se na sledeću iteraciju, kao i u slučaju da je g manje ili jednako najvećoj ranije određenoj vrednosti ove varijable. U suprotnom se P proglašava tekućom najboljom putanjom, njena dužina tekućom najboljom vrednosti za l_C, a trenutno g najboljom *NZD* vrednošću. Ako je l_C najveće moguće u datom kontekstu, poziv *ExtendPath* procedure vraća indikator uspešnosti.

4. Prelazi se na sledeću iteraciju. Ako se došlo do kraja a procedura se nije prekinula sa uspešnim ishodom, vraća se indikator neuspešnog nalaženja najdužeg ciklusa.

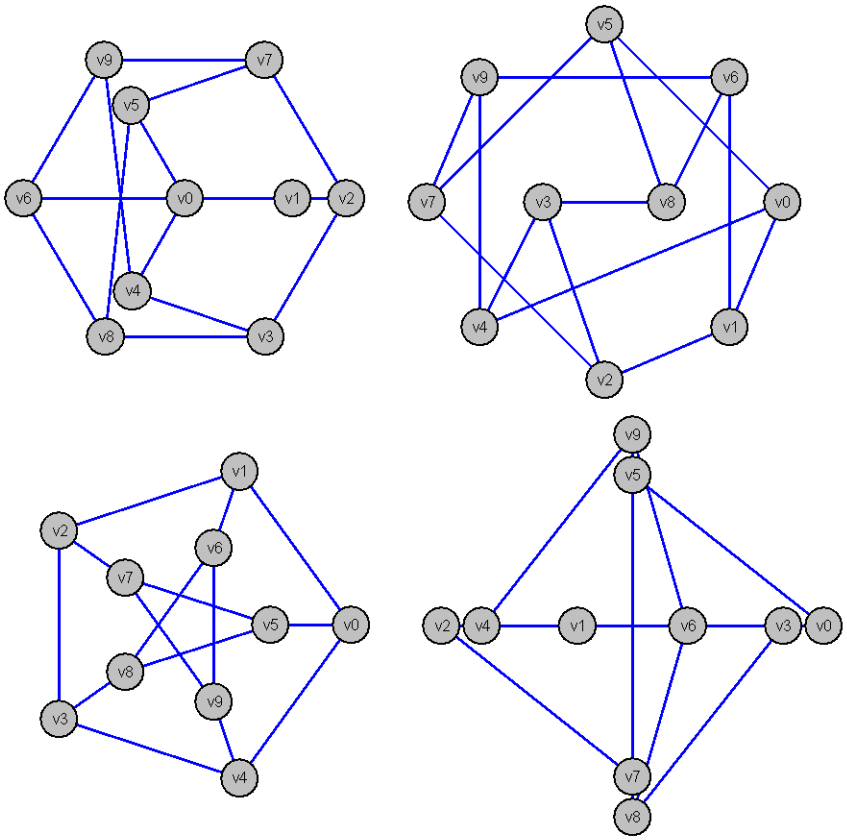
Algoritam, dakle, određuje putanje za cikluse različitih dužina. Ako postoji jedan ciklus kome pripadaju svi čvorovi grafa G, rezultujući crtež se sastoji iz jedne kružnice po kojoj su raspoređeni svi čvorovi i koji je je obično veoma kvalitetan. Ukoliko to nije slučaj, algoritam predlaže raspoređivanje čvorova ciklusa na koncentrične kružnice.

5.6.2 Implementacija simetričnog raspoređivanja

Algoritam Kara i Kokaja, kao što je spomenuto, zahteva prosleđivanje jedne simetrije kao ulaznog parametra. Time se omogućava kreiranje crteža koji će naglasiti baš nju, što je možda upravo ono što korisnici žele. Međutim, automatizacija procesa nalaženja odgovarajuće permutacije svakako znatno povećava iskoristivost ovog algoritma. Autori algoritma kao najbolje izdvajaju permutacije kod kojih postoji najveći broj čvorova u ciklusima iste dužine.

Simetrično raspoređivanje je, uzimajući u obzir upravo navedeno, implementirano tako da se omogući specifikacije željene permutacije, ali i da se automatski nađe odgovarajuća ukoliko korisnik ne želi samostalno izdvojiti nijednu. Permutacije se nalaze upotrebom algoritma Makeja, te se njihovom analizom, uzimajući u obzir sugestiju Kara i Kokaja, izdvaja ona koja se prosleđuje algoritmu. Dalje je postupak isti kao i u slučaju kada se permutacija direktno prosledi.

Upotrebom algoritma Kara i Kokoja nalazi se niz ciklusa, te se on sortira tako da se bliže spoljašnosti postave kružnice koje sadrže duže cikluse. Za svaku od kružnica, na način isti kao kod već opisanog kružnog crtanja računa se optimalni poluprečnik. Ukoliko se za poluprečnik nekog unutrašnjeg kruga dobije veća vrednost od poluprečnika spoljašnjeg, spoljašnji se postavlja na zbir unutrašnjeg i neke određene distance kojom se definišu razmaci između krugova. Na slici 5.19 prikazano je nekoliko crteža Petersonovog grafa generisanih ovim algoritmom. Treći od njih upravo je najpoznatija reprezentacija ovog čuvenog grafa.

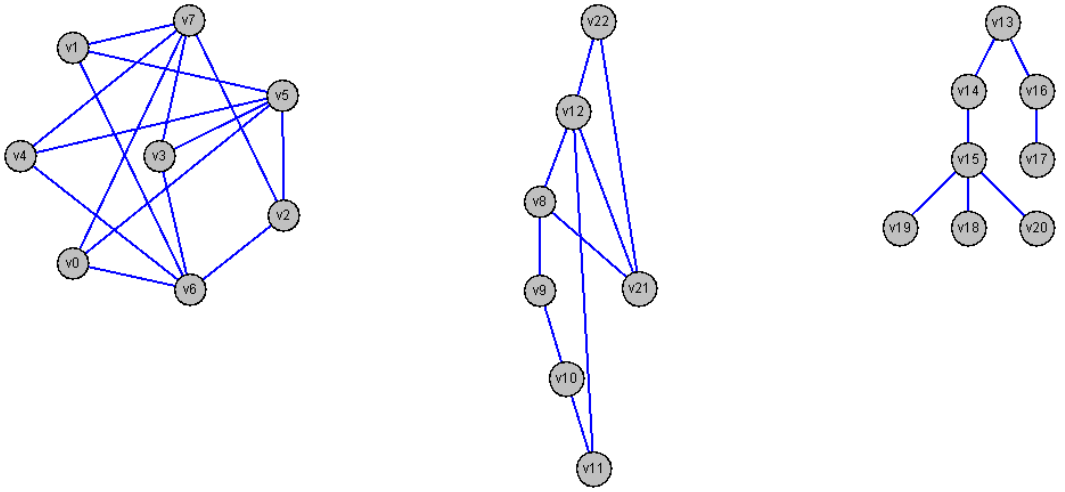


Slika 5.19: Nekoliko crteža Petersonovog grafa generisanih simetričnim algoritmom

5.7 Ostali algoritmi

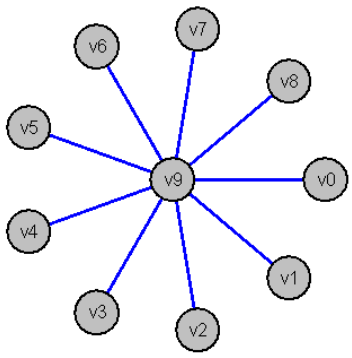
Pored prethodno opisanih, implementirano je još nekoliko jednostavnijih algoritama. Prvi od njih raspoređuje čvorove u strukturu tipa tabele, postavljajući definisani broj čvorova u jedan red, pre prelaska u sledeći. Broj elemenata u jednom redu, kao i željeni razmaci između njih, kako po horizontalnoj, tako i po vertikalnoj osi se mogu konfigurisati. Prilikom pozicioniranja čvorova u obzir se uzimaju njihove dimenzije.

Dakle, rastojanje između centara dva susedna čvora po x-osi jednako je zbiru polovine širina dva čvora i zadatog razmaka. Slično se računa i rastojanje po y-osi, samo se, naravno, u obzir uzimaju visine. Ovaj algoritam je idealan za primenu u situacijama kada se graf sastoji iz više nepovezanih čvorova. Takođe, koristi se i pri kompozitnom raspoređivanju, gde se jedan nepovezani graf sastoji iz više 1-povezanih komponenti. Ukoliko algoritam koji se primenjuje ne pozicionira te komponente na način da se njihovi delovi ne preklope, ili se nad svakom komponentom izvršava drugi algoritam, upravo opisani postupak se primenjuje u cilju dobijanja konačnog crteža grafa. Jedan primer, gde su razdvojene komponente raspoređene različitim algoritmima, prikazan je na slici 5.20.



Slika 5.20: Raspoređivanje komponenti grafa

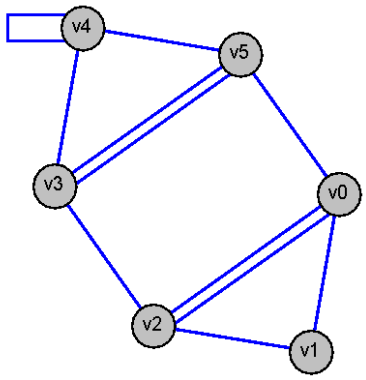
Drugi jednostavniji algoritam kružno raspoređuje čvorove oko jednog, izabranog čvora. Ovaj algoritam dodat je postojećem skupu naprednijih algoritama radi bolje podrške automatskom raspoređivanju i namenjen je primeni u situacijama kada je veliki broj čvorova povezan sa jednim istim čvorom. Na primer, u dijagramima klasa može se uočiti takva situacija, kada, recimo, veći broj konkretnih klasa implementira jedan interfejs. Primer je prikazan na slici 5.21.



Slika 5.21: Kružno crtanje sa jednim čvorom u sredini

Osim implementacije konkretnih algoritama za raspoređivanje elemenata grafova, realizovano je i postprocesiranje dobijenih crteža. Naime, većina algoritama ne vodi

računa o petljama, kao ni višestrukim granama između dva ista čvora. Dakle, nakon primene izabranog algoritma za raspoređivanje, vrši se provera da li u grafu postoje pomenute grane, te se obrađuju na odgovarajući način u slučaju detekcije. Višestruke grane se postavljaju paralelno jedna drugoj, pri čemu razmak među njima zavisi od veličine čvorova koji su povezani, kao i samog broja grana između njih. Procesiranje rekurzivnih grana fokusira se na obezbeđivanje njihove jasne vidljivosti na crtežu. Primer grafa koji sadrži oba tipa grana prikazan je na slici 5.22.



Slika 5.22: Primer crteža grafa sa rekurzivnim i višestrukim granama

5.8 Pregled naučnih i stručnih doprinosa na polju crtanja grafova

Opisane implementacije nekolicine algoritama za crtanje grafova ujedno sadrže i njihova manja poboljšanja, dopune i pojašnjenja:

1. Kod algoritma za kružno raspoređivanje elemenata dijagrama osmišljen je način za automatsko određivanje poluprečnika kruga na osnovu broja i veličine čvorova.

2. Objašenjen je jedan mogući način za automatsko određivanje ulaza u Tutov algoritam. Radi se o periferalnom poligonu i pozicijama čvorova koji mu pripadaju.

3. Kod algoritma za nalaženje konveksnih crteža planarnih grafova uočeno je da, ukoliko je cilj samo naći pozicije čvorova, nije neophodno odrediti sve proširive cikluse lica. Polazeći od definicija Čibe, Jamanoučija i Nišizekija, osmišljen je način kojim se nalazi samo jedan proširivi ciklus. Prednost opisanog rešenja je znatno jednostavnija kasnija implementacija. Takođe, predložen je način za realizaciju neprecizno definisanog koraka nalaženja pozicija čvorova ciklusa lica, tako da zadovolje sve uslove definisane u okviru algoritma.

4. Kod simetričnog raspoređivanja koje određuje niz ciklusa koji treba da budu raspoređeni po koncentričnim kružnicama, opisan je način nalaženja poluprečnika i rasporeda tih kružnica.

5. Predočena je potreba za posebnim raspoređivanjem višestrukih i rekurzivnih grana.

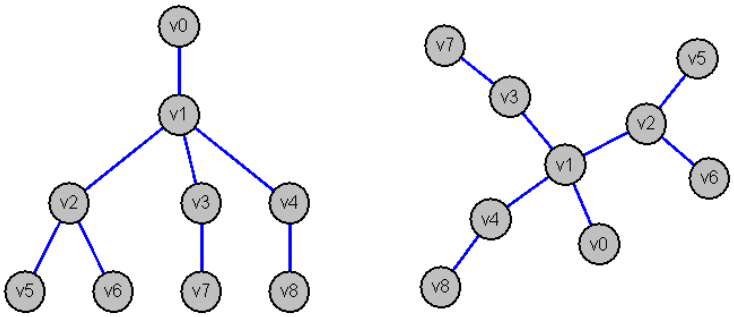
Stručni doprinosi pre svega jesu implementacije algoritama za analizu i crtanje grafova na modernom programskom jeziku - *Javi*.

Poglavlje 6

Automatski izbor algoritma za raspoređivanje

U ovoj sekciji biće opisan postupak automatskog izbora i primene algoritma isključivo na osnovu osobina grafa, bez ikakvog učešća korisnika. Ovakva opcija mogla namenjena je korisnicima grafičkih editora koji nisu upoznati sa oblašću crtanja grafova, te bi želeli bez puno udubljivanja i utrošenog vremena rasporediti elemente svog dijagrama na dovoljno dobar način. Odnosno, tako da se dobije pregledan i čitljiv crtež.

Kao što je ranije napomenuto, razni algoritmi za crtanje grafova fokusirani su na jedan njihov tip, dajući bolje rezultate nego opštiji prilikom primene nad odgovarajućim ulaznim grafom. Na primer, ako je graf stablo, crtež na kome se može uočiti najveći broj estetskih kriterijuma dobija se upravo primenom algoritma specijalizovanog za rad sa stablima. Kako je stablo specijalan slučaj hijerarhije, primena hijerarhijskog algoritma takođe bi dala relativno dobar rezultat, ali za nešto više vremena, dok bi silom usmereni bio loša opcija. Na slici 6.1 prikazana su dva crteža jednog stabla, gde je prvi dobijen primenom algoritma za crtanje stabala baziranog na nivoima, a drugi organskog silom usmerenog. U drugom slučaju se teško primećuje da graf predstavlja hijerarhiju.



Slika 6.1: Dva crteža stabla generisana specijalizovanim i opštijim algoritmom

Slično, ako je graf prsten, prirodno bi bilo rasporediti njegove elemente kružno, a ne recimo hijerarhijski. Osnovna ideja postupka automatskg izbora algoritma za raspoređivanje elemenata grafa sastoji se upravo u detekciji osobina koje kao preduslov za primenu ili dobijanje posebno lepog crteža izdvajaju specijalizovani algoritmi. Naime, ukoliko je moguća primena algoritma dizajniranog posebno za dati tip grafa, može se računati da će rezultat biti bolji, odnosno da će se na dobijenom crtežu moći uočiti više poželjnih estetskih kriterijuma.

6.1 Implementacija automatskog raspoređivanja

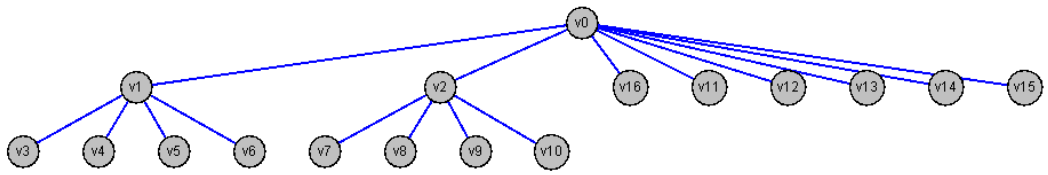
Automatsko raspoređivanje realizovano je, dakle, izdvajanjem osobina koje posebno odgovaraju pojedinim algoritmima za crtanje, te primenom prethodno implementiranih algoritma za analizu grafova kojima se proverava njihovo prisustvo. U skladu sa izloženim opservacijama, provera se vrši od specifičnijih ka opštijim osobinama. Postupak je sledeći:

1. Proverava se da li graf poseduje bar jednu granu. Ako to nije slučaj, čvorovi se raspoređujuu u strukturu tipa tabele. Na primer, dijagram klasa koji na prvom nivou poseduje samo skup nepovezanih paketa zadovoljavao bi ovaj kriterijum.

2. Proverava se da li je graf prsten. Ako jeste, primenjuje se kružni algoritam za raspoređivanje.

3. Proverava se da li je graf takav da su svi njegovi čvorovi povezani sa istim čvorom, pa se u tom slučaju primenjuje kružni algoritam sa centralnim čvorom. Kao što je spomenuto, takva situacija može se sresti, na primer, kod dijagrama klasa, kada se predstavlja slučaj kad više klasa nasleđuju jednog pretka, ili realizuju isti interfejs.

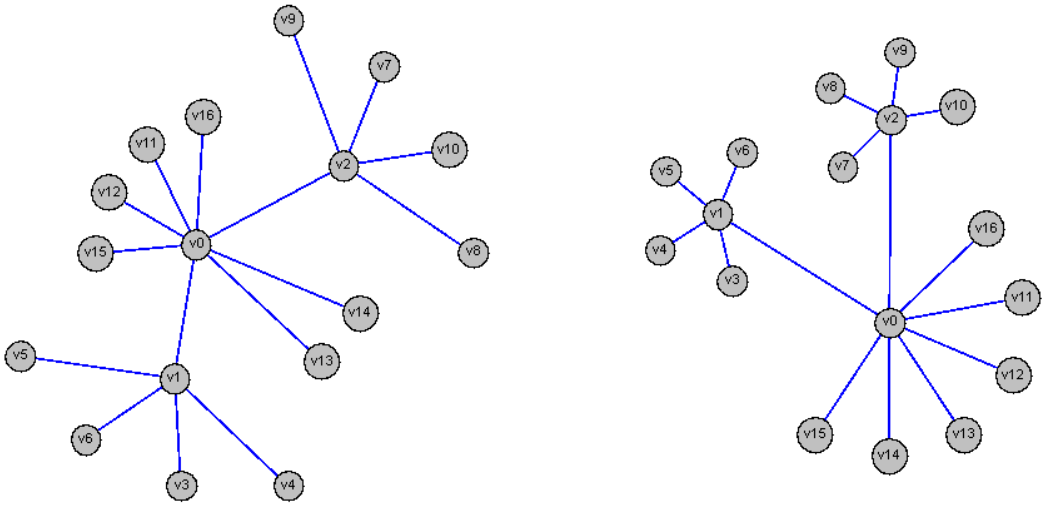
4. Proverava se da li je graf stablo. Ukoliko jeste, vrši se detekcija najpogodnijeg algoritma specijalno dizajniranog za dati tip grafova. Algoritam baziran na nivoima generiše tradicionalni prikaz stabla, kakvog bi većina korisnika verovatno i sama kreirala da ručno raspoređuje čvorove. Međutim, taj pristup može dovesti do veoma širokog, a samim tim i nedovoljno preglednog crteža, što je demonstrirano na slici 6.2. Za takvo stablo primećuje se da se dominantno sastoji iz listova, te da bi radijalno crtanje ili metoda balona bili bolje opcije. Metoda balona nešto bolje izražava strukturu opisanih stabala, što se može uočiti na slici 6.3, gde su prikazani crteži grafa sa slike 6.2, dobijeni prvo radijalnom, a potom i metodom balona.



Slika 6.2: Široko nivovsko stablo

Takođe, crtež stabla dobijen nivovskim pristupom može biti previše širok ako je reč o binarnom izbalansiranom stablu. U tom slučaju se konfigurabilni parametar algoritma koji određuje razdaljinu između čvorova sa zajedničkim roditeljom posebno podešava tako da se pomenuto rastojanje smanji. Sve opisano se može sumirati na sledeći način:

- Za stablo kod koga je veliki procenat čvorova lišće, primenjuje se metoda balona.
- Za ostala stabla, primenjuje se nivovski algoritam.
- Ukoliko je stablo binarno i izbalansirano, konfiguriše se (smanjuje se) rastojanje između čvorova koji dele roditelja.



Slika 6.3: Stablo sa prethodne slike raspoređeno prvo radijalnom, pa metodom balona

5. Ukoliko graf nije stablo, proverava se da li bi mogao biti nacrtan hijerarhijskim algoritmom sa zadovoljavajućim ishodom. Provera je definisana uz oslonac na kreiranje DFS stabla. Naime, ova konstrukcija razvrstava grane na grane stabla i povratne. Ukoliko je udeo povratnih grana u celokupnom skupu grana grafa manji od nekog zadatog procenta, primenjuje se hijerarhijski algoritam. Na slici 6.4 prikazana su dva crteža kreirana hijerarhijskim algoritmom. Dva grafa imaju iste čvorove, ali drugi ima znatno više grana. Uočava se da prvi graf ima svega tri povratne grane, a drugi nekoliko puta više, te dok je prvi crtež jasan i pregledan, za drugi se to ne može tvrditi. Prema navedenom kriterijumu, za prvi graf zaista bi bio odabran hijerarhijski algoritam, za razliku od drugog.

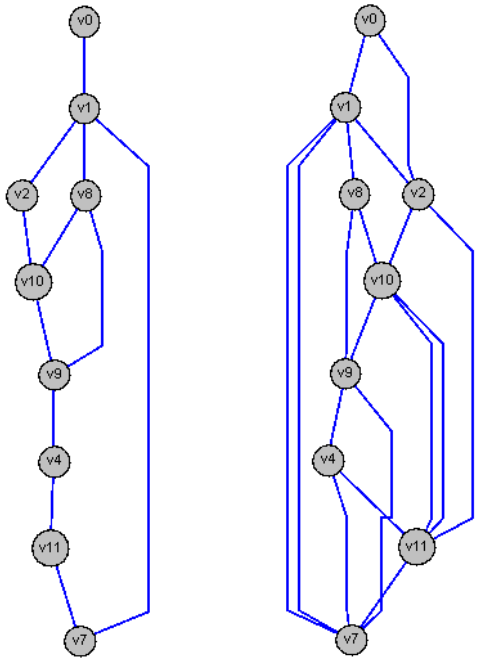
6. Ukoliko prethodne provere nisu dale pozitivan odgovor, posmatra se njegova veličina. Naime, svi algoritmi koji bi mogli biti izabrani u prethodnim razmatranjima veoma su efikasni ukoliko se primene nad pogodnim ulazom, te ne treba brinuti o performansi. Međutim, u slučajevima koji slede ona postaje bitan faktor. Dakle, ako graf ima više od predefinisanog broja čvorova, recimo hiljadu, primenjuje se efikasni ISOM silom usmereni algoritam.

7. Ako graf ne zadovoljava nijedan ranije definisani uslov i nije previše veliki, ispituje se da li se može nacrtati planarnim pravolinijskim algoritmom. Ukoliko ima planarni konveksni crtež, on se određuje.

8. Preostali grafovi crtaju se organskim silom usmerenim algoritmom, koji je izdvojen kao ne preterano efikasan, ali kao algoritam koji proizvodi posebno lepe crteže u opštem slučaju. Njegovi parametri se podešavaju tako da se maksimalno uredi crtež.

Za svaki od odabranih algoritama parametri se podešavaju tako da se pre svega izbegne preklapanje čvorova, odnosno, da se njihove veličine izmu u obzir. Na primer, rastojanje između nivoa kod nivovskog algoritma, kao i čvorova istog nivoa ili slojeva hijerarhijskog algoritma podešavaju se na osnovu maksimalne visine i širine čvorova.

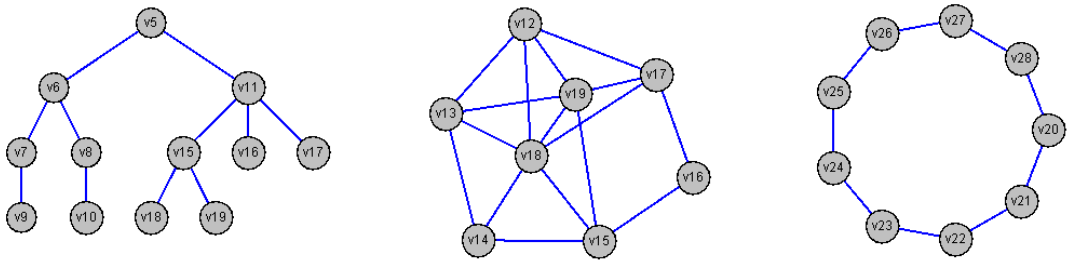
Primećuje se da niti simetrični, niti ortogonalni algoritam nisu razmatrani kao mogući automatski izbori. Razlog za takvu odluku nalazi se u činjenici da je bez



Slika 6.4: Dva crteža grafova nacrtana hijerarhijskim algoritmom

poznavanja konteksta i preferenci korisnika teško preporučiti bilo koji od algoritama tih klasa. Naime, simetrija kao estetski kriterijum je po nekolicini vršenih istraživanja slabije rangirana nego minimizacija broja preseka grana. Stoga postupak automatskog izbora prednost daje upravo drugom spomenutom kriterijumu, koji u obzir uzimaju svi algoritmi koji ulaze u skup kandidata. Slično, ortogonalnost grana dovodi do većeg broja njihovog preloma i često manje kompaktnih crteža. Međutim, konkretna primena, kao što je recimo crtanje električnih šema, može nametnuti optimalnost upravo takvog algoritma. Slično, korisnik može lično doživeti simetriju najbitnijim kriterijumom. Prema tome, automatski izbor na osnovu obeležja nije jedina mogućnost. Naime, da od korisnika ne bi bila preuzeta sva kontrola, osmišljen je jezik specifičan za domen opisa crteža grafa na osnovu kojeg se takođe automatski bira podoban algoritam.

Konačno, treba napomenuti da se, ukoliko se graf sastoji iz više nepovezanih komponenti, izbor i primena algoritma posebno sprovodi za svaku od njih, te da se one potom takođe raspoređuju tako da ne dođe do preklapanja. Jedan takav primer prikazan je na slici 6.5.



Slika 6.5: Automatsko raspoređivanje elemenata grafa koji se sastoji iz više odvojenih komponenti

Poglavlje 7

Jezik specifičan za domen opisa crteža grafa

Potpuno automatizovani izbor i primena algoritma za raspoređivanje elemenata datog grafa, opisan u prethodnom poglavlju, može u određenim situacijama biti koristan, ali se korisniku u potpunosti oduzima mogućnost učešća u tom procesu. Kao što je već diskutovano, korisnikove lične preference ili specifičnost konteksta u kom se algoritam treba primeniti mogu dovesti do situacije gde algoritam izabran prema osobinama grafa nije odgovarajući. Uvođenje jezika za cilj ima premošćavanje upravo takvih situacija, odnosno, nalaženja kompromisa između korisnikovog izbora i konfiguracije algoritma bez ikakve asistencije i punog automatizma. Naime, jezik dozvoljava korisnicima, između ostalog, opis poželjnih osobina crteža, na osnovu kojih se bira, konfiguriše i primenjuje algoritam čiji rezultati su najbliži njegovim željama. Jezik takođe ima za cilj povećavanje efikasnosti programera pri pisanju koda za poziv određenih algoritama. U nastavku poglavlja biće dat detaljan opis jezika, prolaskom kroz sve faze njegovog razvoja.

7.1 Razvojne faze jezika

Razvoj jezika specifičnih za domen sastoji se iz sledećih faza: odluka, analiza, dizajn, implementacija, tesitranje, raspoređivanje (*deployment*) i održavanje [107], [162], [163]. U ovoj sekciji će biti prezentovan JSD za specifikaciju raspoređivanja elemenata grafa i opisana svaka od njegovih razvojnih faza.

7.1.1 Odluka

Cilj faze odluke jeste davanje odgovora na pitanje da li se isplati razvijati JSD ili ne. Drugim rečima, da li pogodnosti koje pruža novi jezik mogu kompenzovati troškove njegovog razvoja. Na ovo pitanje je dat pozitivan odgovor iz sledećih razloga:

1. Jedan od glavnih ciljeva *GRAD* biblioteke jeste pružanje opšteg načina njene integracije sa bilo kojim grafčkim editorom. Ovo se može postići upotrebom jezika specifičnog za domen. U odnosu na rešenje poput konfiguracionog dijaloga, jezik se može iskoristiti unutar proizvoljnog editora razvijenog na programskom jeziku *Java*, nezavisno od toga da li je reč o desktop ili veb aplikaciji, ali i nezavisno od korišćenog razvojnog okvira. Takođe, JSD se može deifinisati tako da podrži i složenije koncepte, uključujući logičke operatore, što je teško postići samo pomoću konfiguracionog dijaloga.

2. JSD može korisnicima grafičkih editora koji nisu niti programeri, niti upoznati sa algoritmima za crtanje grafova, omogućiti deskriptivnu specifikaciju rasporeda elemenata dijagrama. Dakle, poželjnih i nepoželjnih osobina, poput simetrije, ortogonalnosti ili ujednačenog toka grana. Korisnici tada ne bi morali trošiti vreme na isprobavanje različitih podržanih algoritama i nameštanje njihovih parametara.

3. Iako se svaki algoritam *GRAD* biblioteke može konfigurisati i pozvati direktnim pisanjem koda koji se oslanja na *API* (*Application Programming Interface*) pozive ove biblioteke, upotrebom jezika isto se može postići sa znatno manje linije koda (uporediti listing 7.11 sa listingom 7.12, ili listing 7.13 sa listingom 7.14). Takođe, sprovedene studije pokazale su da programeri brže i efikasnije mogu razumeti program kada se koristi JSD, a ne programski jezik opšte namene [164].

4. Moderne biblioteke i metajezici programere snabdevaju alatima pomoću kojih izuzetno brzo mogu razviti sopstveni eksterni JSD [165]. Jedan takav primer je *textX* metajezik pomoću kojeg je kreiran JSD *GRAD* biblioteke.

7.1.2 Analiza domena

Detaljna analiza domena mora prethoditi dizajnu i implementaciji novog JSD [163]. Cilj ove faze jeste definisanje domena i prikupljanje potrebnih informacija, uz opciono kreiranje domenskog modela. Data faza se u većini slučajeva sprovodi neformalno [166], što je bio slučaj i tokom razvoja jezika *GRAD* biblioteke. Domen koji je od interesa jesu teorija grafova i crtanja grafova, koji su uvedeni u prethodnim poglavljima disertacije.

7.1.3 Dizajn jezika

Dizajn jezika uključuje definisanje konstrukata jezika i njegove semantike [163]. Takođe, u ovoj fazi se definiše odnos novog jezika sa drugim jezicima. Odnosno, potrebno je odlučiti da li će jezik biti interni ili eksterni JSD. Kako je jedan od ciljeva razvijenog jezika upotreba i kod korisnika koji nisu programeri, nametnuo se izbor eksternog jezika. U nastavku sekcije biće predstavljena sintaksa i semantika datog JSD, kao i *textX* metajezik na kojem je zasnovan njegov razvoj.

textX alat

textX je metajezik za specifikaciju JSD u programskom jeziku *Python*. Na osnovu opisa gramatike, *textX* automatski konstruiše metamodel u formi *Python* klasa, kao i parser za novokreirani jezik. Parser parsirajući jezik gradi graf *Python* objekata, odnosno model, u skladu sa metamodelom. Iako ciljni jezik ovog alata nije *Java*, pozivanje *Python* koda iz pomenutog jezika ne predstavlja problem, o čemu će više reči biti u sekciji 7.2.

Osnovni gradivni blokovi *textX* jezika su pravila. Pravila su oblika prikazanog u listingu koda 7.1:

Listing 7.1: Izgled *textX* pravila

```
Hello:
  'hello' who=ID;
;
```

Telo pravila sastoji se iz izraza koji se specifikuju upotrebom osnovnih operatora i izraza, od kojih su sledeći korišćeni u razmatranom jeziku:

- Poklapanja, koja su izrazi najnižeg nivoa. Mogu biti tekstualna, kada se prepoznaje ulazni tekst, napisan unutar jednostrukih ili dvostrukih navodnika, i regularni izrazi, kada se prepoznaje svaki ulaz koji ih zadovoljava.

- Sekvence, ili više drugih *textX* izraza navedenih jedan iza drugog razdvojenih razmakom.

- Uređeni izbor, koji se navodi kao skup izraza razdvojenih operatorom "|". Operator pokušava da izvrši prepoznavanja ulaza s leva-na-desno i koristi prvi izraz kod koga se operacija uspešno završi.

- Opciona prepoznavanja, kojima se definiše da se izraz može, a ne mora pojaviti. Koristi se operator "?".

- Ponavljanja, specifikovana operatorima "*" ili "+". Prvi omogućava prepoznavanje sadržanog izraza nula ili više, a drugi jedan ili više puta.

- Reference na druga pravila.

- Dodela, koja je od ključne važnosti za kasniju obradu modela i koja se definiše upotrebom operatora "=". Svaka dodela rezultuje atributom meta-klase kreirane na osnovu datog pravila. Naziv atributa predstavlja levu stranu dodele, dok je desna referenca na druga pravila ili bazično prepoznavanje. Osim običnih dodela, postoji još nekoliko njihovih tipova:

  - Logička ("?="), gde atribut dobija vrednost *True* ukoliko se prepoznavanje desi, inače *False*.
  - Nula ili više ("*="), gde je atribut lista, u koju se dodaje svaki prepoznati objekat sa desne strane pravila. Lista može ostati prazna, ako nijedno prepoznavanje nije izvršeno.
  - Jedan ili više ("*="), koje je isto kao i prethodno, samo se mora izvršiti bar jedno prepoznavanje.

U okviru pravila ponavljanja (bilo običnog, bilo kombinovonog za dodelom), mogu se koristiti modifikatori ponavljanja. Navode se u okviru uglastih zagrada ("[" i "]"). Označavaju da se između uzastopnih prepoznavanja ulaza treba javiti određeni niz karaktera, koji može biti zadat i preko regularnog izraza. Detaljan opis *textX* gramatike može se naći na [167].

U okviru *textX*-a, definisana su posebna ugrađena pravila koje predstavljaju bazične tipove. To su:

1. ID, reprezentovano identifikatorom koji se može sastojati iz slova, cifara i donjih crta.

2. INT, koje definiše cele brojeve.

3. FLOAT, koje definiše realne brojeve.

4. BOOL, koje obuhvata reči *True* i *False*.

5. STRING, niz karaktera naveden između navodnika.

Sintaksa jezika

Jezik je dizajniran imajući u vidu i eksperte i one koji nemaju puno predznanja, nudeći ukupno četiri načina specifikacije raspoređivanja. Od najjednostavnijeg do najkompleksnijeg, to su:

1. Definisanje stila raspoređivanja (kružno, hijerarhijski, simetrično, automatski itd.).

2. Zadavanje jednog ili više estetskih kriterijuma u skladu sa kojima bi raspoređivanje trebalo biti sprovedeno.

3. Navođenje izraza koji se sastoje iz estetskih kriterijuma i logičkih (i, ili, ne) operatora.

4. Direktan izbor algoritma uz opciono podešavanje jednog ili više njegovih parametara.

Poslednji način korisnicima dobro upoznatim sa algoritmima za crtanje grafova obezbeđuje dodatnu alternativu za primenu željenog algoritma. Dakle, umesto pisanja koda kojim bi se algoritam podesio i pozvao, moguće je putem definisanja uputstava u skladu sa jezikom postići isto. Prestale dve opcije teže olakšavanju procesa ostalim korisnicima. Prvi način je najbrži, podržavajući i pozivanje automatskog raspoređivanja na drugačiji način, dok su drugi i treći ekspresivniji i interesantniji. Estetski kriterijumi koji se nalaze u njihovom korenu predstavljaju intuitivan koncept sa jedne, a blizak algoritmima za crtanje grafova sa druge. Kod drugog načina, korisnik navodi niz kriterijuma koje bi želeo da vidi na crtežu, pri čemu će pri nalaženju adekvatnog algoritma prvonavedeni kriterijum imati najviši prioritet, potom drugi itd. Sa druge strane, treći način dozvoljava pisanje kompleksnijih matematičkih izraza, kojima se može specifikovati• i nepoželjnost jednog ili više kriterijuma. Jezik podržava definisanje različitih načina raspoređivanja za različite podgrafove, kao i primenu jednog algoritma nad celim grafom.

Osnovna pravila jezika odražavaju sve prethodno spomenuto i prikazana su u listingu koda 7.2.

Listing 7.2: Osnovna pravila jezika

```
Layout:
        LayoutGraph | LayoutSubgraphs
;
LayoutGraph:
        'lay' 'out' 'graph' layoutType = LayoutEnum
;

LayoutSubgraphs:
        'lay' 'out' layoutSubgraphs += LayoutSubraph[';']
;

LayoutSubraph:
        (subgraph = Subgraph |   'others')
         layoutType = LayoutEnum
;
```

```
Subgraph:
        'subgraph' 'containing'? vertices += Vertex[',']
;

LayoutEnum:
        LayoutStyle | AestheticCriteriaMath |
        AestheticCriteria | LayoutAlgorithm
;

LayoutStyle:
        'using'? 'style' style = LayoutStyleEnum
;

LayoutStyleEnum:
        'automatic' | 'circular' | 'tree' |
         'hierarchical' | 'symmetric' | 'general'
;

LayoutStyle:
        'style' style = LayoutStyleEnum
;

LayoutStyleEnum:
        'automatic' | 'circular' | 'tree' |
         'hierarchical' | 'symmetric' | 'general'
;

AestheticCriteria:
        'criteria' aestheticCriteria += AestheticCriterion[',']
;

AestheticCriterion:
        EdgeCrossings | MinimumAngles | MinimumBands |
        UniformFlow | Symmetry  | NodeDistribution |
        EdgeLengths | EdgeVariation
;

LayoutAlgorithm:
        'algorithm' algorithm = LayoutAlgorithmEnum
;

LayoutAlgorithmEnum:
        TreeAlgorithm | StraightLineAlgorithm |
        HierarchicalAlgorithm |
        CircularAlgorithm | SymmetricAlgorithm |
        ForceDirectedAlgorithm |
        BoxAlgorithm
;

TreeAlgorithm:
```

```
        RadialTreeAlgorithm | LevelBasedTreeAlgorithm |
        CompactTreeAlgorithm |
        NodeLinkTreeAlgorithm | BalloonTreeAlgorithm
;

ForceDirectedAlgorithm:
        KamadaKawai | FruchtermanReingold | Spring |
        FastOrganic | Organic
;

AestheticCriteriaMath:
        expression = CriteriaExpression
;

CriteriaFactor:
        negative ?= 'not' ( criterion = AestheticCriterion |
         ('(' expression = CriteriaExpression ')'))
;

CriteriaTerm:
        termStartFactor = CriteriaFactor
        termFactors *=  CriteriaAndFactor
;

CriteriaAndFactor:
        'and' factor = CriteriaFactor
;

CriteriaExpression:
        expressionStartTerm = CriteriaTerm
        expressionTerms *= CriteriaOrTerm
;

CriteriaOrTerm:
        'or' term = CriteriaTerm
;
```

Može se primetiti da svakoj od četiri navedene opcije za raspoređivanje odgovara jedno pravilo. Prvom i najjednostavnijem od njih pridruženo je i najmanje kompleksno pravilo, `LayoutStyle`, u okviru kojeg se specifikuje opšti stil konstrukcije crteža. Druga alternativa adresirana je pravilom `AestheticCriterion`, koje dozvoljava navođenja jednog ili više estetskih kriterijuma u proizvoljnom rasporedu. Za pojedine kriterijume, kod kojih to ima smisla, opciono se mogu definisati određeni parametri. Na primer, smer ujednačenog toka, što je prikazano u listingu koda 7.3. Iz ovog primera uočava se i da su određene reči takođe opcione, ukoliko ne nose posebno bitne informacije, ali mogu povećati čitljivost i preglednost specifikacije. Naime, kako je, na primer sama reč *flow* dovoljna za identifikaciju kriterijuma, *uniform* nije nužno navesti, što bi odgovaralo korisnicima koji preferiraju kraće zapise. Pojedinim korisnicima duži i deskriptivniji zapis bi mogao ipak biti primarni izbor, te je i to uzeto u obzir.

Listing 7.3: Pravilo za definisanje estetskog kriterijuma ujednačenog toka

```
UniformFlow:
        'uniform'? criterion = 'flow'
        ('direction' '=' direction = Orientation)?
;

Orientation:
        'left' | 'right' | 'up' | 'down'
;
```

Treći mogući način automatskog izbora algoritma definisan je pravilom `AestheticCriteriaMath`. Dato pravilo predviđa navođenje logičkih izraza, u okviru kojih se pojavljuju estetski kriterijumi. Jedan primer prikazan je u listingu 7.4.

Listing 7.4: Primer specifikacije u opisanom jeziku koji obuhvata estetske kriterijume povezane logičkim operatorima

```
lay out graph not(symmetry and angle)
              or planarity or edge crossings
```

Prikazani primer označava da crtež ili treba da zadovoljava kriterijum planarnosti, ili minimizacije broja preseka grana, ili da ne zadovoljava ni simetriju ni kriterijum maksimizacije minimalnih uglova.

Poslednji podržani metod specifikacije načina raspoređivanja jeste direktno navođenje algoritma uz njegovu opcionu konfiguraciju. Dakle, ako algoritam poseduje parametre, mogu se navesti vrednosti jednog ili više njih. Parametri se mogu specifikovati u proizvoljnom redosledu. Primer pravila kojim se definiše jedan konkretan algoritam prikazan je u listingu koda 7.5. Sva konfigurabilna obeležja algoritama izdvojena su u posebna pravila, kako bi se izbeglo zadavanje striktnog redosleda njihovog navođenja i kako bi ih mogli referencirati različiti algoritmi u slučaju da imaju određeni podskup zajedničkih parametara.

Listing 7.5: Pravilo za definisanje algoritma za nivovsko crtanje stabala

```
LevelBasedTreeAlgorithm:
        name = 'level' 'based' 'tree'
        properties *= LevelBasedTreeAlgorithmProperty[',']
;

LevelBasedTreeAlgorithmProperty:
        HorizontalTreeProperty | VerticalTreeProperty
;

HorizontalTreeProperty:
        'horizontal' 'distance'? '=' xDist = INT
;

VerticalTreeProperty:
        'vertical' 'distance'? '=' yDist = INT
;
```

U listingu koda 7.6 prikazan je jedan primer specifikacije na ovom jeziku.

Listing 7.6: Primer specifikacije raspoređivanja u opisanom jeziku

```
   lay out
```

```
        subgraph v 1, v 2 using style symmetric;
        subgraph v 3, v 4 conforming to planarity, bends;
        others using algorithm Kamada–Kawai;
```

Semantika jezika

Semantika nekog jezika može biti specifikovana formalno, korišćenjem metoda poput denotacione semantike i apstraktnih mašina prelaza stanja, ili neformalno, prirodnim jezikom sa opcionim primerima konkretnih programa [107], [163]. Implementirani JSD je jednostavan jezik, sa visokim nivoom apstrakcije. Programi napisani u njemu se transformišu u *Java* kod koji se oslanja na *GRAD API* za raspoređivanje elemenata grafova. Stoga, semantička analiza jezika najefikasnije može biti opisana putem druge metode.

U nastavku će preko tabele 7.1 biti prikazana pojašnjenja glavnih pravila jezika, da bi potom bio dat jedan primer mapiranja unosa napisanog u skladu sa JSD na *API* pozive *GRAD* biblioteke za svaku od četiri raspoložive metode definisanja raspoređivanja.

Kao što je već viđeno, ulazi (programi) napisani u razvijenom JSD se transformišu u *Java* kod koji se većinom sastoji iz *API* poziva *GRAD* biblioteke. Naredni listinzi koda predstavljaju parove koda napisanog u jeziku specifičnom za domen i rezultujućeg *Java* koda. Listing 7.7 prikazuje primer JSD programa napisanog u skladu sa pravilom `LayoutAlgorithm`, dok je odgovarajući *Java*/*GRAD* kod obuhvaćen listingom 7.8. Slično, listing koda 7.9 sadrži primer specifikacije raspoređivanja elemenata grafa primenom pravila `LayoutStyle` dok listing 7.10 prikazuje *Java* kod koji će se izvršiti. Konačno, listinzi koda 7.11 i 7.12, kao i 7.13 i 7.14 jesu parovi ulaza napisanih pridržavajući se pravila `AestheticCriteria` i `AestheticCriteriaMath` jezika, i rezultujućih *API* poziva.

Listing 7.7: Ulaz u skladu sa `LayoutAlgorithm` pravilom

```
radial tree(horizontal distance = 5, vertical distance = 20)
```

Listing 7.8: *Java* kod koji poziva raspoređivanje u skladu sa specifikacijom prikazanom u listingu 7.7

```
//izaberi algoritam
LayoutAlgorithms algorithm = LayoutAlgorithms.RADIAL_TREE;
//postavi obelezja
GraphLayoutProperties layoutProperties = new GraphLayoutProperties();
layoutProperties.setProperty(RadialTreeProperties.X_DISTANCE,5);
layoutProperties.setProperty(RadialTreeProperties.Y_DISTANCE,20);
//inicijalizuj rasporedjivac (layouter) pretpostavljajuci da su
//vertices i edges liste koje sadrze elemente grafa
Layouter<GraphVertex, GraphEdge> layouter = new Layouter<>(
      vertices, edges, algorithm, layoutProperties);
Drawing<GraphVertex, GraphEdge> drawing = layouter.layout();
```

Listing 7.9: Ulaz u skladu sa `LayoutStyle` pravilom

```
lay out graph using style circular
```

Listing 7.10: *Java* kod koji poziva raspoređivanje u skladu sa specifikacijom prikazanom u listingu 7.10

```
//nadji najbolji algoritam na osnovu obelezja grafa,
//ali uzmi u obzir samo one koji pripadaju
```

JSD unos	Semantika
`lay out graph using algorithm NazivAlgoritma (obeležje1 = vrednost1, obeležje2 = vrednost2)`	Izaberi algoritam naziva *NazivAlgoritma* i postavi vrednost obeležja naziva *obeležje1* na *vrednost1*, *obeležje2* na *vrednost2*. Izvrši izabrani algoritam.
`lay out graph using style NazivStila`	Automatski odredi najpogodniji algoritam. Ako navedeni stil nema vrednost *automatic*, ograniči skup razmatranih algoritama samo na one koji pripadaju odgovarajućoj grupi. Ako je stil *circular*, *hierarchical*, *symmetric* ili *tree*, izaberi algoritam koji pripada klasi kružnih, hijerarhijski ili simetričnih algoritama, odnosno algoritama za crtanje stabala. Ako je naveden stil *general* (opšti), izaberi silom usmereni algoritam. Izvrši dati algoritam.
`lay out graph conforming to criteria kriterijum1, kriterijum2, kriterijum3`	Izaberi algoritam koji se može primeniti nad datim grafom, tako da dobijeni crtež poseduje karakteristike koje kao poželjne izdvaja estetski kriterijum *kriterijum1*, te ako je moguće i koje favorizuju i *kriterijum2* i *kriterijum3*. Izvrši izabrani algoritam.
`lay out graph not?(kriterijum1 and kriterijum2) or not?(kriterijum3 and kriterijum4)`	Izaberi algoritam koji zadovoljava ili i kriterijum1 i kriterijum2 (ili ne zadovoljava nijedan od njih, u zavisnosti od toga da li je naveden operator negacije ili ne), ili i kriterijum3 i kriterijum4. Izvrši algoritam.

Tabela 7.1: JSD izrazi i njihova semantika

```
//klasi odredjenoj na osnovu specificaranog stila
LayoutAlgorithms algorithm =
    LayoutPicker.pickAlgorithm(graph, "circular");
GraphLayoutProperties layoutProperties =
                DefaultGraphLayoutProperties
               .getDefaultLayoutProperties(algorithm, graph));
Layouter<GraphVertex, GraphEdge> layouter = new Layouter<>(
    vertices, edges, algorithm, layoutProperties);
Drawing<GraphVertex, GraphEdge> drawing = layouter.layout();
```

Listing 7.11: Ulaz u skladu sa `LayoutCriteria` pravilom

```
lay out graph conforming to planarity, symmetry, flow
```

Listing 7.12: *Java* kod koji poziva raspoređivanje u skladu sa specifikacijom prikazanom u listingu 7.11

```
List<AestheticCriteria> criteria =
        new ArrayList<AestheticCriteria>();
```

```
criteria.add(AestheticCriteria.PLANAR);
criteria.add(AestheticCriteria.SYMMETRIC);
criteria.add(AestheticCriteria.UNIFORM_FLOW);
LayoutAlgorithms algorithm = LayoutPicker.
     pickAlgorithm(graph, criteria);
GraphLayoutProperties layoutProperties  =
     DefaultGraphLayoutProperties.
     getDefaultLayoutProperties(algorithm, graph));
Layouter<GraphVertex, GraphEdge> layouter = new Layouter<>(
    vertices, edges, algorithm, layoutProperties);
Drawing<GraphVertex, GraphEdge> drawing = layouter.layout();
```

Listing 7.13: Ulaz u skladu sa `AestheticCriteriaMath` pravilom

```
lay out graph not(symmetry and angle) or edge crossings
```

Listing 7.14: *Java* kod koji poziva raspoređivanje u skladu sa specifikacijom prikazanom u listingu 7.13

```
List<Pair<List<AestheticCriteria>, List<AestheticCriteria>>>
  orPairs = new ArraList<Pair<List<AestheticCriteria>,
    List<AestheticCriteria>>>();
List<AestheticCriteria> positiveCriteria =
    new ArrayList<AestheticCriteria>();
positiveCriteria.add(AestheticCriteria.MINIMAL_EDGE_CROESSES);
List<AestheticCriteria> negativeCriteria =
  new ArrayList<AestheticCriteria>();
negativeCriteria.add(AestheticCriteria.SYMMETRIC);
negativeCriteria.add(AestheticCriteria.MINIMUM_ANGLES);
orPairs.add(new Pair<List<AestheticCriteria>,
        List<AestheticCriteria>>(positiveCriteria, negativeCriteria));

LayoutAlgorithms algorithm =
        LayoutPicker.pickAlgorithm(graph, orPairs);
GraphLayoutProperties layoutProperties  =
      DefaultGraphLayoutProperties.
  getDefaultLayoutProperties(algorithm, graph));
Layouter<GraphVertex, GraphEdge> layouter = new Layouter<>(
  vertices, edges, algorithm, layoutProperties);
Drawing<GraphVertex, GraphEdge> drawing = layouter.layout();
```

Implementacija

Implementacija je faza razvoja jezika specifičnih za domen kada se bira i realizuje najpogodniji način njihovog razvoja. Najčešće primenjene metode obuhvataju razvoj interpretera, kompajlera i pretprocesora [107]. Implementirani jezik koristi interpreter, čije kreiranje se oslanja na mogućnost *textX* metajezika da napravi graf objekata (model) na osnovu ulaza koji je u skladu sa JSD. Kada se formira model, interpreter ga transformiše u odgovarajuće *API* pozive. Na primer, nazivi i obeležja algoritama se konvertuju u odgovarajuće vrednosti `LayoutAlgorithms` enumeracije i elemente `GraphLayoutProperties` mape. Dok implementacija takvih transformacija nije težak zadatak, složenost jezika enkapsulirana je metodama `PickAlgorithm`. Unutar njih realizovana je selekcija odgovarajućeg algoritma na osnovu obeležja grafa i unosa korisnika. Detalji su izloženi u sekciji 7.2.

Testiranje i raspoređivanje

Testiranje i raspoređivanje (*deployment*) su sledeće faze razvoja jezika specifičnih za domen. Kao što naziv sugeriše, evaluacija JSD se izvršava u ovoj fazi, dok raspoređivanje predstavlja vremenski trenutak kada se počnu koristiti aplikacije kreirane upotrebom jezika [163]. *GRAD* biblioteka i njen JSD su integrisani sa spomenutim *Kroki* alatom, dok je evaluacija sprovedena kroz korisničku studiju. Ova studija je pokazala da bi više učesnika preferiralo da koristi dati jezik za pozivanje raspoređivanja elemenata dijagrama, nego bilo koju drugu od ponuđenih opcija.

Održavanje

Održavanje je poslednja faza konstrukcije jezika, koja postaje aktuelna kada se pojave novi zahtevi. Kako novi algoritmi za automatsko raspoređivanje budu implementirani, tako će nastajati potreba da se proširi JSD. Ovo bi podrazumevalo dodavanje novih pravila gramatike, implementaciju novih transformacija, kao i unapređenje procedure opisane u sekciji 7.2, kako bi se i novi algoritmi uzeli u razmatranje. Imajući u vidu sposobnost *textX* alata da kreira i metamodel i parser na osnovu gramatike, kao i njegovu dinamičku prirodu, proširenje jezika ne predstavlja izazov.

7.2 Integracija jezika sa *GRAD* bibliotekom

Na osnosu opisa koji poštuje definisana pravila jezika automatski se bira, konfiguriše i primenjuje odgovarajući algoritam. Kao što je spomenuto, *textX* na osnovu dobijenog konkretnog programa na datom jeziku, odnosno, modela, konstruiše objektni model, koji se dalje može koristiti za interpretaciju ili generisanje koda. Model za primer iz listinga koda 7.15 prikazan je na slici 7.1.

Listing 7.15: Ulaz u skladu sa jezikom za koji se prikazuje model

```
lay out graph not(symmetry and angle) or planarity or edge crossings
```

U konkretnom slučaju, radi se interpretacija, radi formiranja odgovarajućeg modela na ciljnom programskom jeziku, *Javi*, u kojem je implementiran ostatak biblioteke. Sprega jezika opšte namene koji su faktori u tekućem razmatranju realizovan je upotrebom *Jython* [168] implementacije *Python* jezika za *Java* platformu.

Naredni koraci predstavljaju izbor algoritma upravljan željama korisnika. Postupak je do neke mere sličan opisanom u prethodnom poglavlju, ali osobine grafa nisu jedine smernice koje se koriste. Najkompleksnija od tri opcije koje jezik podržava za specifikaciju načina za raspoređivanje ujedno je i najjednostavnija za implementaciju. Naime, algoritam koji je korisnik direktno naveo biva izvršen, pri čemu se parametri podešavaju ili prema zahtevanim vrednostima, ukoliko ih je korisnik specifikovao, ili dobijaju podrazumevane. Ni raspoređivanje prema zadatom stilu nije posebno složeno. Dakle, ako je korisnik zadao automatski stil, algoritam biva izabran na osnovu obeležja grafa, ukoliko je naveo kružni, primenjuje se kružni algoritam i slično. Sa druge strane, u pozadini izbora prema estetskim kriterijumima ima više logike, te će realizacija ove mogućnosti biti detaljnije opisana.

U osnovi ove funkcionalnosti nalazi se činjenica da je većina algoritama za crtanje grafova dizajnirana vodeći računa o jednom estetskom kriterijumu ili više njih kriterijuma. Na primer, algoritmi za planarno pravolinijsko crtanje akcenat stavljaju na planarnost, konveksni i ortogonalni posvećuju pažnju i uglovima, algoritmi za crtanje stabala vode računa o dužinama grana, površini crteža itd. Dakle, osnovna ideja

Algoritam	Planarnost	Tok	Simetrija	Distribucija čvorova
Organski	±	−	±	+
Radijalno	+	−	±	±
Nivovsko stablo	+	+	±	−
Balon stablo	+	−	±	−
Hijerarhijski	±	+	−	−
Simetrični	−	−	+	−
Tutov	+	−	−	−
Konveksni	+	−	−	−
Ortogonalni	+	−	−	−
Kružni	±	−	−	−

Tabela 7.2: Algoritmi i opšti estetski kriterijumi

Algoritam	Prelomi	Ugao	Varijacija	Dužine
Organski	+	−	±	±
Radijalno	+	−	+	+
Nivovsko stablo	+	−	+	+
Balon stablo	+	−	−	−
Hijerarhijski	±	−	−	−
Simetrični	+	−	−	−
Tutov	+	−	−	−
Konveksni	+	±	−	−
Ortogonalni	±	+	−	−
Kružni	+	−	−	−

Tabela 7.3: Algoritmi i estetski kriterijumi vezane za grane

jeste nalaženje algoritma koji može ispuniti, bar do neke mere, sve navedene estetske kriterijume ili većinu njih. Međutim, treba napomenuti da nije svaka kombinacija tih kriterijuma kompatabilna. Na primer, ako graf nije stablo, nije moguće izdvojiti algoritam koji će obezbediti i simetriju i planarnost. Stoga je u nekim situacijama nužno praviti kompromise, te izabrati algoritam koji najbliže zadovoljava navedeni skup kriterijuma. Prioritet se daje ranije navedenim kriterijumima, čime se teži maksimalnom zadovoljavanju prvog, te ostalih koliko je to moguće. Ako se, recimo, traže i simetrija i planarnost, nalazi se algoritam koji oba zadovoljava do neke mere, pri čemu se simetriji daje prioritet. U tabelama 7.2 i 7.3 objedinjene su informacije povezane s algoritmima i stepenon težnje estetskim kriterijumima. Znak ± označava da algoritam do neke mere poštuje kriterijum, ne garantujući da će za svaki generisani crtež proizvoljnog grafa on biti zadovoljen.

Primećuje se da nisu navedeni svi silom usmereni algoritmi, već je izdvojen njihov predstavnik koji se najbolje pokazuje sa stanovišta estetike. Uzimanjem ovih podataka u obzir, postupak izbora algoritma definisan je na sledeći način:

- Ukoliko je naveden samo jedan estetski kriterijum, traži se algoritam koji je dizajniran imajući upravo njega u vidu:
  - Ukoliko postoji samo jedan takav algoritam i može se primeniti nad datim grafom, on biva izabran. Na primer, za simetrični estetski kriterijum primenjuje se simetrični algoritam.

  - Ako je moguća primena samo jednog algoritma, koji do neke mere zadovoljava kriterijum, odnosno, nema bolje opcije, on se bira. Na primer, ako je specifikovan kriterijum minimalne varijacije dužina grana, a graf nije stablo, izdvaja se organski silom usmereni algoritam.
  - U slučaju da veći broj algoritama vodi računa o datom kriterijumu, uzimaju se u obzir i osobine grafa. Slično kao kod automatskog izbora, za primenu se izdvaja onaj koji je dizajniran baš za graf poput tekućeg. Recimo, ako je planarnost najbitniji faktor, a graf je stablo, najbolja opcija jeste jedan od algoritama za crtanje stabala koji garantuje planarnost.
  - Ako za dati graf nije dostupan algoritam koji vodi računa o navedenom kriterijumu, primenjuje se organski, pošto on generalno vodi računa o najvećem broju kriterijuma.

- Ukoliko je specifikovano više estetskih kriterijuma, traži se algoritam koji zadovoljava što veći njihov broj, pri čemu se vodi računa o redosledu u kojem su navedeni. Naime, redosled navođenja kriterijuma definiše njihov prioritet, tako da prvi ima najviši. Dakle, algoritam bi trebalo bar do neke mere da ispoštuje prvi kriterijum, što sužava skup kandidata, te se ne može garantovati da će baš svi kriterijumi biti zadovoljeni u celosti, ili do bilo koje mere. Kao što je spomenuto, kompromisi se moraju praviti, jer ne postoji algoritam koji sve kriterijume pokriva u celosti, dok korisnike ništa ne sprečava da zadaju nemoguće kombinacije. Ako se, recimo, zahtevaju tok, planarnost i ravnomerna distribucija čvorova, a graf nije stablo, biće izabran hijerarhijski algoritam. Ako se pak isti kriterijumi poređaju drugačije, na primer, distribucija čvorova, planarnost i tok, biće izabran organski.

Po izdvajanju odgovarajućeg algoritma, ukoliko je potrebno, pojedini njegovi parametri se podešavaju kako bi se obezbedilo poštovanje relevantnih estetskih kriterijuma. Na primer, organskom algoritmu, ako treba da optimizuje broj preseka grana, konfiguriše se parametar koji o ovome vodi računa.

7.3 Primeri

U nastavku sledi nekoliko primera specifikacije raspoređivanja upotrebom upravo opisanog jezika i rezultujućih grafova. Listing koda 7.16 sadrži komande napisane u skladu sa jezikom, a slika 7.2 rezultujuće crteže istog grafa. U listingu koda 7.17 prikazan je primer zadavanja načina raspoređivanja grafa koji se sastoji iz odvojenih komponenti, gde se jedna crta na jedan, a druga na drugi način. Dobijeni crtež može se videti na slici 7.2

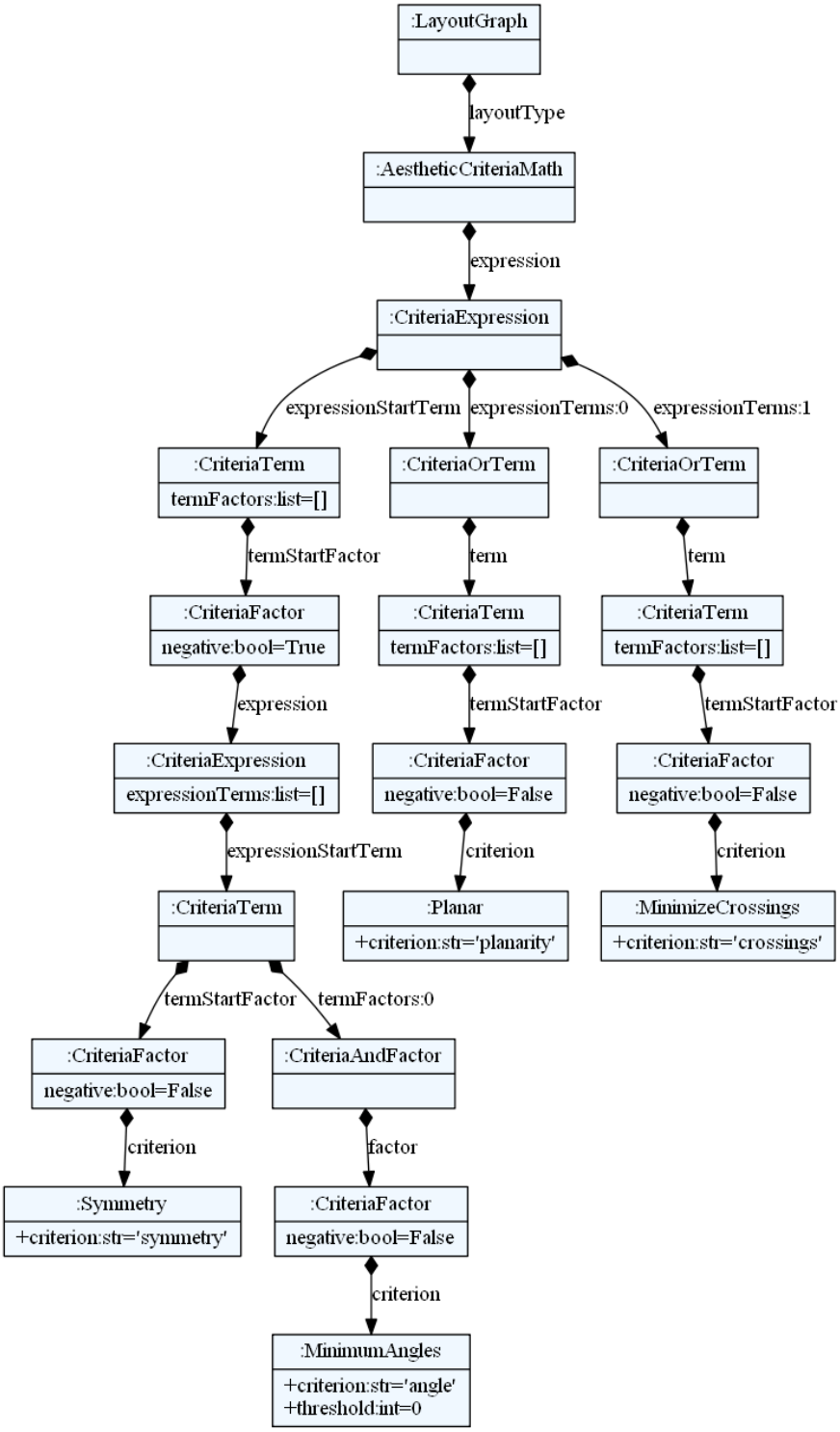
Listing 7.16: Primeri specifikacije raspoređivanja u skladu sa jezikom za graf u celini

```
1. lay out graph using style circular

2. lay out graph using criteria
     planar, symmetric, optimize edges lengths,
     distribute nodes evenly

3. lay out graph using criteria symmetric

4. lay out graph using criteria planar
```

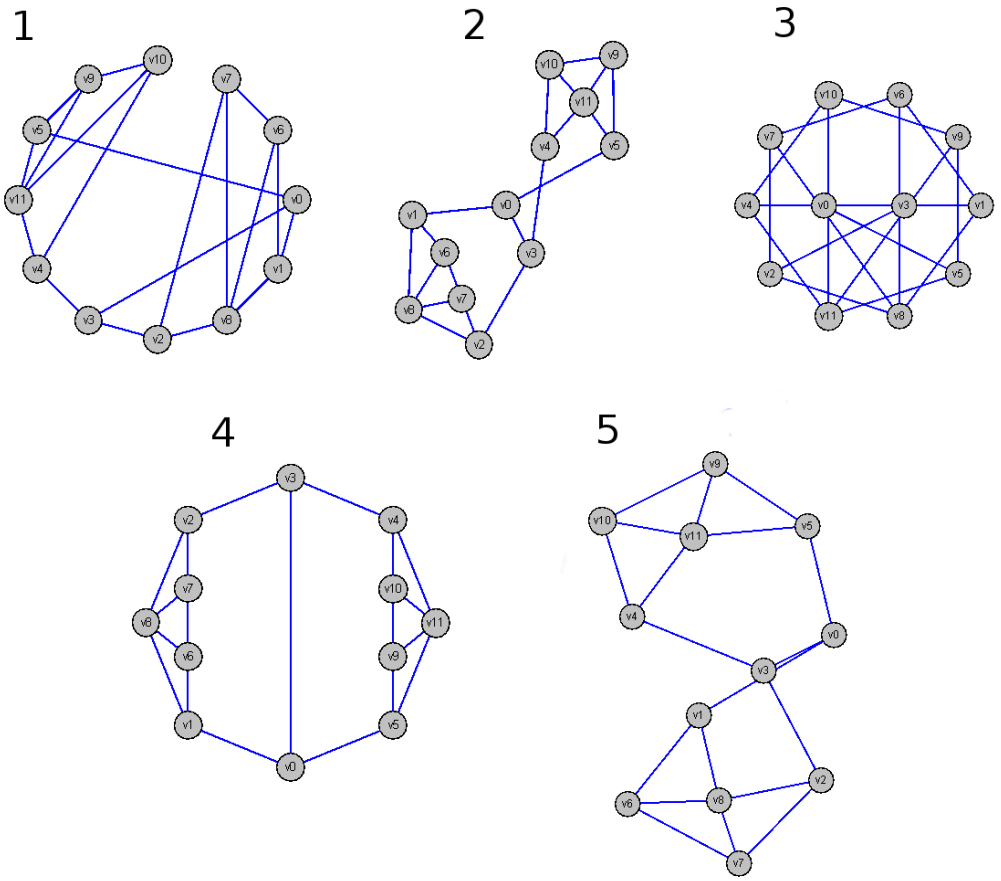
```
5. lay out graph using algorithm Kamada Kawai(length factor = 1)
```

Listing 7.17: Primeri specifikacije raspoređivanja u skladu sa jezikom za graf koji se sastoji iz dve nepovezane komponente

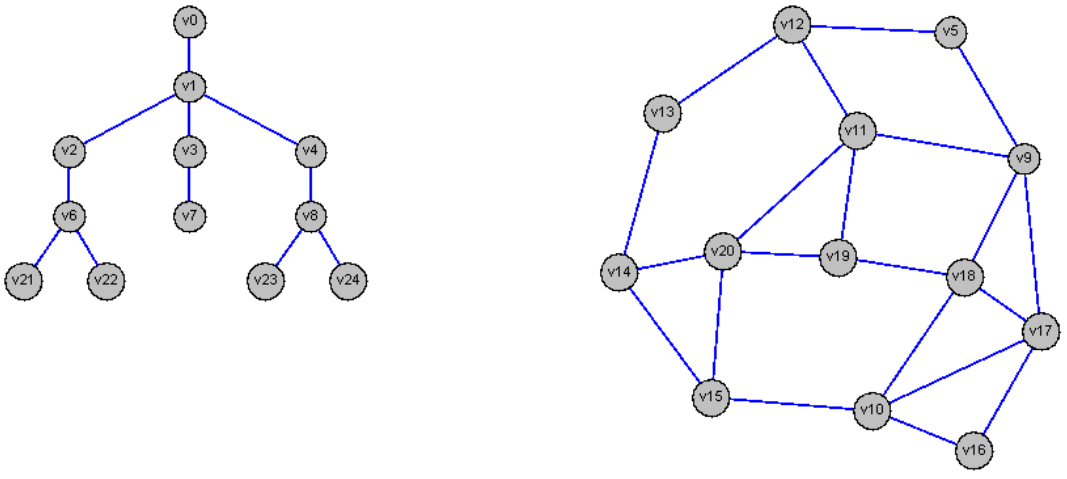
```
lay out
    subgraph
          v0,v1,v2,v3,v4,v6,v7,v8,v21,
          v22,v23,v24
   using style tree;
   others using criteria minimize edge crossings;
```



Slika 7.1: Dijagram objekata koji *textX* generiše za primer iz prethodnog listinga koda 7.15



Slika 7.2: Crteži istog grafa dobijeni na osnovu naredbi napisanih opisanim jezikom



Slika 7.3: Crtež grafa koji se sastoji iz dve nepovezane komponente dobijen na osnovu uputstva napisanog u kreiranom jeziku

Poglavlje 8

Grafički editor

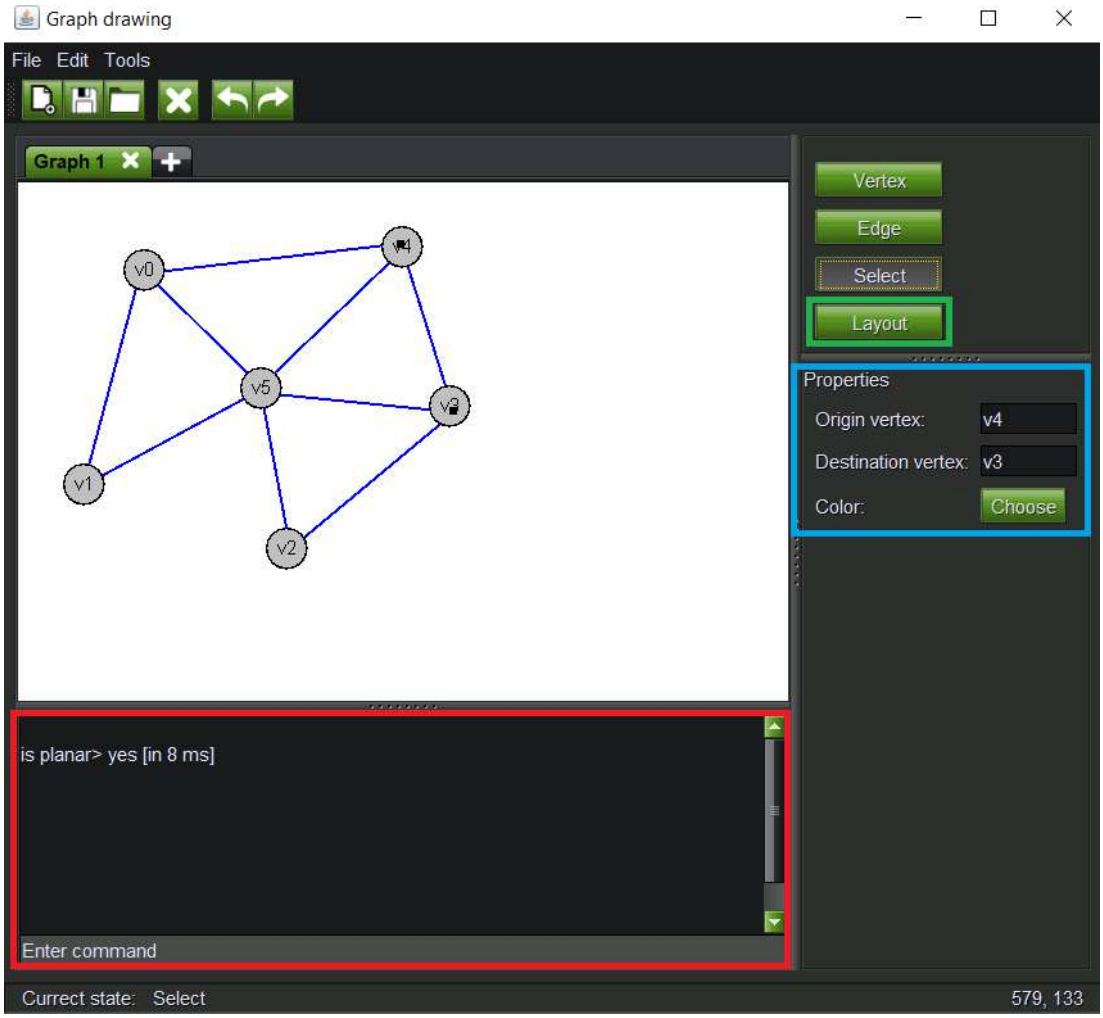
U cilju olakšavanja postupka razvoja, testiranja i ocene algoritama za analizu i crtanje grafova, implementiran je poseban grafički editor. On omogućava kreiranje jednostavnih grafova, kako orijentisanih, tako i neorijentisanih. Upravo su pomoću njega kreirani crteži grafova prikazani u radu. Crtanje grafova pomoću ovog editora brže je nego pisanje koda kojim bi se kreirao željeni graf. Pristustvo editora olakšava i ocenu kvaliteta algoritma, pogotova kada je potrebno analizirati estetiku crteža. Ova jednostavna aplikacija može se upotrebiti i radi upoznavanja sa različitim algoritmima za raspoređivanje i eksperimentisanja sa različitim vrednostima njihovih parametara, te za analizu datog grafa (proveru planarnosti, povezanosti, bipovezanosti, dekompoziciju na komponente itd.). Izvorni kod editora nalati se na [169].

Glavni prozor editora prikazan je na slici 8.1. U njegovom donjem delu (uokvireno crvenom bojom na slici) nalazi se komandna konzola, gde se unose komande kojima se mogu pozvati svi dostupni algoritmi za analizu grafova. Kao što se može videti sa slike, osim rezultata izvršavanja, ispisuje se i vreme koje je za to bilo potrebno.

Unosom komande *help* dobija se spisak svih raspoloživih opcija, dok se u svim ostalim slučajevima unosa validne naredbe izdvojeni algoritam izvršava nad tekućim grafom. Odnosno, nad grafom koji je u datom trenutku u fokusu. Neke od tih komanda su:

- *is planar* - provera planarnosti grafa,
- *is cyclic* - provera cikličnosti grafa,
- *is connected* - provera povezanosti grafa,
- *is biconnected* - provera bipovezanosti grafa,
- *list cut vertices* - lista svih artikulacionih čvorova,
- *list blocks* - lista svih bipovezanih komponenti grafa,
- *automorphisms* - lista automorfizama grafa,
- *lay out* - praćeno opisom u skladu sa definisanim jezikom specifičnim za domen vrši raspoređivanje elemenata grafa prema uputstvima korisnika,
- itd.

Implementacije algoritama, kao što je u uvodnom pogavlju objašnjeno, deo su posebnog projekta i nisu u jakoj sprezi sa editorom. Slično, ni definicija i interpreter jezika nemaju

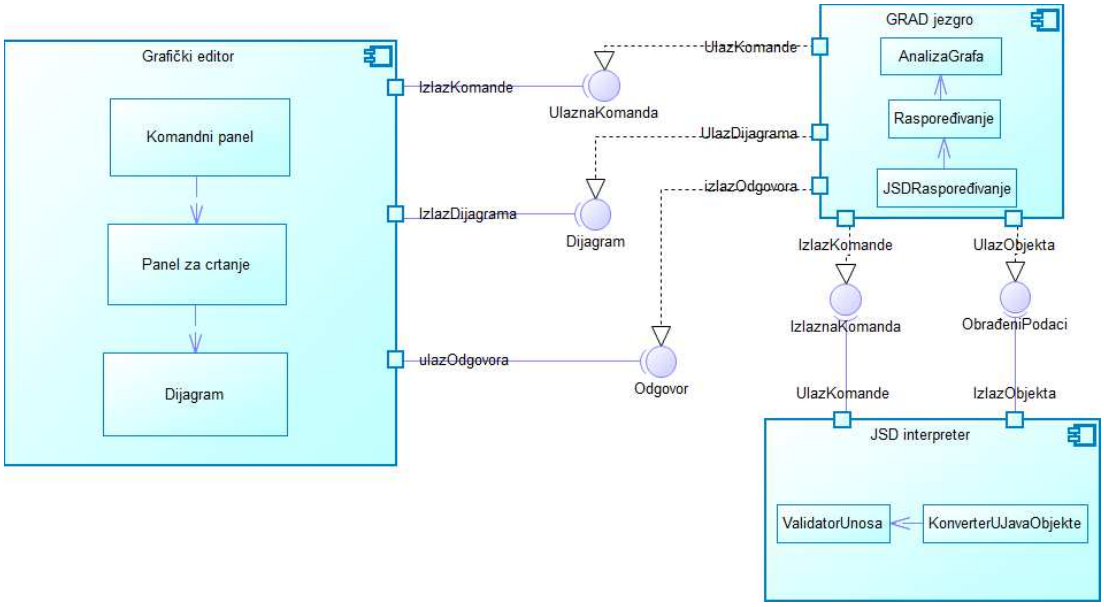


Slika 8.1: Grafički editor za podršku razvoja, testiranja i ocene algoritama

zavisnost takvog tipa ka njemu. Na slici 8.2 prikazan je odnos između editora, jezika specifičnog za domen i jezgra *GRAD* biblioteke (projekta koji sadrži implementirane algoritme za crtanje i analizu grafova).

U komandni panel editora mogu se uneti komande za analizu grafa, kao što su `is planar` ili `is cyclic`, kao i komanda za raspoređivanje elemenata dijagrama. U oba slučaja, grafički editor komandu, zajedno sa informacijama o tekućem dijagramu prosleđuje *GRAD* jezgru, gde se izvršavaju odgovarajući algoritmi. Ukoliko prosleđena komanda počinje rečima `lay out`, klasa *GRAD*biblioteke zadužena za raspoređivanje prosleđuje tu komandu JSD interpreteru. Interpreter kreira *Java* objekat koji reprezentuje komandu, konverzijom *Python* objekata upotrebom *Jython* bilioteke, kao što je objašnjeno u poglavlju 7.2. Po prijemu odgovarajućeg objekta, klasa *GRAD* biblioteke koja se bavi raspoređivanjem bira i primenjuje odgovarajući algoritam, prateći postupak detaljno opisan u upravo spomenutom poglavlju. Konačno, klasa za analizu ili raspoređivanje elemenata grafa koja je odbrađivala komandu šalje odgovor grafičkom editoru. Format odgovora zavisi od komande, te može biti:

- indikator da li graf poseduje neku osobinu, lista ciklusa grafa, lista povezanih komponenti i sl. ukoliko je poslata neka od komandi za analizu grafa, ili,



Slika 8.2: Odnos između editora, jezika specifičnog za domen i jezgra *GRAD* biblioteke

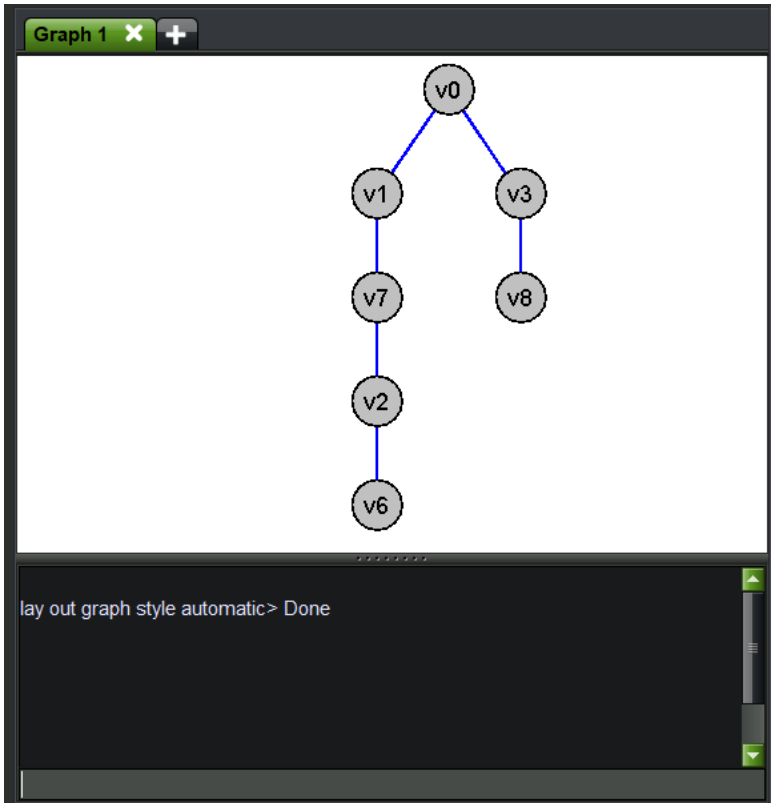
- pozicije elemenata dijagrama ukoliko je poslata komanda za raspoređivanje.

Konačno, grafički editor odgovor ispisuje na konzolu, odnosno, postavlja pozicije elemenata dijagrama. Na slici 8.3 prikazani su panel za crtanje i komandna konzola grafičkog editora, gde se vidi komanda uneta u skladu sa definisanim JSD i rezultujući izgled dijagrama.

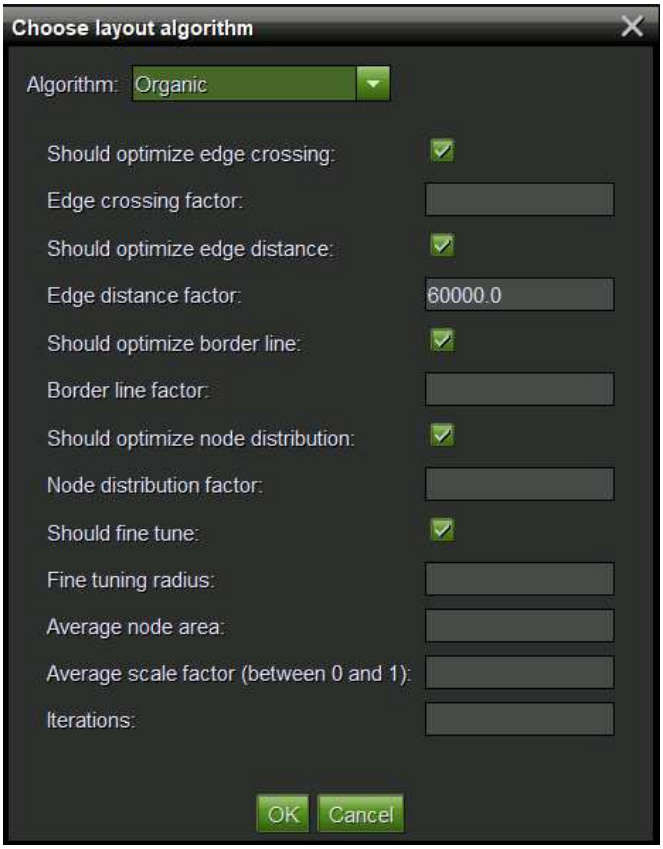
Raspoređivanje elemenata grafa može se pozvati i klikom na dugme označeno zelenom bojom na slici 8.1. elemenata tekućeg grafa. Preciznije, otvara se dijalog preko kojeg se bira algoritam, te za trenutno selektovani opciono podešavaju parametri. Pomenuti dijalog prikazan je na slici 8.4.

Ukoliko su za algoritam definisane podrazumevane vrednosti pojedinih parametara, komponente mapirane na takve parametre bivaju automatski popunjene inicijalnim sadržajem. Na slici je odabran organski algoritam, te su čekirane sve *checkbox* komponente, kako su sve optimizacije prema podrazumevanim podešavanjima omogućene.

Konačno, plavom bojom na slici označen je panel za postavljanje određenih osobina elemenata grafa. Editor podržava specifikaciju osnovnih osobina čvorova, grana, kao i samog grafa. Dakle, mogu se postaviti boje elemenata, sadržaj čvora, te da li je graf orijentisan ili ne. Takođe, nacrtani grafovi se mogu snimiti i kasnije učitati.



Slika 8.3: Komandna konzola sa komandom unetom u skladu sa JSD i rezultujući izgled dijagrama



Slika 8.4: Dijalog za izbor i konfiguraciju algoritma za raspoređivanje

Poglavlje 9

Integracija rešenja sa postojećim grafičkim editorima

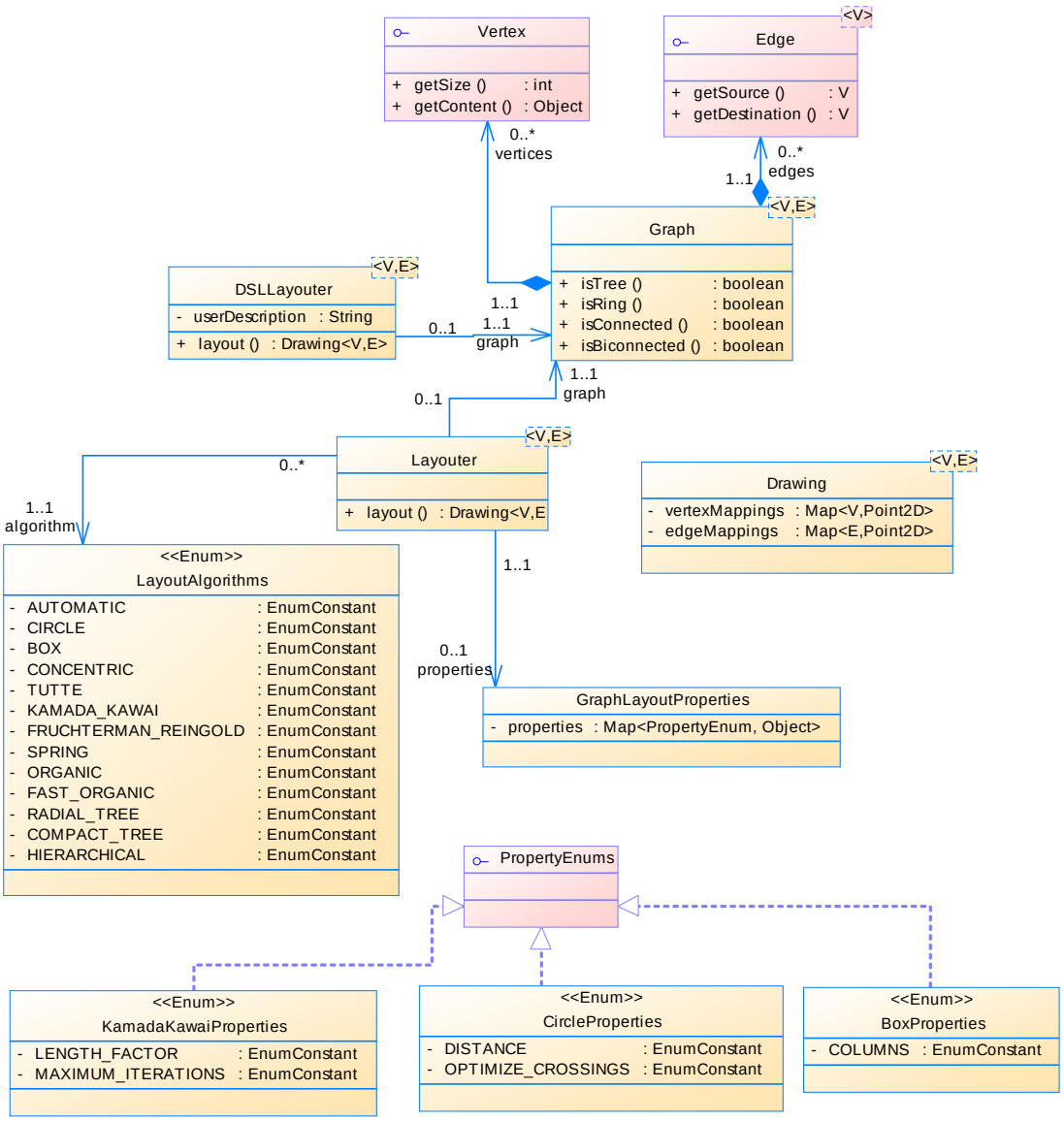
Jedan od glavnih zadataka *GRAD* biblioteke koja je proizišla iz istraživanja jeste veoma jednostavna integracija sa postojećim grafičkim editorima. Prilikom pregleda postojećih biblioteka sa podrškom za crtanje i analizu grafova za programski jezik *Java* predstavljene su mogućnosti korišćenja nekog od algoritama koje implementiraju u okviru zasebno razvijenih vizualizatora. Dakle, ne koristeći komponente koje one same nude. Uočeno je da je ovaj naizgled jednostavan zadatak dosta kompleksniji nego što se očekivalo. Razlog se nalazi u činjenici da su postojeća rešenja usredsređena na omogućavanje prikaza specifikovanog grafa unutar spomenutih komponenata, koje mogu biti i potpuno funkcionalni grafički editori. Cilj razvijene biblioteke, sa druge strane, nije podrška vizualizaciji informacija na način koji nude druga rešenja, već davanje mogućnosti jednostavnog raspoređivanja elemenata postojećih editora, prilikom generisanja statičkih slika dijagrama i slično. U nastavku poglavlja biće predstavljen API (*Application Programming Interface*) za crtanje grafova biblioteke i primeri koda koji ga koriste.

9.1 Pregled API-ja za crtanje grafova

GRAD biblioteka omogućava veoma jednostavno pozivanje željenog algoritma i preuzimanje dobijenog rezultata, odnosno pozicija čvorova i grana grafa. Dijagram klasa koji prikazuje ključne elemente ove funkcionalnosti prikazan je na slici 9.1.

Pre svega, može se primetiti da su klase parametrizovane. Naime, bilo koja klasa koja implementira interfejse `Vertex` i `Edge` može biti tip čvora, odnosno, grane grafa. Ovi interfejsi sadrže metode za preuzimanje informacija neophodnih za izvršavanje algoritama. Pre svega, veličine čvora i početnog i odredišnog čvora grane. U opštem slučaju, proširivanje modela koji predstavlja pomenute elemente grafa proizvoljnog grafičkog editora realizacijom navedenih interfejsa ne bi trebalo da izazove poteškoće. Ukoliko to, međutim, bude slučaj, postoji mogućnost kreiranja po jedne instance klasa `GraphVertex` i `GraphEdge` za svaki element datog dijagrama koji treba biti uzet u obzir pri raspoređivanju.

Centralne klase zadužene za obradu zahteva za raspoređivanjem jesu `Layouter` i `DSLLayouter`. Klasa `Layouter` prima liste čvorova i grana, vrednost enumeracije `LayoutAlgorithms` koja nabraja sve dostupne algoritme i, opciono, vrednosti konfigurabilnih parametara izabranog algoritma. Dakle, klasu koja predstavlja graf *GRAD* biblioteke korisnik ne mora da instancira, nego to kasnije automatski biva



Slika 9.1: Dijagram klasa dela rešenja bitnog za pozivanje algoritama za crtanje grafova

urađeno na osnovu prosleđenih lista. Štaviše, i sam algoritam se kreira na sličan način, na osnovu nabrojive vrednosti, te se time dodatno smanjuje količina koda koji pozivalac mora napisati. Vrednost pomenute enumeracije *AUTOMATIC* može biti iskorišćena za naznačavanje želje za punom automatizacijom raspoređivanja. U tom slučaju ne prosleđuju se vrednosti parametara algoritma, kako će oni biti postavljeni nakon detekcije najboljeg algoritma na osnovu obeležja grafa. Sa druge strane, ukoliko je direktno izabran konkretan algoritam za crtanje, željene vrednosti njegovih parametara mogu se zadati uz pomoć `GraphLayouProperties` klase, koja sadrži mapu koja vezuje identifikatore obeležja i njihove vrednosti. Kako bi se i ovaj korak maksimalno pojednostavio, za svaki konfigurabilni algoritam definisana je posebna enumeracija koja pobraja sva podesiva obeležja. Upravo vrednosti tih enumeracija dodaju se kao ključevi u spomenutu mapu, što smanjuje mogućnost greške i potrebu oslanjanja na dokumentaciju radi postizanja cilja. Pošto biblioteka sadrži poprilično velik broj algoritama za raspoređivanja, pri čemu većina njih može biti bar do neke mere

konfigurisana, na dijagramu nisu prikazane sve enumeracije za specifikaciju parametara. Takođe, svi parametri kojima korisnik ne zada vrednost dobijaju podrazumevane vrednosti.

Klasa `DSLLayouter`, sa druge strane, realizuje raspoređivanje na osnovu specifikacije u skladu sa definisanim jezikom specifičnim za domen. Slično kao `Layouter`, prima liste čvorova i grana, ali umesto algoritma i njegovih parametara, daje opis u vidu običnog teksta koji poštuje pravila JSD-a.

U svakom slučaju, nezavisno od metode raspoređivanja koji se koristi, ono će biti pokrenuto pozivom metode `layout` koju poseduju obe klase. Po završetku ove akcije, koja obuhvata izvršavanje algoritma ili algoritama za raspoređivanje i postprocesiranja, kao povratna vrednost dobija se objekat tipa `Drawing`, iz kojeg se direktno mogu preuzeti mapiranja čvorova na njihove izračunate pozicije i grana na liste pozicija početnih, krajnjih i prelomnih tačaka koje sadrže.

9.2 Primeri koda

U ovoj sekciji biće prikazan po primer pozivanja raspoređivanja direktnim navođenjem algoritma i upotrebom jezika specifičnog za domen. Prvi je dat u listingu koda 9.1, a drugi u listingu 9.2.

Listing 9.1: Pozivanje Kamada-Kavai algoritma i preuzimanje rezultata

```
//izbor algoritma
LayoutAlgorithms algorithm = LayoutAlgorithms.KAMADA_KAWAI;
//zadavanje parametara
GraphLayoutProperties layoutProperties =
   new GraphLayoutProperties();
//Prvi argument identifikuje obelezje, drugi je vrednost
//Posto se poziva algoritam Kamada-Kavai, koristi
//se enumeracija povezana sa njegovim obelezjima
layoutProperties.setProperty(
   KamadaKawaiProperties.LENGTH_FACTOR, 5);
layoutProperties.setProperty(
   KamadaKawaiProperties.DISCONNECTED_DISTANCE_MULTIPLIER, 3);
//Rasporedjivacu se prosledjuju liste elemenata,
//izabrani algoritam i konfiguracija
Layouter<GraphVertex, GraphEdge> layouter =
  new Layouter<>(vertices, edges, algorithm,
                 layoutProeprties);
//izvrsavanje algoritma vraca Drawing objekat, iz koga
//se preuzimaju poziyicije
Drawing<GraphVertex, GraphEdge> drawing = layouter.layout();
Map<GraphVertex, Point2D> vertexPositions =
  drawing.getVertexMappings();
Map<GraphEdge, List<Point2D>> edgePositions =
  drawing.getEdgeMappings();
```

Listing 9.2: Raspoređivanje upotrebom definisanog jezika

```
//String u skladu sa jezikom
String dsl = "lay out graph algorithm Kamada Kawai length factor = 5,
              distance multiplier = 3";
DSLLayouter dslLayouter =
  new DSLLayouter<>(vertices, edges, dsl);
Drawing<GraphVertex, GraphEdge> drawing = dslLayouter.layout();
Map<GraphVertex, Point2D> vertexPositions =
  drawing.getVertexMappings();
```

```
Map<GraphEdge, List<Point2D>> edgePositions =
  drawing.getEdgeMappings();
```

Iz primera se može primetiti da zaista nije potrebno pisati niti puno, niti posebno kompleksnog koda. Svega nekoliko linija dovoljno je za postizanje cilja, posebno ukoliko se koristi implementirani jezik specifičan za domen.

Poglavlje 10

Primena rešenja

Rešenje može imati veoma veliki broj primena, u okviru bilo kojeg grafičkog editora, odnosno, alata za modelovanje kojima bi automatsko raspoređivanje elemenata dijagrama moglo biti od koristi. U pojedinim situacijama ovakva funkcionalnost je neophodna za prijatan rad korisnika. U ovom poglavlju biće prodiskutovane takve situacije, te dati neki konkretni primeri korišćenja.

10.1 Diskusija o mogućim primenama

Kao što je spomenuto, rešenje može biti iskorišćeno u bilo kojem grafičkom editoru, gde postoji potreba za automatskim kreiranjem crteža nekog dijagrama. Međutim, rešenje je od posebnog interesa u situacijama kada već postojeće informacije treba da budu vizualizovane. Jedan od primera kada se navedeno može desiti upravo su jezici specifični za domen.

U sekciji 2.7, gde je dat kratak uvod u jezike pomenutog tipa, napomenuto je da mogu imati različite konkretne sintakse, od kojih su najznačajnije tekstualna i grafička. Obe sintakse imaju određene prednosti i mane, pri čemu je tekstualnu često preferiraju programeri, dok domenski eksperti radije koriste grafičku. Različita istraživanja došla su do zaključka da je stoga idealno podržati i jednu i drugu [112]. Radi podržavanja grafičke sintakse neophodno je obezbediti poseban grafički editor, u koejm bi se kreirali dijagrami čiji elementi odgovaraju konceptima jezika. Postojanje dve sintakse, međutim, dovodi do sledećeg problema: kako prikazati modele specifikovane tekstualnom sintaksom pomoću grafičkog editora radi pregleda i eventualne modifikacije? Naime, elementi dijagrama, kreirani na osnovu tekstualne definicije, nemaju pozicije, budući da taj koncept nije sastavni deo ovih notacija. Dakle, dolazi se do problema automatskog raspoređivanja. Elementi bi se mogli rasporediti nasumično, ili svi postaviti na neku proizvoljnu poziciju, ali bi tada korisnici pre dalje upotrebe morali samostalno urediti dijagram, što bi bilo neprijatno i dugotrajno u slučaju većih modela.

Takođe, pojedini autori jezika specifičnih za domen ne podržavaju grafičku notaciju u celosti, već samo generišu statičke vizuelne predstave modela kreiranih drugom notacijom. Kako navodi studija [112], iako ovo nije optimalno rešenje, grafički prikaz modela može domenskim ekspertima olakšati njegovo razumevanje. Iako tada nije neophodan potpuno funkcionalan grafički editor, i generisanje statičke slike koja bi mogla preneti informacije na odgovarajući način uključuje računanje pozicija elemenata, odnosno, primenu nekog algoritma za automatsko raspoređivanje.

Još jedna okolnost u kojoj se može javiti potreba za upotrebom automatskog crtanja grafova, odnosno, dijagrama, jeste unutar alata za modelovanje koji podržavaju uvoženje

modela razvijenih korišćenjem nekog drugog alata. Na primer, popularni *PowerDesigner* [170] i *MagicDraw* [171] alati poseduju takvu funkcionalnost. Oba alata automatski kreiraju sopstvene modele na osnovu uvezenih, ali ih jedino prvospomenuti iscrtava na estetski lep način, što u velikoj meri može povećati satisfakciju korisnika ovom opcijom.

Dodatno, automatsko raspoređivanje elemenata dijagrama je sastavni *Eclipse GEF* okvira (Graphical Editing Framework) [172]. *GEF* omogućava kreiranje grafičkih editora na programskom jeziku *Java*, nudeći nekoliko algoritama za pozicioniranje elemenata, ali i pružajući mehanizam za proširivanje ovog skupa. Dodati algoritam može biti implementiran pozivanjem nekog od algoritama *GRAD* biblioteke. Demonstracija upotrebe biblioteke, međutim, neće obuhvatati ovu primenu, nego će se koncentrisati na noviji *Eclipse* projekat koji se takođe bavi grafičkim modelovanjem oslanjajući se na *GEF- Sirius* projektnom okviru [16].

Takođe, treba napomenuti da čak i u sklopu grafičkih editora unutar kojih je korisnik ručno kreirao dijagram, automatsko raspoređivanje može biti od koristi. Naime, ukoliko je dijagram sačinjen u žurbi i bez fokusiranja na estetiku, primena algoritma za automatsko raspoređivanje može brzo ulepšati neuredan crtež.

U svakom slučaju, u kojem god kontekstu je potrebno ili korisno automatsko raspoređivanje elemenata grafa, razvijeni jezik specifičan za domen može pomoći pri izboru. Kao što je navedeno u pregledu ciljeva istraživanja, jezik može u velikoj meri olakšati korisniku koji nema puno znanja vezanog za crtanje grafova izbor i konfiguraciju algoritma kojim bi se kreirao crtež u skladu sa njegovim željama.

Konačno, pošto biblioteka implementira veliki broj algoritama za analizu grafova, njih same mogu direktno primeniti drugi projekati koji se bave grafovima. Algoritmi koji se tiču planarnosti grafova i dekompozicija na bipovezane i tripovezane komponente poprilično su komplikovani za implementaciju, imajući pritom veoma široku primenu.

U naredne dve sekcije biće prikazana dva primera korišćenja *GRAD* biblioteke. Jedan u okviru *Kroki* [173] alata za skiciranje poslovnih aplikacija upotrebom različitih alata, od čega je jedan editor UML dijagrama klasa. Drugim primerom biće demonstrirana mogućnost korišćenja biblioteke u okviru *Sirius* razvojnog okvira.

10.2 Primer korišćenja u okviru Kroki alata

Kroki je alat za interaktivni razvoj poslovnih aplikacija baziran na skicama formi. Alat omogućava skiciranje ekranskih formi različitih vrsta, poštujući posebno definisan standard korisničkog interfejsa koji određuje njihov izgled i funkcionalnost. *Kroki* je namenjen kako projektantima, tako i korisnicima različitih struka i nivoa znanja, te je postojanje više pogleda na aplikaciju od velike važnosti. Stoga ovaj alat poseduje više ugrađenih alata pomoću kojih se mogu kreirati forme i uspostavljati veze među njima. To su:

- Editor formi, koji omogućava njihovo kreiranje direktnim prevlačenjem elemenata kao što su tekstualna i druga polja sa palete.
- Komandna konzola, koja pruža mogućnost unosa tekstualnih komandi putem kojih se na brz način mogu kreirati ekranske forme.
- Grafički *UML* editor, kod kojeg svaka klasa predstavlja formu, a asocijacija među klasama vezu jedne forme sa drugom.

Nezavisno od načina na koji se aplikacija modeluje, promene načinjene nad jednim pogledom moraju biti vidljive u okviru drugog. Kako je osnovna ideja *Kroki* alata

kreiranje inicijalne skice zajedno sa krajnjim korisnicima (domenskim ekspertima), to se uglavnom vrši upotrebom editora formi, kako je njima ovaj način najprirodniji. Međutim, programerima bi kasnije moglo biti zgodno pogledati *UML* model klasa, te ga dopuniti i izmeniti prema potrebi. Naime, ovaj način skiciranja je znatno brži ukoliko korisnik ima iskustva sa drugim alatima za modelovanje. Dakle, dolazi se u situaciju kada je potrebno automatski kreirati grafičke elemente, ali ih i rasporediti na pregledan način.

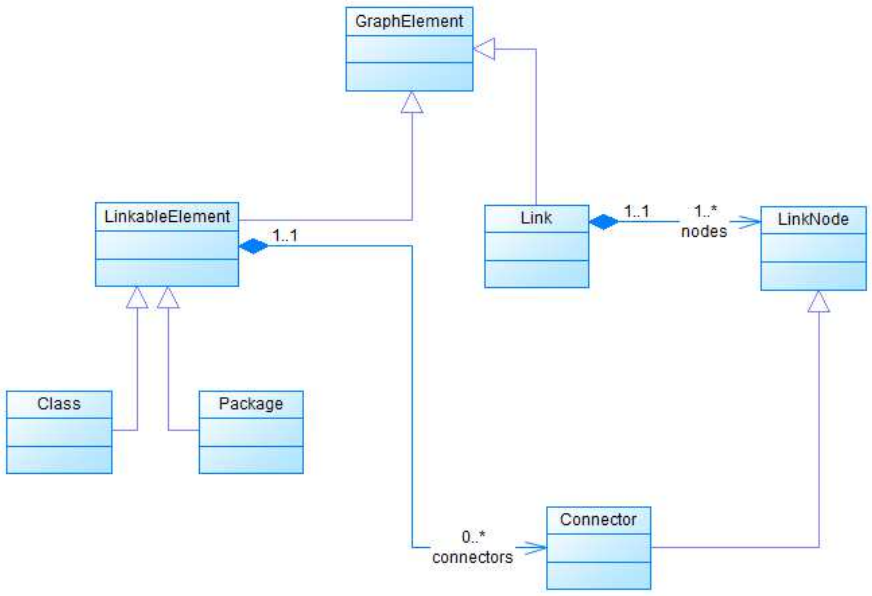
Treba napomenuti da u ovom slučaju sama vizualizacija skicirane aplikacije putem dijagrama *UML* klasa nije dovoljna, te je postojala potreba za razvojem posebnog editora. Naime, svako dodavanje nove klase, promena njenog atributa ili uspostavljanje veze između dva elementa dijagrama treba rezultovati ažuriranjem i pogleda putem editora formi. Odnosno, modela koji se nalazi u pozadini oba relevantna pogleda. Dakle, korišćenje neke od opisanih biblioteka za vizualizaciju radi generisanja editora nije u datom slučaju validno rešenje. Više informacija o grafičkom editoru, modelu koji se nalazi u korenu *Kroki* alata, kao i sinhronizaciji izmena o kojima je bilo reči može se naći u [174].

Pored kreiranja skica na tri nabrojana načina, Kroki podržava i uvoženje *UML* modela razvijenih drugim alatima za modelovanje, kao što su prethodno spomenuti *PowerDesigner* i *MagicDraw* [175]. Ovakvi modeli mogu imati i po nekoliko stotina, pa čak i hiljada klasa, na osnovu kojih se pri uvoženju generišu forme. Pri prvom otvaranju grafičkog *UML* editora formiraju se odgovarajući grafički elementi, čije pozicije inicijalno nisu poznate. Kako korisnik ne bi sam raspoređivao ovako veliki broj elemenata, njihovo automatsko pozicioniranje je veoma značajno. Štaviše, kako je u takvim situacijama teško proceniti koji algoritam bi se mogao primeniti i kako ga treba konfigurisati, poželjna je puna automatzacija ovih akcija.

Jedan primer rada sa većim modelom za koji je korišćen *Kroki* alat jeste za podršku vizualizacije, daljeg razvoja i kreiranja izvršivog prototipa *CERIF* (*(the Common European Research Information Format*)[176] modela. *CERIF* je standard za razvoj nacionalnih sistema za rukovanje istraživanjima. Definiše standardizovani model podataka i predložen je 1991. godine, da bi od tada u više navrata bio poboljšavan i proširivan. *CERIF* specifikacija 1.5 je uvezena u *Kroki* alat, gde se primetilo da se sastoji iz oko 280 klasa, raspoređenih u 31 paket. Inicijalni model prvo je morao biti sređen pre nego što se pristupilo generisanju aplikacije [177]. Ručno pozicioniranje elemenata 31 paketa svakako bi znatno usporilo stizanje do jasne vizualizacije modela, te se i ovde automatsko raspoređivanje pokazalo kao faktor ubrzavanja rada, kojem *Kroki* teži u svim aspektima. Sadržaji dva proizvoljna paketa su uglavnom znatno različiti, te je ovaj primer dobar test za automatsku detekciju i konfiguraciju pogodnog algoritma. Takođe, treba napomenuti da su klase znatno većih dimenzija u odnosu na čvorove grafova korišćene u primerima, te je ovaj problem ujedno i prilika za demonstraciju rada sa čvorovima nezanemarljivih i nejednakih veličina. Takođe, ukoliko su klase paketirane, neophodno je kreirati prečice u okviru svih paketa čiji elementi treba da budu povezani s elementima drugih paketa. Dakle, ukupan broj grafičkih elemenata je često veći od ukupnog broja različitih klasa.

Radi demonstracije načina sprezanja razvijene biblioteke sa *Kroki* alatom, na slici 10.1 prikazan je deo modela grafičkih elemenata *Kroki UML* editora.

Jedan dijagram sadrži elemente kao što su klase i paketi, koji nasleđuju `LinkableElement` klasu i veze, tipa `Link`. Takav dijagram može se predstaviti grafom čiji čvorovi su instance prve, a grane druge klase. Raspoređivanje *Kroki* UML dijagrama klasa na potpuno automatski način prikazan je u listingu koda 10.1. Listing prikazuje jednostavnost korišćenja razvijene biblioteke za dobijanja pozicija na kojima elementi



Slika 10.1: Deo modela klasa grafičkih elemenata *UML* editora *Kroki* alata

dijagrama treba da se nađu i integracije sa daljom obradom u skladu sa specifičnostima editora.

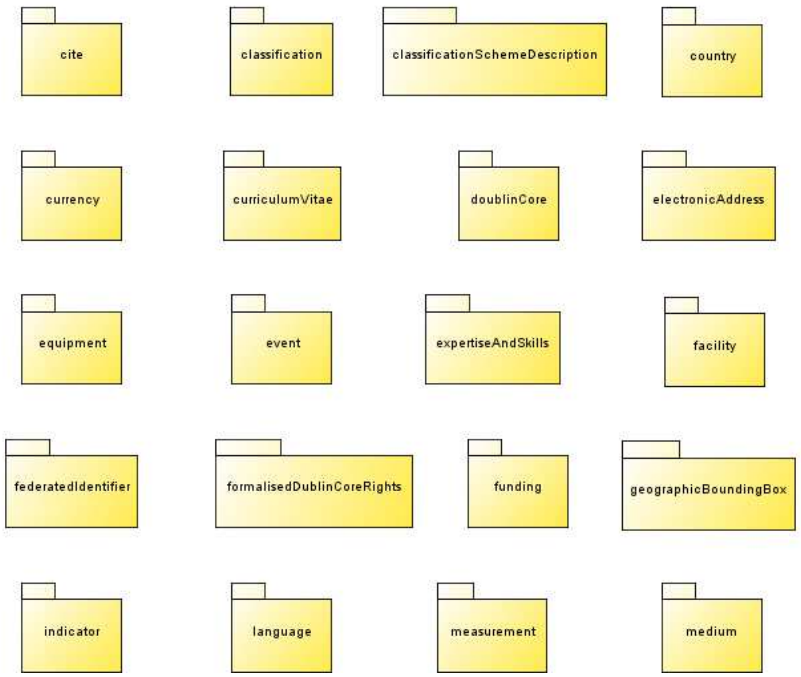
Listing 10.1: Sprega *Kroki* alata sa *GRAD* bibliotekom

```
List<LinkableElement> elementsToLayout =
   new ArrayList<LinkableElement>();
for (GraphElement el : model.getDiagramElements())
   if (el instanceof LinkableElement)
     elementsToLayout.add((LinkableElement) el);
links = model.getLinks();


Layouter<LinkableElement, Link> layouter =
       new Layouter<LinkableElement, Link>(
       elementsToLayout, links, LayoutAlgorithms.AUTOMATIC);
try {
  Drawing<LinkableElement, Link> drawing = layouter.layout();
  Map<LinkableElement, Point2D> elementPositions =
      drawing.getVertexMappings();
  for (LinkableElement el : elementsToLayout){
    Point2D position = elementPositions.get(el);
    setPosition(view, el, (int) position.getX(), (int) position.getY());
    setConnectorLocations(el);
  }
  Map<Link, List<Point2D>> linkPositions = drawing.getEdgeMappings();
    for (Link link : links){
      setLinkNodes(view, link, linkPositions.get(link));
  }
} catch (CannotBeAppliedException e) {
  e.printStackTrace();
}
```

Pri importu *CERIF* modela dobijen je, kao što je rečeno, 31 paket, od kojih svaki unutar sebe sadrži manji model, odnosno, poseban dijagram. Ti dijagrami neretko nemaju puno sličnosti, sastojeći se iz različitog broja elemenata, ponekad sa

velikim brojem veza između sadržanih klasa, ponekad sa znatno manjim. Prema tome, automatsko raspoređivanje daje drugačije rezultate, odnosno, izdvaja različite algoritme za različite pakete, kao i za prikaz dijagrama najvišeg nivoa, koji se sastoji samo iz paketa. Na slikama koje slede prikazano je nekoliko dijagrama *CERIF* modela s elementima koje je *GRAD* biblioteka pozicionirala uz pomoć automatske detekcije i konfiguracije algoritma:

- Slika 10.2 prikazuje dijagram na vrhu, raspoređen tabelarnim algoritmom.
- Slika 10.3 paket sa manjim unutrašnjim dijagramom koji je graf tipa stabla.
- Slika 10.4 paket sa većim dijagramom koji ne poseduje osobine od interesa, te je raspoređen silom usmerenim algoritmom.



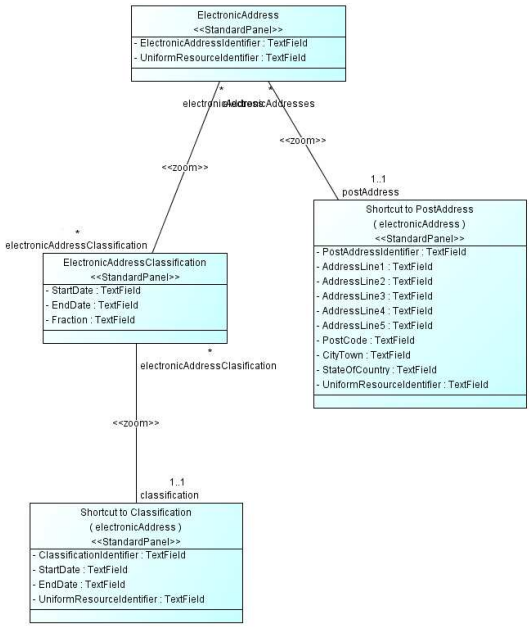
Slika 10.2: Dijagram *CERIF* modela najvišeg nivoa, raspoređen tabelarnim algoritmom

Iz primera se vidi da se zaista vodilo računa o dimenzijama elemenata, što je izuzetno bitno za dijagrame klasa, kod kojih su veličine elemenata nekonzistentne.

10.3 Integracija sa popularnim alatima

Pojedini popularni alati za modelovanje, odnosno, kreiranje grafičkih editora, pored pružanja nekolicine sopstvenih mehanizama za automatsko raspoređivanje elemenata dijagrama ostavljaju mogućnost ugrađivanja dodatnih. Jedan primer je *Eclipse GMP* (*Graphical Modeling Project*, [178] koji obezbeđuje skup generičkih komponenata za razvoj grafičkih editora baziranih na *Eclipse EMF (Eclipse Modeling Framework)* [179] i *GEF (Graphical Editing Framework)* [180] okvirima.

EMF je okvir za modelovanje sa mogućnošću generisanja koda na osnovu modela specifikovanog u odgovarajućem formatu. Konkretnije, *EMF* omogućava generisanje entiteta, njihove grafičke reprezentacije, editora kojim se model može širiti i šablona



Slika 10.3: Manji dijagram unutar paketa raspoređen algoritmom za crtanje stabala

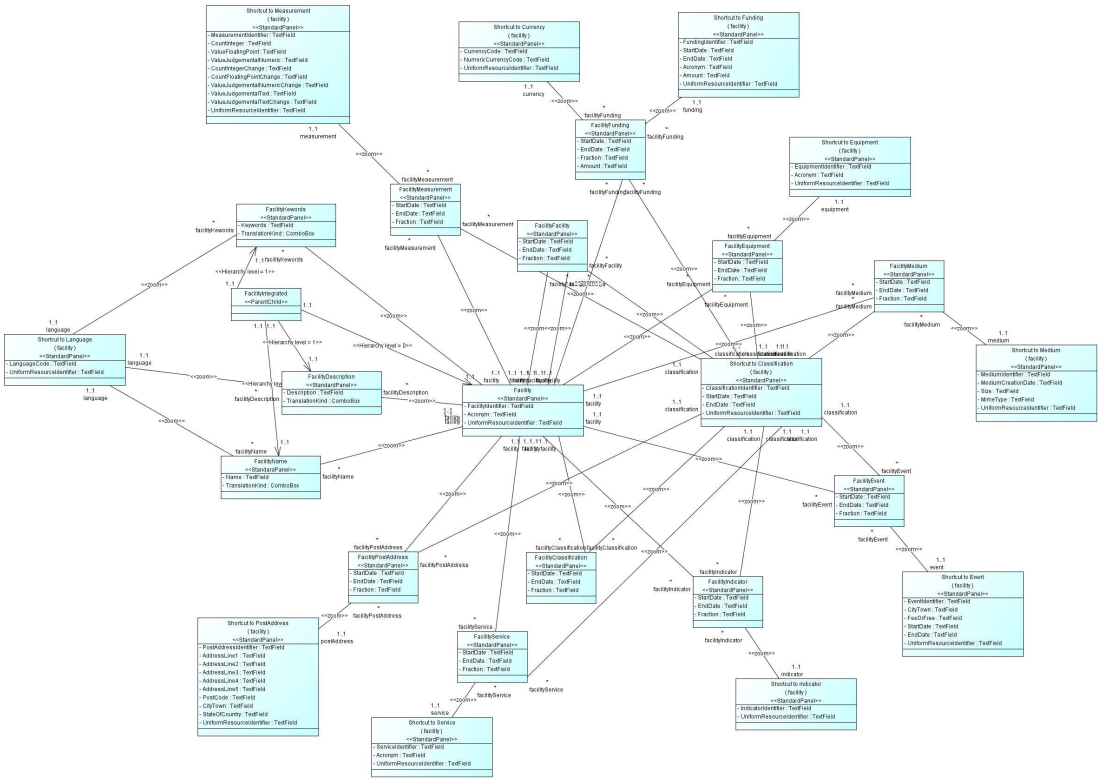
za pisanje testova. *GEF* sa druge strane poseduje funkcionalnosti poput alata za kreiranje *Graphviz DOT* grafova i njihove reprezentacije, kao i komponenata koje se mogu iskoristiti za obogaćivanje grafičkih aplikacija. Između ostalog, *GEF* sadrži skup algoritama za raspoređivanje elemenata dijagrama, definišući pritom interfejse čijom realizacijom se dati skup može proširiti. Algoritmi koje *GEF* obezbeđuje jesu standardni algoritmi za crtanje stabala, raspoređivanje elemenata tabelarno, kao i redom na jednoj liniji, te silom-usmereni algoritam baziran na metodi opruge.

Iako se *EMF* i *GEF* mogu direktno koristiti za razvoj grafičkih editora i drugih aplikacija, jednostavnija opcija je upotreba *Sirius Eclipse* projekta, koji obuhvata pomenuta dva okvira i podiže ih na viši nivo. *Sirius* omogućava kreiranje radnog okruženja za modelovanje koje se sastoji iz editora dijagrama, tabela i stabala za pravljenje, modifikaciju i vizualizaciju *EMF* modela. Editori se definišu modelima koji određuju strukturu radnog okruženja, njegovo ponašanje i alate koje poseduje. Opisano *Sirius* radno okruženje biva dinamički interpretirano.

Specifikacija dijagrama obuhvata određivanje grafičke reprezentacije elemenata modela, kao i alata koji se mogu primeniti - dodavanje novih elemenata i veza među njima, filteri, validaciona pravila, navigacija, te automatsko raspoređivanje elemenata dijagrama. *Sirius* je u velikoj meri proširiv, dozvoljavajući definisanje novih načina reprezentacije, jezika za pretragu, kao i novih mehanizama raspoređivanja, što je od posebnog interesa za tekući pregled i biće detaljnije opisano u nastavku.

Rad u *Sirius* okruženju počinje kreiranjem domenskog modela putem *EMF* koncepata i njegovim pokretanjem kao *Eclipse* aplikaciju. U pokrenutom okruženju potom se kreiraju projekti za modelovanje (*Modeling Project*) i za specifikaciju prezentacije (*Viewpoint Specification Project*), u kome se kreiraju sve klase koje će biti spominjane u narednim pasusima.

Kao što je već spomenuto, *GEF* okvir koji je jedan od temelja *Sirius* projekta nudi kako gotove algoritme za raspoređivanje elemenata dijagrama, tako i mogućnost dodavanja novih. Odnosno, omogućeno je definisanje reakcije na zahtev za raspoređivanje svih elemenata (akcija *Arange All*), te samo svih selektovanih. *Sirius*



Slika 10.4: Veći dijagram raspoređen silom usmerenim algoritmom

dodatno pojednostavljuje postupak, definišući nove apstraktne i konkretne klase usko povezane sa navedenim problemom. Od posebnog interesa su `AbstractLayoutProvider` i `DefaultLayoutProvider`, koje nasleđuju *GEF* `AbstractLayoutEditPartProvider` klasu, redefinišući njene metode neophodne za izvršavanje spomenute akcije. Reč je o sledećem:

- `provides(IOperation operation)` - određuje da li klasa može da primi zahtev prosleđenog tipa.
- `layoutEditParts(GraphicalEditPart containerEditPart, IAdaptable layoutHint)` - raspoređuje elemente prosleđenog roditeljskog elementa, određujući način za izvršavanje ovog zadatka na osnovu naznaka specifikovanih drugim parametrom. Kao rezultat vraća komandu kojom se menjaju pozicije elemenata.
- `layoutEditParts(List selectedObjects, IAdaptable layoutHint)` - raspoređuje selektovane elemente, takođe vraćajući komandu na kraju svog izvršavanja.

Centralni pojmovi u *GMF*-u, su *view* i *edit part*, gde prvi predstavlja gradivne celine prezentacionog sloja, a drugi kontrolore odgovorne za ponašanje editora, korisničku interakciju sa dijagramom, kao i definisanje grafičke reprezentacije elemenata. Notacioni model određuje koji podskup domenskog modela se prikazuje u vizualizatoru, pritom dodajući informacije povezane s konkretnom reprezentacijom poput pozicija elemenata, fonta i drugih stilskih podataka.

Razlikuje se više *edit part* klasa, a dve najbitnije za integraciju sa *GRAD* bibliotekom su `ShapeNodeEditPart`, koja predstavlja čvor grafa, i `ConntectionEditPart`, koja predstavlja granu. Dakle, proverama tipova *edit part* elemenata jednog roditelja može se

kreirati *GRAD* graf, nad kojim se dalje može primeniti bilo koji algoritam koji biblioteka nudi. Primer koda kojim se poziva automatsko raspoređivanje dat je u listingu 10.2.

Listing 10.2: Korišćenje *GRAD* biblioteke za definisanje novih načina automatskog raspoređivanja editora generisanog pomoću *Sirius* alata

```
public Command layoutEditParts(final
   GraphicalEditPart containerEditPart,
   final IAdaptable layoutHint) {
   final List<ConnectionEditPart> connectionEditParts = new
     ArrayList<ConnectionEditPart>();
   final List<GraphEdge> edges = new ArrayList<GraphEdge>();
  //Mapiranje edit part-a na cvor grafa
   final Map<ShapeEditPart, GraphVertex> editPartVertexMap = new
     HashMap<ShapeEditPart, GraphVertex>();

  for (EidtPart next : containerPart.getChildren()){
    if (next instanceof ShapeEditPart &&
      !(next instanceof IBorderItemEditPart)) {
      //Ako je element shape edit part - cvor grafa
      //nalazimo njegove dimenzije i
      //pravimo odgovarajuci cvor GRAD grafa.
      //Posto se ne moze implementirati interfejs,
      //koriste se GraphVertex i GraphEdge klase
      final ShapeEditPart shapeEditPart = (ShapeEditPart) next;
      Dimension size = shapeEditPart.getSize();
      java.awt.Dimension dim = new java.awt.Dimension(size.width, size.height);
      GraphVertex vertex = new GraphVertex(dim);
      editPartVertexMap.put(shapeEditPart, vertex);
    } else if (next instanceof ConnectionEditPart){
      final ConnectionEditPart connectionEditPart = (ConnectionEditPart) next;
      connectionEditParts.add(connectionEditPart);
    }
  }

    for (ConnectionEditPart connectionEditPart : connectionEditParts){
      GraphVertex v1 = editPartVertexMap.get(connectionEditPart.getSource());
      GraphVertex v2 = editPartVertexMap.get(connectionEditPart.getTarget());
      GraphEdge edge = new GraphEdge(v1, v2);
      edges.add(edge);
    }

    //kreiranje kompozitne komande, koja sadrzi pojedinacne komande
    //zaduzene za pomeranje elemenata
    final CompoundCommand result = new CompoundCommand();
    //GRAD centralna klasa za rasporedjivanje.
    //Deifnise se automatski nacin, ali se na slican nacin
    //moze iskoristiti bilo koja druga opcija biblioteke
    Layouter<GraphVertex, GraphEdge> layouter =
      new Layouter<GraphVertex, GraphEdge>(
         (List<GraphVertex>) editPartVertexMap.values(), edges,
              LayoutAlgorithms.AUTOMATIC);

    Drawing<GraphVertex, GraphEdge> drawing;
    try {
      drawing = layouter.layout();
      for (EditPart editPart : editPartVertexMap.keySet()) {
        ShapeEditPart shapeEditPart = (ShapeEditPart) editPart;
        //racunanje pomeraja, na osnovu kog se formira komanda
        final Point ptOldLocation =
```

```
            shapeEditPart.getFigure().getBounds().getLocation();
        Point2D location = drawing.getVertexMappings().get(
          editPartVertexMap.get(shapeEditPart));
        shapeEditPart.getFigure().translateToAbsolute(ptOldLocation);
        final Point ptLocation =
            new Point(location.getX(), location.getY());
        final Dimension delta = ptLocation.getDifference(ptOldLocation);

        final Object existingRequest =
        this.findRequest(shapeEditPart.getNotationView(),
            RequestConstants.REQ_MOVE);
        if (existingRequest == null) {
          final ChangeBoundsRequest request = new
          ChangeBoundsRequest(RequestConstants.REQ_MOVE);
          request.setEditParts(shapeEditPart);
          request.setMoveDelta(
              new PrecisionPoint(delta.width, delta.height));
          request.setLocation(
              new PrecisionPoint(ptLocation.x, ptLocation.y));
          final Command cmd = this.buildCommandWrapper(
              request, shapeEditPart);
          if (cmd != null && cmd.canExecute()) {
            result.add(cmd);
          }
        } else if (existingRequest instanceof ChangeBoundsRequest) {
          final ChangeBoundsRequest changeBoundsRequest =
          (ChangeBoundsRequest) existingRequest;
          changeBoundsRequest.setMoveDelta(
              new PrecisionPoint(delta.width, delta.height));
          changeBoundsRequest.setLocation(
              new PrecisionPoint(ptLocation.x, ptLocation.y));
        }
      }
    } catch (CannotBeAppliedException e) {
      e.printStackTrace();
    }
      return result;
  }
```

Iz listinga se može primetiti da instanciranje odgovarajućih *GRAD* klasa i pozivanje automatskog raspoređivanja ne predstavlja poseban izazov. Kompleksniji delovi koda povezani su s kreiranjem komande na način definisan *GEF* i *Sirius* okvirima, što je nezavisno od samog računanja i preuzimanja pozicija elemenata. Koristeći istu logiku kao u listingu, može se pozvati bilo koji konkretni algoritam ili raspoređivanje na osnovu opisa koji poštuje pravila jezika, te rasporediti i kompleksniji dijagrami sa elementima koji sadrže druge ili ivične elemente. Ivični elementi povezani su s običnim, tako što se nalaze na njihovim granicama sa okolinom. Kako u tekućoj diskusiji akcenat nije na upoznavanju sa *Sirius* okruženjem, neće biti posvećeno više pažnje određenim specifičnostima i načinu definisanja raspoređivanja kompozitnih dijagrama.

Naredni korak u zadavanju korišćenja sopstvenog raspoređivača u okviru *Sirius* projekata podrazumeva kreiranje klase koja implementira interfejs `LayoutProvider`, koji definiše metodu `getLayoutNodeProvider`. Metoda prima roditeljski *edit part*, i vraća instancu klase zaduženu za određivanje pozicija sadržanih elemenata. Dakle, može se upotrebiti upravo prethodno napisana klasa koja nasleđuje `AbstractLayoutEditPartProvider`. Poslednja neophodna akcija jeste dodavanje `layoutProvider` ekstenzije u `plugin.xml` uz specifikaciju putanje do kreirane `LayoutProvider` klase kao vrednosti `providerClass` obeležja.

Poglavlje 11

Evaluacija

Evaluacija *GRAD* biblioteke izvršena je sprovođenjem korisničke studije, tokom koje su korisnici (programeri sa do 25 godina iskustva) ocenjivali čitljivost i izgled istih modela automatski kreiranih i vizualizovanih pomoću različitih alata. Takođe, učesnici studije davali su svoj sud o jasnoći nekoliko primera koda za pozivanje raspoređivanja elemenata grafa, uključujući i kod napisan primenom *GRAD* biblioteke.

Cilj studije je bio da se potvrdi da:

1. Dijagrami prikazani unutar grafičkog editora koji za automatsko raspoređivanje elemenata koristi *GRAD* u proseku su vizuelno atraktivniji korisnicima u odnosu na dijagrame sa automatski raspoređenim elementima vizualizovanih pomoću vodećih komercijalnih alata i alata otvorenog koda za modelovanje.

2. *API GRAD* biblioteke je lakši za razumevanje i korišćenje u poređenju sa ovim segmentom drugih *Java* biblioteka za analizu i crtanje grafova.

3. Veliki procenat programera (blizu 50%) preferirao bi korišćenje eksternog JSD-a *GRAD* biblioteka u odnosu na pozivanje algoritma za raspoređivanje elemenata dijagrama na neki drugi način.

11.1 Evaluaciona procedura

Evaluacija je bazirana na vizualizaciji *Cerif* modela upotrebom četiri različita alata za modelovanje UML dijagrama klasa. Reč je o vodećim komercijalnim alatima spomenutog tipa *SAP Sybase PowerDesigner* [170] i *MagicDraw*, sve popularnijem alati otvorenog koda *Pypurus*, te editoru dijagrama klasa *Kroki* alata, koji je integrisan sa *GRAD* bibliotekom. Izabrano je 10 od 31 *Cerif* paketa i otvoreno u svakom od editora.

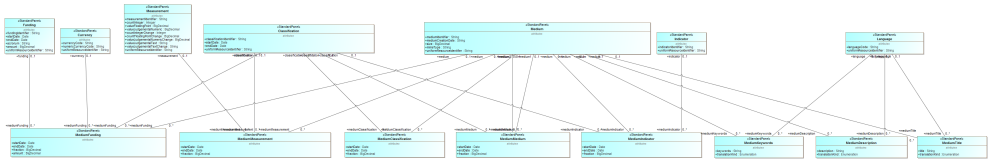
PowerDesigner i *Kroki* automatski raspoređuju elemente dijagrama, dok je u slučaju *MagicDraw* i *Papyrus* alata datu fukcionalnost neophodno ručno pozvati. Međutim, i *MagicDraw* i *Papyrus* raspoređivanje rade potpuno automatski, bez potrebe za učešćem korisnika u izboru i konfiguraciji algoritma. Sva obeležja elemenata dijagrama, osim njihovih pozicija, postavljena su na iste vrednosti unutar svih editora. Ovo je urađeno kako bi se minimizovao uticaj ličnih ukusa učesnika studije, poput preferiranja jedne boje u odnosu na drugu. Učesnicima je prikazano 10 skupova od po 4 dijagrama a nije im rečeno koji dijagram je nastao upotrebom kog editora (po jedan dijagram za svaki alat, za svaki izabrani paket). Od njih se tražilo da:

1. Ocene estetski utisak dijagrama korišćenjem 1-5 Likertove skale, predstavljene gradacijom od odgovora "vrlo loše" do "odlično".

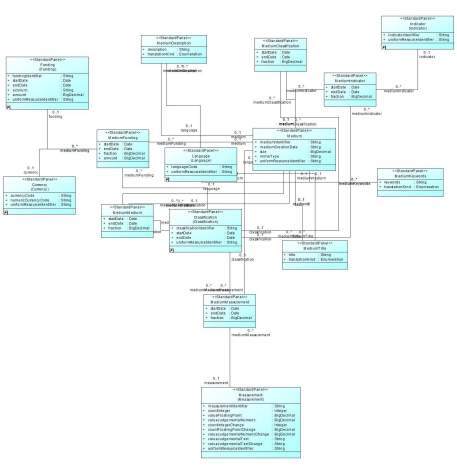
2. Probaju da reše relativno jednostavni zadatak i daju sud o njegovoj težini za svaki dijagram. Radilo se o, na primer, nalaženju putanja između klasa, prebrojavanju broja grana između dve klase (čime se testira koliko dobro koji alat vodi računa o višestrukim granama), prebrojavanju broja grana koje ulaze u neku klasu ili izlaze iz nje itd. Učesnici su mogli izabrati odgovor na skali od "vrlo teško" (1) do "vrlo lako" (5).

Druga grupa pitanja uvedena je kako bi se mogla porediti čitljivost dijagrama, imajući u vidu da dijagram može estetski lepo izgledati, ali da pritom nije lako uočite bitne informacije na njemu. Na primer, dijagram kod kojeg su višestruke grane u potpunosti preklopljene može da izgleda preglednije od onih gde su ove grane razdvojene, ali stoga i skriva bitne relacije između klasa. Prilikom izbora 10 paketa za evaluaciju, vođeno je računa o raznovrsnosti. Naime, zastupljeni su i manji dijagrami (sa 5 ili 6 klasa), veći sa preko 20 klasa, kao i dijagrami sa višestrukim vezama između istih klasa. Jedan primer prikaza istog paketa kroz sva četiri alata dat je na slici 11.1.

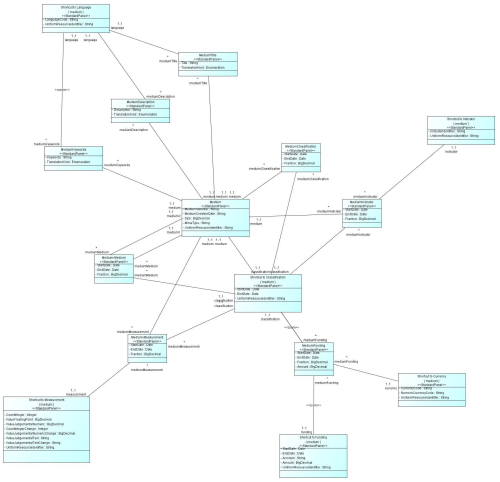
(a) Papyrus



(b) MagicDraw



(c) PowerDesigner



(d) Kroki i GRAD

Slika 11.1: Model paketa srednje veličine prikazan od strane sva 4 alata

U drugoj fazi studije učesnicima je predstavljeno pet primera koda za pozivanje jednog algoritma za raspoređivanje elemenata grafa i preuzimanje rezultata. Među uzorcima koda našli su se *JGraphX*, *JUNG*, *prefuse*, te dva primera koda *GRAD*

biblioteke, gde jedan uključuje JSD, a drugi ne. Svi dijagrami i primeri koda korišćeni u studiji, kao i dobijeni rezultati dostupni su na *Mendeley Data* repozitorijumu naučnih podataka [181].

11.2 Učesnici studije

U studiji je učestvovao ukupno 31 softverski inženjer, sa određenim iskustvom u oblasti *UML* modelovanja. Na samom početku evaluacije, učesnici su popunjavali anketu sa opštim pitanjima, koja su se odnosila na njihovu procenu sopstvenih veština na poljima relevantim za oblast studije. Učesnici su svoje znanje ocenjivali brojem od 1 do 10. Dobijeni rezultati (prosečna vrednost svakog odgovora) prikazani su u tabelama 11.1 i 11.2.

Godine	Godine iskustva programiranja u *Javi*	Godine iskustva u modelovanju
31,4	10	5,6

Tabela 11.1: Prosečna vrednost odgovora na opšta pitanja

Pitanje	1	2	3	4	5	6	7	8	9	10	Prosek
Koliko biste rekli da ste vešti pri modelovanju UML dijagrama klasa?	3%	0%	0%	6%	0%	10%	16%	23%	29%	13%	7.71/10
Da li ste upoznati sa oblašću crtanja grafova?	6%	10%	13%	6%	13%	16%	19%	13%	3%	0%	5.19/10
Kako biste ocenili svoju veštinu u oblasti grafičkog dizajna?	6%	6%	16%	16%	19%	10%	3%	10%	13%	0%	5.03/10

Tabela 11.2: Rezultati upitnika o proceni veštine učesnika studije

Učesnici studije bili su upitani i da li bi, u slučaju da razvijaju sopstveni grafički editor, radije implementirali algoritam za raspoređivanje elemenata dijagrama samostalno, ili koristili gotova rešenja. Velika većina (84%) odgovorila je da bi se radije oslonila na postojeće implementacije.

Rezultati

Rezultati prve faze evaluacije prikazani su u tabelama 11.3 i 11.4. U prvoj od dve tabele nalaze se prosečne ocene estetike crteža sa standardnim devijacijama, dok su u drugoj prikazane procene težine izvršavanja zadatih zadataka. Na slici 11.2 prikazani su *box chart* dijagrami sva četiri alata, kreiranih na osnovu podataka iz tabele 11.3, dok se na slici 11.3 mogu videti dijagrami nastali na osnovu tabele 11.4.

Prosečne ocene pokazuju da su se učesnicima sa estetske strane najviše svideli dijagrami čiji su elementi raspoređeni pomoću *GRAD* biblioteke. Naime, 9 od 10 dijagrama dobilo je ocenu 4/5 ili veću. Takođe se može primetiti da je razlika u ocenama u korist *GRAD* biblioteke u odnosu na druge alate pogotovo velika u slučaju većih dijagrama, kao što su modeli 3, 4, 9 i 10. *GRAD* je dobio najbolje ocene i po pitanju jednostavnosti rešavanja postavljenih zadataka.

Rezultati intuitivnosti korišćenja različitih biblioteka za raspoređivanje elemenata grafa prikazani su u tabeli 11.5. Od učesnika studije tražilo se i da izaberu primer koda

Broj modela	*GRAD*		*PowerDesigner*		*MagicDraw*		*Papyrus*	
	Ocena	SD	Ocena	SD	Ocena	SD	Ocena	SD
1	3,65	4,17	2,71	5,11	3,61	3,19	1,97	5,49
2	4,55	7,39	3,81	4,71	4,03	4,75	2,48	4,26
3	4,29	5,88	2,45	5,42	3,45	4,17	1,65	6,14
4	4,13	5,15	2,1	5,53	3,32	3,87	1,68	6,14
5	4,1	5,19	2,03	5,08	3,58	6,34	1,90	6,65
6	4,1	4,92	2,0	5,67	3,84	4,45	1,81	5,98
7	4,06	6,52	2,32	4,92	2,35	5,95	1,81	5,74
8	4,42	6,85	2,45	4,66	3,87	6,21	2,26	4,83
9	4,29	5,84	2,77	5,0	3,61	5,78	2,52	5,74
10	4,06	4,96	2,48	5,78	3,39	5,88	1,84	5,38

Tabela 11.3: Prosečne ocene estetike crteža i standardne devijacije

Model no	*GRAD*		*PowerDesigner*		*MagicDraw*		*Papyrus*	
	Ocena	SD	Ocena	SD	Ocena	SD	Ocena	SD
1	4,29	6,4	2,9	2,32	2,35	4,49	3,35	4,17
2	4,68	8,45	4,32	6,11	4,1	4,87	2,71	2,23
3	4,52	8,06	2,19	4,96	3,9	4,31	1,61	7,11
4	3,74	4,75	1,97	4,92	3,19	3,76	1,52	7,65
5	4,39	6,68	1,94	5,04	4,1	5,64	1,97	4,83
6	4,35	6,24	1,97	4,83	4,03	4,66	2,0	4,45
7	4,65	8,13	2,26	4,02	3,19	4,17	2,42	3,76
8	4,68	9,06	1,84	5,27	3,32	2,48	2,71	4,17
9	4,13	5,81	2,23	4,02	3,45	4,92	2,48	3,37
10	4,23	5,71	2,23	5,04	3,52	5,56	1,65	6,14

Tabela 11.4: Prosečne ocene težine rešavanja zadataka i standardne devijacije

koji bi najradije koristili i ovi procenti mogu se videti u tabeli 11.6.

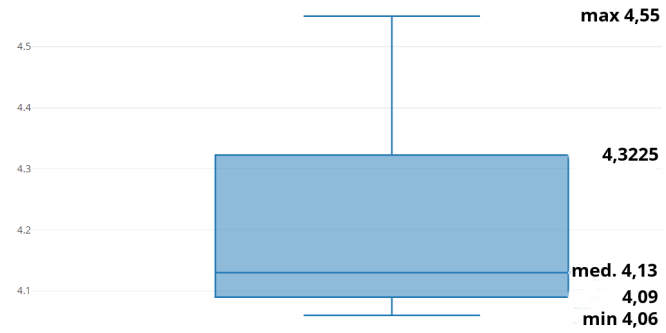
JGraphX		*JUNG*		*GRAD*		*prefuse*		*GRAD* JSD	
Ocena	SD	Ocena	SD	Ocena	SD	Ocena	SD	Ocena	SD
3.06	3.82	3.48	5.78	4.06	6.27	2.45	3.31	3.94	4.31

Tabela 11.5: Prosečne ocene i standardne devijacije intuitivnosti primera koda

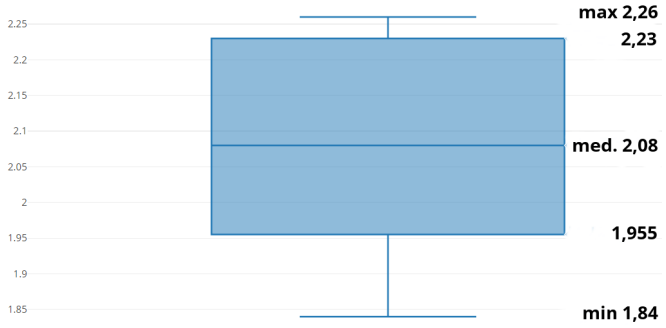
Može se zapaziti da je običan način za pozivanje algoritma za raspoređivanje i preuzimanje rezultata njegovog izvršavanja *GRAD* biblioteke (bez korišćenja jezika specifičnog za domen) dobio najbolje ocene. Sa druge strane, metodu *GRAD* biblioteke koja uključuje JSD izabrao je je najveći broj učesnika studije kao metodu koju bi preferirali da koriste. Kako korišćenje upravospomenutog načina podrazumeva učenje sintakse jezika, nije iznenađujuće da mu neki učesnici studije nisu dali visoku ocenu. Međutim, ova metoda je i najkraća od svih ponuđenih načina, što je skoro polovina ispitanika posebno cenila.

JGraphX	*JUNG*	*GRAD*	*prefuse*	*GRAD* DSL
6%	16%	32%	3%	42%

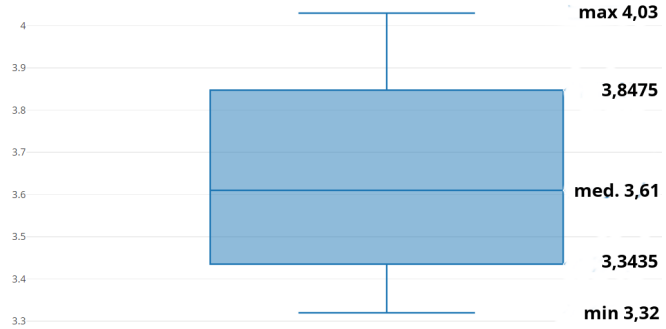
Tabela 11.6: Procenat korisnika koji bi preferirao upotrebu datog primera koda



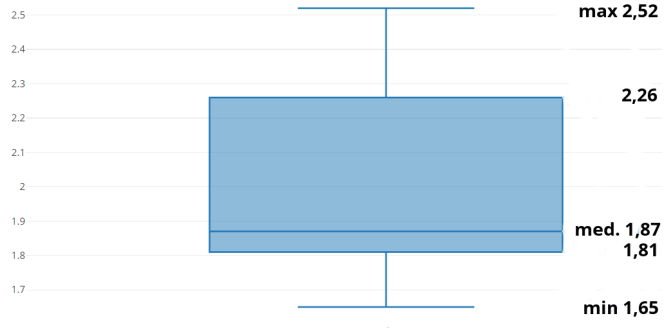
(a) GRAD



(b) PowerDesigner

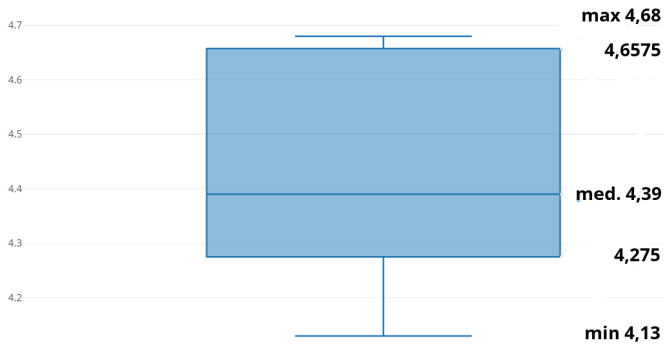


(c) MagicDraw

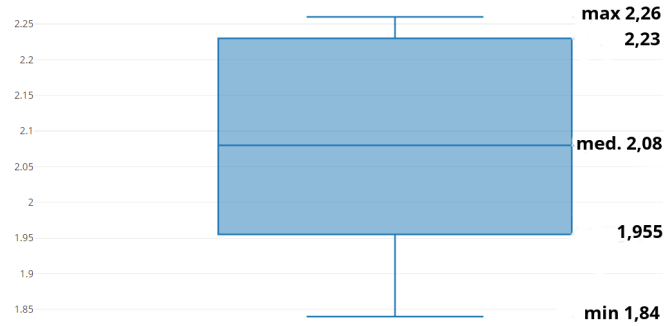


(d) Papyrus

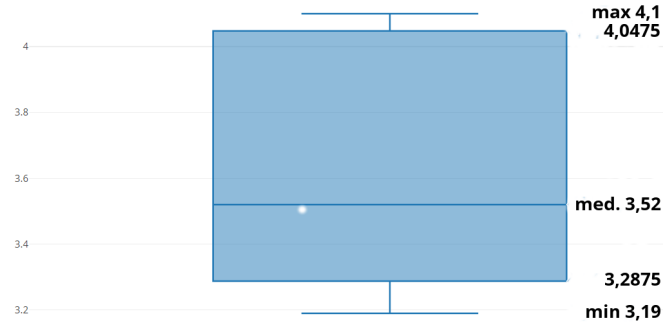
Slika 11.2: *Box chart* dijagrami za sva četiri alata kreirani na osnovu podataka iz tabele 11.3



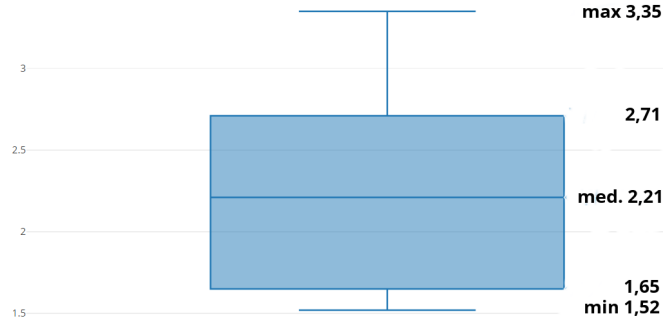
(a) GRAD



(b) PowerDesigner



(c) MagicDraw



(d) Papyrus

Slika 11.3: *Box chart* dijagrami za sva četiri alata krairani na osnovu podataka iz tabele 11.4

Poglavlje 12

Zaključak

U ovoj disertaciji je predstavljeno istraživanje usmereno ka problemu automatskog raspoređivanja elemenata grafova, obuhvatajući izučavanje i implementaciju raznih algoritama ovog tipa, te osmišljavanje i realizaciju načina za pojednostavljenu selekciju, konfiguraciju i primenu odgovarajućeg algoritma.

Crtanje grafova predstavlja spoj matematičkih metoda teorije grafova i vizualizacije informacija i odlikuje se velikom praktičnom primenom. Za jedan graf može se kreirati više crteža, ali se oni mogu znatno razlikovati po pitanju razumljivosti informacija koje prikazuju. Upravo je kreiranje crteža na estetski prihvatljiv način cilj algoritama za automatsko raspoređivanje. Postoji veliki broj ovih algoritama, od koji su neki osmišljeni tako da daju najbolji mogući rezultat ukoliko se primene nad grafovima koji poseduju određene osobine, dok su drugi znatno opštiji i teže kreiranju prihvatljivih crteža bez obzira na tip i veličinu grafa. Izbor i konfiguracija odgovarajućeg algoritma bez poznavanja teorijskih osnova zahteva dosta eksperimentisanja, što je inspirisalo cilj automatizacije datog postupka.

Predložena su dva načina za rešavanje ovog problema, gde prvi obuhvata učešće korisnika, ali na način koji ne zahteva dobro poznavanje oblasti, dok drugi predstavlja potpunu automatizaciju izbora. Druga opcija realizovana je analizom grafa radi provere da li poseduje osobine koji pojedini algoritmi za automatsko raspoređivanje zahtevaju, te izdvajanjem optimalnog na osnovu dobijenih rezultata. Sa druge strane, prva korisniku pruža mogućnost navođenja osobina koje bi dobijeni crtež trebalo da poseduje, te izdvaja algoritam koji će proizvesti rezultat koji je potpuno ili najviše moguće u skladu sa njegovim željama. Ovo je postignuto dizajnom i implementacijom jezika specifičnog za domen opisa prostornog raspoređivanja elemenata grafa.

Prethodno pomenuti načini za automatski izbor i primenu algoritama ne bi bili mogući bez postojanja implementacija niza algoritama za analizu i crtanje grafova. Kako već postoji određen broj biblioteka za programski jezik Java, koje se, između ostalog, bave i crtanjem grafova, one su bile analizirane sa ciljem pronalaska nekih od potrebnih implementacija. Uočena je potreba za implementacijom većeg broja algoritama za analizu grafova, kao i da su određene klase algoritama za automatsko raspoređivanje slabo zastupljene, ako uopšte. Recimo, crtanje stabala i silom usmereni algoritmi su među funkcionalnostima koje svaka ozbiljna biblioteka pomenute namene nudi, dok simetrično i ortogonalno crtanje nije prioritet nijedne od njih. Takođe, primećeno je i da one mahom čvrsto vezuju raspoređivanje elemenata sa komponentama za vizualizaciju, tako da je njihova integracija sa posebno razvijenim grafičkim editorima kompleksnija nego što bi mogla biti. Navedeni problemi inspirisali su razvoj nove biblioteke za crtanje i analizu grafova za programski jezik *Java* nazvanu *GRAD*.

GRAD biblioteka uključuje implementacije bar jednog predstavnika najvažnijih klasa algoritama za crtanje grafova, kao i neophodnih algoritama za njihovu analizu. Tu spadaju ispitivanje planarnosti, povezanosti, cikličnosti, rastavljanje na komponente, itd. Algoritmi za implementaciju birani su tako da obuhvate samo one sa linearnim vremenom izvršavanja, ukoliko za datu klasu problema postoji takvo rešenje. Takođe, predloženi su efikasni načini za realizaciju koraka koji nisu precizno definisani unutar pojedinih implementiranih algoritama, te za proširenje određenih algoritama za raspoređivanje elemenata grafova tako da se vodi računa i o veličini čvorova, kao i rekurzivnim i višestrukim granama. Biblioteka obuhvata jednostavni grafički editor za isprobavanje i upoznavanje sa raznim algoritmima, ali i spomenuta dva načina za pojednostavljeni izbor i konfiguraciju odgovarajućeg. Takođe, omogućena je veoma jednostavna integracija bilo koje od funkcionalnosti *GRAD* biblioteke sa postojećim grafičkim editorima.

Mogućnost primene rešenja je izuzetno velika. Naime, algoritmi za crtanje grafova, kao i njihova automatska selekcija mogu se iskoristiti unutar bilo kojeg grafičkog editora ukoliko je pisan u programskom jeziku *Java* ili podržava spregu sa njim. To može biti bilo editor koji se trenutno razvija, poput grafičkog *UML* editora *Kroki* alata, bilo alat koji putem mehanizma priključaka predviđa uključivanje dodatnih algoritama za raspoređivanje, kao što je popularni *Sirius* okvir. Takođe, algoritmi za analizu grafova mogli bi se bez ograničenja i direktno iskoristiti, za razvoj drugih bibliteka koje se bave teorijom grafova.

Pravci daljeg razvoja pre svega se odnose na implementaciju dodatnih algoritama za crtanje grafova, čime bi se postavila podloga i za širenje jezika i postupka izbora najboljeg algoritma. Bila bi reč pre svega o boljim simetričnim i ortogonalnim algoritmima. Takođe, dalji razvoj bio bi fokusiran i na omogućavanje raspoređivanja svih čvorova, ali samo podskupa svih grana grafa određenim algoritmom, dok bi preostali bili na odgovarajući način rutirani, tako da ne dođe do velikog broja njihovog preseka niti preklapanja. Na primer, omogućilo bi se da se graf nacrta upotrebom algoritma za crtanje stabala, uz kasnije raspoređivanje svih grana koje ne pripadaju njegovom razapinjućem stablu. Po dodavanju ove funkcionalnosti, i jezik i automatska detekcija algoritma na osnovu osobina grafa bi bili prošireni. Dodatno, istraživao bi se i problem automatskog pozicioniranja novih čvorova i grana u okviru grafa čiji postojeći elementi ne bi trebalo da tom prilikom budu pomereni.

Dodatno, buduća unapređenja odnosila bi se i na dizajn i implementaciju jezika specifičnih za domen koji bi omogućili i specifikovanje semantike, kao faktora koji u velikoj meri može uticati na optimalni raspored elemenata. Implementirani jezik predstavlja polaznu tačku za razvoj ovih JSD. Takođe, planirano je i prevođenje *Grad* biblioteke na programski jezik *Pharo*, uz oslonac na *Bloc* projekat [182]. Ideja je zasnovana na činjenici da na datom programskom jeziku još nisu implementirani kompleksniji algoritmi za rasrepođivanje elemenata dijagrama, iako je razvijeno više alata za vizualizaciju.

Bibliografija

[1] R. Shields, "Cultural topology: The seven bridges of konigsburg, 1736," *Theory, Culture and Society*, vol. 29, no. 4-5, pp. 43–57, 2012.

[2] V. Ivančević and I. Luković, "A systematic mapping study on the usage of software tools for graphs within the edm community," in *Workshop on Tools and Technologies in Statistics, Machine Learning and Information Retrieval for Educational Data Mining, SMLIR@EDM2015*, 2015, pp. 6–9.

[3] J. Bondy and U. S. Murty, *Graph Theory with Applications*. Elesevier Science Publishing Co., 1976, ISBN: 0444194517.

[4] J. Anderson, *Automata Theory with Modern Applications*. Cambridge University Press, 2006, ISBN: 0521848873.

[5] *International symposium on graph drawing*, Posećeno: 20.5.2018. [Online]. Available: http://www.graphdrawing.org.

[6] G. D. Battista, P. Eades, R. Tamassia, and I. Tollis, "Algorithms for drawing graphs: An annotated bibliography," *Computational Geometry: Theory and Applications*, vol. 4, pp. 235–282, 5 1994. DOI: 10.1016/0925-7721(94)00014-X.

[7] M. Petre, "Cognitive dimensions 'beyond the notation'," *Journal of Visual Languages and Computing*, vol. 17, no. 4, pp. 292–301, 2006. DOI: 10.1016/j.jvlc.2006.04.003.

[8] M. Schrepfer, J. Wolf, J. Mendling, and H. Reijers, "The impact of secondary notation on process model understanding," in *Proceeding of the Working Conference on the Practice of Enterprise Modeling*, Springer, 2009, pp. 161–175. DOI: 10.1007/978-3-642-05352-8_13.

[9] *Gephi*, Posećeno: 20.5.2018. [Online]. Available: https://gephi.org.

[10] *Graphviz - graph visualization software*, Posećeno: 20.5.2018. [Online]. Available: http://www.graphviz.org.

[11] *Jung framework*, Posećeno: 20.5.2018. [Online]. Available: http://jung.sourceforge.net.

[12] *Jgraphx*, Posećeno: 20.5.2018. [Online]. Available: https://github.com/jgraph/jgraphx.

[13] *G*, Posećeno: 20.5.2018. [Online]. Available: http://geosoft.no/graphics/.

[14] I. Dejanović, R. Vaderna, G. Milosavljević, and Željko Vuković, "Textx: A python tool for domain-specific languages implementation," *Knowledge-Based Systems*, vol. 115, pp. 1–4, 2017. DOI: 10.1016/j.knosys.2016.10.023.

[15] G. Milosavljević, M. Filipović, V. Marsenić, D. Pejaković, and I. Dejanović, "Kroki: A mockup-based tool for participatory development of business applications," in *Intelligent Software Methodologies, Tools and Techniques (SoMeT), 2013 IEEE 12th International Conference on*, IEEE, 2013, pp. 235–242.

[16] *Sirius*, Posećeno: 20.5.2018. [Online]. Available: http://www.eclipse.org/sirius/.

[17] *The mit license*, Posećeno: 20.5.2018. [Online]. Available: https://opensource.orge/licences/MIT.

[18] *Grad izvorni kod*, Posećeno: 20.5.2018. [Online]. Available: https://github.com/renatav/GraphDrawing.

[19] M. Patrignani, "Planarity testing and embedding," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed. Chapman and Hall/CRC, 2007, ch. 1, pp. 1–42, ISBN: 1584884126.

[20] D. Cvetković and M. Milić, *Teorija grafova i njene primene*. Beogradski izdavačko-grafički zavod, 1971.

[21] S. Skiena, "Implementing discrete mathematics: Combinatorics and graph theory with mathematica," in. Addison-Wesley, 1990, ch. 5, p. 189.

[22] E. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*. Prentice Hall College Div, 1977, ISBN: 013152447X.

[23] *Graph automorphisms*, Posećeno: 20.5.2018. [Online]. Available: http://mathworld.wolfram.com/GraphAutomorphism.html.

[24] D. West, *Introduction to graph theory*, Posećeno: 20.5.2018., 2000.

[25] C. Kuratowski, "Sur le problème des courbes gauches en topologie," *Fundamenta Mathematicae*, vol. 15, no. 1, pp. 271–283, 1930.

[26] V. Adamchik, *Algorithmic complexity*, Posećeno: 20.5.2018. [Online]. Available: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Algorithmic%20Complexity/complexity.html.

[27] *Np problem*, Posećeno: 20.5.2018. [Online]. Available: http://mathworld.wolfram.com/NP-Problem.html.

[28] *Np hard problem*, Posećeno: 20.5.2018. [Online]. Available: http://mathworld.wolfram.com/NP-HardProblem.html.

[29] H. Purchase, "Which aesthetic has the greatest effect on human understanding?" In *Proceedings of the 5th International Symposium on Graph Drawing*, Springer-Verlag, 1997, pp. 248–261, ISBN: 3-540-63938-1.

[30] H. Purchase, "Which aesthetic has the greatest effect on human understanding?" In *Proceedings of the 5th International Symposium on Graph Drawing*, ser. GD '97, Springer-Verlag, 1997, pp. 248–261.

[31] ——, "Computer graphics and multimedia: Applications, problems and solutions," in, J. DiMarco, Ed. Idea Group Publishing, 2004, ch. 8, pp. 110–144.

[32] C. Ware, H. Purchase, L. Colpoys, and M. McGill, "Cognitive measurements of graph aesthetics," *Information Visualization*, vol. 1, no. 2, pp. 103–110, 2002. DOI: 10.1057/palgrave.ivs.9500013.

[33] H.Purchase, "Metrics for graph drawing aesthetics," *Journal of Visual Languages and Computing*, vol. 13, pp. 129–137, 2001.

[34] C. Buchheim, M. Chimani, C. Gutwenger, M. Jünger, and P. Mutzel, "Crossings and planarization," in *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2007, ch. 2, pp. 43–86, ISBN: 1584884126.

[35] M. Coleman and S.Parker, "Aesthetics-based graph layout for human consumption," *Software - Practice and Experience*, vol. 26, pp. 1415–1438, 1996. DOI: 10.1002/(SICI)1097-024X(199612)26:12<1415::AID-SPE69>3.3.CO;2-G.

[36] C. Gutwenger and P. Mutzel, "Planar polyline drawings with good angular resolution," in *Proceedings of Graph Drawing Symposium 1998*, Springer, 1998, pp. 167–182.

[37] R. Tamassia, "On embedding a graph in the grid with the minimum number of bends," *SIAM Journal of Computing*, vol. 16, pp. 421–444, 3 1987. DOI: 10.1137/0216030.

[38] A. Papakostas and I. Tollis, "Efficient orthogonal drawings of high degree graphs," *Algorithmica*, vol. 26, pp. 100–125, 2000. DOI: 10.1007/s004539910006.

[39] P. Eades and S.-H. Hong, "Symmetric graph drawing," in *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2007, ch. 3, pp. 87–114, ISBN: 1584884126.

[40] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information Processing Letters*, vol. 31, no. 1, pp. 7–15, 1989. DOI: 10.1016/0020-0190(89)90102-6.

[41] A. Rusu, "Tree drawing algorithms," in *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2007, ch. 5, pp. 155–192, ISBN: 1584884126.

[42] T. Chan, M. Goodrich, S. R. Kosaraju, and R. Tamassia, "Optimizing area and aspect ratio in straight-line orthogonal tree drawings," *Computational Geometry: Theory and Applications*, vol. 23, no. 2, pp. 153–162, 2002. DOI: 10.1016/S0925-7721(01)00066-9.

[43] G. D. Battista, P. Eades, R. Tamassia, and I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, 1998, ISBN: 0133016153.

[44] E. Reingold and J. Tilford, "Tidier drawings of trees," *IEEE Transactions on Software Engineering*, vol. 7, no. 2, pp. 223–228, 1981. DOI: 10.1109/TSE.1981.234519.

[45] C. Wetherell and A. Shannon, "Tidy drawings of trees," *IEEE Transactions on Software Engineering*, vol. 5, no. 5, pp. 514–520, 1979. DOI: 10.1109/TSE.1979.234212.

[46] J. W. II, "A node-positioning algorithm for general trees," *Software—Practice & Experience*, vol. 20, no. 7, pp. 685–705, 1990. DOI: 10.1002/spe.4380200705.

[47] A. Bloesch, "Aesthetic layout of generalized trees," *Software Practice and Experience*, vol. 23, no. 8, 817—827, 1993.

[48] C. Buchheim, M. Jünger, and S. Leipert, "Improving walker's algorithm to run in linear time," in *Revised Papers from the 10th International Symposium on Graph Drawing*, Springer-Verlag, 2002, pp. 344–353, ISBN: 3-540-00158-1.

[49] K. Marriott and P. Sbarski, "Compact layout of layered trees," in *Proceedings of the Thirtieth Australasian Conference on Computer Science*, Australian Computer Society, Inc., 2007, pp. 7–14, ISBN: 1-920-68243-0.

[50] C. Bachmaier, "A radial adaptation of the sugiyama framework for visualizing hierarchical information," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 3, 2007. DOI: 10.1109/TVCG.2007.1000.

[51] M. Bernard and S. Mohammed, "Labeled radial drawing of data structures," in *Proceedings of 7th International Conference on Information Visualisation*, 2003, 479—555.

[52] P. Crescenzi, G. D. Battista, and A. Piperno, "A note on optimal area algorithms for upward drawings of binary trees," *Computational Geometry: Theory and Applications*, 187—2002, 1992. DOI: 10.1016/0925-7721(92)90021-J.

[53] P. Metaxas, G. Pantziou, and A. Symvonis, "A note on parallel algorithms for optimal h-v drawings of binary trees," *Computational Geometry: Theory and Applications*, vol. 9, no. 3, pp. 145–158, 1998. DOI: 10.1016/S0925-7721(96)00018-1.

[54] C.-S. Shin, S. K. Kim, and K.-Y. Chwa, "Area-efficient algorithms for straight-line tree drawings," *Computational Geometry*, vol. 15, pp. 175–202, 2000. DOI: 10.1016/S0925-7721(99)00053-X.

[55] A. Garg and A. Rusu, "Area-efficient order-preserving planar straight-line drawings of ordered trees," in *Proceedings of the 9th Annual International Conference on Computing and Combinatorics*, Springer-Verlag, 2003, pp. 475–486, ISBN: 3-540-40534-8.

[56] P. Eades, "Drawing free trees," *Bulletin of the Institute for Combinatorics and its Applications*, vol. 5, pp. 10–36, 1992.

[57] S. Grivet, D. Auber, J.-P. Domenger, and G. Melancon, "Bubble tree drawing algorithm," in *Proceedings of International Conference on Computer Vision and Graphics*, Springer-Verlag, 2004, 633—641.

[58] C.-C. Lin and H.-C. Yen, "On balloon drawings of rooted trees," in *Proceedings of the 13th International Conference on Graph Drawing*, Springer-Verlag, 2006, pp. 285–296. DOI: 10.1007/11618058_26.

[59] A. Rusu, C. Santiago, and R. Jianu, "Real-time interactive visualization of information hierarchies," in *In Proceedings of 11th International Conference on Information Visualisation*, 2007, 117—123.

[60] I. Halupczok and A. Schulz, "Pinning balloons with perfect angles and optimal area," in *Proceedings of the 19th International Conference on Graph Drawing*, Springer-Verlag, 2012, pp. 154–165. DOI: 10.1007/978-3-642-25878-7_16.

[61] A. Garg and A. Rusu, "Straight-line drawings of general trees with linear area and arbitrary aspect ratio," in *Proceedings of the 2003 International Conference on Computational Science and Its Applications: Part III*, Springer-Verlag, 2003, pp. 876–885, ISBN: 3-540-40156-3.

[62] A. Rusu and C. Santiago, "A practical algorithm for planar straight-line grid drawings of general trees with linear area and arbitrary aspect ratio," in *Proceedings of 17th International Conference on Information Visualisation*, 2007, 743—750. DOI: 10.1109/IV.2007.14.

[63] A. Garg and A. Rusu, "A more practical algorithm for drawing binary trees in linear area with arbitrary aspect ratio," in *Proceedings of 11th International Symposium on Graph Drawing*, 2003, 159––165.

[64] P. Healy and N. Nikolov, "Hierarchical drawing algorithms," in *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2007, ch. 13, pp. 383–408, ISBN: 1584884126.

[65] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981. DOI: 10.1109/TSMC.1981.4308636.

[66] J. Six and I. Tollis, "Circular drawing algorithms," in *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2007, ch. 9, pp. 155–192, ISBN: 1584884126.

[67] J. Six and I. Tollis, "Effective graph visualization via node grouping," in *Software Visualization: From Theory to Practice*, K. Zhang, Ed., Kluwer Academic Publishers, 2003, ch. 14, pp. 413–337.

[68] E. Gansner and Y. Koren, "Improved circular layouts," in *Proceedings of the 14th International Conference on Graph Drawing*, Springer-Verlag, 2007, pp. 386–398, ISBN: 978-3-540-70903-9.

[69] J. Six and I. Tollis, "A framework and algorithms for circular drawings of graphs," *Journal of Discrete Algorithms*, vol. 4, 25—50, 1 2006. DOI: 10.1016/j.jda.2005.01.009.

[70] J. Manning, "Geometric symmetry in graphs," PhD thesis, 1990.

[71] S.-H. Hong and P. Eades, "Symmetric layout of disconnected graphs," in *Prooceedings of International Symposium on Algorithms and Computation*, Springer, 2003, pp. 405–414.

[72] ——, "Drawing planar graphs symmetrically ii: Biconnected planar graphs," *Algorithmica*, vol. 42, no. 2, 159––197, 2005.

[73] ——, "Drawing planar graphs symmetrically, iii: One-connected planar graphs," *Algorithmica*, vol. 44, no. 1, 67—100, 2006.

[74] S.-H. Hong, B. McKay, and P. Eades, "A linear time algorithm for constructing maximally symmetric straight-line drawings of planar graphs," *Discrete & Computational Geometry*, vol. 36, pp. 283–311, 2. DOI: 10.1007/s00454-006-1231-5.

[75] W. Tutte, "How to draw a graph," *Proceedings of the London Mathematical Society*, vol. 13, pp. 743–768, 1963.

[76] S.-H. Hong, B. McKay, and P. Eades, "A linear time algorithm for constructing maximally symmetric straight line drawings of triconnected planar graphs," *Discrete & Computational Geometry*, vol. 36, no. 2, 283—311, 2006. DOI: 10.1007/s00454-006-1231-5.

[77] H. Carr and W. Kocay, "An algorithm for drawing a graph symmetrically," *Bulletin of the Institute of Combinatorics and its Applications*, vol. 27, pp. 19–25, 1999.

[78] L. Vismara, "Planar straight-line drawing algorithms," in *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2007, ch. 6, pp. 193–222, ISBN: 1584884126.

[79] N. Chiba, K. Onoguchi, and T. Nishizeki, "Linear algorithms for convex drawings of planar graphs," *Progress in Graph Theory*, vol. 173, pp. 155–173, 1984.

[80] S. Hong and H. Nagamochi, "A linear-time algorithm for symmetric convex drawings of internally triconnected plane graphs," *Algorithmica*, vol. 58, pp. 433–460, 2 2010. DOI: 10.1007/s00453-008-9275-y.

[81] ——, "Convex drawings of hierarchical planar graphs and clustered planar graphs," *Journal of Discrete Algorithms*, vol. 8, pp. 282–295, 3 2010. DOI: 10.1016/j.jda.2009.05.003.

[82] I. García-Marco and K. Knauer, "Drawing graphs with vertices and edges in convex position," *Computational Geometry*, vol. 58, pp. 25–33, 2016. DOI: 10.1016/j.comgeo.2016.06.002.

[83] C. Duncan and M. Goodrich, "Planar orthogonal and polyline drawing algorithms," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed. Chapman and Hall/CRC, 2007, ch. 7, pp. 223–246.

[84] R. H.J. M. Otten and J. G. van Wijk, "Graph representations in interactive layout design," in *Proceedings of the IEEE Internatational Symposium on Circuits and Systems*, 1978, pp. 914–918.

[85] P. Rosenstiehl and R. Tarjan, "Rectilinear planar layouts and bipolar orientations of planar graphs," *Discrete & Computational Geometry*, vol. 1, pp. 343–353, 4 1986. DOI: 10.1007/BF02187706.

[86] H. Zhang and X. He, "Improved visibility representation of plane graphs," *Computational Geometry*, vol. 30, pp. 29–39, 1 2005. DOI: 10.1016/j.comgeo.2004.08.002.

[87] C.-Y. Chen, Y.-F. Hung, and H.-I. Lu, "Visibility representations of four-connected plane graphs with near optimal heights," *Computational Geometry: Theory and Applications*, vol. 42, pp. 865–872, 9 2009. DOI: 10.1016/j.comgeo.2009.01.006.

[88] F. Brandenburg, "1-visibility representations of 1-planar graphs," *Journal of Graph Algorithms and Applications*, vol. 18, no. 3, pp. 421–438, 2014. DOI: JournalofGraphAlgorithmsandApplications.

[89] A. Garg and R. Tamassia, "On the computational complexity of upward and rectilinear planarity testing," *SIAM Journal of Computing*, vol. 31, no. 2, pp. 601–625, 2002. DOI: 10.1137/S0097539794277123.

[90] S. Cornelsen and A. Karrenbauer, "Accelerated bend minimization," in *Proceedings of the 19th International Conference on Graph Drawing*, Springer-Verlag, 2012, pp. 111–122. DOI: 10.1007/978-3-642-25878-7_12.

[91] T. Nishizeki and S. Rahman, "Rectangular drawing algorithms," in *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC, 2007, ch. 10, pp. 317–348, ISBN: 1584884126.

[92] S. Rahman, S.-I. Nakano, and T. Nishizeki, "Rectangular grid drawings of plane graphs," *Computational Geometry: Theory and Applications*, vol. 10, no. 3, pp. 203–220, 1998. DOI: 10.1016/S0925-7721(98)00003-0.

[93] G. Kant, "Drawing planar graphs using the canonical ordering," *Algorithmica*, vol. 16, pp. 4–32, 1996. DOI: 0.1007/BF02086606.

[94] C. Duncan and S. Kobourov, "Polar coordinate drawing of planar graphs with good angular resolution," *Journal of Graph Algorithms and Applications*, vol. 7, no. 4, pp. 311–333, 2003, ISSN: 1526-1719.

[95] D. Auber, N. Bonichon, P. Dorbec, and C. Pennarun, "Rook-drawing for plane graphs," in *Revised Selected Papers of the 23rd International Symposium on Graph Drawing and Network Visualization*, Springer-Verlag New York, Inc., 2015, pp. 180–191. DOI: 10.1007/978-3-319-27261-0_15.

[96] S. Kobourov, "Force-directed drawing algorithms," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed. Chapman and Hall/CRC, 2007, ch. 12, pp. 383–408.

[97] *Spring embedders and force directed graph drawing algorithms*, Posećeno: 20.5.2018., 2012. [Online]. Available: http://arxiv.org/abs/1201.3011.

[98] P. Eades, "Heuristic for graph drawing," *Congressus Numerantium*, vol. 42, pp. 149–160, 1984.

[99] T. Fruchterman and E. Reingold, "Graph drawing by force-directed placement," *Software—Practice & Experience*, vol. 21, no. 11, pp. 1129–1164, 1991. DOI: 10.1002/spe.4380211102.

[100] R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing," *ACM Transactions on Graphics*, vol. 15, no. 4, pp. 301–331, 1996. DOI: 10.1145/234535.234538.

[101] R. Hadany and D. Harel, "A multi-scale algorithm for drawing graphs nicely," *Discrete Applied Mathematics*, vol. 113, 3—21, 1 2001. DOI: 10.1016/S0166-218X(00)00389-9.

[102] S. Hachul and M. Jünger, "Drawing large graphs with a potential-field-based multilevel algorithm," in *Proceedings of the 12th International Conference on Graph Drawing*, Springer-Verlag, 2004, pp. 285–295. DOI: 10.1007/978-3-540-31843-9_29.

[103] Y. Hu, "Efficient and high quality force-directed graph drawing," vol. 10, pp. 37–71, 2005.

[104] M. Khoury, Y. Hu, S. Krishnan, and C. Scheidegger, "Drawing large graphs by low-rank stress majorization," *Computer Graphics Forum*, vol. 31, 3 2012. DOI: 10.1111/j.1467-8659.2012.03090.x.

[105] J. Hua, M. L. Huang, and Q. V. Nguyen, "Drawing large weighted graphs using clustered force-directed algorithm," in *Proceedings of 18th International Conference on Information Visualisation*, 2014. DOI: 10.1109/IV.2014.24.

[106] G. Liotta, "Proximity drawings," in *Handbook of Graph Drawing and Visualization*, R. Tamassia, Ed. Chapman and Hall/CRC, 2007, ch. 4, pp. 1–42, ISBN: 1584884126.

[107] M. Mernik, J. Heering, and A. Sloane, "When and how to develop domain-specific languages," *ACM Computing Surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005.

[108] I. Dejanović, "Prilog metodama brzog razvoja softvera na bazi proširivih jezičkih specifikacija," PhD thesis, 2011.

[109] A. van Deursen and P. Klint, "Little languages: Little maintenance," *Journal of Software Maintenance*, vol. 10, no. 2, pp. 75–92, 1998. DOI: 10.1002/(SICI)1096-908X(199803/04)10:2<75::AID-SMR168>3.0.CO;2-5.

[110] J. Sprinkle, M. Mernik, J.-P. Tolvanen, and D. Spinellis, "What kinds of nails need a domain-specific hammer?" *IEEE Software*, vol. 26, no. 4, pp. 15–18, 2009. DOI: 10.1109/MS.2009.92.

[111] M. Fowler, *Domain specific languages*, Posećeno: 20.5.2018. [Online]. Available: http://martinfowler.com/dslwip/.

[112] U. Zdun and M. Strembeck, "Architectural decisions for dsl design: Foundational decisions in dsl development," in *Proceedings of 14th European Conference on Pattern Languages of Programs*, Germany, 2009, pp. 1–37.

[113] *Yfiles for java*, Posećeno: 20.5.2018. [Online]. Available: http://www.yworks.com/products/yfiles-for-java-2.x.

[114] *Bsd license*, Posećeno: 20.5.2018. [Online]. Available: www.linfo.org/bsdlicense.html.

[115] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Stanford InfoLab, Technical Report, 1999.

[116] B. Meyer, "Self-organizing graphs - a neural network perspective of graph layout," in *In Neural Computers, 393–406, ECKMILLER*, Springer, 1998, pp. 246–262.

[117] *The dot language*, Posećeno: 20.5.2018. [Online]. Available: http://www.graphviz.org/doc/info/lang.html.

[118] *Smetana*, Posećeno: 20.5.2018. [Online]. Available: https://github.com/plantuml/smetana/.

[119] *Setting custom sizes of the jung graph vertices*, Posećeno: 20.5.2018. [Online]. Available: http://sourceforge.net/p/jung/discussion/252062/thread/0b98adc5.

[120] A. Bergel, S. Maass, S. Ducasse, and T. Girba, "A domain-specific language for visualizing software dependencies as a graph," in *Second IEEE Working Conference on Software Visualization (VISSOFT)*, 2014, pp. 45–49. DOI: 10.1109/VISSOFT.2014.17.

[121] *Pharo programming language and environment*, Posećeno: 20.5.2018. [Online]. Available: http://pharo.org.

[122] *Dsl based approach to input graph data in graph theory based java programs*, Posećeno: 20.5.2018. [Online]. Available: https://sanaulla.info/2013/07/19/dsl-based-approach-to-input-graph-data-in-graph-theory-based-java-programs.

[123] S. Hong, H. Chafi, E. Sedlar, and K. Olukotun, "Green-marl: A dsl for easy and efficient graph analysis," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 349–362. DOI: 10.1145/2150976.2151013.

[124] J. Hopcroft and R. Tarjan, "Efficient algorithms for graph manipulation," Tech. Rep., 1971.

[125] S. Even, *Graph Algorithms*, 2nd. Cambridge University Press, 2011, ISBN: 0521736536, 9780521736534.

[126] *Klasifikacija dfs grana*, Posećeno: 20.5.2018. [Online]. Available: https://courses.csail.mit.edu/6.006/fall11/rec/rec14.pdf.

[127] E. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. DOI: 10.1007/BF01386390.

[128] E. Reingold, J. N. Jurg, and N. Deo, *Combinatorial Algorithms: Theory and Practice*. Prentice Hall College Div, 1977, ISBN: 013152447X.

[129] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Communications of the ACM*, vol. 16, pp. 372–378, 1973. DOI: 10.1145/362248.362272.

[130] F. Harary, *Graph Theory*. Avalon Publishing, 1969, ISBN: 0201410338.

[131] J. Hopcroft and R. Tarjan, "Dividing a graph into triconnected components," *SIAM Journal on Computing*, vol. 2, 135–158, 1973.

[132] "On separation pairs and split components of biconnected graphs," Tech. Rep., 2011.

[133] C. Gutwenger and P. Mutzel, "A linear time implementation of spqr-trees," in *Proceedings of the 8th International Symposium on Graph Drawing*, Springer-Verlag, 2001, pp. 77–90, ISBN: 3-540-41554-8.

[134] G. D. Battista and R. Tamassia, "Incremental planarity testing," in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1989, pp. 436–441, ISBN: 0-8186-1982-1. DOI: 10.1109/SFCS.1989.63515.

[135] P. Bertolazzi, G. D. Battista, G. Liotta, and C. Mannino, "Optimal upward planarity testing of single-source digraphs," *SIAM Joural on Computing*, vol. 27, no. 1, 132—169, 1998.

[136] E. Dahlhaus, "A linear time algorithm to recognize clustered planar graphs and its parallelization," in *Proceedings of the 3rd Latin American Symposium on Theoretical Informatics*, 1998, pp. 239–248.

[137] P. Bertolazzi, G. D. Battista, and W. Didimo, "Computing orthogonal drawings with the minimum number of bends," in *Proceedings of the 5th Workshop on Algorithms and Data Structures*, 1997, 331—344.

[138] G. D. Battista and R. Tamassia, "On-line maintenance of triconnected components with spqr-trees," *Algorithmica*, vol. 15, 302—318, 1996.

[139] N. Gibbs, "Basic cycle generation [h]," *Communications of the ACM*, vol. 18, no. 5, pp. 275–276, 1975, ISSN: 0001-0782. DOI: 10.1145/360762.360778.

[140] C. Gotlieb and D. Corneil, "Algorithms for finding a fundamental set of cycles for an undirected linear graph," *Communications of the ACM*, vol. 10, no. 12, pp. 780–783, 1967, ISSN: 0001-0782. DOI: 10.1145/363848.363861.

[141] K. Paton, "An algorithm for finding a fundamental set of cycles of a graph," *Communications of the ACM*, vol. 12, no. 9, pp. 514–518, Sep. 1969, ISSN: 0001-0782. DOI: 10.1145/363219.363232.

[142] R. Tarjan, "Enumeration of the elementary circuits of a directed graph," Tech. Rep., 1972.

[143] A. Ehrenfeucht, L. Fosdick, and L. Osterweil, "An algorithm for finding the elementary circuits of a directed graph," Tech. Rep., 1973.

[144] D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM Journal on Computing*, vol. 4, no. 1, pp. 77–84, 1975.

[145] A. Lempel, S. Even, and I. Cederbaum, "An algorithm for planarity testing of graphs," in *Theory of graphs: International symposium*, vol. 67, 1967, pp. 215–232.

[146] S. Even and R. Tarjan, "Computing an st-numbering," *Theoretical Computer Science*, vol. 2, no. 3, pp. 339–334, 1976. DOI: 10.1016/0304-3975(76)90086-4.

[147] J. Hopcroft and R. Tarjan, "Efficient planarity testing," *Hournal of the Association for Computing Machinery*, vol. 21, no. 4, pp. 549–568, 1974.

[148] L. Auslander and V. Parter, "On imbedding graphs in the sphere," *Journal of Mathematics and Mechanics*, vol. 10, no. 3, pp. 517–523, 1961.

[149] H. de Fraysseix, P. O. de Mendez, and P. Rosenstiehl, "Trémaux trees and planarity," *International Journal of Foundations of Computer Science*, vol. 17, 1017–1030, 2006.

[150] K. Booth and G. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms," *Journal of Computer and System Sciences*, vol. 13, no. 3, pp. 335–379, 1976. DOI: 10.1016/S0022-0000(76)80045-1.

[151] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa, "A linear algorithm for embedding planar graphs using pq-trees," *Journal of Computer and System Sciences*, vol. 30, no. 1, pp. 54–76, 1985. DOI: 10.1016/0022-0000(85)90004-2.

[152] J. Boyer and W. Myrvold, "On the cutting edge: Simplified o(n) planarity by edge addition," *Journal of Graph Algorithms and Applications*, vol. 8, 241–273, 2004.

[153] J. Westbrook, "Fast incremental planarity testing," in *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, Springer-Verlag, 1992, pp. 342–353, ISBN: 3-540-55719-9.

[154] J. A. La Poutré, "Alpha-algorithms for incremental planarity testing (preliminary version)," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '94, ACM, 1994, pp. 706–715, ISBN: 0-89791-663-8. DOI: 10.1145/195058.195439.

[155] S. Fialko and P. Mutzel, "A new approximation algorithm for the planar augmentation problem," in *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '98, Society for Industrial and Applied Mathematics, 1998, pp. 260–269, ISBN: 0-89871-410-9.

[156] B. McKay and A. Piperno, "Practical graph isomorphism, {ii}," *Journal of Symbolic Computation*, vol. 60, pp. 94–112, 2014. DOI: 10.1016/j.jsc.2013.09.003.

[157] B. D. McKay, "Practical graph isomorphism," *Congressus Numerantium*, vol. 30, pp. 45–87, 1981.

[158] S. Hartke and A. Radcliffe, *McKay's Canonical Graph Labeling Algorithm.* American Mathematical Society, 2009, vol. 479, pp. 99–111.

[159] H. S. Coxeter, "Barycentric coordinates," in *Introduction to Geometry*, 1969, ch. 13, pp. 216–221.

[160] *Wolfram baricentrične koordinate*, Posećeno: 20.5.2018. [Online]. Available: http://mathworld.wolfram.com/BarycentricCoordinates.html.

[161] C. Gotsman, *Tutte's embedding theorem reproven and extended*, Posećeno: 20.5.2018. [Online]. Available: http://www3.math.tu-berlin.de/geometrie/ps/ddg07/slides/Gotsman.pdf.

[162] M. Mernik, D. Hrnčić, B. R. Bryant, and F. Javed, "Applications of grammatical inference in software engineering : Domain specific language development," *Scientific applications of language methods*, vol. 2, pp. 421–457, 2011. DOI: 10.1142/9781848165458_0008.

[163] I. Čeh, M. Crepinsek, T. Kosar, and M. Mernik, "Ontology driven development of domain-specific languages," *Computer Science and Informaion Systems*, vol. 8, pp. 317–342, 2011. DOI: 10.2298/CSIS101231019C..

[164] T. Kosar, M. Mernik, and J. C. Carver, "Program comprehension of domain-specific and general-purpose languages: Comparison using a family of experiments," *Empirical Software Engineering*, vol. 17, pp. 276–304, 2012. DOI: 10.1007/s10664-011-9172-x.

[165] W. Cazzola and E. Vacchi, "Language components for modular dsls using traits," *Computer Languages, Systems and Structures*, vol. 45, pp. 16–34, 2015. DOI: 10.1016/j.cl.2015.12.001.

[166] T. Kosar, S. Bohra, and M. Mernik, "Domain-specific languages: A systematic mapping study," *Information and Software Technology*, vol. 71, pp. 77–91, 2016. DOI: doi.org/10.1016/j.infsof.2015.11.001.

[167] *Textx gramatika*, Posećeno: 20.5.2018. [Online]. Available: http://www.igordejanovic.net/textX/grammar/.

[168] *Jython*, Posećeno: 20.5.2018. [Online]. Available: http://www.jython.org.

[169] *Izvorni kod grafičkog editora*, Posećeno: 6.4.2018. [Online]. Available: https://github.com/renatav/GraphDrawing/tree/master/GraphEditor.

[170] *Powerdesigner*, Posećeno: 20.5.2018. [Online]. Available: https://www.sap.com/westbalkans/product/data-mgmt/powerdesigner-data-modeling-tools.html.

[171] *Magicdraw*, Posećeno: 20.5.2018. [Online]. Available: https://www.nomagic.com/products/magicdraw.

[172] *Gef*, Posećeno: 20.5.2018. [Online]. Available: http://www.eclipse.org/gef/.

[173] *Kroki mockup tool*, Posećeno: 20.5.2018. [Online]. Available: http://www.kroki-mde.net.

[174] R. Vaderna, G. Milosavljević, I. Dejanović, M. Filipović, and Željko Ivković, "Enhancing kroki tool by providing a graphical uml editor," in *Proceedings of the 2014 YuInfo Conference*, 2014.

[175] Željko Ivković, R. Vaderna, and I. D. Milorad Filipović Gordana Milosavljević, "Implementation of a basis for cooperation between kroki tool and uml modeling tools," in *Proceedings of the 2014 YuInfo Conference*, 2014.

[176] *Cerif model*, Posećeno: 20.5.2018. [Online]. Available: http://www.eurocris.org/cerif/introduction/.

[177] M. Filipović, R. Vaderna, Željko Ivković, S. Kaplar, Željko Vuković, I. Dejanović, G. Milosavljević, and D. Ivanović, "Application of kroki mockup tool to implementation of executable cerif specification," in *13th International Conference on Current Research Information Systems, CRIS 2016*, 2017, pp. 245–252.

[178] *Gmp*, Posećeno: 20.5.2018. [Online]. Available: https://www.eclipse.org/modeling/gmp/?project=gmf-tooling\#gmf-tooling.

[179] *Emf*, Posećeno: 20.5.2018. [Online]. Available: http://www.eclipse.org/modeling/emf/.

[180] *Graphical modeling framework*, Posećeno: 20.5.2018. [Online]. Available: http://www.eclipse.org/modeling/gmp/.

[181] R. Vaderna, *Grad evaluation results, mendeley data*, Posećeno: 20.5.2018., 2017. DOI: 10.17632/yswvvz8hjs.1.

[182] *Bloc - low-level ui infrastructure and framework for pharo*, Posećeno: 6.6. 2018. [Online]. Available: https://github.com/pharo-graphics/Bloc.