Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Departman za matematiku i informatiku

Milana Gatarić

NOVI ALGORITAM ZA KOMBINATORNO PARTICIONISANJE HIPERGRAFOVA

Master rad

Novi Sad, oktobar 2010

Sadržaj

Predgovor

Problem kojim se bavim u svom master radu jeste problem particionisanja hipergrafova. Značaj proučavanja ovog problema proističe iz njegovih mnogobrojnih praktičnih primena, od kojih su možda najpoznatije primene kod dizajna VLSI kola i održavanja velikih softverskih sistema. Poznato je da je problem particionisanja hipergrafa uz minimizaciju broja prelomljenih hipergrana i uslov balansa NP-težak. Dakle, za sada ne znamo algoritam koji ovaj problem rešava u polinomnom vremenu, niti je izvesno da li takav algoritam uopšte i postoji. Međutim, radi neophodnosti rešavanja ovog problema za hipergrafove koji imaju i po nekoliko stotina hiljada čvorova, razvijene su razne heuristike sa približno linearnim vremenom izvršavanja. Njima dobijamo samo lokalno rešenje za koje je teško proceniti kvalitet s obzirom da optimalno rešenje nije poznato. U ovom radu dat je pregled nekih od postojaćih algoritama za rešavanje problema particionisanja hipergrafa, a zatim je data i ideja novog algoritma koji bi trebao rešavati dati problem do optimuma. Izloženi su neki od rezultata nakon implementacije ovog algoritma u programskom jeziku C++. Takođe, ostavljene su smernice za neki dalji rad na poboljšavanju ovog algoritma.

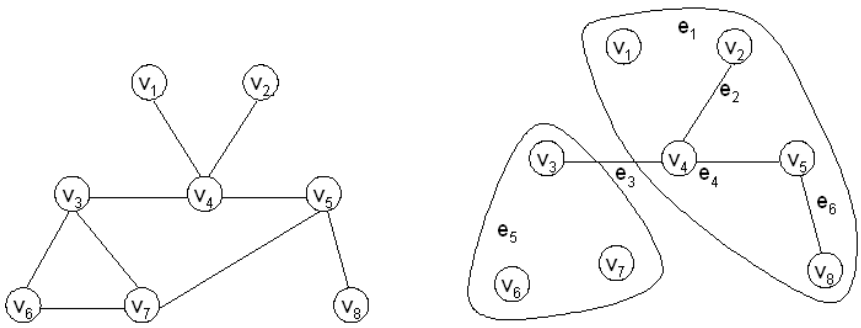
Kratak pregled po poglavljima izgledao bi ovako. U uvodnom poglavlju pokazujemo kako se realni problemi iz prakse mogu modelirati problemom particionisanja hipergrafa i izdvajamo neke od njegovih praktičnih primena rukovodeći se pregledom datim u [6]. Na kraju uvodnog poglavlja dajemo definicije osnovnih pojmova koje koristimo u radu i govorimo o računskoj složenosti, a zatim svrstavamo problem particionisanja hipergrafova u odgovarajuću klasu. U drugom poglavlju dajemo pregled najpoznatijih algoritama kojima se ovaj problem može rešiti, pri čemu smo se vodili klasifikacijom datom u [1]. U trećem poglavlju izlažemo ideju zajedno sa rezultatima implementacije novog algoritma koji rešava problem particionisanja hipergrafova do optimuma. U poslednjem, četvrtom poglavlju dajemo zaključak.

Ovde želim da se zahvalim na veoma korisnim savetima, suštinskim predlozima i podršci doc. dr Nebojši Gvozdenoviću. Tema mog master rada koju mi je on predložio bila je veoma izazovna za mene. Međutim, završavanjem ovog rada uz njegovu pomoć dobila sam mnogo u pogledu proširivanja svog znanja. Takođe, zahvaljujem se i svom mentoru prof. dr Draganu Mašuloviću koji mi je u pogledu završavanja ovog master rada izažao u susret. Ovim putem želim da mu se zahvalim i na divnim predavanjima iz predmeta Diskretna matematika tokom mojih osnovnih studija, koja su me motivisala da se dublje zainteresujem za ovu oblast. Na kraju, ogromnu zahvalnost želim da izrazim i prof. dr Nataši Krejić za neiscrpno razumevanje i pomoć, za zadržavanje zainteresovanosti za matematičku analizu kao i za sve one mnogobrojne napisane preporuke tokom mojih osnovnih i master studija.

1. Uvod

U praksi se često susrećemo sa problemom podele neke grupe objekata na više manjih delova, modula, pri čemu je poželjno da se objekti koji su međusobno jako povezani nađu u istom modulu, nasuprot objektima koji nisu povezani i koji mogu da se nađu u različitim modulima. Time dolazimo do matematičkog problema particionisanja skupa nekakvih elemenata, tj. grupe objekata čiju povezanost možemo predstaviti koristeći strukture kao što su grafovi i hipergrafovi. Na osnovu povezanosti objekata predstavljene grafom možemo zatim da definišemo funkciju cilja koju želimo da optimizujemo prilikom particionisanja, kao što je na primer broj presečenih grana u particionisanom grafu.

Objekti koje želimo da podelimo u module su u grafovima i hepergrafovima predstavljeni čvorovima kojima može biti dodeljena neka vrednost, tj. težina čvora. Postojanje veze između objekata predstavljena je granom u grafu odnosno hipergranom u hipergrafu. U hipergrafovima je moguće predstaviti relaciju grupe objekata, dok su u grafu prisutne samo binarne relacije. Dakle, u grafu grana spaja dva čvora pa je kardinalnost grane dva, dok u hipergrafu hipergrane mogu biti proizvoljne kardinalnosti. Primeri grafa i hipergrafa dati su na slici (1).



Slika 1: Primer grafa (levo) i primer hipergrafa (desno)

Grafovi i hipergrafovi mogu biti particionisani tako da se optimizuje neka funkcija cilja. Obično se particioniše skup čvorova, dok se funkcija cilja definiše nad skupom grana, odnosno skupom hipergrana. U ovom radu ćemo se baviti problemom particionisanja hipergrafa tako da se minimizuje broj prelomljenih hipergrana, tj. onih hipergrana čiji se čvorovi nakon particionisanja nalaze u više od jednog modula. Pri tome dodajemo uslov, takozvani uslov balansa, da ukupna veličina svakog od modula, recimo broj čvorova u svakom od modula mora biti ograničen.

Mnogi problemi iz prakse se mogu interpretirati kao problemi particionisanja hipergrafova i ovde izdvajamo neke od njih.

Primena kod podele softvera

Održavanje velikog informacionog sistema ili softvera jeste obiman posao za samo jednog menadžera. Posao se olakšava ukoliko se takav sistem podeli na nekoliko manjih modula o kojima se potom može brinuti samo po jedan menadžer. Prilikom podele važno je voditi računa o tome koje informacije ostaju dostupne izvan modula. Pri tome troškovi se optimizuju ukoliko je takvih informacija što manje, dakle ukoliko se komunikacija među modulima svede na minimum. Dobrom patricijom ovakvog sistema veličina svakog od modula je ograničena, a ukupan broj informacija dostupnih izvan matičnog modula je minimizovan.

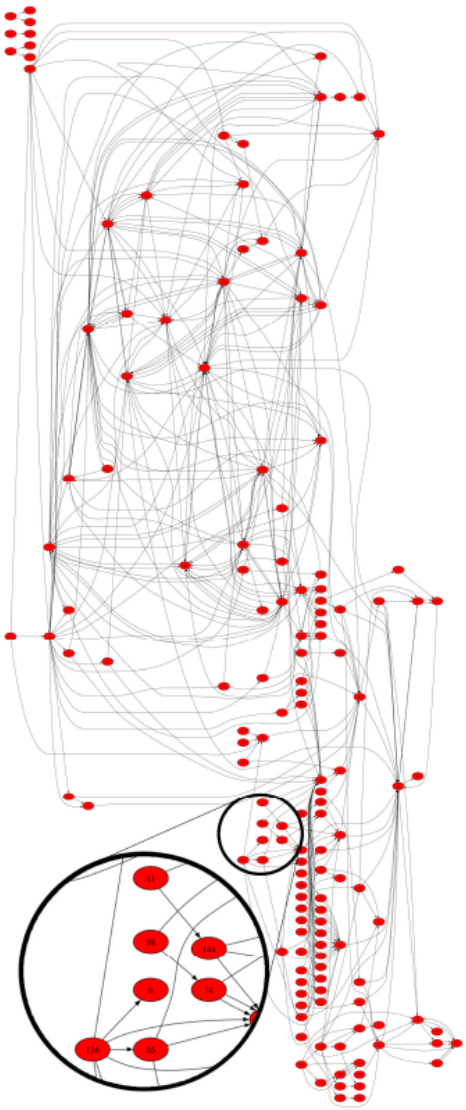
Ovakav softverski sistem možemo predstaviti jednim orjentisanim grafom $D = (V, E)$, tako da su čvorovima u grafu D pretstavljene neke programske jedinice, kao što su na primer procedure ili klase, dok je granama predstavljena komunikacija između njih, slika (2). Tako grana (u, v), koja vodi od čvora u do čvora v, znači da procedura u poziva proceduru v. Pri tome, svakom čvoru w možemo dodeliti težinu $\omega(w)$, koja predstavlja na primer broj linija koda ili zahtevnost održavanja.

Mi želimo da podelimo ovakav graf u određeni broj modula, tj. disjunktne podskupove čvorova, pri čemu svaki od modula ne sme imati preveliku težinu. Pri tome želimo da minimizujemo broj onih čvorova koji se nakon podele nalaze u modulu različitom od onog u kom se nalazi neki od čvorova koji ga pozivaju. Takav čvor nakon podele mora ostati dostupan javnosti jer se poziva iz modula različitog od onog kom pripada, pa njega nazivamo interfejs. Dakle, pri podeli grafa želimo da svedemo broj interfejsa na minimum. Ovim smo dobili problem particionisanja orjentisanog grafa uz minimizaciju interfejsa ili CGP (eng. *call graph partitioning*) problem [13]. Problem particionisanja orjentisanog grafa uz minimizaciju broja interfejsa je ekvivalentan problemu particionisanja hipergrafa, što ćemo pokazati u nastavku ovog rada.

Primena kod VLSI kola

VLSI je engleska skraćenica od pojma *Very Large Scale Integration* i odnosi se na integrisana kola velike gustine. S obzirom na veliki broj komponenata ovakvih sistema njihovo dizajniranje postaje komplikovano. Mogućnost podele u klastere, takve da se minimizuje povezanost među delovima koji čine te klastere, optimizovala bi način njihovog pakovanja i integrisanja.

Korespondenciju sa hipergrafovima možemo dati na primeru lokičkog kola prikazanog na slici (3). Logička kola se sastoje od kapija povezanih metalnim žicama koje izvršavaju logičke operacije. Jadan električni signal može da se postire od jedne do nekoliko drugih kapija i tada za te kapije kažemo da čine mrežu. Ove mreže predstavljamo hipergranom u odgovarajućem hipergrafu, a kapije čvorovima. Tako možemo iskoristiti hipergraf da



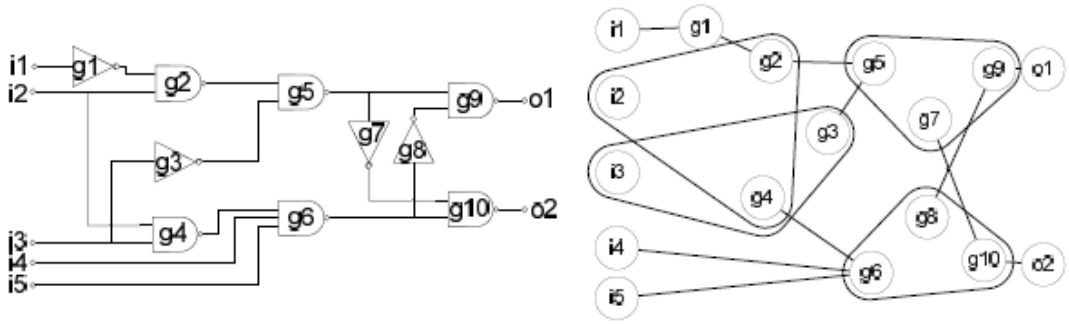
Slika 2: Primer informacionog sistema prdastavljenog orjentisanim grafom

predstavimo povezanost među delovima u kolu.

Razvoj algoritama za particionisanje hipergrafova radi VLSI dizajna podstican je već dugi niz godina, videti na primer [9], [10]. Njihova uspešnost testirana je na hipergrafovima koji sadrže milione čvorova i za koje je karakteristično da im se broj hipergrana kreće između $0.8|V|$ i $1.5|V|$, gde je $|V|$ broj čvorova, a pri tome hipergrane sadrže u proseku 3 do 5 čvorova.

Primena u numeričkoj linearnoj algebri

Vreme izvršavanja operacija linearne algebre kao što je množenje matrica se ubrzava kada se one primenjuju na blok-dijagonalne matrice. Takve operacije se tada mogu izvršavati paralelno nad odgovarajućim blokovima, jer operacije nad različitim blokovima

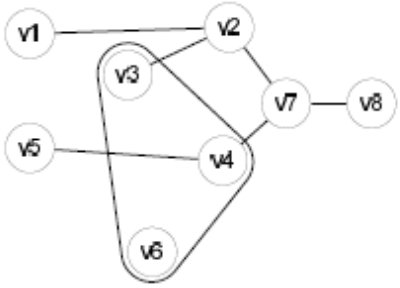


Slika 3: Primer logičkog kola i odgovarajućeg hipergrafa

su međusobno nezavisne. Poznato je da se rezultat operacije nad permutovanom matricom razlikuje za istu permutaciju koja je izvršena nad matricom. Zbog toga je pogodno kada radimo sa retkim matricama permutovati njihove kolone kako bi elemente različite od nule grupisali oko dijagonale i time dobili blok-dijagonalne matrice.

Jedan od načina da dobijemo blok-dijagonalnu matricu jeste primena algoritma particije hipergrafa nad matricom kojom je zapravo predsavljen neki hipergraf. Ovde dajemo objašenje kako se to matricom može predstaviti hipergraf. Vrstama matrice A odgovaraju hipergrane, a kolonama čvorovi, tako da element različit od nule $A_{e,v} \neq 0$ tumačimo da čvor v pripada hipergrani e. Kada primenimo algoritam za particiju hipergrafa, kolone matrice će biti permutovane tako da se čvorovi iste particije nalaze jedan do drugog. Kako se prilikom particije hipergrafa vodi računa da što manje hipergrana bude presečeno, dobićemo matricu sa elementima različitim od nule grupisanim oko dijagonale. Primer ovako dobijene blok-dijagonalne matrice i njenog odgovarajućeg hipergrafa dat je na slici (4).

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



Slika 4: Primer približno blok-dijagonalne matrice i odgovarajućeg hipergrafa

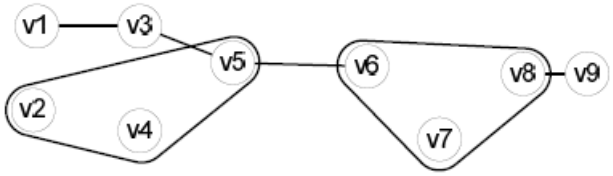
Primena kod Bulovih formula

Konjuktivna normalna forma Bulove formule sastoji se od Bulovih promenljivih i njihovih komplemenata grupisanih operacijom "i" u klauzule koje su međusobno povezane operacijom "ili". Ovakvu formulu možemo predstaviti kao hipergraf tako što svaki literal (promenljiva je jedan literal, a njen komplement drugi literal) predstavljamo čvorom u hipergrafu, a svaku klauzulu hipergranom. Primer konjuktivno normalne forme Bulove

formule i njenog odgovarajućeg hipergrafa dat je na slici (5).

Algoritmi koji manipulišu Bulovim formulama postaju efikasniji ukoliko smo u stanju da grupišemo Bulove promenljive tako da se minimizuje presek hipergrana koje odgovaraju klauzulama date formule. I tu opet nailazimo na primenu particije hipergrafa.

$$\phi = (v1 + v3)(v3 + v5)(v2 + v5 + v4)$$
$$(v5 + v6)(v6 + v7 + v8)(v8 + v9)$$



Slika 5: Primer Bulove formule i odgovarajućeg hipergrafa

1.1. Notacija i osnovni pojmovi

Ovde se upoznajemo sa oznakama i osnovnim pojmovima koje ćemo koristiti u daljem radu, a i formalno definišemo problem particionisanja hipergrafova.

Definicija 1. *Hipergraf je uređena dvojka $H = (V, E)$, gde je $V = \{v_1, ..., v_n\}$ neprazan konačan skup, koji nazivamo skup čvorova, a $E = \{e_1, ..., e_m\} \subset \mathcal{P}(V)$ je skup hipergrana.*

Napomenimo da ukoliko u gornjoj definiciji uzmemo da je $E \subset V^{(2)}$, $V^{(2)} = \{\{u, v\} | u, v \in V, u \neq v\}$ svaka hipergrana će biti kardinalnosti dva i dobićemo sistem $G = (V, E)$ poznat pod nazivom *graf*. Ukoliko su grane u grafu uređene dvojke (u, v), dakle $E \subset V^2 = \{(u, v) | u, v \in V, u \neq v\}$, onda je takav graf orjentisan i njega još zovemo *digraf*, a obeležavamo sa $D = (V, E)$. Kod grane (u, v) orjentisanog grafa kažemo da je u prethodnik čvora v i v sledbenik čvora u, a granu možemo zapisivati $u \to v$.

Pogodan oblik matrice za predstavljanje hipergrafa je dat sledećom definicijom.

Definicija 2. *Matrica hipergrafa $H = (V, E)$, za $V = \{v_1, ..., v_n\}$ i $E = \{e_1, ..., e_m\}$, je matrica $A = [a_{ij}]$ dimenzija $n \times m$ data sa*

$$a_{ij} = \begin{cases} 1 & , \quad ako\ v_i \in e_j \\ 0 & , \quad inače. \end{cases}$$

Za razliku od hipergrafova, grafove je pogodno predstavljati matricom susedstva koju dajemo u sledećoj definiciji.

Definicija 3. *Matrica susedstva grafa $G = (V, E)$, za $V = \{v_1, ..., v_n\}$ i $E = \{e_1, ..., e_m\}$, je matrica $A = [a_{ij}]$ dimenzija $n \times m$ data sa*

$$a_{ij} = \begin{cases} 1 & , \quad ako\ postoji\ e \in E\ tako\ da\ v_i \in e, v_j \in e \\ 0 & , \quad inače. \end{cases}$$

Definicija 4. *Stepen čvora $v \in V$ hipergrafa $H = (V, E)$ je broj različitih hipergrana u E koje sadrže v. Dužina hipergrane $e \in E$ je njena kardinalnost $|e|$.*

Definicija 5. *Particija hipergrafa $H = (V, E)$ je podskup partitivnog skupa nad skupom čvorova $\Pi \subset \mathcal{P}(V)$, tako da je $P_i \cap P_j = \emptyset$ za sve $P_i, P_j \in \Pi$ $i \neq j$, $i \bigcup_i P_i = \Pi$.*

Napomenimo ovde da ćemo delove P_i particije $\Pi = \{P_1, ..., P_n\}$ nazivati još i modulima particije. Dalje, ukoliko je $|\Pi| = k$ i $k > 2$, particija Π se naziva k-particija ili multi-particija, a ako je $k = 2$ onda je u pitanju biparticija. Proces izračunavanja k-particije hipergrafa naziva se k-particionisanje hipergrafa, a proces izračunavanja biparticije hipergrafa biparticionisanje. Pri tome napomenimo da se izrčunavanje k-particije često u algoritmima ne izvršava direktno, već rekurzivnim biparticionisanjem.

Kako bi problem particionisanja hipergrafovava učinili smislenim u pogledu modeliranja problema iz prakse, uvešćemo funkciju cilja particionisanja kao i skup ograničenja čime ćemo dobiti podskup dopustivih particija iz skupa svih mogućih particija hipergrafa. Radi ovog koncepta uvodimo težine na čvorove i/ili na hipergrane u hipergrafu. Dakle, u hipergrafu $H = (V, E)$ svakom čvoru $v \in V$ može biti dodeljena neka celobrojna ili realna vrednost $\omega(v)$ koju nazivamo težina čvora v. Na primer, kao što smo videli, u problemu iz prakse ta vrednost može oslikavati broj linija koda neke procedure kod softverskog sistema. Isto tako, svakoj grani $e \in E$ može biti dodeljena vrednost $\omega(e)$ koja, na primer, u praktičnim problemima može da oslikava stepen povezanosti među čvorovima te grane. U ovom radu uglavnom ćemo posmatrati hipergrafove kod kojih je samo čvorovima dodeljena neka težina.

Za početak ćemo formalno uvesti funkciju cilja particionisanja hipergrafa, koju možemo izraziti kao funkciju hipergrafa $H(V, E)$ i njegove particije Π tako da nam ona daje meru kvaliteta te particije Π nad hipergrafom H.

Definicija 6. *Neka $f_c : (\mathcal{P}(V), E) \to \mathbb{R}$ tako da $f_c(\Pi, E)$ predstavlja cenu (ili neku drugu kvantitativnu meru) particije Π primenjene na hipergraf $H(V, E)$. Funkcija f_c se naziva funkcija cilja particionisanja hipergrafa.*

Funkcija cilja može biti definisana na različite načine što zavisi od konkretnog problema. Nas će zanimati ona koja oslikava kvalitet particije hipergrafa u smislu da su čvorovi unutar istih delova particije jako povezani a čvorovi različitih delova slabo, jer ćemo posmatrati situaciju kada je skupo održavati vezu izmeču čvorova različitih delova particije. Dakle, nakon particionisanja poželjno bi bilo imati što manje hipergrana koje imaju čvorove u više različitih delova date particije, tj. što više hipergrana koje su cele smeštene u neki od modula. Radi preciziranja poželjne situacije uvodimo dodatnu definiciju prelomljene hipergrane.

Definicija 7. *Za hipergranu $e \in E$ hipergrafa $H(V, E)$ kažemo da je prelomljena particijom Π ako postoje $P_i, P_j \in \Pi$, $i \neq j$, i različiti čvorovi $v_i, v_j \in e$ takvi da $v_i \in P_i$ i $v_j \in P_j$.*

Dakle, nama će biti ineresantna funkcija cilja koja minimizira kardinalnost skupa prelomljenih hipergrana particijom Π hipergrafa $H(V, E)$, a pri tome za hipergrane ćemo smatrati da nemaju dodeljene težine. Pa ako takav skup prelomljenih hipergrana označimo sa E_Π, funkciju cilja možemo zapisati kao

$$f_c(\Pi, E) = |E_\Pi|.$$

Složenije funkcije cilja particionisanja hipergrafa bile bi one koje uzimaju u obzir u koliko različitih podsistema se našla prelomljena hipergrana. Vidimo da nas to u slučaju particionisanja hipergrafa koji smo dobili na osnovu digrafa neće interesovati, jer ukoliko je čvor v_i interfejs on postaje vidljiv za sve ostale čvorove i neće nas zanimati koliko tačno drugih modula ga vidi.

Sada ćemo definisati skup ograničenja koji će nam dati podskup dopustivih particija iz skupa svih mogućih particija hipergrafa. Smatraćemo da su u hipergrafu svim čvorovima dodeljene težine pa stoga možemo uvesti težine svakog od delova date particije.

Definicija 8. *Neka je funkcije $f_\omega : \mathcal{P}(V) \to \mathbb{R}$ takva da za svaki skup $A \subseteq V$, $f_\omega(A)$ definiše težinu datog podskupa čvorova. Ovu funkciju koristimo da definišemo težine svakog dela particije $P \in \Pi$.*

Konkretno, nas će interesovati slučaj kad je $f_\omega(A)$ zbir težina čvorava iz A, a skup ograničenja će nam osigurati da težine različitih delova date particije budu balansirane. Dakle, imaćemo prosečnu težinu jednog dela particije Π datu sa W_{sr}, u zavisnosti od ukupne težine svih čvorova i kardinalnosti particije Π, a pojedinačnim delovima particije Π nećemo dozvoliti da pređu tu težinu za više od nekog unapred datog faktora ν, gde je $0 < \nu < 1$.

Sada možemo formalno da definišemo problem particionisanja hipergrafa.

Definicija 9. *(Problem particionisanja hipergrafa) Posmatrajmo hipergraf $H(V, E)$. Neka je dato $k > 1$ i parametar prekoračenja $0 < \nu < 1$, cilj je naći particiju $\Pi = \{P_1, ..., P_k\}$ sa odgovarajućim težinama $W_i = f_\omega(P_i)$, $1 \leq i \leq k$, tako da*

$$W_i < (1 + \nu)W_{sr} \tag{1}$$

važi za sve $1 \leq i \leq k$, i tako da je funkcija cilja $f_c(\Pi, E)$ optimizovana. Pri tome, vidimo da je ovde $W_{sr} = \sum_{i=1}^{k} W_i/k$.

1.2. Računska složenost problema particionisanja hipergrafova

U ovom poglavlju ćemo se najpre kratko osvrnuti na pojmove i rezultate teorije računske složenosti, a zatim svrstati problem particionisanja hipergrafova u konkretnu klasu složenosti.

Definisanje računske složenosti algoritamski rešivog problema se vrši korišćenjem Tjuringove mašine koja predstavlja apstraktni matematički model računara. Tjuringova mašina je između ostalog specifizirana konačnim skupom stanja Q i konačnim skupom ulaznih simbola Σ (azbukom). Algoritam kreće sa izvršavanjem na Tjuringovoj mašini u nekom početnom stanju $q_0 \in Q$, a dalji nastavak njegovog izvršavanja se odvija po nekoj funkciji prelaza $\delta : Q \times \Sigma \to Q \times \Sigma$. Algoritam se završava kada se mašina nađe u nekom od stanaja zaustavljanaja iz Q.

Ovim smo opisali rad determinističke Tjuringove mašine, kod koje je savako naredno stanje jednoznačno, tj. deterministički određeno. Međutim, pored ovih postoje i nedeterministički modeli gde u svakom trenutku mašina ima na raspolaganju nekoliko mogućnosti među kojima bira naredno stanje. Dakle, kod nedeterminističke Tjuringove mašine, umesto funkcije prelaza imamo relaciju, tako da za svaki par stanje-simbol možemo imati nijedan, jedan ili više odgovarajućih sledbenika iz $Q \times \Sigma$.

Ovakve Tjuringove mašine koristimo za rešavanje problema odlučivanja, problema kod kojih je izlazni podatak binaran (da/ne, $\top/\bot$), tj. kod kojih se pitamo da li objekti predstavljeni ulaznim podacima imaju neko svojstvo ili ne. Naš problem optimalnog particionisanja hipergrafa može se preformulisati tako da bude problem odlučivanja na sledeći način. Za dati hipergraf pitamo se da li postoji particija koja zadovoljava uslov (1) tako da vrednost funkcije cilja f_c bude manja ili jednaka od nekog zadatog broja m. Optimalna particija se nalazi ponavljanjem rešavanja ovog problema birajići pogodno m.

Sada ćemo uvesti pojmove vremenske i prostorne složenosti. Polazimo od toga da pod korakom algoritma podrazumevamo izvršenje jedne komande Tjuringove mašine koja implementira odgovarajući problem. Vremenska složenost problema je broj koraka koji je potreban da bi se problem rešio na Tjuringovoj mašini korišćenjem najefikasnijeg algoritma, a izražen u zavisnosti od veličine ulaznog podatka. Pri tome nas ne zanima tačna složenost, već samo ocena reda veličine razmatrane složenosti za koju se koristi asimptotska, takozvana $\mathcal{O}$-notacija. Slično se definiše i prostorna složenost kojom procenjujemo potrošnju memorije mašine od strane razmatranog programa, kao što je broj polja trake koji koristi data Tjuringova mašina tokom određenog izračunavanja.

Za probleme koji imaju sličnu vremensku (ili prostornu) složenost kažemo da pripadaju istoj klasi složenosti. Pomoću tih klasa složenosti možemo vršiti klasifikaciju različitih problema odlučivanja. Najznačajnija klasa algoritama bila bi klasa polinomnih algoritama, odlučivih u vremenu koje se može oceniti polinomnom funkcijom po veličini ulazne reči. Oni su značajni u smislu da se sa stanovišta vremena smatraju dobrim, vremenski efikasnim. Ovu klasu problema obeležavamo sa $\mathbf{P}$. Analogno deterministčkom slučaju, imamo klasu problema koji mogu biti rešeni u polinomnom vremenu na nedeterminističkoj Tju-

ringovoj mašini, a tu klasu obeležavamo sa **NP**.

Jedno od najznačajnijih i najintrigantnijih otvorenih pitanja moderne matematike i teorijskog računarstva jeste pitanje da li važi $\mathbf{P} = \mathbf{NP}$? Kako je svaka deterministička mašina specijalan slučaj nedeterminističke, klasa jezika svih nedeterminističkih sadrži klasu jezika determinističkih Tjuringovih mašina. Prema tome, inkluzija $\mathbf{P} \subseteq \mathbf{NP}$ važi, pa se ovaj problem svodi na pitanje da li se svaki nedeterministički algoritam može determinizovati tako da pri tome ostane sačuvana polinomna vremenska složenost.

U klasi **NP** nalaze se u izvesnom smislu reprezentativni problemi, tzv **NP**-kompletni problemi. Njihov značaj je u tome što ukoliko se za ma koji **NP**-kompletan problem odlučivanja pokaže egzistencija polinomnog algoritma za rešavanje, time se pokazuje jednakost klasa **P** i **NP**.

Ako su $\mathcal{A}$ i $\mathcal{B}$ dva problema odlučivanja, kažemo da se $\mathcal{A}$ polinomno redukuje na $\mathcal{B}$, u oznaci $\mathcal{A} \leq_P \mathcal{B}$, ukoliko postoji redukciona funkcija koja je izračunljiva u determinističkom polinomnom vremenu i koja svaku instancu problema $\mathcal{A}$ transformiše u ekvivalentnu instancu probema $\mathcal{B}$. Ovako se definiše (polinomna) redukcija jednog problema odlučivanja na drugi. Intuitivno, $\mathcal{A} \leq_P \mathcal{B}$ znači da problem $\mathcal{A}$ nije teži od problema $\mathcal{B}$.

Kažemo da je problem $\mathcal{A}$ **NP**-težak ako za sve probleme $\mathcal{B} \in \mathbf{NP}$ važi $\mathcal{B} \leq_P \mathcal{A}$. Ako je problem $\mathcal{A}$ **NP**-težak i pri tome još $\mathcal{A} \in \mathbf{NP}$, onda kažemo da je on **NP**-kompletan. Dakle, ovi **NP**-kompletni problemi jesu najteži problemi klase **NP**. Njihov značaj dat je sledećom teoremom. Za dokaz teoreme pogledati [11].

Teorema 1. *Neka je $\mathcal{A}$ **NP**-kompletan problem. Tada važi*

$$\mathcal{A} \in \boldsymbol{P} \Leftrightarrow \boldsymbol{P} = \boldsymbol{NP}.$$

Prema tome, kako bi se potvrdila jednakost $\mathbf{P} = \mathbf{NP}$, dovoljno je za bilo koji od **NP**-kompletnih problema pronaći polinomni algoritam. Obratno, ako bi bilo moguće za ma koji od tih problema pokazati nepostojanje polinomnog algoritma, odmah bi sledilo $\mathbf{P} \neq \mathbf{NP}$.

Prvi otkriven **NP**-kompletan problem bio je problem zadovoljivosti iskaznih formula, a njegovu kompletnost pokazali su Stiven Kuk i Leonid Levin. S obzirom na teoremu koju navodimo odmah u nastavku (za dokaz videti [11]), moguće je naći i mnogo drugih **NP**-kompletnih problema.

Teorema 2. *Neka je $\mathcal{A}$ **NP**-kompletan problem. Ako je $\mathcal{A} \leq_P \mathcal{B}$ i $\mathcal{B} \in \boldsymbol{NP}$, tada je i problem $\mathcal{B}$ takođe **NP**-kompletan.*

Dakle, ukoliko imamo neki problem $\mathcal{B}$ za koji znamo da je iz klase **NP** a želimo da pokažemo da je **NP**-kompletan, uzmemo neki drugi problem $\mathcal{A}$ za koji znamo da je **NP**-kompletan i pokažemo da važi $\mathcal{A} \leq_P \mathcal{B}$.

Jedan od načina da pokažemo da je neki problem u klasi **NP** jeste da nađemo njegov verifikujući problem koji je u klasi **P**. Za detaljnije objašnjenje verifikacije i sertifikata pogledati [11]. Ako želimo da pokažemo da je optimalno biparticionisanje hipergrafa u

NP, njegov sertifikat bio bi podskup skupa čvorova $S \subseteq V$, a verifikujući problem bi se sastojao u proveri da li je $\sum_{v \in S} \omega(v) \leq K$, gde je K dat kapacitet jednog modula, i da li je odgovarajuća funkcija cilja manja od neke date vrednosti x, tj. da li je $f_c(S) < x$, gde funkcija f_c broji prelomljene hipergrane particijom $\{S, V \setminus S\}$. Provera nejednakosti $\sum_{v \in S} \omega(v) \leq K$ i $f_c(S) < x$ se može izvršiti u polinomnom vremenu po veličini $|V|$, stoga verifikujući problem jeste u **P**. Iz ovoga zaključujemo da je problem optimalnog bipartikcionisanja hipergrafa u klasi **NP**.

Na osnovu Teoreme 2, da bi pokazali da je problem optimalnog biparticionisanja hipergrafa (BPH) **NP**-kompletan, potrebno je još pokazati da važi $\mathcal{A} \leq_P$BPH, za neki poznati **NP**-kompletan problem $\mathcal{A}$. Za problem $\mathcal{A}$ možemo uzeti problem polovljenja koji jeste **NP**-kompletan problem [11].

Problem Polovljenja. Dat je konačan skup S i funkcija težine $t : S \to \mathbb{N}$. Utvrditi da li postoji podskup $S' \subseteq S$ tako da je

$$\sum_{x \in S'} t(x) = \sum_{x \in S \setminus S'} t(x).$$

Uređenim parom (S, t) je ovde zapravo definisan multiskup prirodnih brojeva, a mi se pitamo da li postoji način da se multiskup prirodnih brojeva podeli na dva disjunktna podskupa tako da sume elemenata u njima budu jednake? U nastavku pokazujemo da važi

Problem Polovljenja $\leq_P$ Problem Biparticionisanja Hipergrafa.

Pođimo od multiskupa (S, t) i konstruišimo instancu problema biparticionisanja hipergrafa $H = (V, E)$. Neka je $\Sigma = \sum_{x \in S} t(x)$. Skup čvorova V će sadržati ćvorove težine jedan tako da je $|V| = \Sigma$. Skup disjunktnih hipergrana u E obrazujemo tako što za svaki elemenat $x \in S$ uvodimo hipergranu koja povezuje $t(x)$ čvorova. Sada vidimo da važi: postoji tražena biparticija $\{S', S \setminus S'\}$ skupa S ako i samo ako postoji biparticija hipergrafa H tako da nijedna hipergrana nije presečena.

Šta više, ako u skup hipergrana E pored ovih $|S|$ disjunktnih hipergrana, uvedemo još m hipergrana (koje ne moraju biti disjunktne), tako da je težina svake od njih bar $\Sigma/2$, onda važi: postoji tražena biparticija $\{S', S \setminus S'\}$ skupa S ako i samo ako postoji biparticija hipergrafa H tako da nije presečeno više od m hipergrana.

Pokazavši da je problem biparticionisanja hipergrafa u klasi **NP** i da se **NP**-kompletan problem polovljenja može polinomno redukovati na njega, pokazali smo da je i problem biparticionisanja hipergrafa **NP**-kompetan. Slično i za problem particionisanja hipergrafa se može pokazati da je **NP**-kompletan, dakle problem kada tražimo particiju kardinalnosti k hipergrafa H. Ovaj problem dat Definicijom 9 kada preformulišemo kao problem odlučivanja, može se dobiti na isti način redukcijom u nastavku datog problema koji je **NP**-kompletan, videti [11].

Problem Pakovanja u Korpe (eng. *Bin Packing*). Dat je konačan skup S i funkcija zapremine $t : S \to \mathbb{N}$, zapremina korpe B i prirodan broj $k \geq 1$. Utvrditi da li je moguće rasporediti elemente skupa S u ne više od k korpi.

2. Različiti algoritmi za rešavanje problema particionisanja hipergrafova

U mnogim praktičnim primenama particionisanja hipergrafa, veličina hipergrafova stalno i veoma brzo raste. Na primer, broj čvorova u hipergrafovima koji modeliraju VLSI kola je reda veličine deset miliona, a očekuje se da će taj broj vrlo brzo dostići i red veličine sto miliona. S obzirom na ogromne brojke ulaznih podataka, od algoritama za rešavanje ovakvih problema očekuje se da imaju bar blizu linearnu složenost.

S obzirom na složenost problema particionisanja hipergrafova, a u cilju brzog rešavanja radi praktične primene, razvijene su mnogobrojne heuristike, algoritmi koji jesu dovoljno brzi ali daju samo lokalno rešenje zavisno od početno zadate particije. Analitičko poređenje ovakvih algoritama je teško jer greške dobijenog rešenja od optimalnog su često neizračunljive. Radi poređenja ovakvih heuristika, tipično je da se veći broj puta primene na probleme uzete iz prakse i zatim uzme prosečna dobijena vrednost funkcije cilja i prosečno vreme izvršavanja.

Ovde ćemo razmatrati nekoliko različitih heuristika grupe algoritama baziranih na pomeranju čvorova. Ovakvi algoritmi su se pokazali pogodni za rešavanje problema particionisanja hipergrafova do 200 čvorova. Zatim ćemo razmatrati algoritme sa multilevel pristupom koji su se pokazali efikasnim za rešavanje problema i preko 200 čvorova, pa i do nekoliko stotina hiljada čvorova.

Kasnije u radu pokazaćemo kako se problem particionisanja orjentisanog grafa uz minimizaciju broja interfejsa može interpretirati kao problem particionisanja hipergrafa uz minimizaciju broja prelomljenih hipergrana, a na kraju ovog poglavlja pokazaćemo kako se taj problem može prevesti u problem celobrojnog linearnog programiranja i potom rešiti do optimuma, videti [13]. Kako je problem celobrojnog linearnog programiranja NP-težak, za sad ne postoji algoritam za rešavanje ovog problema u polinomnom vremenu. Zbog složenosti problema celobrojnog programiranja, na ovaj način se mogu tretirati samo grafovi do 1500 čvorova.

Spomenimo da je jedan od načina rešavanja problema particije hipergrafa da se hipergraf najpre pogodno transformiše u običan graf i onda primeni algoritam za particionisanje grafa tako da se minimizuje broj prelomljenih grana. Pri tome, za particionisanje običnog grafa mogu se primeniti isti metodi koji će ovde biti razmatrani u kontekstu hipergrafova, algoritmi bazirani na pomeranju čvorova kao i oni bazirani na multilevel pristupu. Takođe, pored ovih postoje i mnogi drugi algoritmi razvijeni za particionisanje običnih grafova.

2.1. Algoritmi bazirani na pomeranju čvorova

Algoritmi ove kategorije istražuju prostor rešenja krećući se od jednog ka drugom dopustivom rešenju. Oni itetrativno konstruišu novo dopustivo rešenje time što trenutno dopustivo rešenje izmene tako da se dobije susedno rešenje iz skupa dopustivih rešenja. Spomenuta izmena rešenja podrazumeva pomeranje jednog ili više čvorova između različitih delova particije. Pri tome, susedno dopustivo rešenje je ono dopustivo rešenje koje je moguće dobiti od trenutnog jednom izmenom. Tako se prostor rešenja istražuje iterativnim pomeranjem od trenutnog ka susednom dopustivom rešenju sve dok se ne zadovolji neki uslov zaustavljanja.

U ovom postupku čuvanje predašnjih pokreta može biti iskorišteno radi navođenja kretanja po prostoru rešenja. Ta ideja je primenjena kod iterativnih algoritama KL i FM koje ćemo ovde razmatrati. Oni generišu naredno rešenje rukovodeći se samo najboljom mogućom izmenom trenutnog rešenja, u smislu da najbolje poboljšava trenutnu vrednost funkcije cilja. Ovi algoritmi se zaustavljaju kada dobijemo rešenje takvo da sva susedna dopustiva rešenja ne daju bolju vrednost funkcije cilja. Stoga zaključujemo da ovi algoritmi konvergiraju ka lokalnom minimumu s obzirom na početno uzeto rešenje. Postoje izvesne dorade ovih algoritama u cilju da se postigne efekat premošćivanja lokalnog minimuma, što ih je učinilo primenljivim u realnim problemima.

Algoritmi bazirani na pomeranju čvorova se veoma često sreću u literaturi i praktičnim primenama, a to je tako iz nekoliko razloga. Genaralno ovi algoritmi su vrlo intuitivni, dato rešenje se na logičan način poboljšava primenjujući male izmene, te se stoga lako objašnjavaju i implementiraju. Sa druge strane, velika prednost ovih algoritama je u tome što je koncept optimizacije koji koriste nezavistan od funkcije cilja koja se optimizuje. S obzirom na to, ovi algoritmi se uspešno mogu primeniti na mnoge probleme kombinatorne optimizacije, a ne samo na problem particionisanja hipergrafova.

2.1.1. Kernighan-Lin (KL) algoritam

Ovaj algoritam razvili su Keringhan i Lin 1970 godine, pogledati [4], radi rešavanja problema biparticionisanja grafa. On se može primeniti na ovde razmatrani problem k-particionisanja hipergrafa rekurzivnim biparticionisanjem kada je k neki stepen dvojke. Kod njega izmena trenutne particije Π kojom se dobija susedno dopustivo rešenje podrazumeva zamenu para čvorova $u, v \in V$ koji pripadaju različitim delovima trenutne particije, $\Pi(u) \neq \Pi(v)$, s tim da je tako dobijena particija dopustiva.

Ovaj algoritam sastoji se od obilazaka čvorova u toku kojih se vrše izmene trenutne particije, sa kojima prekidamo kada više ne možemo dobiti susednu dopustivu particiju koja će poboljšati trenutnu vrednost funkcije cilja. Na početku svakog od obilazka svi čvorovi su slobodni da se pomere, tj. svi su otključani. Obilazak se zatim izvodi sve dok

svi čvorovi ne postanu zaključani. U toku obilaska, u svakom koraku naparavi se ona zamena čvorova koja vodi ka najboljoj mogućoj dopustivoj particiji, u smislu da najviše poboljšava vrednost funkcije cilja. Kada se zamena čvorova obavi, zapamti se doprinos funkciji cilja kao razlika njene pređašnje i dobijene vrednosti, i čvorovi koji su učestvovali u zameni se zaključaju tako da se ne mogu ponovo pomerati u narednim koracima istog prolaza. Kada se više ne može napraviti zamena otključanih čvorova tako da dobijemo dopustivu particiju, prolaz se završava i tada ispitujemo sve doprinose dobijene zamenama čvorova tokom tog prolaza. Uzme se parcijalna suma onih doprinosa koji najviše poboljšavaju funkciju cilja, što će odgovarati zamenama čvorova koji vode do najbolje moguće particije dobijene u toku prolaza. Ukoliko je ta najveća parcijalna suma pozitivna, zamene čvorova koje ne učestvuju u toj sumi se odrade unazad i potom se dobijena particija uzme za startnu u sledećem prolazu. U suprotnom, ako je ta suma negativna, to znači da u toku prolaza nismo dobili susednu dopustivu particiju koja će poboljšati funkciju cilja, te stoga sve zamene izvršene u poslednjem prolazu se odrade unazad i KL algoritam se završava.

Uočimo to da u toku prolaza, u svakom koraku se zamenjuju čvorovi koji daju najveći doprinos funkciji cilja, bez obzira da li je taj doprinos pozitivan ili ne. Na kraju je moguće dobiti da parcijalna suma sa najvišim pozitivnim doprinosom uključuje i one zamene čvorova koje dovode do povećavanja funkcije cilja. Dakle, dopuštaju se i nepovoljne zamene ako će one na kraju, zajedno sa narednim zamenama u datom prolazu, rezultirati najvećim doprinosom. Ovo nam govori da KL algoritam može da se izvuče iz lokalnog minimuma tokom jednog prolaza.

KL algoritam po prolazu treba $O(n^3)$ vremena, s obzirom da se za izračunavanje najvišeg doprinosa vrši $O(n^2)$ poređenja. Složenost ovog algoritma po prolazu je u [8] svedena na $O(\max\{m \log n, d_{max} m\})$, gde je sa d_{max} označen maksimalan stepen čvora u grafu.

Sledi pseudo-kod KL algoritma:

Uzmi početnu dopustivu particiju Π;

repeat

$i := 0$;

$S_i := 0$;

otključaj sve čvorove $v \in V$;

while postoji dopustiva zamena otključanih čvorova

napravi najbolju dopustivu zamenu;

zaključaj čvorove koji su učestvovali u zameni;

zapamti doprinos g_i dobijen zamenom čvorova;

$S_{i+1} := S_i + g_i$;

$i := i + 1$;

end while;

izračunaj najbolju parcijalnu sumu doprinosa S_j;

if $S_j < 0$ **then**

sve zamene odradi unazad;

else

sve zamene od j-tog do i-tog koraka odradi unazad;

end if;

until $S_j < 0$.

2.1.2. Fidduccia-Mattheyses (FM) algoritam

Fidduccia i Mattheyses su objavili ovaj algoritam u [5] 1982 godine. On smanjuje složenost biparticionisanja hipergrafa na $O(N_H)$, gde je N_H broj ne-nula u matrici kojom je predstavljen hipergraf. Za razliku od KL algoritma, u FM algoritmu izmena trenutne particije podrazumeva pomeranje samo jednog čvora. Inače, FM algoritam isto tako izvršava prolaske kroz čvorove tako da se u okviru jednog prolaska svaki čvor pomeri tačno jednom, a zatim vraća rešenje dobijeno tokom prolaza koje je najviše optimizovalo funkciju cilja. I ovaj algoritam završava sa radom kada se u okviru jednog prolaska ne uspe popraviti vrednost funkcije cilja.

Razlog zbog koga je postignuta veća brzina ovog algoritma jeste korišćenje specijalne strukture podataka za čuvanje doprinosa funkciji cilja pomeranjem svakog od čvorova, koju nazivamo *gain bucket* struktura, i to dve ovakve strukture za kretanje u svakom od mogućih smerova. Ona omogućava izbor čvora čije pomeranje rezultira najvećim doprinosom u konstantnom vremenu, i takođe brzo apdejtovanje vrednosti doprinosa nakon svakog pomeranja.

Na početku svakog prolaska kroz čvorove izračunavaju se potencijalni doprinosi za svakog od n čvorova u $O(N_H)$ vremena čime se inicijalizuje *gain bucket* struktura. Ako hipergrana nije prelomljena, tada se doprinosi svih njenih čvorova smanjuje za jedan. A ukoliko je u jednoj od particija samo jedan čvor neke hipergrane, doprinos tog čvora se povećava za jedan. Sledi pseudo-kod dela algoritma kojim se izračunavaju doprinosi eventualnim pomeranjem čvorova. Ovde ćemo broj čvorova koje grana e ima u delu particije P označavati sa $P(e)$.

Zahtevi: dopustiva particija Π hipergrafa $H(V, E)$;

svaki od čvorova $v \in V$ je otključan (slobodan da se pomeri);

for all $v \in V$ **do**

$g(v) := 0$ (inicijalizacija doprinosa);

$P := \Pi(v)$;

for all $e \in E$ takve da $v \in e$ **do**

if $P(e) = 1$ **then**
g(v):=g(v)+1;
end if;
if $\bar{P}(e) = 0$ **then**
g(v):=g(v)-1;
end if;

end for;

end for.

Zatim se bira čvor sa najvećim doprinosom i on se izmesti u drugi deo particije, a vrednosti doprinosa za sve čvorove koji pripadaju istim granama kao i on se apdejtuju u konstantnom vremenu. Sledi pseudo-kod dela algoritma kojim se apdejtuju vrednosti doprinosa nakon izvršenog pomeranja čvora.

Zahtevi: $v \in V$ i v je otključan;
$P = \Pi(v)$;

$\Pi(v) := \bar{P}$;

zaključaj v ;

for all $v \in V$ **do**
$P(e) := P(e) - 1$;
$\bar{P}(e) := \bar{P}(e) + 1$;

if $P(e) = 0$ **then**

for all $v' \in e$ **do**
if (v' je otključan i $\Pi(v') = \bar{P}$ **then**
$g(v') := g(v') - 1$
end if
end for;

else if $\bar{P}(e) = 1$ **then**

for all $v' \in e$ **do**
if v' je otključan i $\Pi(v') = P$ **then**
$g(v') := g(v') + 1$
end if
end for;

end if;

end for.

2.1.3. Simulated annealing (SA) algoritam

Simulated annealing algoritam je stohastički algoritam koji se prvi put primenio za rešavanje problema iz optimizacije u [14]. Ovaj algoritam je baziran na pristupu koji je analogan pristupu tretiranja mikroskopskih procesa preuređivanja iz statističke mehanike.

Za razliku od prethodno navedenih algoritama particionisanja hipergrafova, SA algoritam pokušava da izbegne zaustavljanje u lokalnom minimumu povremeno prihvatajući i rešenja koja ne poboljšavaju funkciju cilja. Polazeći od početno uzete dopustive particije, slučajno se uzima susedno dopustivo rešenje. Izračuna se time dobijena promena funkcije cilja δf_c i ako je $\delta f_c \leq 0$, tj. ako smo poboljšali vrednost funkcije cilja, razmatrano susedno dopustivo rešenje se uzima za početno rešenje u sledećoj iteraciji. A ako je $\delta f_c \geq 0$, razmatranu particiju prihvatamo sa određenom verovatnoćom koja se u svakoj sledećoj iteraciji smanjuje za neki određeni korak.

Verovatnoća da će loša particija biti prihvaćena je data sa $e^{-\delta f_c/T}$, gde je T parametar temperature koji kontroliše promenu ove verovatnoće po određenom rasporedu. Nju smanjujemo posle svakih nekoliko iterativnih koraka za neki koeficient r i kako se T približava nuli, SA algoritam će prihvatati samo particije koje poboljšavaju trenutnu vrednost funkcije cilja.

U [3] pokazano je da će SA algoritam konvergiati ka globalnom rešenju ako i samo ako parametar T smanjujemo dovoljno polako i inicijalno uzeto dovoljno veliko.

Sledi pseudo-kod simulated annealing algoritma.

Uzmi za početno rešenje neku slučajnu particiju Π;
Postavi početnu temperaturu $T > 0$;
while $(T > T_{min})$
Neka je za Π' slučajno uzeto susedno rešenje rešenja Π;
$\delta f_c = f_c(\Pi') - f_c(\Pi)$
if $(\delta f_c \geq 0)$
$\Pi = \Pi'$;
if $(\delta f_c < 0)$
$\Pi = \Pi'$ sa verovatnoćom $e^{-\delta f_c/T}$;
$T = rT$ (smanji temperaturu);
end while.

2.2. Algoritmi bazirani na multilevel pristupu

U nazivu ove grupe algoritama stoji *multilevel* što bi na engleskom značilo više nivoa, a smisao ovog naziva postaće jasan u nastavku rada. Drugi naziv za ove algoritme bio bi algoritmi zasnovani na klasterizaciji. Prvi algoritmi razvijeni na ovom pristupu pojavili su se 1997 godine i do sad su se pokazali najefikasniji za rešavanje problema particionisanja hipergrafova preko 200 čvorova.

Povećavanjem broja čvorova hipergrafa povećava se i broj lokalnih rešenja problema njegovog particionisanja. Stoga, smanjuje se verovatnoća da će heuristika konvergirati ka lokalnom minimumu koji je blizu globalnog rešenja. Jedan od načina da se dođe do lokalnog minimuma koji je blizu globalnom minimumu jeste da se poveća broj izvršavanja algoritma, što je prilično nepraktično.

U multilevel pristupu, početni hipergraf se sukcesivno aproksimira manjim hipergrafovima. Prostor dopustivih rešenja time se smanjuje, kao i broj lokalnih rešenja odgovarajućom heuristikom. Smanjivanjem početnog hipergrafa postaje lakše da se izračuna optimalna particija, koja, ako je aproksimacija dobro izvršena, bi trebala odgovarati optimalnoj particiji početnog hipergrafa.

Aproksimacija početnog hipergrafa može se napraviti klastrizacijom čvorova, tako da klasteri koje dobijemo čine čvorove novog hipergrafa. Klasterizacijom se očuvava vrednost funkcije cilja particionisanja, jer hipergrane koje se prelome particijom manjeg hipergrafa će ostati prelomljene i nakon projekcije dobijene particije na veći hipergraf.

Multilevel pristup ima tri glavne faze: klasterizacija, inicijalno particionisanje i anklasterizacija. U toku klasterizacije početni hipergraf se aproksimira u više nivoa, stoga i naziv multilevel, čime dobijamo sve manje i manje hipergrafove. Inicijalno particionisanje zatim podrazumeva particionisanje hipergrafa dobijenog klasterizacijom na najvišem nivou. Na kraju, tokom anklasterizacije, dobijena particija sa najvišeg nivoa se projektuje na originalni hipergraf opet prolazeći kroz različite nivoe aproksimacija. U nastavku dajemo detaljnije objašnjenu svaku od ovih faza.

2.2.1. Klasterizacija

Tokom ove faze algoritma originalan hipergraf $H(V, E)$ se aproksimira sukcesivnim nizom manjih hipergrafova $H_i(V_i, E_i)$, $1 \leq i \leq a$, tako da je $|V_i| < |V_j|$ za $i > j$ i tako da hipergraf H_a ima neki fiksiran broj čvorova, koji obično iznosi nekoliko stotina. Dakle, klasterizacija hipergrafa H_i bi se mogla zapisati kao preslikavanje $g_i : V_i \to V_{i+1}$ tako da je $|V_i|/|V_{i+1}| = r$, $r > 1$.

Poželjno bi bilo da algoritam za klasterizaciju obezbedi da se za $i > j$ particija dobijena na nivou i lako može projektovati na hipergraf sa nivoa j. Isto tako, poželjno je da vrednost funkcije cilja na nižem nivou j ne bude veća od vrednosti funkcije cilja na višem nivou i. Jedan od načina da se zadovolje ovi kriterijumi jeste grupisanje čvorova

datog hipergrafa u klastere koji se potom proglašavaju za čvorove u novodobijenom hipergrafu. Pri tome, suma težina čvorova koji se klasterizuju se pripisuje kao težina novog, klasterizacijom dobijenog čvora. Tokom klasterizacije hipergrane sa samo jednim čvorom se zanemaruju pošto one mogu biti smeštene u bilo koji deo particije a da se pri tome ne menja vrednost funkcije cilja. Sa druge strane, klasterizacijom cilj je je smanjiti broj velikih hipergrana sa kojima se heuristike teško nose.

Neki generalan pristup bi bio da se teži klasterizaciji čvorova koji su snažno povezani u hipergrafu. To snažno povezani bi se odnosilo na broj zajedničkih hipergrana kojima pripadaju. Kako bi mogli razmatrati stepen povezanosti čvorova u hipergrafu uvodi se graf povezanosti $G_i(V_i, E_i')$ za hipergraf $H_i(V_i, E_i)$, tako da grana $e \in E_i'$ postoji između čvorova $v, u \in V_i$ ako i samo ako postoji hipergrana $h \in E_i$ tako da $v \in h$ i $u \in h$. Težine granama u G_i se dodeljuju tako da oslikavaju stepen povezanosti čvorova u hipergrafu H_i. Zatim kad algoritam obilazi sve čvorove, one koje još nisu klasterizovani, spaja sa susednim čvorom tako da se maksimizuje neka metrika koja oslikava povezanost među čvorovima.

U najranijim pristupima klasterizaciji čvorova, klasterizovanje je obavljano grupisanjem povezanih čvorova na potpuno slučajnoj bazi, tako da se za svaki čvor bira neki od susednih za spajanje u jedan klaster. Pri tome se vodi računa da se ne spajaju klasteri velike težine. Složeniji pristup je da se uzima u obzir veličina hipergrana i težina čvorova prilikom konstruisanja odgovarajuće metrike u G_i. Ovde se čvor spaja sa onim susednim čvorovima sa kojim ima najveću poveznost u smislu da su povezani granom najveće težine. Kako je pogodno skloniti hipergrane manje kardinalnosti s obzirom da ne moraju biti prelomljene, veze među čvorovima manjih hipergrana se rangiraju više. Uobičajno je da se za granu $e' \in E_i'$ između čvorova u i v u grafu povezanosti $G_i(V_i, E_i')$ uzima da je

$$\omega(e') = \sum \frac{1}{|e| - 1},$$

gde je suma izvršena po svim hipergranama e koje su zajedničke za čvorove u i v u H_i. Time se postiže i to da ako čvorovi pripadaju većem broju zajedničkih hipergrana, grana koja ih povezuje u G_i imaće i veću težinu.

Pri klasterizaciji čvorova može se koristiti i sofisticiranija metrika. Tako se za granu $e' \in E_i'$ koja povezuje čvorove u i v u grafu G_i može postaviti težina

$$\omega(e') = \frac{1}{\omega(v)\omega(u)} \sum \frac{1}{|e|},$$

gde se suma uzima po svim hipergranama e koje su zajedničke za čvorove u i v u H_i. Ovim se postiže izbegavanje grupisanja prevelikih čvorova u jedan klaster.

Spomenućemo još jednu varijantu za klasterizaciju hipergrafa implemetiranu u paketu Mondriaan, sa kojim ćemo uporediti naše rezultate u sledećem poglavlju. Ovde su tokom klasterizacije spajani čvorovi slične poveznosti u smislu da pripadaju što većem broju istih hipergrana. Ako hipergraf predstavimo matricom datom u Definiciji 2, gde su kolone te matrice hipergrane tog hipergrafa, onda vrste koje na istim mestima imaju jedinice impliciraju da čvorovi koji odgovaraju tim vrstama pripadaju istim hipergranama. Stoga,

velika vrednost unutrašnjeg proizvoda vrsta ukazuje na to da odgovarajući čvorovi pripadaju velikom broju istih hipergrana. U Mondoriaan paketu upravo je velika vrednost ovog unutrašnjeg proizvoda korišćena kao kriterijum za spajanje odgovarajućih čvorova.

Na višim nivoima klasterizacije moguće su veće varijacije u težinama novodobijenih čvorova, čime se samnjuje broj dopustivih particija klastrizovanog hipergrafa. Parametar koji ograničava maksimalnu težinu klastra bi stoga trebao da igra značajnu ulogu na efikasnost algoritma klasterizacije. Takođe i parametar r koji određuje brzinu smanjivanja hipergrafova bi trebao da ima uticaj na efikasnost algoritma i kvalitet konačnog rešenja. Sa manjom vrednošću ovog parametra dobijamo više nivoa klasterizacije zbog čega je očekivati veći kvalitet rešenja, ali time se povećava vremenska i prostorna složenost algoritma.

Postavlja se pitanje kako odrediti kada prestati sa klasterizacijom. Ukoliko se prekine vrlo brzo, proces pronalaženja particije hipergrafa H_a dobijenog na najvišem nivou klasterizacije može biti suviše težak. Sa druge strane, ukoliko dobijemo suviše mali hipergraf, prostor dopustivih rešenja može jako da se smanji tako da optimalno rešenje za hipergraf H_a bude daleko od optimalnog rešenja početnog hipergrafa H. U nekim radovima predlagano je da se sa klasterizacijom završi kada se dobije hipergraf sa ne više od $30k$ čvorova, gde je k broj delova particije. Takođe, klasterizaciju je pogodno završiti kada više nismo u mogućnosti dovoljno smanjiti veličinu hipergrafa H_i.

2.2.2. Inicijalno particionisanje

U ovoj fazi algoritama zasnovanih na multilevel pristupu cilj je izračunati particiju najmanjeg hipergrafa $H_a(V_a, E_a)$ dobijenog na poslednjem nivou klasterizacije, ali tako da ta particija zadovoljava uslove balansiranja, tj. da bude dopustiva i da optimizuje funkciju cilja. Dobijena inicijalna particija će se zatim u fazi anklasterizacije projektovati sa najmanjeg hipergrafa H_a na sve veće hipergrafove, sve dok ne stignemo do početnog hipergrafa H.

Inicijalno particionisanje može biti dobijeno bilo kojom od heuristika navedenih u prethodnom poglavlju. Ali pre nego što se primeni neka od heuristika, potrebno je generisati startnu dopustivu particiju koju ćemo zatim poboljšati izabranom heuristikom, poboljšati u smislu optimizacije funkcije cilja.

Postoji puno različitih načina da se odredi startna dopustiva particija. Jedna od ideja je da se vrši bisekcija slučajno sortiranih čvorova. Takođe, moguće je izabrati čvorove na slučajan način pa oko njih obrazovati delove particije. Jedan od metoda za nalaženje startne dopustive particije jeste REM (Randomized Engineering Method). On najpre sortira čvorove opadajuće po težini, a zatim ih pripisuje delovima particije na slučajan način. Svakom od čvorova se dodeljuje verovatnoća pripisivanja delu particije koja je proporcionalna razlici između maksimalnog kapaciteta dela particije i njene trenutne težine uvećane za težinu posmatranog čvora. Ovo održava delove particije približno jednakim po težini, a da se pri tom očuvava faktor slučajnosti.

2.2.3. Anklasterizacija

Kada smo izračunali inacijalnu particiju nad hipergrafom H_a, tokom anklasterizacije tu particiju projektujemo unazad sve dok ne dobijemo particiju Π nad početnim hipergrafom H:

$$H_a^{\Pi_a} \to H_{a-1}^{\Pi_{a-1}}, \ldots, H_{i+1}^{\Pi_{i+1}} \to H_i^{\Pi_i}, \ldots, H_1^{\Pi_1} \to H^{\Pi}.$$

Projekcija particije na hipergraf $H_i(V_i, E_i)$ se vrši tako što se za svaki čvor $v \in V_i$ odredi kojem klasteru pripada u hipergrafu H_{i+1}, dakle identifikuje se klaster $g_i(v) \in V_{i+1}$. Čvor v će pripadati delu particije Π_i koji odgovara onom delu particije Π_{i+1} gde je uočen klaster $g_i(v)$.

Kada smo projektovali particiju Π_{i+1} na hipergraf sa većim brojem čvorova H_i, vrlo je verovatno da postoji mogućnost daljeg poboljšanja dobijene particije Π_i u smislu optimizacije funkcije cilja. Ovo se može izvesti bilo kojim od heuristika navedenih u prethodnom poglavlju. Dakle, sprovodimo sličan princip kao kod inicijalnog particionisanja koje smo imali na najvišem nivou. Kada projekcijom dobijemo particiju Π_i nju uzimamo za startnu particiju, koja je u svakom slučaju mnogo kvalitetnija od neke proizvoljne slučajno dobijene particije, pa je nekom od heuristika još poboljšavamo. Osim toga što krećemo sa kvalitetnom particijom kao početnom, sada kada pomeramo čvorove primenom heuristike nad H_i ustvari pomeramo klastere koji su sačinjeni od jako povezanih čvorova. Time je poboljšanje funkcije cilja još veće nego što bi dobili pomeranjem pojedinačnih čvorova.

Na kraju ovog poglavlja o multilevel algoritmima, daćemo pseudo-kod koji je korišćen u softverskim paketima za particionisanje hipergrafova MLPart i hMETIS [6].

```
    nivo = 0;
    KlasterizovanGraf[nivo] = Hipergraf;
// klasterizacija
    while (KlasterizovanGraf[nivo].BrojCvorova() > 200)
        SledeciGraf = Klastrizuj (KlasterizovanGraf[nivo]);
        nivo = nivo + 1;
        KlasterizovanGraf[nivo] = SledeciGraf;
// inicijalno particionisanje
    Particija[nivo] = slučajna incijalna particija;
    FM (KlasterizovanGraf[nivo], Particija[nivo]);
// anklasterizacija
```

```
    while (nivo > 0)
        nivo = nivo - 1;
        Particija[nivo] = Projektuj (Particija[nivo+1], KlasterizovanGraf[nivo]);
        FM (KlasterizovanGraf[nivo], Particija[nivo]);
    return Particija[0];
```

2.3. Rešavanje celobrojnim linearnim programiranjem

U narednom poglavlju gde dajemo predlog novog algoritma, tretiraćemo CGP problem tj. problem particionisanja orjentisanog grafa uz minimizaciju čvorova pozivanih iz drugih modula, takozvanih interfejsa, za koji ćemo pokazati da je ekvivalentan problemu particionisanja hipergrafova. Rezultate koje ćemo tamo izložiti uporedićemo sa rezultatima dobijenim rešavanjem pomoću celobrojnog linearnog programiranja koje nam daje optimalno rešenje. Zato ovde dajemo kratko izlaganje o tome kako se problem particionisanja orjentisanih grafova uz minimizaciju interfejsa interpretira kao problem celobrojnog linearnog programiranja sa 0-1 varijablama koji potom može biti rešen nekim od postojaćih softverskih paketa za rešavanje ovakvih problema.

Dakle, posmatramo orjentisani graf $D(V, E)$ gde je skupom čvorova V zapravo predstavljen skup programa nekog informacionog sistema, a pozivi među njima predstavljeni su skupom grana $E = \{(u, v) \in V \times V | u \text{ poziva } v\}$. Ako uzmemo podskup skupa čvorova $U \subseteq V$, skup interfejsa tog podskupa smatraćemo skup svih čvorova iz U koji su pozivani od strane čvorova koji nisu iz U. Skup interfejsa skupa U možemo zapisati na sledeći način:

$$I_D(U) = \{u \in U | (u, v) \in E \text{ za neko } v \in V \setminus U\}.$$

Kao što je uobičajeno, particijom orjentisanog grafa smatramo kolekciju podskupova $\{V_1, ..., V_L\}$ skupa čvorova V tako da je presek svaka dva takva podskupa prazan skup, $V_i \cap V_j = \emptyset$ za $i \neq j$, i tako da je unija svih upravo početni skup čvorova V. Sada CGP problem možemo matematički formulisati sledećom definicijom.

Definicija 10. *Neka je dat orjentisani graf $D(V, E)$. Za dati broj modula $L \in \mathbb{N}$ i kapacitet svakog od njih $K \in \mathbb{N}$, naći particiju $\{V_1, ..., V_L\}$ skupa čvorova V tako da je $|V_l| \leq K$ za svako l i tako da je funkcija cilja $\sum_{l=1}^{L} |I_D(V_l)|$ minimizovana.*

Formulišimo zatim opšti slučaj problema binarnog linearnog celobrojnog programiranja, tj. problema celobrojnog linearnog programiranja kod koga sve promenljive uzimaju vrednosti 0 ili 1.

Definicija 11. *Neka je data matrica $A \in \mathbb{Z}^{m,n}$ i vektori $b \in \mathbb{Z}^m, c \in \mathbb{Z}^n$. Naći vektor $x \in \mathbb{Z}^n$ tako da je minimizovana funkcija cilja $c^T x$ i tako da su zadovoljena ograničenja $Ax \leq b$ i $x \in \{0, 1\}^n$.*

Sada možemo da formulišemo problem particionisanja orjentisanog grafa uz minimizaciju interfejsa kao problem binarnog linearnog celobrojnog programiranaj na sledeći

način:

$$\begin{aligned}
\min & \sum_{l=1}^{L} \sum_{v \in V} x_{vl}, \\
\text{uz ograničenja: } & \sum_{l=1}^{L} y_{vl} = 1, \text{ za sve } v \in V, \\
& \sum_{v \in V} y_{vl} \leq K, \text{za } l = 1, \ldots, L, \\
& x_{vl} \leq y_{vl}, \text{ za } l = 1, \ldots, L, \text{ i za sve } v \in V, \\
& y_{vl} \leq y_{ul} + x_{vl}, \text{ za } l = 1, \ldots, L, \text{ i za sve } (u, v) \in E, \\
& x_{vl}, y_{vl} \in \{0, 1\}, \text{ za } l = 1, \ldots, L, \text{ i za sve } v \in V,
\end{aligned}$$

gde promenljive y_{vl} i x_{vl} interpretiramo kao:

$$y_{vl} = \begin{cases} 1 & , \text{ ako } v \in V_l, \\ 0 & , \text{ inače,} \end{cases} \qquad x_{vl} = \begin{cases} 1 & , \text{ ako je } v \text{ interfejs od } V_l, \\ 0 & , \text{ inače.} \end{cases}$$

Prvo navedeno ograničenje nam kaže da svaki od čvorova v može se naći samo u jednom od modula V_l. Drugo ograničenje se odnosi na kapacitet svakog od modula. Treće ograničenje tumačimo tako da ako je čvor v interfejs modula V_l onda se on nalazi u tom modulu, a četvrto tako da za svako $v \in V_l$ postoje dve mogućnosti koje isključuju jedna drugu: v je ili interfejs od V_l ili su svi čvorovi u koji pozivaju v u istom modulu V_l.

Kada dobijemo rešenje ovako formulisanog problema, uzimanjem za

$$V_l = \{v \in V | y_{vl} = 1\}, \quad l = 1, \ldots, L$$

$$I = \{v \in V | x_{vl} = 1 \text{ za neko } l\}$$

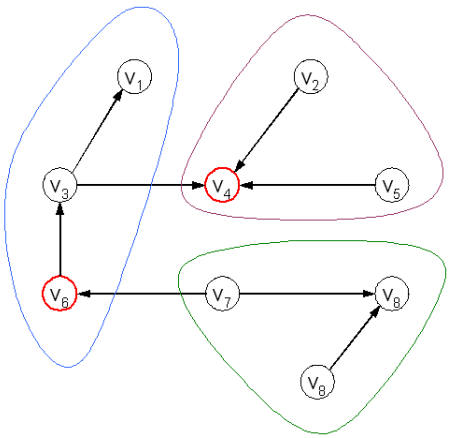
dobijamo rešenje $\{V_1, \ldots V_L\}$ problema particionisanja orjentisanog grafa sa minimizovanim skupom interfejsa I.

3. Novi algoritam za particionisanje hipergrafova

U ovom poglavlju dajemo novi algoritam koji rešava problem particionisanja hipergrafova do optimuma. Kako se naš algoritam izvršava nad orjentisanim grafom, ovde najpre pokazujemo ekvivalentnost problema particionisanja hipergrafa (HP) i problema particionisanja orjentisanog grafa uz minimizaciju takozvanih interfejsa (CGP problem).

HP problem $\Leftrightarrow$ CGP problem

Posmatrajmo najpre problem CGP. Dakle, bavimo se problemom particionisanja orjentisanog grafa, tj. digrafa, $D(V, E)$ sa skupom čvorova V i skupom grana E koji sadrži uređene dvojke iz skupa čvorova. Svaki od čvorova v može imati dodeljenu mu težinu $\omega(v)$. Particionišemo skup čvorova V u dati broj podskupova, tj. modula, tako da je ukupna težina svakog modula ograničena. Pri tome znamo da je težina modula zbir težina čvorova koji ga sačinjavaju ili samo broj takvih čvorova ako ne uvodimo funkciju težine ω. Nakon particionisanja želimo da je što manje čvorova koji imaju dolaznu granu iz modula različitog od onog kom pripadaju. Takve čvorove koji nakon podele u module imaju dolaznu granu iz nekog drugog modula nazivamo intefejsima, prema tome mi prilikom particionisanja orjentisanog grafa želimo da minimizujemo broj interfejsa. Na slici (6) dat je primer orjentisanog grafa sa obeleženim čvorovima koji su nakon particionisanja postali interfejsi. Vidimo da je particionisanje izvršeno na tri dela i da smo nakon toga dobili dva interfejsa (obeleženi crveno).

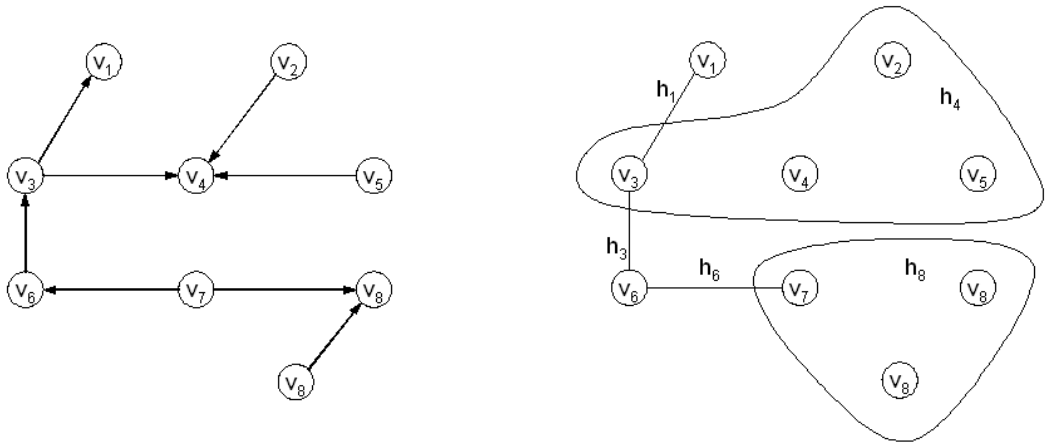


Slika 6: Primer particionisanog orjentisanog grafa sa prikazanim interfejsima

Sada dajemo transformaciju orjentisanog grafa $D(V, E)$ u hipergraf $H(V, E')$ i pokazujemo da se problem CGP svodi na problem HP. Za svaki čvor $v_i \in V$ formiramo skup

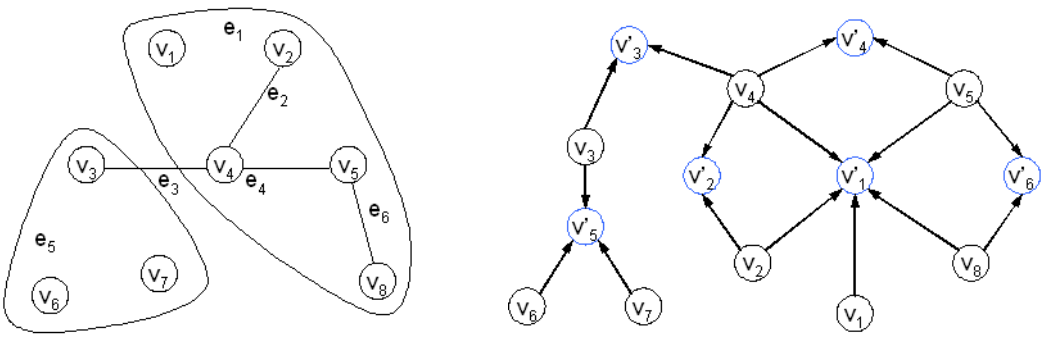
$$h_i = \{v_i\} \cup \{v_j | (v_j, v_i) \in E\},$$

dakle h_i sadrži čvor v_i i sve čvorove v_j koji ga pozivaju. Ove skupove uzimamo za hipergrane u hipergrafu, dakle za svaki $v_i \in V$ uzimamo da $h_i \in E'$. Pa važi sledeće: hipergrana h_i je prelomljena nakon particionisanja hipergrafa, tj. odgovarajući skup h_i nije postavljen ceo u jedan modul, ako i samo ako v_i je interfejs. Dakle, vidimo da je problem particionisanja orjentisanog grafa uz minimizaciju broja interfejsa postao problem particionisanja hipergrafa uz minimizaciju broja prelomljenih hipergrana. Transformacija orjentisanog grafa u hipergraf je data na slici (7).



Slika 7: Transformacija orjentisanog grafa (levo) u hipergraf (desno)

Sa druge strane, svaki hipergraf $H(V, E)$ možemo transformisati u orjentisani graf $D(V', E')$ tako da se problem HP svede na problem CGP. Za svaku hipergranu $h_i \in E$ dodamo čvor v'_i i pridružimo mu težinu nula $\omega(v'_i) = 0$. Zatim sve čvorove hipergrane h_i usmerimo prema čvoru v'_i i izaberemo jedan od njih, recimo $v_i \in h_i$, za koji možemo da zamislimo da se uvek nakon particionisanja grafa D nađe u istom modulu kao i v'_i. Tada možemo da tvrdimo sledeće: v'_i je interfejs ako i samo ako neki od čvorova grane h_i nije u istom modulu kao i čvor v'_i (kao i v_i), što znači ako i samo ako hipergrana h_i je prelomljena. Na slici (8) dajemo kako izgleda opisana transformacija hipergrafa u orjentisani graf.



Slika 8: Transformacija hipergrafa (levo) u orjentisani graf (desno)

Zaključujemo da su problemi HP i CGP ekvivalentni, tako da algoritme koji rešavaju problem particionisanja orjentisanog grafa uz minimizaciju interfejsa nakon odgovarajuće transformacije hipergrafa možemo koristiti za rešavanje problema particionisanja hipergrafa i obratno.

Nama će ulaz biti orjentisan graf $D(V, E)$ u kome svaki čvor ima pridruženu mu težinu $v \mapsto \omega(v)$. Kapacitet svakog od modula možemo odmah da odredimo na osnovu ukupne težine grafa, datog broja modula i nekog stepena slobode prekoračenja ν koji je obično u praksi između $1,2$ i $1,4$. Neka je $\omega(D) = \sum_{v \in V} \omega(v)$ ukupna težina grafa, kapacitet svakog od modula izračunavaćemo prema sledećoj formuli:

$$K = \lfloor \frac{\nu \omega(D)}{L} \rfloor.$$

Ukoliko posle podele grafa postoji grana (u, v) takva da su čvorovi u i v u različitim modulima, čvor v mora ostati dostupan javnosti pa ga proglašavamo interfejsom. Dakle, nama će osnovni kriterijum prilikom podele biti minimizacija broja interfejsa. Postojanje takvih čvorova je gotovo nemoguće izbeći, s obzirom da je za to dovoljno da je graf povezan i da je $K < \omega(D)$.

Osnovna ideja u nastavku izloženog algoritma bila je ta da se početni graf smanjuje u smislu uklanjanja pojedinih čvorova iz razmatranja prilikom proglašavanja interfejsa. To možemo da radimo jer na osnovu pogodnih kriterijuma smo u stanju da zaključimo koji čvorovi neće biti interfejsi. Tako korišćenjem pogodnih kriterijuma za ostavljanje pojedinih čvorova sa strane i skupljanje jednih čvorova u druge, pretraživanje grafa radi pronalaženja interfejsa se ubrzava. Dakle, tokom pretrage mi zapravo pokušavamo da klasterizujemo graf dobijajući sve manji i manji graf sa čvorovima koji su u suštini klasteri čvorova početnog grafa.

U nastavku ćemo najpre izložiti kriterijume na osnovu kojih početni graf u startu pokušavamo da smanjimo, a to su upravo i kriterijumi koje nameravamo dalje da koristimo tokom pretraživanja interfejsa.

3.1. Klasterizacija

Polazimo od ulaznog orjentisanog grafa $D(V, A)$ sa zadatim težinama $\omega : V \to \mathbb{N}$. Pretpostavljamo da je graf D bez multi-grana, dakle kod svaka dva čvora u i v mogu postojati grane (u, v) i (v, u) ali ne i višestruke grane u istom smeru. Takođe, pretpostavljamo da nema grana tipa (v, v). Kako nam je unapred zadat broj modula L i stepen slobode prekoračenja ν, odmah izračunavamo i kapacitet svakog od modula K. Sada redom izlažemo kriterijume na osnovu kojih možemo da smanjimo ovaj ulazni graf.

Proglašavanje neminovnih interfejsa

Najpre prolazimo kroz sve čvorove i ukoliko naiđemo na neki čvor v čija težina zajedno sa težinama njegovih dolaznih čvorova premašuje kapacitet modula K, taj čvor proglašavamo interfejsom tj. ukupan broj interfejsa povećavamo za jedan. Dakle, u ovakvoj situaciji sigurni samo da će nakon podele čvor v biti pozivan iz nekog spoljašnjeg modula. Njegove dolazne grane zatim brišemo jer one nemaju više nikakvu funkciju. Time dobijamo graf koji je ređi nego početni graf. U nastavku dajemo pseudo-kod kojim se proglašavaju sigurni interfejsi.

for all $v \in V$ **do**

$t :=$ težina(v) + težina(svi dolazni čorovi čvora v)

if $(t > K)$

ZapamtiInterfejs(v);

BrojInterfejsa++;

Obriši dolazne grane čvora v;

end if;

end for.

Uklanjanje komponenti

U ovoj fazi određujemo komponente povezanosti grafa i ispitujemo da li postoji komponenta dovoljno mala da se interfejsi u njoj mogu izbeći. Napomenimo da svaki put kada obrišemo neke grane iz grafa treba primeniti ovaj postupak, jer se možda nove komponente stvaraju. Sa ispitivanjem krećemo od komponente najmanje težine.

Neka je $D'(V', E')$ komponenta povezanosti grafa D težine $\omega(D') = \sum_{v \in V'} \omega(v)$. Ako ostatak grafa $D \setminus D'$ proizvoljno podelimo u L modula svaki od njih će imati težinu najviše $\lfloor \frac{\omega(D \setminus D')}{L} \rfloor$ što je manje od K. Pa ako težina posmatrane komponente zadovoljava uslov da je

$$\omega(D') \leq K - \lfloor \frac{\omega(D \setminus D')}{L} \rfloor$$

to znači da celu komponentu D' možemo izmestiti u neki od modula prilikom deljenja ostatka grafa, a da pri tome ne pravimo dodatne interfejse. Dakle, ukoliko je ovaj uslov zadovoljen, možemo izbeći interfejse u posmatranoj komponenti D' i nju ostavljamo sa strane. Graf D se ovim umanjuje za komponentu D', pa postupak ponavljamo za sledeću komponentu sada grafa $D \setminus D'$. Sledi deo koda kojim smo ovo implementirali, a koji ćemo u nastavku rada podrazumevati da se izvršava pozivanjem procedure *UklanjanjeKomponenti()*.

Nađi komponente grafa D;

Sortiraj komponete grafa po težini: $[K_1, ..., K_N]$;

for (i = 0; i < N; i++)

// izračunaj kapacitet prilikom podele grafa $D \setminus K_i$
$NovoK = \lfloor$ TežinaGrafa(D) − TežinaKomponente(K_i) / L $\rfloor$;
if (($K - NovoK$) ≥ TežinaKomponente(K_i))
Obriši K_i iz grafa;
end if;

end for.

Skupljanje čvorova

U ovoj fazi za svaki čvor grafa proveravamo da li u slučaju njegovog proglašenja interfejsom postoje čvorovi koji sigurno staju u modul zajedno sa njim izbegavajući nove interfejse. To nam dozvoljava da te čvorove skupimo u posmatrani čvor povećavajući njegovu težinu. Dakle, time dobijamo novi graf sa manjim brojem čvorova, ali iste težine. Takođe, istu proveru mogućnosti skupljanja vršimo i za čvorove koji ulaze u trenutno posmatrani čvor kod koga smo ulazne grane privremeno izbrisali.

Neka je v proizvoljan čvor grafa D. Privremeno obrišemo grane koje ulaze u njega i nađemo komponente povezanosti tako dobijenog grafa. Odredimo komponentu u kojoj se nalazi posmatrani čvor v, recimo da je to komponenta $D_v(V_v, E_v)$. Ako važi uslov da je

$$\omega(D_v) \leq K - \lfloor \frac{\omega(D \setminus D_v)}{L} \rfloor$$

to znači da komponentu D_v možemo izmestiti u modul sa slobodnim prostorom koji ostaje posle podele ostatka grafa na L modula. Ovim izmeštanjem postižemo da se čitava komponenta D_v nađe u istom modulu čime interfejs među njima može postati jedino čvor v. To nam dozvoljava da komponentu D_v skupimo u jedan čvor v sa težinom $\omega(D_v)$.

Neka je sada (u, v) jedna od privremeno obrisanih grana koje ulaze u posmatrani čvor v, ali takva da se čvor u nakon ovog privremenog brisanja našao u komponenti D_u koja je različita od komponente D_v. Ukoliko je ispunjen uslov

$$\omega(D_u) \leq K - \lfloor \frac{\omega(D \setminus D_u)}{L} \rfloor,$$

komponentu D_u možemo izmestiti u neki modul sa slobodnim prostorom. Time ne stvaramo nove interfejse, i dalje jedini eventualni interfejs može biti čvor v. Čvorove komponente D_u skupljamo u čvor u sa težinom $\omega(D_u)$ i vraćamo granu (u, v).

Na sledećoj strani dajemo pseudo-kod procedure kojom se izvršava gore objašnjenja klasterizacija. U nastavku rada ovaj postupak će se odrađivati pozivanjem procedure *SkupljanjeČvorova()*.

for all $v \in V$

// privremeno brisanje grana:
dolazni = UzmiDolazneČvorove(v);
Obriši grane iz grafa između v i *dolazni*;
Nađi komponente grafa $[K_1, ...K_N]$;
Vrati grane u grafa između v i *dolazni*;
K_v = NađiKomponentuGdeJe(v);
$NovoK = \lfloor$ TežinaGrafa(D) − TežinaKomponente(K_v) / $L \rfloor$;

if (TežinaKomponente(K_v) ≤ ($K - NovoK$) && $|K_v| > 1$)
// skupljanje komponente K_v:
Obriši iz grafa sve čvorove $K_v \setminus v$;
Težina(v) := TežinaKomponente(K_v);
Apdejtuj grane čvora v;
Apdejtuj *dolazni*;
end if;

while (NijePrazno(*dolazni*))
u = UzmiPrviČvor(*dolazni*);
IzbaciPrviČvor(*dolazni*);
K_u = NađiKomponentuGdeJe(u);
$NovoK = \lfloor$ TežinaGrafa(D) − TežinaKomponente(K_u) / $L \rfloor$;
if (TežinaKomponente(K_u) ≤ ($K - NovoK$) && $|K_u| > 1$)
// skupljanje komponente K_u:
Težina(u) := TežinaKomponente(K_u);
Obriši iz grafa sve čvorove $K_u \setminus u$;
Apdejtuj grane čvora u;
Apdejtuj grane čvora v;
Apdejtuj *dolazni*;
end if;

end while;

end for.

3.2. Početno rešenje

Ovde ćemo izložiti ideju koja nas je dovela do nekog prvog rešenja. Rešenje koje smo dobili ovim algoritmom nije optimalno, ali se za neke primere pokazalo da je prilično blizu optimalnog.

Prvo što smo uradili jeste proglašavanje interfejsima onih čvorova koji to moraju biti jer im je težina zajedno sa dolaznim čvorovima prevelika da bi se smestili u jedan modul. Zatim proveravamo da li je težina najveće komponente u grafu veća od kapaciteta K. Dokle god to jeste slučaj pokušavamo sa daljim proglašavanjem interfejsa da dođemo do toga da se i najveća komponeta može smestiti cela u jedan od modula. Svakim proglašenjem sledećeg interfejsa brišemo njegove dolazne grane i dobijamo novu situaciju.

Ovde je veliko pitanje kako izabrati odgovarajući čvor koji će biti naredni interfejs. Odgovarajući u smislu da je što manje takvih potrebno, dakle da što pre dobijemo komponentu koja je najveće težine, a koja cela može stati u jedan modul.

Prvo što možemo uraditi jeste da primenimo uslove klasterizacije koje smo opisali u prethodnom tekstu. Dakle, uklonićemo dovoljno male komponente i skupićemo čvorove koji ne moraju biti interfejsi. Time dalje u odabiru interfejsa izbacujemo iz opticaja čvorove za koje nema ni potrebe da budu interfejsi.

Kako smo dalje birali interfejse? Kriterijum nam je bio broj ulaznih grana u čvor, jer kada taj čvor proglasimo interfejsom mi brišemo njegove ulazne grane koje tada nemaju nikakvu funkciju. Uvek smo birali čvor sa najviše ulaznih grana, a onda očekujemo da brisanjem tih grana dolazimo do toga da se stvaraju nove komponente i to takve da će i najveća od njih moći da se smesti u modul kapaciteta K.

U nastavku dajemo pseudo-kod gore objašnjenog algoritma.

Odredi sigurne intrfejse;

Nađi komponente grafa;

T := NađiTežinuNajvećeKomponente();

while $(T > K)$

UklanjanjeKomponenti();
SkupljanjeČvorova();
v = NađiČvorSaNajvišeDolaznih();
ObrišiDolazneGraneČvora(v);
ZapamtiInterfejs(v);
BrojInterfejsa0++;
Nađi komponente grafa;

T := NađiTežinuNajvećeKomponente();

end while;

return BrojInterfejsa0;

3.3. Optimalno rešenje

U prethodnom poglavlju dali smo algoritam sa kojim smo dobili neko rešenje *BrojInterfejsa0*, koje nije optimalno. To rešenje možemo dalje iskoristiti kao neko početno u daljem traganju za optimalnim rešenjem koje podrazumeva manji broj interfejsa. Dakle, sad pokušavamo da nađemo rešenje koje je manje od *BrojInterfejsa0*.

Ovde smo najpre odradili klasterizaciju grafa, dakle proglasili interfejsima čvorove koji to moraju biti i eliminisali iz razmatranja čvorove koji to nisu. Zatim smo preostale čvorove u grafu sortirali prema kriterijumu broja ulaznih grana, koji smo koristili u nalaženju početnog rešenja u prethodnom poglavlju. Ovaj sortirani niz smo dalje pretraživali rekurzijom birajući interfejse u cilju smanjivanja početnog rešenja *BrojInterfejsa0*, sve dokle god je bilo moguće dobiti bolje rešenje od prethodnog.

U nastavku dajemo pseudo-kod rekurzije koja se poziva sa prvim čvorom u sortiranom nizu. U prvom pozivu stavljamo da nam je *NajRez* = *BrojInterfejsa0*. Ako nam ona vrati bolji rezultat, tj. *NajRez*−1, ponavljamo postupak sada sa novom vrednošću za *NajRez* koja je za jedan manja od prethodne. Ako ne dobijemo bolji rezultat, pokušavamo sa sledećim u sortiranom nizu. Ako dođemo do kraja sortiranog niza bez boljeg rezultata znači da je trenutna vrednost koju imamo u *NajRez* i najniža moguća, tj. to je optimalno rešenje.

int **Rekurzija** (int i)

// privremeno proglasi čvor $sortirani[i]$ interfejsom i

// privremeno obriši dolazne grane kod njega;

ZapamtiInterfejs($sortirani[i]$);

BrojInterfejsa++;

$dolazni$ = DolazniČvorovi($sortirani[i]$);

Obriši grane iz grafa između $sortirani[i]$ i $dolazni$;

if (BrojInterfejsa < NajRez−1)

for (j = i+1; j < $|V|$; j++)
PreNaj = NajRez;
NajRez = **Rekurzija**(j);

```
            if (PreNaj != NajRez)
               return NajRez;
            end if;
        end for;
        //smanji broj interfejsa i vrati privremeno obrisano
        BrojInterfejsa−−;
        Vrati grane u graf između sortirani[i] i dolazni;
    else // BrojInterfejsa = NajRez−1
        T := NađiTežinuNajvećeKomponente();
        if (T ≤ K)
            NajRez = BrojInterfejsa;
            return NajRez;
        else
            //smanji broj interfejsa i vrati privremeno obrisano
            BrojInterfejsa−−;
            Vrati grane u graf između sortirani[i] i dolazni;
            return NajRez;
        end if;
    end if;
end Rekurzija.
```

Dakle, uzmemo prvi iz sortiranog niza čvorova i privremeno ga proglasimo interfejsom. To znači da povećavamo broj interfejsa i privremeno mu brišemo ulazne grane. Ako je broj interfejsa i dalje manji od trenutno najboljeg rezultata za minimum 2, uzimamo sledeći iz sortiranog niza i njega privremeno proglašavamo interfejsom. Kada tako nakupimo $NajRez-2$ interfejsa, treba da izaberemo poslednji interfejs koji bi nam time dao bolji novi rezultat a to je $NajRez-1$. Kada izaberemo taj poslednji interfejs proveravamo da li se najveća komponenta u tako dobijenom grafu može smestiti u jedan modul. Ako može, onda smo dobili bolji rezultat, izlazimo iz ove gore ispisane rekurzije i ponovo sprovodimo ceo postupak ali sada sa manjim najboljim rezultatom koji smo zapamtili u globalnoj promenljivoj $NajRez$.

Ako posle biranja $(NajRez-1)$.-og interfejsa nismo uspeli da najveću komponentu smestimo u jedan modul, to znači da nemamo dobru kombinaciju interfejsa koja bi nam dala bolji rezultat. Zato, tom poslednje izabranom interfejsu vraćamo grane, smanjujemo broj interfejsa i izlazimo iz poslednje pozvane rekurzije. U for petlji smo i biramo sledeći iz sortiranog niza da nam bude poslednji interfejs. Ako se for petlja završi a uslov najveće komponente ne prođe ni za jednog u sortiranom nizu, to znači da treba promeniti $(NajRez-2)$.-i interfejs. Zato posle for petlje vraćamo grane kod $(NajRez-2)$.-og interfejsa

i smanjujemo broj interfejsa na *NajRez*−3. Posle toga se vraćamo u for petlju gde smo birali (*NajRez*−2).-i interfejs i uzimamo sledeći u sortiranom nizu pa proveravamo istu priču za njega.

Tako primenjujući ovu rekurziju na sortirani niz ispitujemo sve kombinacije interfejsa, kojih dozvoljavamo da bude uvek za jedan manje od prethodno najboljeg rezultata. Kako broj ovakvih kombinacija veoma brzo raste povećavanjem ulaznog grafa, ovaj algoritam možemo smatrati neefikasnim. Ali u ovako napisanom algoritmu postoji dosta prostora za dalju doradu i ubrzavanje petrage korišnjenjem postupaka klasterizacije opisanih ranije.

3.4. Numerički rezultati

Ovde ćemo izložiti rezultate koje smo dobili primenjujući prethodno opisane postupke na nekoliko test primera. Najpre smo testirali postupke klasterizacije, dakle redom smo proglašavali neminovne interfejse, uklanjali male komponente i skupljali čvorove na osnovu opisanih kriterijuma. Dobijeni rezultati prikazani su u tabeli (1). Test primeri su sortirani prema broju čvorova $|V|$, a sa $|E|$ je dat broj grana. Grafove smo particionisali na $L = 8$ modula, a za parametar tolerancije kapaciteta modula ν smo uzeli da je 1,2. U poslednjoj koloni dato je postignuto samnjenje početnog grafa u procentima. Vidimo da smo u dva slučaja dobili smanjenje i za preko 50%.

problem	**|V|**	**|E|**	**L**	ν	**smanjenje** (%)
graf 1	15	39	8	1.2	47
graf 2	449	659	8	1.2	66
graf 3	947	1900	8	1.2	42
graf 4	1100	2951	8	1.2	24
graf 5	1145	2686	8	1.2	20
graf 6	2142	2436	8	1.2	59

Tabela 1: Rezultati klasterizacije

Zatim smo na nekoliko test primera pustili algoritam za nalaženje početnog rešenja, dobijeni rezultati su dati u tabeli (2). Za primere za koje smo imali rezultat dobijen celobrojnim linearnim programiranjem i jednom od heuristika stavili smo u kolone optimalno rešenje, odnosno rešenje heuristikom. Rešenje heuristikom dobijeno je pomoću paketa Mondoriaan, koga smo spominjali u odeljku o multilevel algoritmima. U poslednjoj koloni je naše početno rešenje. Vidimo da smo u nekim slučajevima dobili rešenje bliže optimalnom nego ono što je dobijeno heuristikom.

Što se tiče našeg algoritma za dobijanje optimalnog rešenja, njega smo za sad uspešno testirali na malim grafovima, do 30 čvorova. Uz dugo vreme izvršavanja testirali smo ga i na test primeru 2 koji ima 449 čvorova, gde smo već početno rešenje dobili blizu optimalnog. Dalja testiranja ćemo obaviti nakon dorade algoritma ostavljene za neki budući rad. Doradom algoritma očekujemo znatno ubrzanje izvršavanja.

problem	**|V|**	**|E|**	**L**	ν	**optimalno rešenje**	**rešenje heuristikom**	**početno rešenje**
graf 1	15	39	8	1.2	11	11	11
graf 2	449	659	8	1.2	6	10	7
graf 3	947	1900	8	1.2	13	17	16
graf 4	1100	2951	8	1.2	32	52	57
graf 5	1145	2686	8	1.2	51	69	94
graf 6	2142	2436	8	1.2	-	-	90

Tabela 2: Rezultati prvog rešavanja

4. Zaključak

U ovom radu smo se bavili problemom particionisanja hipergrafa. Glavni deo rada bio je posvećen ideji novog algoritma za rešavanje problema particionisanja orjentisanog grafa uz minimizaciju broja interfejsa, problema za koji smo pokazali da je ekvivalentan problemu particionisanja hipergrafa.

Kriterijumi za početnu klasterizaciju grafa su testirani na primerima grafova koji imaju do 2000 čvorova. Na nekim od njih smo dobili smanjenje preko 50% broja čvorova ulaznog grafa. Doradom ovog algoritma očekujemo da će klasterizacija proći i za mnogo veće grafove.

Što se tiče nalaženja optimalnog rešenja, za sada smo uspeli našim algoritmom da dobijemo optimalno rešenje za grafove do 500 čvorova ali pod uslovom kada krenemo od početnog rešenja koje je blisko optimalnom. Ubacivanjem kriterijuma za smanjivanje grafa, koji su korišteni na početku, tokom pretraživanja optimalne kombinacije interfejsa rekurzijom, očekujemo ubrzanje ovog algoritma i njegovu uspešnost na većim grafovima.

Literatura

[1] A. Trifunović, *Parallel Algorithms for Hypergraph Partitioning*, Ph.D. Thesis , University of London, Imperial College of Science, Technology and Medicine Department of Computing, 2006

[2] A. Drozdek, *Data Structures and Algorithms in C++*, Brooks/Cole, 2000

[3] B. Hajek, *Cooling schedules for optimal annealing*, Mathematics of Operations Research, 13(2) : 311329, 1988

[4] B. W. Kernighan, S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal, 2(49): 291-307, 1970

[5] C. M. Fiduccia, R. M. Mattheyses, *A linear time heuristic for improving network partitions*, In Proc. 19th IEEE Design Automation Conference, 175181, 1982

[6] D. Papa, I. Markov, *Hypergraph Partitioning and Clustering*, University of Michigan,EECS Department, Ann Arbor, MI 48109-2121

[7] Dj. Paunić, *Strukture podataka i algoritmi*, Univerzitet u Novom Sadu, Prirodno-matematički fakultet, 1997

[8] D. G. Schweikert, B. W. Kernighan, *A proper model for the partitioning of electrical circuits*, In Proc. ACM/IEEE Design Automation Conference, 5762, 1972

[9] G. Karypis, V. Kumar, *Multilevel k-way hyprgraph partitioning*, University of Minnesota, 1998

[10] G. Karypis, V. Kumar, R. Aggarwal, S. Shekhar, *Multilevel Hypergraph Partitioning: Applications in VLSI Domain*, University of Minnesota, 1998

[11] I. Dolinka, *Analiza algoritama*, Univerzitet u Novom Sadu, Prirodno-matematički fakultet, 2008

[12] L. Tao, Y. C. Zahao, K. Thulasiraman, M. N. S. Swamy, *Simulated Annealing and Tabu Search Algorithms for Multiway Graph Partitioning*, Concordia University, Canada, 1992

[13] R. Bisseling, J. Byrka, S. Cerav-Erbas, N. Gvozdenović, M. Lorenz, R. Pendavingh, C. Reeves, M. Roger, A. Verhoeven, *Partitioning a Call Graph*, Study Group Mathematics with Industry 2005, Amsterdam

[14] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, *Optimization by Simulated Annealing*, Science, 220(4598):671680, 1983

Kratka biografija

Milana Gatarić rođena je u Somboru 24. avgusta 1986. godine, gde je završila osnovnu školu i Gimnaziju "Veljko Petrović" kao vukovac. Prirodno-matematički fakultet Univerziteta u Novom Sadu upisuje 2005 godine. Na Departmanu za matematiku i informatiku opredeljuje se za smer primenjene matematike sa naglaskom primene u tehnici. Septembra 2009. godine završava osnovne studije sa prosečnom ocenom deset, čime stiče zvanje Diplomirani inženjer matematike. Iste godine na istom fakultetu upisuje master studije primenjene matematike, modul tehnomatematika. Zaključno sa junskim ispitnim rokom 2010. godine, uspešno polaže sve predviđene ispite master studija sa prosečnom ocenom deset.

UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI FAKULTET
KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj (RBR):	
Identifikacioni broj (IBR):	
Tip dokumentacije (TD):	monografska dokumentacija
Tip zapisa (TZ):	tekstualni štampani materijal
Vrsta rada (VR):	master rad
Autor (AU):	Milana Gatarić
Mentor (ME):	dr Nataša Krejić
Naslov rada (NR):	Novi algoritam za kombinatorno particionisanje hipergrafa
Jezik publikacije (JP):	srpski (latinica)
Jezik izvoda (JI):	s/en
Zemlja publikovanja (ZP):	Republika Srbija
Uže geografsko područje (UGP):	Vojvodina
Godina (GO):	2010
Izdavač (IZ):	autorski reprint
Mesto i adresa (MA):	Novi Sad, Trg D. Obradovića 4
Fizički opis rada (FOR):	(4/40/14/2/8/0/0) (broj poglavlja/strana/lit.citata/tabela/slika/grafka/priloga)
Naučna oblast (NO):	matematika
Naučna disciplina (ND):	diskretna mateamtika

Predmetne odrednice, ključne reči (PO, UDK):	Problem particionisanja hipergrafova, Klasterizacija, Hipergrafovi, Kombinatorna optimizacija
Čuva se (ČS):	
Važna napomena (VN):	
Izvod (IZ):	U radu je razmatran problem particioniranja hipergrafova. Navedene su najznačajnije primene ovog problema u praksi i pokazana je računska složenost ovog problema koji je NP-kompletan. Dat je pregled nekih od postojaćih algoritama za rešavanje problema particionisanja hipergrafa, a zatim je data i ideja novog algoritma koji bi trebao rešavati dati problem do optimuma. Izloženi su neki od rezultata nakon implementacije ovog algoritma u programskom jeziku C++.
Datum prihvatanja teme od strane NN veća (DP):	septembar 2010
Datum odbrane (DO):	oktobar 2010
Članovi komisije (KO):	
Predsednik:	dr Nataša Krejić, redovni profesor Prirodno-matematičkog fakulteta u Novom Sadu
Član:	dr Nebojša Gvozdenović, docent Ekonomskog fakulteta u Subotici
Mentor:	dr Dragan Mašulović, redovni profesor Prirodno-matematičkog fakulteta u Novom Sadu

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCE
KEY WORDS DOCUMENTATION

Serial number (SNO):	
Identification number (INO):	
Document type (DT):	monograph type
Type of record (TR):	printed text
Contents code (CC):	master's thesis
Author (AU):	Milana Gatarić
Mentor (ME):	prof. dr Dragan Mašulović
Title (TI):	New algorithm for combinatorial hypergraph partitioning
Language of text (LT):	Serbian (Latin)
Language of abstract (LA):	s/en
Country of publication (CP):	Republic of Serbia
Locality of publication (LP):	Vojvodina
Publication year (PY):	2010
Publisher (PU):	author's reprint
Publication place (PP):	Novi Sad, Trg D. Obradovića 4
Physical description (PD):	(4/40/14/2/8/0/0) (chapters/pages/lit./tables/pictures/graphs/add.lists)
Scientific field (SF):	mathematics
Scientific discipline (SD)	discrete mathematics

Subject, key words (SKW):	Hypergraph partitioning problem, Clustering, Hypergraphs, Combinatorial optimization
Holding data (HD):	
Note (N):	
Abstract (AB):	In this work, we have dealt with the hypergraph partitioning problem. The main practical applications are presented and NP-completeness of the problem is shown . We discussed about some existing algorithms for solving this kind of problem and then we gave the idea of new algorithm. Some of the results are reported after implementation of this algorithm in programming language C++.
Accepted on Scientific board on (AS):	September 2010
Defended (DE):	October 2010
Thesis Defend board (DB):	
President:	dr Nataša Krejić, full professor, Faculty of Sciences, University of Novi Sad
Member:	dr Nebojša Gvozdenović, docent, Faculty of Economics, University of Novi Sad
Mentor:	dr Dragan Mašulović, full professor, Faculty of Sciences, University of Novi Sad