UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Petar Milikić

PARALELNI ALGORITMI ZA REŠAVANJE PROBLEMA BOJENJA GRAFOVA

master rad

Beograd, 2024.

Mentor:

dr Vesna Marinković, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

prof. dr Filip Marić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Mirko Spasić, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Porodici

Naslov master rada: Paralelni algoritmi za rešavanje problema bojenja grafova

Rezime: Problem bojenja grafa predstavlja problem dodeljivanja boja čvorovima grafa tako da nikoja dva susedna čvora ne budu obojena istom bojom. Najčešće se postavlja zahtev da broj upotrebljenih boja za bojenje bude minimalan. Algoritmi za rešavanje ovog problema imaju širok spektar primene, a neke od primena su: alokacija registara u memoriji, detekcija nepotpune LU dekompozicije matrice i planiranje rasporeda časova. U ovom radu opisane su različite sekvencijalne i paralelne heuristike za bojenje grafova, pri čemu je akcenat stavljen na paralelne heuristike. U radu neće biti razmatrani algoritmi za određivanje bojenja koje koristi minimalni broj boja. Umesto toga razmotrićemo algoritme zasnovane na heuristikama koje za bojenje grafa koriste broj boja koji je blizak minimalnom potrebnom broju boja za bojenje grafa.

U poglavlju koje se odnosi na sekvencijalne algoritme prikazana je heuristika *First Fit* koja predstavlja osnovu gotovo svih heuristika u nastavku. U ovom poglavlju su predstavljene i neke pohlepne heuristike za bojenje grafa, kao i heuristike zasnovane na odabiru nezavisnih skupova u grafu. Na kraju poglavlja je opisana modifikacija *First Fit* heuristike koja predstavlja potpuno drugačiji pristup od svih prethodno navedenih heuristika, a bazira se na reorganizaciji redosleda bojenja čvorova i ponovnim bojenjima čvorova grafa.

U poglavlju koje se bavi paralelnim algoritmima izložena je *GM* heuristika kao jedna od najjednostavnijih paralelnih heuristika za bojenje grafa. Takođe, predstavljene su paralelne heuristike koji su zasnovane na odabiru nezavisnih skupova u grafu, kao i paralelna *Luby-MIS* heuristika za konstrukciju maksimalnog nezavisnog skupa u grafu, koja je potrebna za realizaciju prethodno navedenih heuristika. Opisane su i druge paralelne heuristike poput *Jones Plassman* heuristike i *LDF* heuristike, koje funkcionišu po sličnom principu kao i paralelne heuristike zasnovane na odabiru nezavisnih skupova. Na kraju poglavlja su predstavljene heuristike zasnovane na blokovskom particionisanju grafa.

U okviru poslednje glave izvršena su eksperimentalna poređenja svih sekvencijalnih i paralelnih algoritama na velikim instancama slučajnih grafova i predstavljena je analiza dobijenih rezultata. Algoritmi su poređeni uzimajući u obzir dva aspekta: vreme izvršavanja i broj iskorišćenih boja.

Svi algoritmi su realizovani u jeziku *C#* i mogu se naći na adresi:
`https://github.com/PetarMilikic/parallel-graph-coloring`.

Ključne reči: algoritmi, heuristike, bojenje grafova, paralelizam

Sadržaj

Glava 1

Uvod

1.1 Motivacija

Problem bojenja grafa predstavlja problem dodeljivanja boja čvorovima grafa tako da nikoja dva susedna čvora ne budu obojena istom bojom. Problem bojenja grafova je jedan od ključnih problema teorije grafova sa dosta primena u rešavanju praktičnih problema. Motivaciju za rešavanje ovog problema ćemo, za početak, naći u primeru bojenja kartografske mape. Pretpostavimo da želimo da pomognemo kartografu da oboji geografsku mapu. Kartograf želi da oboji sve države prisutne na mapi. Boja kojom će biti obojena konkretna država nije bitna, ali se zahteva da države koje se graniče ne smeju biti obojene istom bojom. Kartograf je štedljiv i želi da iskoristi najmanji mogući broj boja za bojenje mape.

Godine 1852. *Francis Guthrie*[1] je, pokušavajući da oboji svoju mapu, uočio svojstvo poznato pod nazivom *4-obojivost* (eng. *4-colorability*). Postavio je hipotezu da je za pravilno bojenje bilo koje mape dovoljno četiri boje (slika 1.1). Međutim, ovu hipotezu nije bilo ni malo lako dokazati. Posle mnogo pokušaja i lažnih dokaza, teoremu o četiri boje prvi su dokazali engleski matematičar i logičar *Kenneth Appel* i njegov kolega *Wolfgang Haken* 1976. godine. Zanimljivo je napomenuti da je i blaža verzija teoreme - teorema o pet boja dokazana negde oko 1800. godine.

Prethodno prikazani problem je opisan u terminima bojenja kartografske mape i odnosi se na planarne grafove (videti definiciju 2.3.1). U nastavku rada problem bojenja grafova biće predstavljen u terminima pojmova teorije grafova i u nešto opštijem obliku - u radu će se razmatrati bojenje i planarnih grafova i onih grafova

[1] Francis Guthrie je bio južnoafrički matematičar i botaničar koji je prvi postavio problem četiri boje.

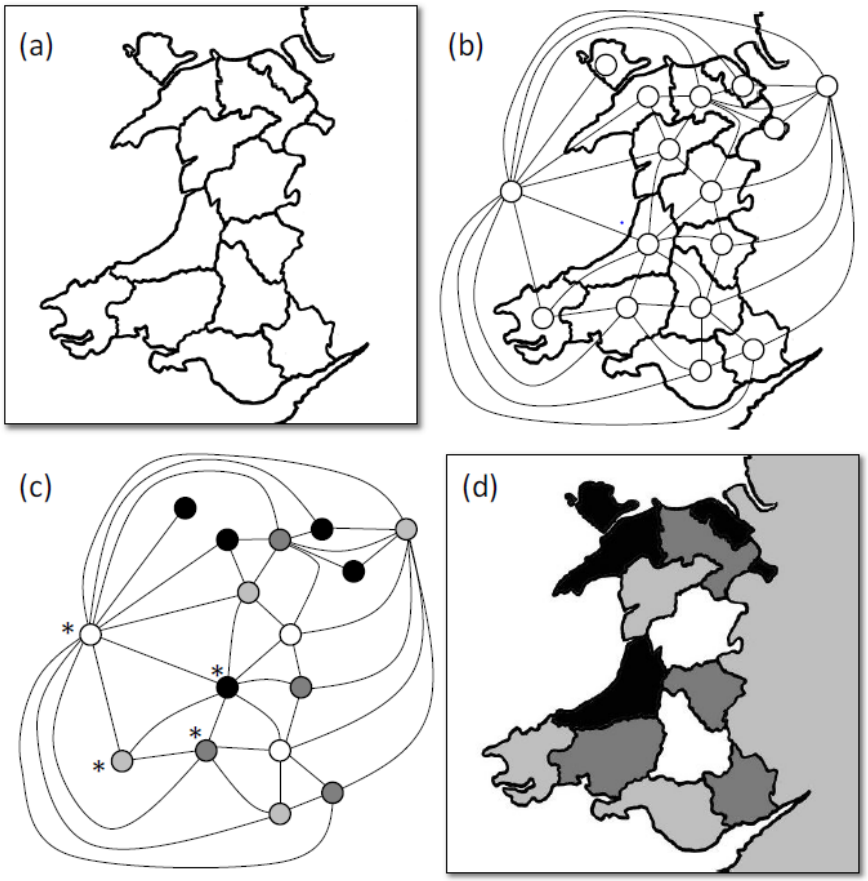
koji to nisu.

Slika 1.1: Primer pravilnog bojenja mape sa 4 boje

1.2 Neke primene bojenja grafova

1.2.1 Problem bojenja kartografske mape

U prethodnom tekstu razmotrili smo problem bojenja kartografske mape. Razmotrimo sada na koji način problem bojenja mape možemo svesti na problem bojenja grafa i kako da na osnovu dobijenog bojenja grafa dobijemo bojenje polazne mape. Na slici 1.2, u delu pod a), je prikazana neobojena mapa. Na istoj slici, u delu pod b), dat je neusmeren graf konstruisan na osnovu mape tako da se svakom regionu dodeljuje tačno po jedan čvor grafa. U odgovarajućem grafu dva čvora su povezana ukoliko se regioni kojima su oni dodeljeni graniče. Ovakvi čvorovi su povezani granom jer je potreno da se oni, kao i regioni koje oni predstavljaju, oboje različitim bojama. Nad ovim grafom pokrećemo algoritam bojenja. Na istoj slici, u delu pod c), prikazan je graf obojen pomoću datog algoritma dok u delu pod d) vidimo bojenje mape dobijeno dodeljivanjem boje čvora regionu koji taj čvor predstavlja.



Slika 1.2: Primer konstrukcije grafa za datu kartografsku mapu i bojenja grafa i mape (preuzeto iz [4]).

1.2.2 Problem raspoređivanja i zakazivanja časova

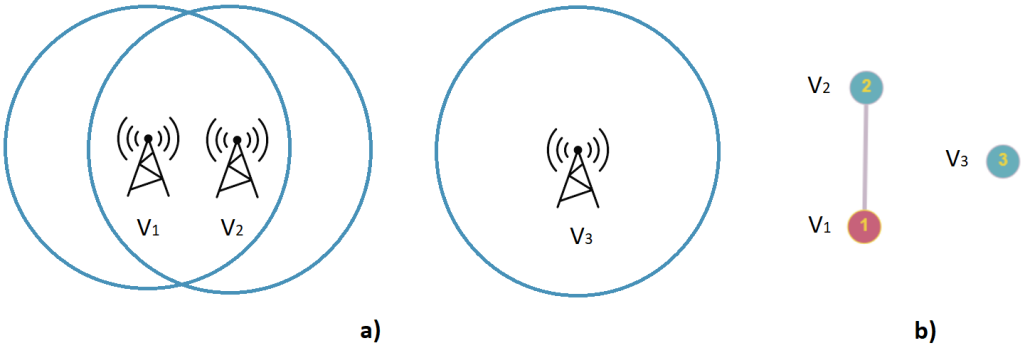
Prilikom pravljenja rasporeda časova na nekom od fakulteta potrebno je poštovati izvesna ograničenja. Na primer, dva kursa koje drži isti profesor ne smeju se držati u istom terminu. Takođe, u istom terminu se ne smeju držati ni dva kursa koje sluša ista grupa studenata. Pretpostavimo da nam je prilikom pravljenja rasporeda časova na raspolaganju dovoljan broj sala. Problem određivanja minimalnog broja vremenskih termina koji poštuju prethodno navedena ograničenja može se svesti na problem bojenja grafova. U ovom slučaju kurseve možemo predstaviti čvorovima u grafu. U datom grafu dva kursa treba povezati granom ako ih sluša ista grupa studenata ili ako ih drži isti profesor. Nakon što kurseve predstavimo čvorovima i povežemo ih granama dobićemo neusmereni graf nad kojim se može primeniti algoritam bojenja. Dobijenu boju čvora možemo shvatiti kao termin koji je dodeljen

kursu - termin u kome će kurs biti održan.

1.2.3 Problem dodeljivanja frekvencija

Razmotrimo problem dodeljivanja radio-frekvencija korisnicima elektromagnetnog spektra i primenu bojenja grafa u rešavanju ovog problema. Korisnici radio-frekvencija mogu biti sateliti, televizijske, radio-stanice i slično. Njihov zadatak je da emituju signal na datoj radio-frekvenciji. S obzirom na ograničenost raspoloživog spektra frekvencija, posebno u slučajevima gustih urbanih područja ili velikog broja uređaja, postoji potreba za efikasnom raspodelom frekvencija kako bi se izbegle smetnje i kolizije signala. Korisnicima koji su blizu jedan drugog moramo dodeliti razliite frekvencije, dok onima koji su na većim udaljenostima možemo dodeliti iste frekvencije. Ovaj problem se takođe može svesti na problem bojenja grafa. U tom slučaju korisnike možemo predstaviti čvorovima u grafu. Ukoliko su korisnici blizu jedan drugog između njih u grafu treba da postoji grana, te će u ispravnom bojenju grafa biti obojeni različitim bojama (i na taj način biće im dodeljene različite frekvencije).

Na slici 1.3, u delu pod a), date su tri radio-stanice označene sa v_1, v_2 i v_3. Radio-stanice v_1 i v_2 se nalaze blizu jedna drugoj i potrebno im je prilikom dodeljivanja radio-frekvencija dodeliti različite frekvencije. Radio-stanica v_3 je na dovoljno velikoj udaljenosti od stanica v_1 i v_2, pa prilikom dodeljivanja radio-frekvencija za nju ne postoji takvo ograničenje. Prilikom formiranja grafa $G = (V, E)$ na koji želimo da primenimo algoritam bojenja potrebno je u skup E dodati granu $\{v_1, v_2\}$. Na istoj slici, u delu pod b) dat je graf formiran na osnovu donesenih zaključaka i ispravno bojenje odgovarajućeg grafa.



Slika 1.3: Primer radio-stanica kao korisnika radio-frekvencije i kolizije između stanica v_1 i v_2; primer grafa kolizije konstruisanog na osnovu datih stanica.

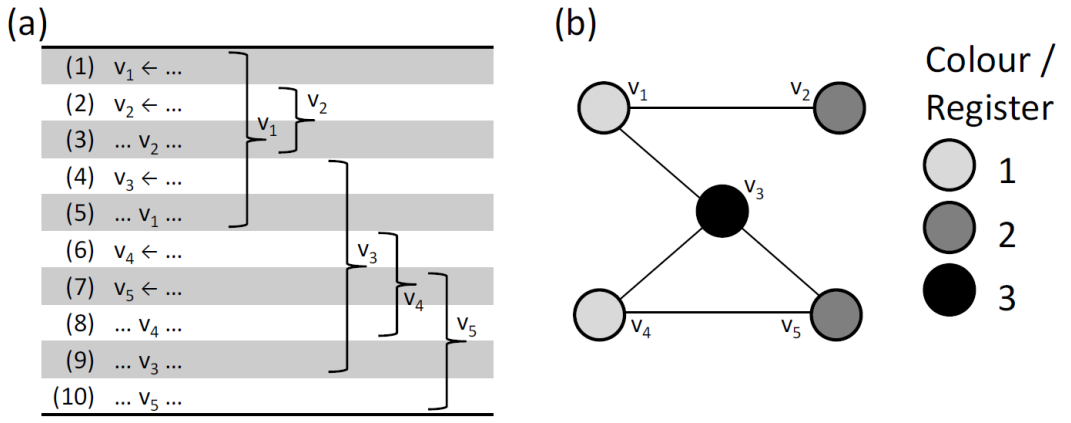
1.2.4 Alokacija registara procesora

Prilikom izvršavanja programa u računaru potrebno je izvršiti alokaciju registara procesora. Ona podrazumeva dodeljivanje hardverskih registrara (kojih je konačno mnogo) promenljivim. Smeštanje promenljivih u registre procesora, u odnosu na smeštanje u RAM memoriji, ima niz prednosti koje doprinose bržem i efikasnijem izvršavanju programa. Naime, registri su najbrži memorijski elementi procesora i najbliži elementi procesoru. Pristupanje registrima gotovo je trenutno, dok pristupanje RAM memoriji zahteva više vremena zbog različitih nivoa keš memorije i fizičke udaljenosti od procesorskog jezgra. Ako se promenljive često koriste, smeštanjem u registre smanjuje se potreba za stalnim pristupom RAM memoriji. To može smanjiti opterećenje memorije i olakšati njen brzi pristup drugim delovima programa.

Najčešći slučaj u praksi je onaj u kome postoji mnogo više promenljivih kojima treba dodeliti registre nego samih registara. Ovde postoji jedno ograničenje: određene promenljive i izrazi ne smeju biti smešteni u isti registar zbog konflikata u vremenskom ili prostornom korišćenju. Ukoliko za neke promenljive važi da su aktivne u istom delu koda a dodeljene su istom registru kažemo da su u konfliktu. Koncept koji daje odgovor na pitanje kada dve promenljive mogu deliti isti registar zove se *koncept životnosti promenljiih* (eng. *liveness concept*). Ovaj koncept je ključan za efikasno upravljanje registrima, a govori o tome da dve promenljive mogu deliti isti registar samo ako se njihovi *životni opsezi* (eng. *live ranges*), kao periodi u kojima su promenljive aktivne, ne presecaju. Cilj problema alokacije registara je nekonfliktno dodeljivanje promenljivih registrima tako da se minimizuje broj promenljivih kojima nisu dodeljeni registri, što za ishod ima minimizaciju upotrebe memorije i optimizaciju izvršavanja programa. Pokazano je da ovaj problem može da se svede na problem bojenja grafova. U tom slučaju potrebno je predstaviti promenljive i njihove međusobne zavisnosti grafom. Čvorovi predstavljaju promenljive, dok grane predstavljaju njihove međusobne zavisnosti, odnosno konflikte. Zatim se primenjuje algoritam bojenja grafa kako bi se promenljivima dodelile boje, odnosno u ovom slučaju registri promenljivima. Algoritam bojenja primenjen nad ovako definisanim grafom će pridružiti jedno validno bojenje istom. Svakoj od boja u dobijenom bojenju pridružuje se jedan registar. Kao rezultat datog pridruživanja nikoje dve promenljive koje se koriste u okviru istog regiona neće biti pridružene istom registru.

Na slici 1.4, u delu pod a), dat je kod dela programa čije je promenljive potreb-

no dodeliti registrima. Promenljive su označene sa v_1, v_2, v_3, v_4 i v_5. Iste oznake su iskorišćene za predstavljanje dela koda u kome su promenljive prisutne. Komande oblika „$v_i \leftarrow \ldots$" predstavljaju dodelu vrednosti promenljivoj v_i, dok komande oblika „$\ldots v_i \ldots$" predstavljaju bilo kakvu operaciju koja se izvodi nad promenljivom v_i. Na primer, promenljiva v_3 se koristi od 4. do 9. linije koda i deo koda u kome je ona aktivna je označen sa v_3. Možemo videti da se, na primer, delovi koda u kojima su promenljive v_1 i v_4 aktivne ne preklapaju dok se delovi u kojima su v_1 i v_2 aktivne preklapaju. Ovo znači da promenljive v_1 i v_4 mogu biti prisutne u okviru istog registra, dok za promenljive v_1 i v_2 to ne bi trebalo da važi. Na istoj slici, u delu pod b), dat je graf koji modeluje konflikte između promenljivih: $v_1 - v_5$. Na graf je primenjen algoritam bojenja koji je obojio graf sa tri boje. Shodno ovim bojama, obeleženim brojevima 1, 2 i 3, promenljivim će biti dodeljena 3 registra indeksirana brojevima 1, 2 i 3. Na primer, promenljivoj v_2 je dodeljena boja indeksa 2 i registar indeksa 2.



Slika 1.4: Primer grafa pridruženog promenljivim i njima pridruženi registri/boje (preuzeto iz [4]).

Glava 2

Osnovni pojmovi

2.1 Definicije osnovnih pojmova

Definicija 2.1.1. *Usmereni graf* (eng. *ordered graph*) G je uređeni par dva skupa $G = (V, E)$, gde je V neprazan i konačan skup, a E podskup Dekartovog proizvoda $V \times V$. Skup V grafa $G = (V, E)$ se naziva *skupom čvorova* (eng. *vertex set*) dok se skup E naziva *skupom grana* (eng. *edge set*).

Definicija 2.1.2. *Nesmereni graf* (eng. *unordered graph*) G je uređeni par dva skupa $G = (V, E)$, gde je V neprazan i konačan skup, a E skup dvočlanih podskupova skupa V.

Često se za skup čvorova V uzima neki konačan podskup skupa prirodnih brojeva. Na taj način dobijamo i numeraciju čvorova grafa, što je najčešći slučaj obeležavanja čvorova grafa. Čvorovima i granama grafa često se pripisuju neka važna svojstva. Najčešća svojstva koja se pripisuju granama i čvorovima su težina grane i boja čvora. U ovom radu fokus će biti na određivanju pravilnog načina za bojenje čvorova grafa uz odgovarajuća ograničenja.

Definicija 2.1.3. U usmerenom grafu $G = (V, E)$ kažemo da je čvor v sused (eng. *neighbour*) čvora u ako važi da je $(u, v) \in E$. Na sličan način se definišu susedni čvorovi u neusmerenim grafovima - u i v su *susedni* ako važi da je $\{u, v\} \in E$. U slučaju da su čvorovi u i v u neusmerenom grafu susedni tada kažemo da je čvor u sused čvora v i da je v sused čvora u.

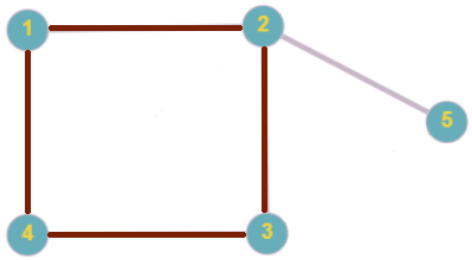
Primetimo da za usmerene grafove ne važi da ako je u sused čvora v da je v sused čvora u. Za čvorove u neusmerenom grafu koji su susedni se često kaže i da su *povezani* granom (eng. *connected by an edge*).

Definicija 2.1.4. *Stepen čvora* v u neusmerenom grafu G (eng. *node degree*), u oznaci $deg(v)$, je broj čvorova koji su susedi sa čvorom v. Maksimalni stepen čvora u grafu G označavaćemo sa $\Delta(G)$, a minimalni stepen čvora u G sa $\delta(G)$.

Na slici 2.1 prikazan je graf za koji važi $\delta(G) = 1$ i $\Delta(G) = 3$.

Definicija 2.1.5. *Put* (eng. *path*) u grafu G je niz čvorova u kome je svaki par uzastopnih čvorova međusobno povezan. Put predstavlja putanju kojom se može kretati kroz graf prateći njegove grane i posećujući čvorove u određenom redosledu.

Definicija 2.1.6. *Ciklus* (eng. *cycle*) u grafu G je put u G koji počinje i završava se istim čvorom. Dakle, ciklusi su zatvoreni putevi u grafovima. Ako graf nema cikluse u sebi kažemo da je on *acikličan* (eng. *acyclic*).



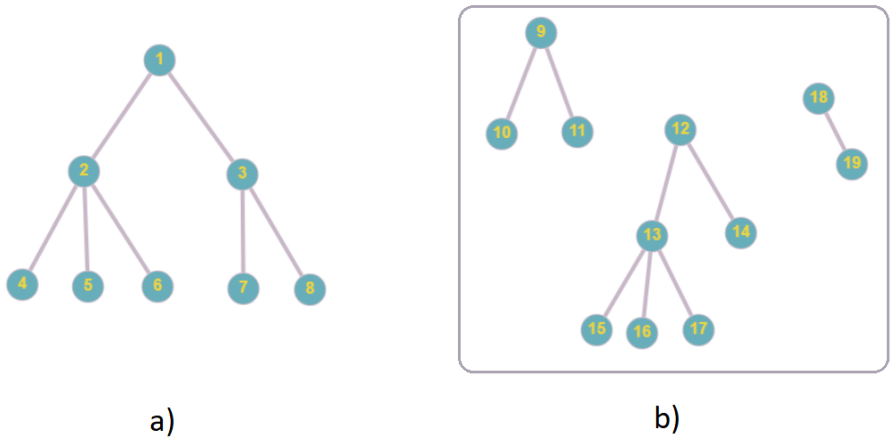
Slika 2.1: Primer ciklusa u grafu

Na slici 2.1 prikazan je neusmeren graf koji sadrži ciklus dužine 4: $[1 \to 2 \to 3 \to 4 \to 1]$. Dati ciklus je podebljan na slici. Kao primere puteva koji nisu ciklusi navodimo dva puta od čvora sa oznakom 1 do čvora sa oznakom 5: $[1 \to 2 \to 5]$ i $[1 \to 4 \to 3 \to 2 \to 5]$.

Definicija 2.1.7. *Povezan graf* (eng. *connected graph*) je graf u kome između svaka dva čvora postoji put.

Definicija 2.1.8. *Stablo* (eng. *tree*) je povezan aciklični graf.

Definicija 2.1.9. *Šuma* (eng. *forest*) je bilo koji aciklični graf.



Slika 2.2: Primer stabla i šume

Drugim rečima, šuma je disjunktna unija stabala. Na slici 2.2, u delu pod a), prikazan je primer stabla dok je u delu pod b) prikazan primer šume.

Definicija 2.1.10. Neka je $G = (V, E)$ neusmereni graf i neka je $V' \subset V$. *Indukovani podgraf* (eng. *induced subgraph*) grafa $G = (V, E)$ je graf $G' = (V', E')$ gde je $E' = \{(u, v) | (u, v) \in E,\ u, v \in V'\}$. Kažemo da je podgraf G' indukovan skupom V'.

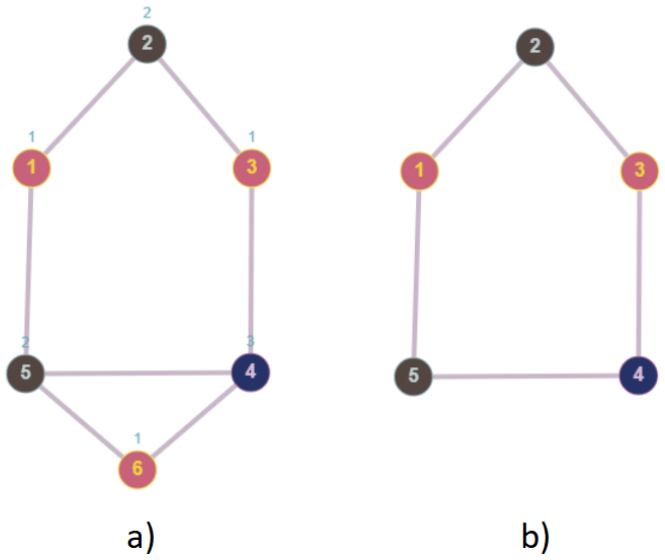
Definicija 2.1.11. *Nezavisan skup čvorova* (eng. *independent set*) grafa $G = (V, E)$ je podskup S skupa čvorova V za koji važi da nikoja dva čvora iz skupa S nisu susedna u grafu G.

Na slici 2.3, u delu pod a), dat je graf koji sadrži 6 čvorova koji su numerisani brojevima od 1 do 6. U odnosu na dati graf navodimo neke od nezavisnih skupova: $S_1 = \{1, 3, 6\}$, $S_2 = \{2, 5\}$ i $S_3 = \{4\}$. Na istoj slici, u delu pod b), vidimo podgraf datog grafa koji je indukovan skupom čvorova $\{1, 2, 3, 4, 5\}$.

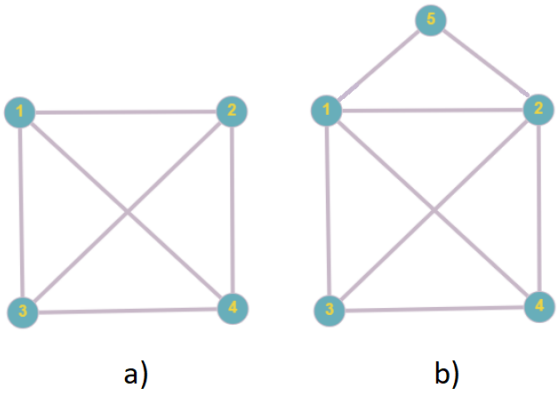
Definicija 2.1.12. *Kompletan graf* (eng. *complete graph*) je graf $G = (V, E)$ u kome su svaka dva čvora $u, v \in V$ međusobno povezana granom. Za kompletan graf sa n čvorova koristimo oznaku K_n.

Definicija 2.1.13. *Klika* (eng. *clique*) u neusmerenom grafu $G = (V, E)$ je skup čvorova $V' \subseteq V$ takav da za svaka dva čvora u, $v \in V'$ važi da su međusobno povezani granom.

Na slici 2.4, u delu pod a), prikazan je jedan kompletan K_4 graf. Na istoj slici, u delu pod b), prikazan je graf sa skupom čvorova $V = \{1, 2, 3, 4, 5\}$. Jednu kliku u datom grafu čini skup čvorova $C = \{1, 2, 3, 4\}$.



Slika 2.3: U delu pod a): primeri nekih nezavisnih skupova čvorova u grafu ($S_1 = \{1, 3, 6\}$, $S_2 = \{2, 5\}$ i $S_3 = \{4\}$). U delu pod b): podgraf grafa iz dela pod a), a koji je indukovan skupom čvorova $\{1, 2, 3, 4, 5\}$

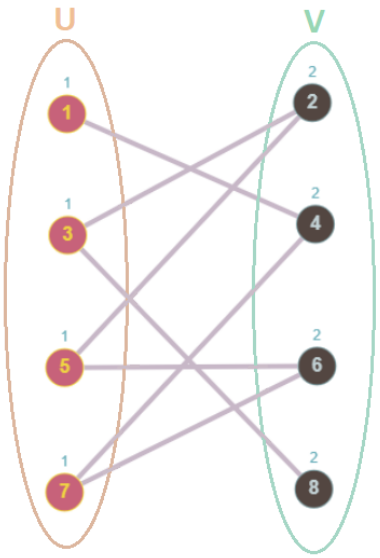


Slika 2.4: Primer kompletnog grafa i grafa koji sadrži kliku veličine 4

Definicija 2.1.14. *Bipartitni graf* (eng. *bipartite graph*) je graf čiji se čvorovi mogu podeliti u dva disjunktna skupa U i V tako da svaka grana iz skupa E povezuje jedan

čvor iz skupa U sa jednim čvorom iz skupa V. Za bipartitan graf tada koristimo notaciju $G = (U, V, E)$.

Na slici 2.5 prikazan je jedan bipartitan graf. Sa U i V su označeni partitivni skupovi datog grafa.



Slika 2.5: Primer bipartitnog grafa

Definicija 2.1.15. *Izomorfizam* (eng. *isomorphism*) grafova $G = (V, E)$ i $G' = (V', E')$ je bijekcija $f : V \to V'$ takva da važi:

$$(\forall u, v \in V)((u, v) \in E) \iff (f(u), f(v)) \in E').$$

Kažemo da su grafovi *izomorfni* (eng. *isomorphic*) ako postoji izomorfizam između njih.

Izomorfne grafove možemo poistovetiti, s obzirom na to da imaju ista svojstva.

Definicija 2.1.16. *Bojenje grafa* predstavlja dodeljivanje boja čvorovima grafa tako da nikoja dva susedna čvora grafa nisu obojena istom bojom.
Takođe, bojenje može da se shvati i kao funkcija: $f : V \to Colors = \{c_1, c_2, \ldots, c_k\}$ za koju važi $(u, v) \in E \Rightarrow f(u) \neq f(v)$, gde je sa $Colors$ označen skup boja koje se dodeljuju čvorovima.

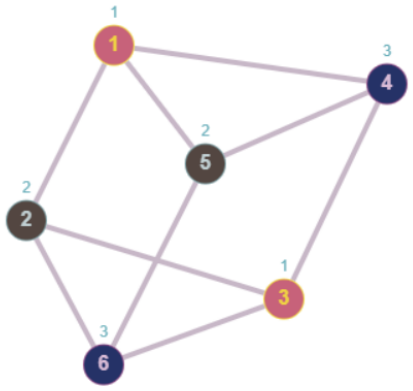
Bojenje još može da se shvati i kao dodeljivanje prirodnih brojeva čvorovima. Često se zahteva da iskorišćeni broj boja za bojenje grafa bude minimalan.

Definicija 2.1.17. Najmanji broj boja potreban za bojenje datog grafa naziva se *hromatski broj grafa* i označava se sa $\chi(G)$.

Na slici 2.5 prikazano je 2-bojenje bipartitnog grafa. Hromatski broj svakog bipartitnog grafa iznosi 2.

Definicija 2.1.18. *Optimalno bojenje* (eng. *optimal coloring*) grafa $G = (V, E)$ je ono bojenje koje koristi $\chi(G)$ boja.

Na slici 2.6 prikazano je jedno 3-bojenje grafa. Hromatski broj datog grafa je 3, jer dati graf nije moguće obojiti sa dve boje pošto sadrži trouglove (tri čvora međusobno povezana granama). Ovo jeste jedno optimalno bojenje grafa.



Slika 2.6: Primer optimalnog bojenja grafa

Slučajan graf (eng. *random graph*) $G = (V, E)$, gde je $V = \{v_1, v_2, \ldots, v_n\}$, je graf za koji važi da je za bilo koja 2 čvora $v_i, v_j \in V$ (gde je $i \leq j \leq n$) verovatnoća da između njih postoji grana jednaka p, gde je $p \in (0, 1)$. Za različite vrednosti parametra p dobijaju se grafovi sa različitim očekivanim brojem grana. Na primer, ukoliko je $p = \frac{1}{2}$ očekivani broj grana u grafu je jednak $\frac{1}{2} \cdot \frac{n(n-1)}{2} = \frac{n(n-1)}{4}$. Primetimo da se pri ovakavom odabiru parametra p mogu dobiti jako gusti grafovi.

2.2 Složenost problema bojenja grafova

U ovom radu ćemo razmotriti konstrukciju heuristika za bojenje grafova. Pritom ćemo nastojati da konstruisano bojenje bude što bliže optimalnom bojenju grafa.

Određivanje hromatskog broja grafa predstavlja jedan od 21 problema koji se nalaze na poznatoj *Karpovoj*[1] listi NP-kompletnih problema, formulisanoj 1972. godine [1]. Stoga, određivanje hromatskog broja grafa kao i optimalnog bojenja grafa nisu jednostavni problemi. Tokom godina razvijene su brojne heuristike za određivanje bojenja grafa koje ne garantuju dobijanje optimalnog bojenja, ali daju prilično dobre rezultate. Vremenom su razvijeni i razni *randomizovani algoritmi* (eng. *randomized algorithms*) koji daju odlične rezultate kako po pitanju vremena izvršavanja tako i po pitanju broja iskorišćenih boja.

Jedan od mogućih načina za ubrzanje ovakvih algoritama je njihova paralelizacija. Treba imati na umu da paralelizacija ne može NP-kompletan problem pretvoriti u problem iz klase P. Naime, paralelizacija algoritma može dovesti do ubrzanja izvršavanja na paralelnim arhitekturama ili smanjiti vreme izvršavanja za određene instance problema, ali to ne menja suštinsku složenost problema. NP-kompletni problemi smatraju se teškim za rešavanje u opštem slučaju, bez obzira na to da li koristimo paralelne ili sekvencijalne algoritme.

2.3 Osnovna tvrđenja vezana za bojenje grafova

Definicija 2.3.1. *Planarni grafovi* (eng. *planar graphs*) su oni grafovi koji se mogu nacrtati u ravni tako da im se grane ne seku. Preciznije, zahteva se da je graf moguće predstaviti u ravni tako da zajednička tačka dve grane može biti samo čvor koji predstavlja zajedničku krajnju tačku tih grana. Ako je planaran graf predstavljen na opisan način u ravni, on deli ravan na više konačnih zatvorenih oblasti i jednu beskonačnu oblast.

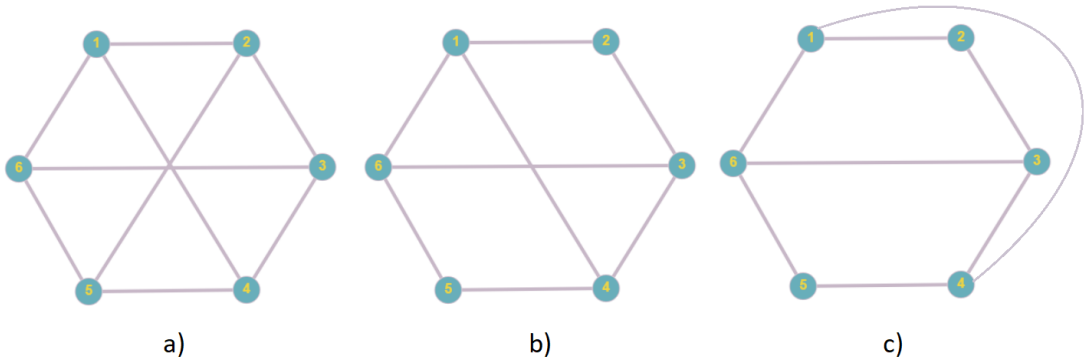
Na slici 2.7, u delu pod a), dat je primer neplanarnog grafa. Na istoj slici, u delu pod b), prikazan je graf koji se dobija uklanjanjem jedne grane iz grafa koji je prikazan u delu pod a). Graf dat na istoj slici, u delu pod c), je očigledno planaran. Primetimo da su grafovi prikazani u delu pod b) i pod c) izomorfni, pa je i graf prikazan u delu pod b) planaran.

Teorema 2.3.1. *Svaki planarni graf je 4-obojiv.*

Dokaz prethodno navedene teoreme nećemo navoditi. Detalji dokaza se mogu pronaći u [2].

[1]Richard Manning Karp, američki naučnik, poznat po istraživanjima u oblasti teorije algoritama

Tvrđenje ove teoreme nam može pomoći da procenimo rešenja dobijena različitim heuristikama za bojenje grafova. Naime, teorema kaže da je hromatski broj planarnog grafa najviše 4. Ukoliko konstruisana heuristika za bojenje grafova oboji planarni graf sa mnogo više od četiri boje, to znači da postoje mogućnosti za dalja unapređenja heuristike.



Slika 2.7: Primer neplanarnog grafa i dva (izomorfna) planarna grafa

Teorema 2.3.2. *Za svaki povezani graf G važi: $\chi(G) \leq \Delta(G) + 1$, gde je $\Delta(G)$ maksimalni stepen čvora u grafu G.*

U najgorem slučaju, kada su svi susedi čvora v obojeni različitom bojom, potrebno je iskoristiti $deg(v) + 1$ boja za bojenje čvora v zajedno sa njegovim susedima. Broj $deg(v) + 1$ je odozgo ograničen sa $\Delta(G) + 1$, pa iz te činjenice sledi i zaključak teoreme. Ova teorema se može koristiti u sličnom kontekstu kao i prethodno navedena teorema. Ukoliko konstruisana heuristika za bojenje povezanog grafa G koristi dosta više od $\Delta(G) + 1$ boja, to znači da treba razmotriti na koji način bi heuristika mogla biti unapređena.

Navedimo još par zanimljivih tvrđenja koja se tiču bojenja grafova:

1. Važi nejednakost: $0 \leq \chi(G) \leq n$, gde je $n = |V|$.
2. Jedini grafovi koji su obojivi jednom bojom su grafovi bez grana.
3. Za kompletan graf K_n sa n čvorova važi $\chi(K_n) = n$.
4. Ukoliko graf sadrži kliku veličine k onda za taj graf važi: $k \leq \chi(G) \leq n$.

5. Graf je 2-obojiv ako i samo ako je bipartitan. Jedini 2-obojivi grafovi su bipartitni grafovi. Stabla i šume su 2-obojivi grafovi, jer su bipartitni grafovi.

6. Grafovi sa velikim brojem klika imaju visok hromatski broj dok obrnuto ne mora da važi.

Sva prethodno navedena tvrđenja su očigledna, sem poslednjeg tvrđenja koje je uvedeno neformalno, bez namere da se prikaže dokaz.

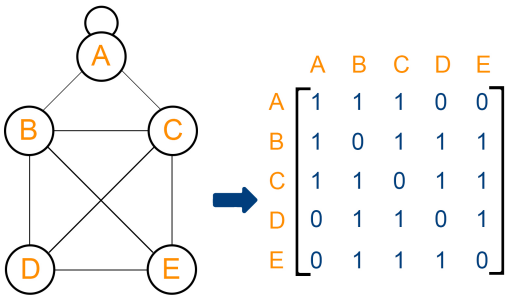
2.4 Reprezentacija grafova u računaru

Bez smanjenja opštosti možemo pretpostaviti da je graf sa kojim radimo usmeren. Slučaj neusmerenog grafa se može svesti na slučaj usmerenog grafa tako što svaku granu neusmerenog grafa razmatramo kao dve grane usmerenog grafa, po jednu u svakom smeru. Postoje dva ustaljena načina za predstavljanje grafova u računaru: pomoću matrice povezanosti i pomoću liste povezanosti.

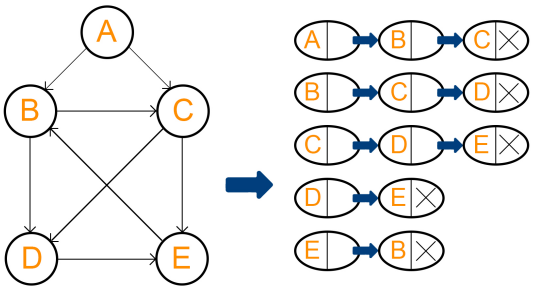
Neka je $|V| = n$ i $V = \{v_1, v_2, \ldots, v_n\}$. *Matrica povezanosti* (eng. *adjacency matrix*) grafa G je kvadratna matrica $A = (a_{ij})$ reda n sa elementima a_{ij} koji su jednaki 1 ako i samo ako $(v_i, v_j) \in E$, dok u suprotnom važi da je $a_{ij} = 0$. Primetimo da važi da je matrica A simetrična ako je graf G neusmeren. Nedostatak predstavljanja grafa matricom povezanosti je u tome što matrica uvek zauzima prostor veličine n^2, nezavisno od toga koliko grana ima graf. Na slici 2.8 je dat primer neusmerenog grafa i njegove reprezentacije pomoću matrice povezanosti.

Drugi način reprezentacije grafa je pomoću *liste susedstava* (eng. *adjacency list*). Svakom čvoru grafa se dodeljuje povezana lista njegovih suseda - čvorova do kojih postoji grana iz datog čvora. Graf se tada predstavlja nizom lista. Svaki element niza sadrži ime (indeks) čvora i pokazivač na njegovu listu suseda. Na slici 2.9 je dat primer usmerenog grafa i njegove reprezentacije pomoću liste povezanosti. Sa matricama povezanosti je jednostavnije raditi. S druge strane, liste povezanosti su efikasnije za grafove sa malim brojem grana. U praksi se često radi sa grafovima koji imaju znatno manje grana od maksimalnog mogućeg broja grana, i tada je efikasnije koristiti liste povezanosti. Dodavanje novog čvora u graf je jednostavnije u slučaju liste povezanosti nego u slučaju matrice povezanosti. Prednost reprezentacije grafa pomoću matrice povezanosti je u brzoj proveri postojanja grane u grafu - za vreme $O(1)$, dok u slučaju liste povezanosti to vreme iznosi $O(|V|)$. Takođe, prednost matrica povezanosti je u brzom dodavanju i brisanju nove grane u graf -

obe operacije se mogu obaviti za vreme $O(1)$. U slučaju reprezentacije grafa pomoću liste povezanosti, dodavanje nove grane u graf se može obaviti za vreme $O(1)$, dok se brisanje grane iz grafa može obaviti za vreme $O(|V|)$. U slučaju reprezentacije grafa pomoću liste povezanosti prolazak kroz sve grane u grafu zahteva $O(|E|)$ operacija, dok je u slučaju reprezentacije pomoću matrice povezanosti potrebno $|V|^2$ operacija.



Slika 2.8: Reprezentacija neusmerenog grafa pomoću matrice povezanosti



Slika 2.9: Reprezentacija usmerenog grafa pomoću liste povezanosti

2.5 Probabilistički algoritmi

Probabilistički algoritmi (eng. *Probabilistic Algorithms*) predstavljaju koncept koji se bitno razlikuje od koncepta deterministčkog algoritma. Deterministički algoritmi se karakterišu time što je njihov način izvršavanja uvek isti za isti ulaz algoritma. Takođe, i rezultat rada ovakvih algoritama, odnosno izlaz algoritma, je uvek isti za iste ulaze algoritma. Drugim rečima, koraci algoritma su u potpunosti određeni ulazom algoritma. Kod probabilističkih algoritama koraci algoritma zavise od ulaza algoritma i od nekih slučajnih promenljivih. Često korišćeni termin za pro-

babilističke algoritme je *randomizovani algoritmi* (eng. *randomized algorithms*). Postoje dve klase ovakvih algoritama: *Monte Karlo* i *Las Vegas*. Monte Karlo algoritmi se karakterišu mogućnošću dobijanja netačnog rezultata, obično sa jako malom vero vatnoćom, ali im je vreme izvršavanja bolje od najboljeg determinističkog algoritma za rešavanje tog problema. Monte Karlo algoritam generiše nasumične instance problema i računa rešenje za svaku od njih. Preciznost se može povećati povećanjem broja generisanih instanci. Analizom ovakvih rezultata algoritam donosi zaključak ili daje procenu traženog rešenja.

S druge strane, Las Vegas algoritmi predstavljaju potpuno drugačiji koncept. Oni su karakteristični po tome što uvek daju ispravan rezultat, ako se algoritam završi. Kod Las Vegas algoritama postoji mogućnost da se algoritam nikada ne završi, ali je verovatnoća za to obično jako mala.

Monte Karlo algoritmi se po pravilu brže izračunavaju, ali bez garancije tačnosti. Sa druge strane Las Vegas algoritmi uvek daju tačan rezultat, ali ne postoji garancija za vreme izvršavanja. Probabilistički algoritmi zahtevaju razumevanje teorije verovatnoće u kontekstu analize vremenske složenosti. U tom smislu vremenska složenost algoritma se često posmatra kao slučajna promenljiva $T(n)$. Tada je od suštinskog značaja za analizu algoritma pronalaženje njenog matematičkog očekivanja $E[T(n)]$. Dodatno, ako je u pitanju paralelni algoritam, sve navedene veličine zavise od broja procesora p dostupnih algoritmu.

2.6 Modeli paralelnog izvršavanja

Paralelizacija je moćan mehanizam za poboljšanje performansi algoritma kod koga se teži što boljoj iskorišćenosti postojeće arhitekture računara. U slučaju sekvencijalnih algoritama, osnovne mere složenosti pri ispitivanju algoritma su vreme izvršavanja algoritma i količina korišćene memorije. Kod paralelnih algoritama se, pored vremena izvršavanja i količine korišćene memorije, vodi računa o broju procesora koje algoritam koristi. U praksi je broj procesora ograničen, pa se mora voditi računa na koji način se koriste dati procesori.

Sledeći važan aspekt o kome se mora voditi računa prilikom razvoja paralelnih algoritama je komunikacija između procesora. Pod terminom udaljenost između procesora podrazumevamo fizičku udaljenost između njih. Ona predstavlja bitan faktor koji utiče na konstrukciju paralelnih algoritama - potrebno je organizovati komunikaciju na efikasan način i svesti je na minimum kako bi se dobilo optimalno

iskorišćenje resursa.

Na kraju, jako važan aspekt je i sama sinhronizacija. *Sinhronizacija* (eng. *synchronization*) kod paralelnih algoritama je proces koordinacije izvršavanja različitih niti ili procesa kako bi se osiguralo ispravno izvršavanje algoritma. Bez odgovarajuće sinhronizacije mogu se pojaviti problemi kao što su *uzajamno blokiranje* (eng. *deadlock*) i narušavanje konzistentnosti podataka. Uzajamno blokiranje podrazumeva situacije do kojih dolazi kada dve (ili više) niti (ili procesa) čekaju jedni druge da završe sa izračunavanjem da bi mogli da nastave sa radom. U takvim situacijama dolazi do blokiranja njihovog rada jer ni jedna nit ne može nastaviti rad. Narušavanje konzistentnosti podataka podrazumeva situacije u kojima prilikom istovremene obrade istog podatka od strane više niti, zbog nedostatka sinhronizacije, dolazi do upisivanja neispravnog rezultata na memorijsku lokaciju.

Osnovni modeli paralelnih računara su *SIMD* i *MIMD*. *SIMD* (*Single-Instruction Multiple-Data*) model predstavlja klasu paralelnih računara kod kojih važi ograničenje da svi procesori u jednom koraku izvršavaju jednu istu instrukciju (ali ne nužno nad istim podacima). Nasuprot tome, model *MIMD* (*Multiple-Instruction Multiple-Data*) predstavlja klasu paralelnih računara kod koje je omogućeno svakom procesoru da izvršava različit program. U nastavku, sem ukoliko se eksplicitno ne kaže drugačije, podrazumeva se da je reč o *MIMD* modelu.

2.7 Složenost paralelnih algoritama

Kod sekvencijalnih algoritama vreme izvršavanja algoritma zavisi od veličine ulaza algoritma i označava se sa $T(n)$. Kod paralelnih algoritama, sem veličine ulaza na vreme izvršavanja algoritma utiče i broj procesora koje algoritam ima na raspolaganju. Oznaka koju ćemo koristiti za vreme izvršavanja algoritma je $T(n, p)$, gde je sa n označena veličina ulaza, a sa p broj procesora. Ukoliko je paralelni algoritam konstruisan kako treba onda će sa porastom broja procesora p vrednosti $T(n, p)$ padati.

Definicija 2.7.1. *Ubrzanje algoritma* (eng. *algorithm acceleration*) je odnos

$$S(p) = \frac{T(n, 1)}{T(n, p)}$$

gde je $T(n, 1) = T(n)$ vreme izvršavanja najboljeg sekvencijalnog algoritma za rešavanje datog problema.

Primetimo da važi: $1 \leq S(p) \leq p$ i da je paralelni algoritam najefikasniji kada je $S(p) = p$ jer je tada vreme koje je potrebno sekencijalnom algoritmu (koji rešava isti problem) smanjeno p puta. U slučaju da važi $S(p) = p$ kažemo da algoritam dostiže savršeno ubrzanje.

Definicija 2.7.2. *Efikasnost* (eng. *efficiency*) paralelnog algoritma se definiše izrazom:

$$E(n,p) = \frac{S(p)}{p} = \frac{T(n,1)}{pT(n,p)}$$

Efikasnost je, dakle, odnos vremena izvršavanja algoritma na jednom procesoru i ukupnog vremena izvršavanja algoritma na p procesora. Efikasnost ukazuje na udeo procesorskog vremena koji se efektivno koristi u odnosu na sekvencijalni algoritam. Za efikasnost važi nejednakost:

$$\frac{1}{p} \leq E(n,p) \leq 1.$$

Pri konstrukciji paralelnog algoritma se teži maksimizaciji njegove efikasnosti. U slučaju kada je $E(n,p) = 1$, paralelni algoritam obavlja istu količinu računanja na svim procesorima tokom svog izvršavanja, što se poklapa sa količinom računanja potrebnom za izvršavanje odgovarajućeg sekvencijalnog algoritma. Ukoliko važi da je $E(n,p) = 1$, kažemo da je postignuto optimalno iskorišćenje procesora. Iako je ova osobina jako poželjna, predstavlja redak slučaj. Naime, paralelni algoritmi su po pravilu kompleksniji od sekvencijalnih algoritama i često moraju izvršiti dodatna izračunavanja koja nisu potrebna kod sekvencijalnih algoritma.

Često se prilikom konstrukcije paralelnog algoritma fiksira broj p procesora koji su na raspolaganju, a zatim se izraz $T(n,p)$ minimizuje. Nedostatak ovog pristupa je preosetljivost na promenu broja procesora p. Ovo naime znači da bi pri promeni broja procesora često morao da se konstruiše novi algoritam. Bolji pristup je konstruisati algoritam koji radi sa što je više moguće različitih vrednosti parametra p.

Posmatrajmo sada algoritam koji ima osobinu $T(n,p) = X$. Transforimišimo ga u algoritam za koji ćemo iskorisiti $\frac{p}{k}$ (gde je $k > 1$) procesora umesto originalnih p. Novi algoritam konstruišemo zamenom svakog koraka polaznog algoritma sa k novih koraka. U okviru svakog koraka jedan procesor oponaša izvršavanje jednog paralelnog koraka na k procesora. U tom slučaju važi:

$$T(n, \frac{p}{k}) = kX.$$

Ovo znači da će vreme izvršavanja algoritma biti k puta veće ukoliko broj korišćenih procesora smanjimo na $\frac{p}{k}$. Drugim rečima, može se smanjiti broj procesora ne menjajući bitno efikasnost algoritma. Međutim, ovaj pristup nije uvek primenljiv - u opštem slučaju p ne mora biti deljivo sa k. Takođe, donošenje odluke koje procesore treba imitirati jednim procesorom nije uvek jednostavna odluka.

Gorenavedeni *princip imitiranja paralelizma* (eng. *parallelism emulation*), pored svih svojih nedostataka, predstavlja vrlo koristan koncept.

2.7.1 Modeli sa zajedničkom memorijom

Modeli paralelnog izvršavanja sa *zajedničkom memorijom* (eng. *shared memory*) podrazumevaju postojanje zajedničke memorije kojoj svi procesori imaju jednako brz i ravnomeran pristup. Prethodno navedena pretpostavka predstavlja teorijski koncept i nije čest slučaj u praksi. Postoje razni modeli sa zajedničkom memorijom čije se razlike ispoljavaju u načinu na koji se rešavaju konflikti prilikom pristupa memoriji. U odnosu na to imamo modele sa blažim i modele sa strožijim kriterijumima kada govorimo o dozvolama za istovremeno čitanje i pisanje na istu memorijsku lokaciju.

Osnovne komponente modela sa zajedničkom memorijom su procesor i zajednička memorija. Svaki procesor ima svoje resurse i može pristupiti zajedničkoj memoriji za konstantno vreme. Pod pojmom memorijskog konflikta podrazumevamo pokušaj istovremenog pristupa memoriji od strane dva (ili više) procesora. Prilikom konstrukcije paralelnih algoritama posebna pažnja se mora obratiti na obradu potencijalnih memorijskih konflikata. Ukoliko memorijski konflikti nisu obrađeni na ispravan način, ispravnost podataka kojima pristupaju različiti procesori se ne može garantovati. Naredni primer ilustruje kako neobrađeni memorijski konflikt može ostaviti podatak u nekonzistentnom stanju. Pretpostavimo da je u registru R_1 upisana vrednost promenljive X i neka ona iznosi 1. Neka procesori P_1 i P_2 istovremeno pristupaju registru R_1 sa zadatkom uvećanja vrednosti promenljive za 1. Vrednost koja treba da bude upisana u registar R_1 nakon uvećanja od strane oba procesora je 3. Međutim, u praksi je moguć sledeći niz akcija:

1. Procesor P_1 čita vrednost iz registra R_1 koja iznosi 1.

2. Procesor P_2 čita vrednost iz registra R_1 koja iznosi 1.

3. Procesor P_1 uvećava vrednost registra R_1 za 1 i tu vrednost upisuje u registar. Tada je u registar R_1 upisana vrednost 2.

4. Pošto je procesor P_2, u trenutku označenim rednim brojem 2, pročitao vrednost 1 iz registra R_1, on tu vrednost uvećava za 1 i (umesto vrednosti 3) upisuje u registar R_1 vrednost 2.

Prethodni primer ukazuje na to da se prilikom konstrukcije paralelnih algoritama posebna pažnja mora obratiti na deljene promenljive (na promenljive kojima različiti procesori mogu pristupati u isto vreme).

Modele sa zajedničkom memorijom razlikujemo po tome kako obrađuju memorijske konflikte. Razlikujemo naredne modele: *EREW*, *CREW* i *CRCW*.

Model *EREW* (*Exclusive-Read Exclusive-Write*) se karakteriše ograničenjem da je u jednom vremenskom trenutku moguće da samo jedan procesor pristupa zajedničkoj memorijskoj lokaciji - da iz nje čita ili da u nju upisuje neki sadržaj.

Model *CREW* (*Concurent-Read Exclusive-Write*) dozvoljava da više procesora istovremeno čitaju sadržaj sa iste memorijske lokacije, ali ne dozvoljava da dva procesora istovremeno pišu na istu lokaciju.

Kao najslabiji model, po pitanju restrikcija, izdvajamo *CRCW* model. Model *CRCW* (*Concurent-Read Concurent-Write*) ne zadaje nikakva ograničenja za pristup procesora memoriji. On se karakteriše time što dozvoljava da više procesora istovremeno pišu na istu lokaciju samo ako zapisuju istu vrednost.

Postoji nekoliko metoda za obradu istovremenih operacija pisanja od strane više procesora. Kada dva procesora pokušaju da istovremeno upišu različit sadržaj na istu memorijsku lokaciju, takvo ponašanje se smatra nevalidnim i zaustavlja izvršavanje algoritma. Jedan od načina za prevazilaženje ovog problema je dodeljivanje prioriteta procesorima. U ovom slučaju, kada više procesora zahteva istovremeni pristup zajedničkoj memoriji, dozvoljava se pristup procesoru sa najvišim rednim brojem ili prioritetom. Drugi način za rešavanje istog problema je korišćenje mehanizma katanca (eng. *locks*) ili semafora (eng. *semaphore*). Ovaj pristup obavezuje procesore da zahtevaju i čekaju na dozvolu za pristup memoriji, čime se sprečavaju istovremeni pristupi i konflikti. Kada jedan procesor pristupi memoriji, drugi procesori će morati da sačekaju dok se resurs ne oslobodi. Ovaj pristup obezbeđuje bezbedan pristup zajedničkoj memoriji, ali može dovesti do potencijalnih problema kao što je na primer uzajamno blokiranje.

Glava 3

Sekvencijalni algoritmi za bojenje grafova

U ovom poglavlju ćemo razmotriti nekoliko primera sekvencijalih heuristika za rešavanje problema bojenja grafova. U radu neće biti razmatrani algoritmi za određivanje bojenja koje koristi minimalni broj broja boja, nego heuristike koje u praksi daju dobre rezultate. U nastavku ćemo pretpostaviti da je dat neusmereni graf $G = (V, E)$, gde je $V = \{v_1, v_2, \ldots, v_n\}$, $E = \{e_1, e_2, \ldots, e_m\}$, $n = |V|$ i $m = |E|$.

Kažemo da je kvalitet bojenja grafa *zadovoljajući* ako se za bojenje koristi broj boja približan broju boja koje koristi optimalno bojenje grafa. Prethodno navedena definicija zadovoljavajućeg kvaliteta bojenja grafa, iako uvedena neformalno, vrlo je intuitivna. Jedan od najboljih načina za merenje kvaliteta bojenja jeste korišćenje metrike definisane sa $\psi(G, c) = |Colors| - \chi(G)$, gde je c bojenje grafa G čiji se kvalitet meri, a $Colors$ skup boja koje koristi bojenje c. Međutim, problem određivanja njene vrednosti je težak problem, s obzirom na to da je određivanje vrednosti $\chi(G)$ NP-kompletan problem.

Neka je dat skup boja C koje smo iskoristili za bojenje čvorova grafa. Tada boje iz skupa C možemo numerisati brojevima. Definišemo *indeks boje* $c_i \in C$ kao njen redni broj (indeks boje c_i je i). Obično numeraciju boja određuje redosled dodeljivanja boja čvorovima - prvododeljena boja ima najniži indeks, dok boja koja je poslednja dodeljena ima najviši indeks.

3.1 Heuristike za bojenje jednog čvora

Pretpostavimo da struktura podataka koja predstavlja čvor grafa sadrži polje *Color* koje u sebi čuva boju čvora. Neka je inicijalna vrednost tog polja postavljena na neku nevalidnu vrednost. Tu vrednost označićemo sa *empty* - ona će služiti kao indikator da čvor nije obojen, tj. da mu je potrebno dodeliti boju. Pretpostavimo, radi jednostavnosti, da su boje čvorova predstavljene prirodnim brojevima - tj. da je vrednost polja *Color* strukture podataka koja predstavlja čvor grafa celobrojnog tipa. Kažemo da je boja c *zabranjena* za čvor v ako je neki njegov sused već obojen tom bojom.

U nastavku je data heuritika za bojenje čvora v bojom sa minimalnom vrednošću indeksa tako da bojenje bude ispravno. Niz $colorMask$ se koristi za čuvanje informacija o zabranjenim bojama koje se odnose na čvor v. Ukoliko važi $colorMask[c] = false$ onda je boja c zabranjena za čvor v. U suprotnom važi da je $colorMask[c] = true$, što označava da se čvor v može obojiti bojom c. Niz $colorMask$ je na početku heuristike inicijalizovan tako da sadrži $|V|$ elemenata, gde je inicijalna vrednost svih elemanata postavljena na $true$.

Heuristika 1 Heuristika za određivanje boje najmanjeg indeksa čvora v

Naziv heuristike: *CalculateNodeColor(v, G)*
Ulaz: Neusmeren graf $G = (V, E)$, čvor $v \in V$

1: **for** *each* $w \in Neighbours(v, G)$ **do**
2: $\quad colorMask[w.Color] = false$
3: **end for**
4: $c = min\{i > 0 \mid colorMask[i] \neq false\}$
5: $v.Color = c$

3.2 Pohlepni algoritmi

Pohlepni algoritmi (eng. *greedy algorithms*) se karakterišu time što se u svakom koraku pronalazi neko lokalno optimalno rešenje problema. Na osnovu dobijenih lokalnih rešenja se gradi krajnji rezultat - globalno optimalno rešenje problema. Konstrukciju ovakvih algoritama treba razmotriti samo u slučajevima kada postoji garancija da će pronalaženje lokalnog optimalnog rešenja na kraju dovesti do globalno optimalnog resenja problema. U nastavku će biti opisane naredne pohlepne heuristike: *First Fit*, *LDO*, *SDO* i *IDO*.

3.2.1 First Fit heuristika

First Fit heuristika je najjednostavnija i najbrža sekvencijalna heuristika za bojenje grafa, koja za bojenje grafa najčeše koristi veliki broj boja. Ona pretpostavlja da je fiksiran redosled kojim će čvorovi biti posećivani; neka je to redosled $(v_1, v_2, ..., v_n)$. Neka su boje čvorova indeksirane brojevima: 1, 2, 3... Heuristika prolazi redom kroz čvorove u zadatom redosledu i svakom čvoru dodeljuje najmanju po indeksu boju kojom se on može obojiti, a da pritom bojenje bude ispravno.

Heuristika 2 *First Fit* Heuristika bojenja

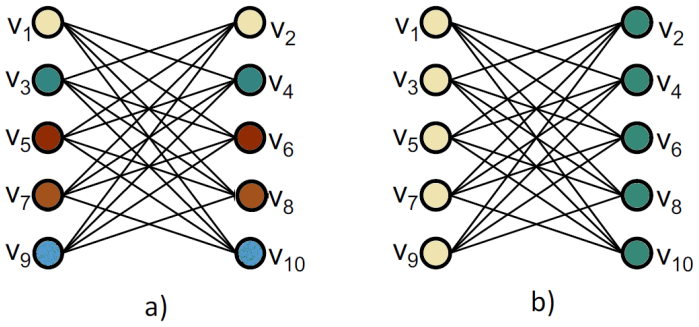
Naziv heuristike: *FirstFit(G)*
Ulaz: Neusmeren graf $G = (V, E)$

```
1: for i = 1 to n do
2:     CalculateNodeColor(v_i, G)
3: end for
```

Iako je heuristika dosta jednostavana i jako brza, kvalitet bojenja u određenim situacijama može biti jako loš. Na slici 3.1 su prikazana dva bojenja istog bipartitnog grafa realizovana pomoću *First Fit* heuristike. U delu pod a) dato je bojenje koje je određeno heuristikom *First Fit* prateći redosled čvorova: $v_1, v_2, \ldots, v_{10}$. Primetimo da je ovo bojenje dosta lošeg kvaliteta - graf je obojen sa pet boja. Na istoj slici, u delu pod b), prikazano je bojenje određeno *First Fit* heuristikom prateći redosled čvorova: $v_1, v_3, v_5, v_7, v_9, v_2, v_4, v_6, v_8, v_{10}$. Ovo bojenje jeste jedno optimalno bojenje datog grafa - iskorišćene su samo dve boje.

Upravo su bipartitni grafovi visokog maksimalnog stepena $\Delta(G)$ primer grafova kod kojih *First Fit* heuristika može dati bojenje lošeg kvaliteta. Najbolji primer za to su bipartitni grafovi kod kojih važi: $G = (U, V, E)$, $U = \{v_1, v_3, \ldots, v_{n-1}\}$, $V = \{v_2, v_4, \ldots, v_n\}$ i kod kojih je čvor $v_i \in U$ povezan sa svim čvorovima iz skupa V sem sa čvorom v_{i+1}, a n je paran broj. U najgorem slučaju ova heuristika će obojiti takav graf sa $\frac{n}{2}$ boja umesto sa dve boje. Taj scenario je za $n = 10$ ilustrovan na slici 3.1, gde je $n = 10$.

U proseku *First Fit* heuristika za kratko vreme daje bojenje zadovoljavajućeg kvaliteta. U radu [6] je dokazano da u slučaju slučajnih grafova očekivani broj boja koji *First Fit* heuristika iskoristi za bojenje nije veći od $2\chi(G)$. U i-tom koraku heuristika obavi $deg(v_i)$ operacija, pa je njena ukupna složenost $\sum_{i=1}^{n} deg(v_i) = O(m)$.



Slika 3.1: Primer različitih bojenja istog grafa dobijenih pomoću *First Fit* heuristike praćenjem različitih redosleda čvorova

3.2.2 Opšta pohlepna heuristika

Pohlepne heuristike karakteriše korišćenje strategije za izbor čvora iz skupa neobojenih čvorova. Odgovarajuću strategiju zovemo *strategija izbora*. Izabranom čvoru se tada dodeljuje boja tako da bojenje bude ispravno. Uz odgovarajuće odabranu strategiju izbora, u opštem slučaju možemo dobiti bolje rezultate nego u slučaju kada koristimo *First Fit* heuristiku.

Heuristika 3 Opšta pohlepna heuristika

Naziv heuristike: *GreedyColoring(G)*
Ulaz: Neusmeren graf $G = (V, E)$

1: $U = V$
2: **while** $U \neq \emptyset$ **do**
3: Izabrati čvor $v \in U$ koristeći odgovarajuću strategiju izbora
4: $CalculateNodeColor(v, G)$
5: $U = U \setminus \{v\}$
6: **end while**

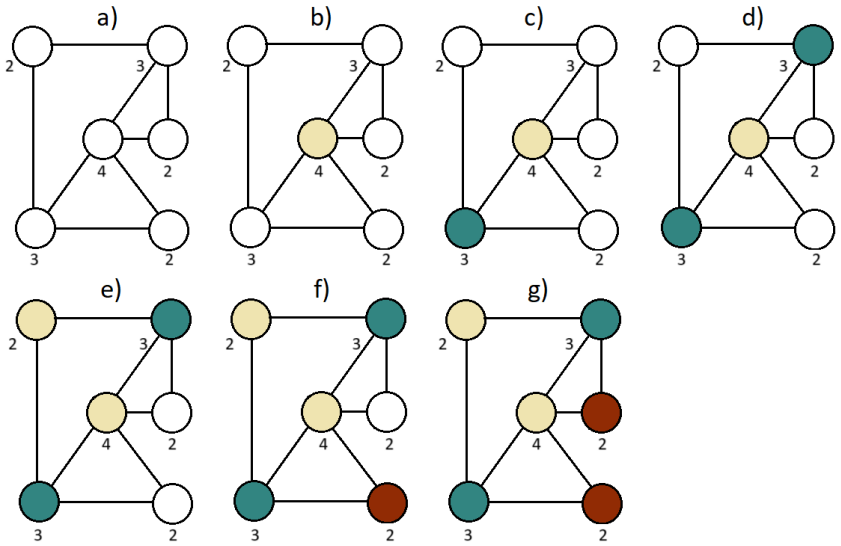
Za grafove sa maksimalnim stepenom Δ možemo dati gornju granicu broja boja koje heuristika koristi za bojenje. Naime, u svakoj iteraciji `while` petlje naredni čvor koji treba obojiti ima najviše Δ suseda te ga sigurno možemo obojiti bojom koja ima jedan od indeksa iz skupa: $\{1, 2, \ldots, \Delta + 1\}$. Stoga, ovakve heuristike koriste najviše $\Delta+1$ boja za bojenje grafa. Postoje razne strategije za izbor narednog čvora. U zavisnosti od strategije koju izaberemo možemo dobiti, u opštem slučaju, manje ili više efikasne heuristike za bojenje. U nastavku će biti opisane razne strategije izbora narednog čvora.

Pretpostavimo da su čvorovi $v_1, v_2, \ldots, v_{i-1}$ obojeni u prethodnoj fazi naše pohlepne heuristike. Razmotrimo na koji način možemo odabrati naredni čvor i dobiti odgovarajuće nove heuristike.

3.2.3 Sortiranje po stepenu opadajuće

Sortiranje po stepenu opadajuće (eng. *Largest Degree Ordering* (LDO)) je jedan od prvo predstavljenih kriterijuma izbora čvorova. Naredni čvor v_i biramo tako da ima najveći stepen među svim neobojenim čvorovima. Ovo znači da je pre početka bojenja redosled u kome će čvorovi biti bojeni u potpunosti određen - sortiranjem niza čvorova po njihovom stepenu opadajuće. Nakon određivanja redosleda bojenja čvorova, tekući čvor v_i bojimo tako što mu dodeljujemo boju najmanjeg indeksa tako da bojenje bude ispravno. Ova strategija daje bolje rezultate u smislu kvaliteta bojenja nego *First Fit* heuristika. Primetimo da ovu heuristiku možemo implementirati tako da ima složenost $O(m)$.

Na slici 3.2 je ilustrovan rad prethodno opisane heuristike po iteracijama. Na svakom grafu je pored čvora prikazan broj koji predstavlja njegov stepen. Na istoj slici, pod a) je prikazan neobojeni graf. U svakoj narednoj iteraciji (na istoj slici u delovima pod b), c), d), e), f) i g)) pored prethodno obojenih čvorova je obojen još jedan čvor - onaj koji je među neobojenim čvorovima na kraju prethodne iteracije imao maksimalni stepen.



Slika 3.2: Primer bojenja grafa pomoću *LDO* heuristike

Heuristika 4 LDO heuristika
Naziv heuristike: *LDO(G)*
Ulaz: Neusmeren graf $G = (V, E)$

```
1: A = ToArray(V)
2: PerformDescendingSortByDegree(A)

3: for i = 1 to n do
4:     v = A[i]
5:     CalculateNodeColor(v, G)
6: end for
```

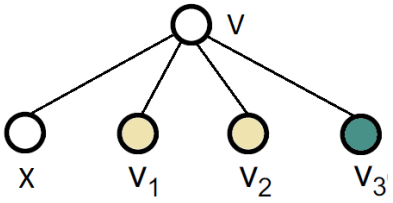
U ovakvoj implementaciji *LDO* heuristike (Heuristika 4) pomoćna funkcija $ToArray(V)$ skup čvorova V pretvara u niz čvorova A kome možemo lako da pristupimo na osnovu indeksa i. Pomoćna funkcija $PerformDescendingSortByDegree(A)$ sortira niz čvorova A opadajuće po stepenu.

3.2.4 Sortiranje po stepenu zasićenja

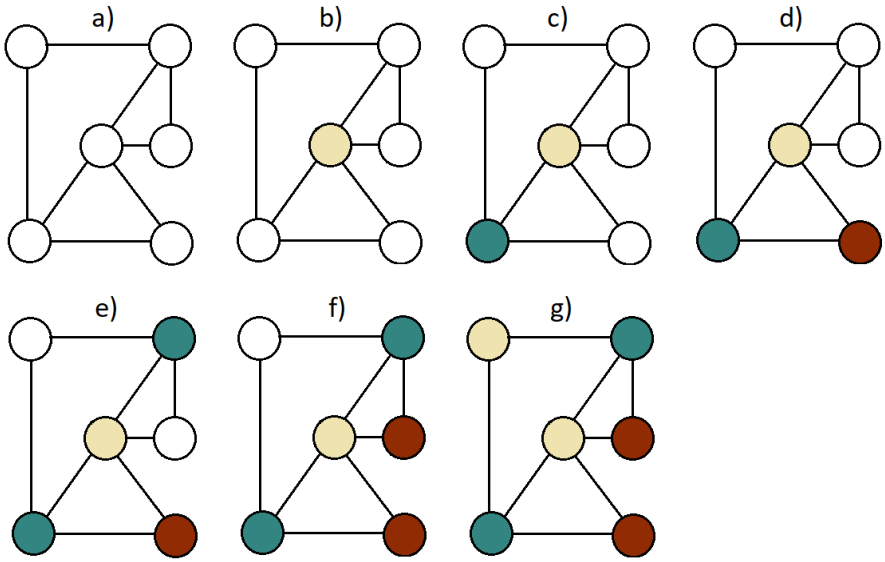
Sortiranje po stepenu zasićenja (eng. *Saturation Degree Ordering* (SDO)) je heuristika nastala sa namerom da se obezbedi drugačiji kriterijum izbora u odnosu na onaj koji koristi *LDO* heuristika tako da dobijemo kvalitetnije bojenje u odnosu na *LDO* heuristiku. Definišemo *stepen zasićenja* (eng. *saturation degree*) čvora kao broj različito obojenih čvorova koji su susedni čvoru v_i. U i-tom koraku heuristike čvor v_i biramo kao čvor maksimalnog stepena zasićenja i čvoru v_i dodeljujemo boju minimalnog indeksa tako da bojenje bude ispravno. Na slici 3.3 je ilustrovan primer čvora v koji ima stepen zasićenja dva. Njegov sused sa oznakom x nije obojen, susedi v_1 i v_2 su obojeni žutom bojom, dok je njegov susedni čvor v_3 obojen zelenom bojom - susedi čvora v su obojeni u dve različite boje pa stepen zasićenja čvora v iznosi dva. Ova heuristika pruža kvalitetnije bojenje od prethodno opisane *LDO* heuristike, s obzirom na to da prvo boji one čvorove koji među svojim susedima imaju najviše prethodno obojenih čvorova koji su obojeni različitim bojama. *LDO* heuristika nakon sortiranja čvorova po stepenu opadajuće obrađuje čvorove u fiksnom redosledu, što može odvesti ka bojenju lošijeg kvaliteta jer se prilikom odlučivanja koji će čvor sledeći biti obojen u razmatranje ne uzimaju čvorovi koji su trenutno obojeni. *SDO* heuristika prevazilazi problem obrade u fiksnom redosledu. Naredni čvor koji treba obojiti se bira dinamički, na osnovu rezultata parcijalnog bojenja koje smo dobili u prethodnoj iteraciji (u prethodnom koraku heuristike). Ova heuristika se može

implementirati tako da ima složenost $O(n^2)$.

Na slici 3.4 je ilustrovan primer bojenja grafa pomoću *SDO* heuristike. U delu pod a) je prikazan neobojeni graf. U delovima pod b), c), d), e), f) i g) su, redom, ilustrovani koraci bojenja grafa - iteracije heuristike. Iako je kvalitet bojenja isti kao i u primeru bojenja pomoću *LDO* heuristike sa slike 3.2, primetna je razlika u redosledu bojenja čvorova.



Slika 3.3: Primer grafa čiji čvor v ima stepen zasićenja 2 i stepen incidencije 3



Slika 3.4: Primer bojenja grafa pomoću *SDO* heuristike

U pseudokodu *SDO* heuristike (Heuristika 5), koji je dat u nastavku, čvor $InvalidNode \notin V$ predstavlja inicijalnu vrednost promenljive *maxSaturationDegreeNode*. *saturationDegreeByNode* predstavlja mapu u kojoj su kljucevi čvorovi, a njihove vrednosti u mapi su odgovarajući stepeni zasićenja. Ovu mapu koristimo kako bismo u svakoj iteraciji spoljne `while` petlje pronašli čvor sa najvećim stepenom zasićenja. Funkcija *ToZeroValueMap(U)*, koja za ulaz ima skup čvorova U, kreira

mapu u kojoj će ključevi biti čvorovi skupa U, dok inicijalna vrednost svakog čvora u mapi iznosi 0. Ovu funkciju koristimo kako bismo mapi *saturationDegreeByNode* dodelili inicijalnu vrednost i na taj nacin postavili inicijalni stepen zasićenja svih čvorova na vrednost 0. Nakon što je u spoljnoj iteraciji `while` petlje određen i obojen čvor sa maksimalnim stepenom zasićenja - *maxSaturationDegreeNode*, potrebno je ažurirati stepene zasićenja svih njegovih suseda u grafu G. Drugim rečima, potrebno je ažurirati *saturationDegreeByNode* mapu tako što će se za svaki sused čvora *maxIncidencyDegreeNode* izračunati novi stepen zasićenja (kao broj različito obojenih čvorova - suseda) i informacija upisati u *saturationDegreeByNode* mapu. Ovaj posao obavlja `foreach` petlja u liniji pod redim brojem 14 SDO heuristike (Heuristika 5). Pomoćna funkcija $Colors(X)$ kao ulaz ima skup čvorova X a na izlazu daje skup boja kojima su obojeni ovi čvorovi skupa X.

Heuristika 5 SDO heuristika
Naziv heuristike: *SDO(G)*
Ulaz: Neusmeren graf $G = (V, E)$

1: $U = V$
2: $saturationDegreeByNode$ = ToZeroValueMap(U)

3: **while** $U \neq \emptyset$ **do**
4: $maxSaturationDegree = -1$
5: $maxSaturationDegreeNode = InvalidNode$

6: **for** each v in U **do**
7: $saturationDegree = maxSaturationDegreeNode[v]$
8: **if** $saturatonDegree > maxSaturationDegree$ **then**
9: $maxSaturationDegreeNode = v$
10: $maxSaturationDegree = saturationDegree$
11: **end if**
12: **end for**

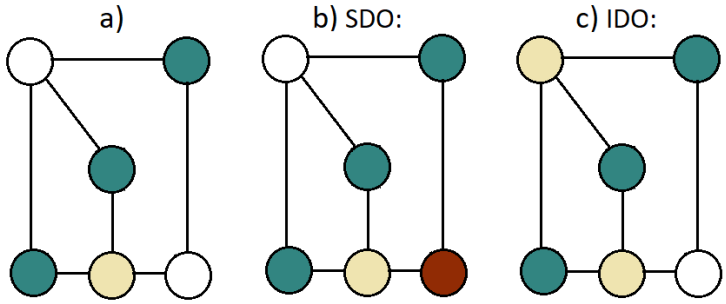
13: $CalculateNodeColor(maxSaturationDegreeNode, G)$
14: **for** each v in $Neighbours(maxIncidencyDegreeNode, G)$ **do**
15: $neighbourColors = \bigcup_{i=1}^{deg(v)} Colors(Neighbours(v_i, G))$
16: $saturationDegreeByNode[v] = |neighbourColors \setminus \{emptyColor\}|$
17: **end for**

18: $U = U \setminus \{maxSaturationDegreeNode\}$
19: **end while**

3.2.5 Sortiranje po stepenu incidencije

Sortiranje po stepenu incidencije (eng. *Incidence Degree Ordering* (IDO)) predstavlja modifikaciju prethodno navedene heuristike. Definišemo *stepen incidencije* čvora kao broj njegovih suseda koji su obojeni. U i-toj iteraciji petlje, za čvor v_i biramo onaj čvor čiji je stepen maksimalan u podgrafu grafa G indukovanim skupom čvorova $\{v_1, v_2, \ldots, v_{i-1}\} \cup \{v_i\}$. Drugim rečima, u i-tom koraku heuristike se bira čvor v_i sa najvećim stepenom incidencije među ostalim, neobojenim čvorovima. Na slici 3.3 čvor v ima stepen incidencije tri, s obzirom na to da ima tri obojena suseda - v_1, v_2 i v_3, i jednog suseda (čvor x) koji nije obojen. Ova heuristika se može implementirati u vremenu $O(m)$ - linearno po broju grana grafa G.

Slika 3.5 prikazuje kako se razlikuje proces izbora čvora za bojenje u trenutnoj iteraciji *SDO* heuristike i *IDO* heuristike. U delu pod a) je prikazan parcijalno obojen graf. Na istoj slici, u delu pod b), je prikazan rezultat rada jedne iteracije *SDO* heuristike. Na istoj slici, u delu pod c), je prikazan i rezultat rada jedne iteracije *IDO* heuristike. Primetimo da je, u ovom slučaju, *SDO* heuristika odabrala drugačiji čvor za bojenje u odnosu na onaj koji je odabrala *IDO* heuristika.



Slika 3.5: Ilustracija razlike između *SDO* heuristike i *IDO* heuristike

U praksi se pokazalo da SDO heuristika gotovo uvek daje bolji kvalitet bojenja u odnosu na IDO heuristiku, dok IDO heuristika ima kraće vreme izvršavanja u odnosu na SDO heuristiku. Heuristike koje koriste svojstva stepena čvorova u grafu su najčešće korišćene sekvencijalne heuristike za bojenje grafova. Razlog za to su njihova jednostavnost i relativno dobre performanse. U nastavku su dati pseudokodovi za dve prethodno predstavljene heuristike bojenja.

Implementacija *IDO* heuristike (Heuristika 6) je vrlo slična implementaciji *SDO* heuristike. Suštinska razlika je u tome što se umesto stepena zasićenja u svakoj iteraciji spoljne `while` petlje (pomoću mape *incidencyDegreeByNode*) pronalazi čvor

maxIncidencyDegreeNode sa maksimalnim stepenom incidencije. Nakon što je u tekućoj iteraciji spoljne `while` petlje određen i obojen čvor *maxIncidencyDegreeNode*, potrebno je ažuritati vrednosti stepena incidencije svih njegovih suseda. Drugim rečima, potrebno je uvećati za 1 stare vrednosti prethodno pomenutih čvorova-suseda u mapi *incidencyDegreeByNode*.

Heuristika 6 IDO heuristika
Naziv heuristike: *IDO(G)*
Ulaz: Neusmeren graf $G = (V, E)$

1: $U = V$
2: $incidencyDegreeByNode = ToZeroValueMap(V)$

3: **while** $U \neq \emptyset$ **do**
4: $maxIncidencyDegree = -1$
5: $maxIncidencyDegreeNode = InvalidNode$

6: **for** each v in U **do**
7: $incidencyDegree = incidencyDegreeByNode[v]$
8: **if** $incidencyDegree > maxIncidencyDegree$ **then**
9: $maxIncidencyDegreeNode = v$
10: $maxIncidencyDegree = incidencyDegree$
11: **end if**
12: **end for**

13: $CalculateNodeColor(maxIncidencyDegreeNode, G)$

14: **for** each v in $Neighbours(maxIncidencyDegreeNode, G)$ **do**
15: $incidencyDegreeByNode[v]{+}{+}$
16: **end for**

17: $U{=}U \setminus \{maxIncidencyDegreeNode\}$
18: **end while**

3.3 Heuristike bazirane na odabiru nezavisnih skupova čvorova

Glavna ideja kojom se vodimo pri konstrukciji heuristika ovog tipa je da na dobar način odaberemo nezavisne skupove $I_1, I_2, \ldots, I_k$ čvorova datog grafa. Kada odaberemo takve skupove koristićemo činjenicu da se svi čvorovi u okviru neza-

visnog skupa I_j mogu obojiti istom bojom, s obzirom na to da između čvorova u datom skupu ne postoji grana. Pošto su svi skupovi I_j, gde je $1 \leq j \leq k$, međusobno nezavisni, za bojenje grafa će se iskoristiti k boja.

Od suštinske važnosti često je nalaženje *maksimalnog nezavisnog skupa* (eng. *maximal independent set*) u grafu. Algoritmi za nalaženje ovakvih skupova su najčešće osnova za algoritme bojenja koji se baziraju na odabiru nezavisnih skupova. Dokazano je da je problem nalaženja maksimalnog nezavisnog skupa u grafu NP-kompletan problem. Stoga, u slučaju algoritama za bojenje koji se baziraju na odabiru nezavisnih skupova ćemo morati da koristimo razne heuristike za određivanje nezavisnih skupova kako bismo se osigurali da algoritam ima zadovoljavajuće performanse. Veoma često se za nalaženje maksimalnog nezavisnog skupa u grafu koriste pohlepne heuristike. Pohlepna heuristika postepeno, u iteracijama, gradi rezultat tako što u svakoj iteraciji (koristeći precizno definisan kriterijum izbora) odabere jedan čvor, proglasi ga za favorita i unira ga sa prethodno odabranim čvorovima. Opšta pohlepna heuristika za nalaženje maksimalnog nezavisnog skupa u grafu je data u nastavku.

U pseudokodu heuristike *GreedyMIS(G)* (Heuristika 7), skup U predstavlja skup čvorova koji se trenutno razmatraju kao kandidati za čvorove koje će sadržati rezultujući nezavisni skup čvorova I. $G' = G[U]$ predstavlja podgraf grafa G koji je indukovan čvorovima iz skupa U. Petlja će iterirati sve dok je graf G' neprazan, odnosno, dok je skup čvorova-kandidata U neprazan. Funkcija $Neighbours(v, G)$ izračunava skup čvorova koji predstavljaju susede čvora v u grafu G, koji ne mogu pripadati skupu I. Skup X je pomoćni skup koji koristimo za ažuriranje skupa U.

Postoje primeri grafova za koje heuristika *GreedyMIS(G)* radi sporo. Međutim, u prosečnom slučaju heuristika pokazuje dobre rezultate. Ovu heuristiku (upotrebom reda sa prioritetom) možemo implementirati tako da ima složenost $O(|V| \cdot \log |V|)$. Ispostavlja se da je heuristika jako koristna u slučaju grafova sa malim maksimalnim stepenom Δ. Ako je maksimalni stepen čvora u grafu Δ, onda će svaki korak heuristike smanjiti graf za ne više od $\Delta + 1$ čvorova. Stoga će ovaj metod pronaći nezavisan skup veličine bar $\lceil \frac{n}{\Delta+1} \rceil$.

Jedan od mogućih načina za poboljšanje ove heuristike je strategija biranja takozvanih *nezavisnih skupova minimalnog stepena* (eng. *independent sets with minimal degree*). Ona podrazumeva da se, prilikom odabira čvorova za obradu, prednost daje onim čvorovima grafa koji imaju najmanji stepen. U svakoj iteraciji `while` petlje uzimamo čvor najmanjeg stepena koji je nezavisan od ostalih čvorova skupa I i do-

dajemo ga u skup I. Motivacija za primenu ove strategije leži u činjenici da odabir čvorova niskog stepena minimizuje skup X i samim tim smanjuje broj čvorova koje nećemo razmatrati kao potencijalne članove nezavisnog skupa I.

Heuristika 7 Pohlepna heuristika za nalaženje maksimalnog nezavisnog skupa u grafu

Naziv heuristike: *GreedyMIS(G)*
Ulaz: Neusmeren graf $G = (V, E)$
Izlaz: Maksimalni nezavisni skup čvorova I u grafu G

1: $I = \emptyset$
2: $U = V$
3: $G' = G$
4: **while** $G' \neq \emptyset$ **do**
5: Izaberi čvor v iz skupa U
6: $I = I \cup \{v\}$
7: $X = \{v\} \cup Neighbours(v, G')$
8: $U = U \setminus X$
9: $G' = G[U]$
10: **end while**
11: **return** I

Pretpostavimo da je data heuristika za nalaženje jednog nezavisnog skupa u grafu G i neka njega izračunava funkcija *GetIndependentSet(G)*. U nastavku je data heuristika bojenja bazirana na odabiru nezavisnih skupova grafa.

Heuristika 8 Heuristika za bojenje bazirana na odabiru nezavisnih skupova grafa

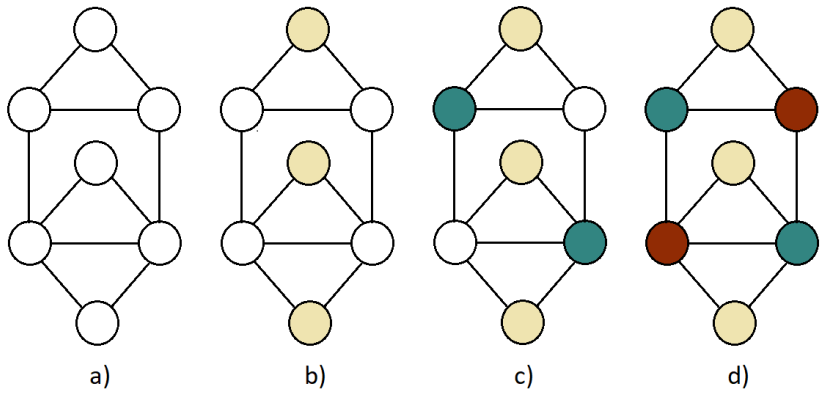
Naziv heuristike: *ColoringByMIS(G)*
Ulaz: Neusmeren graf $G = (V, E)$

1: $U = V$
2: $G' = G$
3: **while** $G' \neq \emptyset$ **do**
4: $I = GetIndependentSet(G')$
5: Obojiti sve čvorove skupa I novoodabranom bojom
6: $U = U \setminus I$
7: $G' = G[U]$
8: **end while**

U svakoj iteraciji `while` petlje generiše se po jedan nezavisni skup čvorova I u odnosu na graf G' (čija je inicijalna vrednost ceo graf G). Svi čvorovi skupa I se boje istom bojom nakon čega se oni izbacuju iz skupa U i u odnosu na skup U se indukuje novi graf G' za narednu iteraciju petlje.

Primetimo da performanse navedene heuristike direktno zavise od heuristike za odabir nezavisnih skupova. Od nje zavise i kvalitet dobijenog bojenja grafa i vremenska složenost konačne heuristike za bojenje.

Na slici 3.6 je ilustrovan primer bojenja grafa pomoću prethodno opisane heuristike po iteracijama `while` petlje. U delu pod a) je dat primer neobojenog grafa. U ovom primeru heuristika u svakoj iteraciji `while` petlje pronalazi maksimalni nezavisni skup čvorova među neobojenim čvorovima datog grafa i boji ga. U delovima pod b), c), d) su ilustrovani rezultati rada prve, druge i treće iteracije `while` petlje.



Slika 3.6: Primer bojenja grafa pomoću heuristike zasnovane na odabiru nezavisnih skupova

3.4 Heuristike lokalnog poboljšanja bojenja

Heuristike lokalnog poboljšanja bojenja predstavljaju nov i potpuno drugačiji pristup bojenju grafa u odnosu na prethodno opisane heuristike bojenja. Prvi korak kod ovakvih heuristika je naivno bojenje grafa. Nazovimo ovaj korak inicijalnim bojenjem grafa. Nakon inicijalnog bojenja grafa čvorove možemo prirodno podeliti po grupama: i-toj grupi pripadaju čvorovi koji su obojeni bojom indeksa i. Grupe su organizovane tako da se nakon čvorova grupe sa rednim brojem i nižu čvorovi grupe sa rednim brojem $i+1$. Nakon toga se, u iteracijama, čvorovi premeštaju po grupama u nadi da će se tako postići kvalitetnije bojenje grafa. Za ovakve heuristike potrebno je obezbediti, kao uslov izlaska iz petlje, *meru (funkciju)* koja će nam reći da li je moguće da heuristika obezbedi bolje bojenje - tj. da li je moguć progres. Ukoliko ocenimo da nije moguće obezbediti bolje bojenje grafa, heuristika prekida sa radom.

Na slici 3.7 možemo videti četiri grupe čvorova formirane po bojama koje su dodeljene čvorovima. U grupi G_i sadržani su čvorovi kojima je dodeljena boja indeksa i, gde je $1 \leq i \leq 4$.

Slika 3.7: Primer grupisanja čvorova u 4 grupe boja (od G_1 do G_4)

Za razliku od pohlepnih heuristika, čvor može promeniti svoju boju nekoliko puta u toku lokalnog poboljšanja, dok se kod pohlepnih heuristika boja čvora ne menja nakon njegove evaluacije.

Kod heuristika lokalnog poboljšanja od ključne je važnosti odrediti sledeće dve stvari: *funkciju cilja* f (eng. *objective function*) koja će na dobar način izmeriti kvalitet rešenja i reći nam da li postoji mogućnost za napredak u bojenju ili se napredak ne može ostvariti i *kriterijum izbora* pomoću koga se odlučuje koje čvorove treba razmeniti. Za razmenu čvorova kažemo da je dobra ako dovodi do progresa u redukciji broja boja koje se koriste.

U nastavku će biti data *Modifikovana First Fit heuristika* (eng. *Modified First Fit*) koja spada u klasu heuristika lokalnog poboljšanja.

3.4.1 Modifikovana First Fit heuristika

Neka su dati graf $G = (V, E)$ i permutacija π skupa $\{1, 2, \ldots, n\}$. Redosled obrade čvorova u koraku inicijalnog bojenja određuje permutacija π. U ovoj verziji heuristike *First Fit* uvodimo izmenu: redosled obrade čvorova u svakoj iteraciji petlje je određen prethodno dobijenim bojenjem. Informacije dobijene bojenjem u prethodnoj iteraciji se koriste za dobijanje novog, kvalitetnijeg bojenja.

Sledeća lema kaže da ukoliko je permutacija π takva da su u njoj indeksi čvorova koji odgovaraju istoobojenim čvorovima jedan do drugog, ukoliko primenimo First Fit heuristiku koja obrađuje čvorove u redosledu koje zadaje permutacija π onda ćemo dobiti bojenje koje je makar onoliko kvalitetno koliko i bojenje koje smo imali u prethodnoj iteraciji.

Lema 1. *Neka je* $C : V \to Colors$ *funkcija bojenja grafa* G *i neka važi da je* $k = |Colors|$. *Neka je* π *permutacija čvorova grafa* G *i neka važi da ako je* $C(v_{\pi(i)})$

$= C(v_{\pi(m)}) = c$, tada za sve j takve da je $i < j < m$ važi $C(v_{\pi(j)}) = c$, gde je $\pi_{(j)}$ j-ti element permutacije π. Ukoliko primenimo First Fit heuristiku na čvorove grafa G u redosledu permutacije π onda će heuristika First Fit konstruisati bojenje koje koristi maksimalno k boja.

Dokaz: Dokaz ćemo sprovesti koristeći princip matematičke indukcije. Tvrdimo da je za bojenje svih elemenata prvih i grupa obojenih čvorova potrebno najviše i boja.
Baza indukcije: za prvu obojenu klasu čvorova očigledno je dovoljna jedna boja za bojenje svih elemenata.
Induktivna pretpostavka: pretpostavimo da se prvih $i-1$ grupa može obojiti sa $i-1$ boja.
Korak indukcije: pretpostavimo suprotno - da je za bojenje nekog čvora i - te grupe potrebna boja indeksa $i+1$. Induktivna hipoteza kaže da je za bojenje čvorova od prve do grupe sa rednim brojem $i-1$ potrebno maksimalno $i-1$ boja. To znači da je on povezan sa nekim čvorom koji je obojen i -tom bojom. Ovo dalje implicira da je on obojen istom bojom kao i neki drugi čvor iz svoje klase - da je različite boje od boje čvorova svoje klase što je kontradikcija sa činjenicom da je C jedno validno bojenje grafa G.
□

Od interesa nam je da petlja sa što manje iteracija da što kvalitetnije bojenje. Potrebno je definisati meru progresa bojenja - potrebno je obezbediti indikator da se u i-toj iteraciji dogodio progres, tj. da se manifestovala promena bojenja u grafu. Neka je funkcija f, koja predstavlja meru progresa, definisana na sledeći način:

$$f(G, p_1, p_2, \ldots, p_k) = \begin{cases} true, & \text{ako je moguće ostvariti progres u bojenju grafa } G \\ false, & \text{u suprotnom} \end{cases}$$

Ova funkcija kao parametre ima neusmeren graf G i opcione parametre: $p_1, p_2, \ldots, p_k$ pomoću kojih se može ustanoviti da li je došlo do progresa u bojenju grafa.

Kao indikator se, na primer, može koristiti broj boja koje smo iskoristili nakon i-te iteracije petlje. Tada bi pokazatelj progresa bio različit broj iskorisćenih boja u i-toj i $(i+1)$-voj iteraciji petlje. Tada bi funkcija cilja f imala sledeći oblik:

$$f(G, previousCount) = \begin{cases} true, & \text{kada je } G.ColorCount \neq previousCount \\ false, & \text{kada je } G.ColorCount = previousCount \end{cases}$$

Problem u ovako definisanim indikatorom progresa je taj što su nekada potrebne hiljade (ili više) iteracija da bi se manifestovala promena broja boja.

Još jedan način da izmerimo progres je numerisanje boja celim brojevima i formiranje sume brojeva dodeljenih čvorovima. Tada kao meru progresa možemo koristiti prethodno definisanu sumu. Tada bi funkcija cilja f imala sledeći oblik:

$$f(G, previousSum) = \begin{cases} true, & \text{kada je } \sum_{i=1}^{n} v.Color < previousSum \\ false, & \text{kada je } \sum_{i=1}^{n} v.Color \geq previousSum \end{cases}$$

Parametar *previousSum* predstavlja sumu numeričkih vrednosti boja čvorova dobijenu u prethodnoj iteraciji bojenja. Ovim je dobro definisana jedna funkcija cilja za meru progresa, jer ukoliko se suma numeričkih vrednosti boja čvorova smanjuje onda je broj boja koji je iskorišćen za bojenje grafa manji ili su nekim čvorovima dodeljene boje manjeg indeksa u odnosu na boje koje su imali u prethodnoj iteraciji bojenja.

Heuristika 9 Modifikovana First Fit heuristika
Naziv heuristike: *ModifiedFirstFit(G,π)*
Ulaz: Neusmeren graf $G = (V, E)$, permutacija π skupa $\{1, 2, \ldots, n\}$

Izračunati parametre: $p_1, p_2, \ldots, p_k$

while $f(G, p_1, p_2, \ldots, p_k)$ **do**
1. Primeni *First Fit* heuristiku na graf G gde se čvorovi obrađuju u redosledu koji određuje permutacija π
2. Permutaciju π formirati grupisanjem čvorova prema njihovim bojama (unutar svake grupe očuvaj redosled čvorova koji odgovara prethodnom bojenju)
3. Formirati nove parametre $p_1, p_2, \ldots, p_k$ saglasno dobijenom bojenju grafa G

end while

U nekim slučajevima funkcija cilja može prevremeno detektovati da ne dolazi do progresa u bojenju. Ovo za posledicu ima bojenje lošijeg kvaliteta kao rezultat rada algoritma. Da bi se u takvim situacijama popravio kvalitet bojenja `while` petlja se može modifikovati tako da dozvoli dodatnih m iteracija nakon što funkcija cilja f detektuje da ne postoji progres u bojenju grafa, gde se broj m unosi kao parametar sa ulaza heuristike.

Glava 4

Paralelni algoritmi za bojenje grafova

Sekvencijalne heuristike opisane u okviru prethodne glave se karakterišu svojom jednostavnošću i mogućnošću lake implementacije. Pohlepne sekvencijalne heuristike u proseku obezbeđuju bojenje dobrog kvaliteta za relativno kratko vreme. Međutim, problem je u njihovoj asimptotskoj složenosti. Za konstrukciju velikog broja heuristika u nastavku će nam biti potrebna funkcija $CalculateNodeColor(v, G)$ (Heuristika 1), opisana u okviru prethodne glave koja je bila osnova većine prethodno opisanih sekvencijalnih heuristika. Navedena funkcija čvoru v grafa $G = (V, E)$ dodeljuje najmanju po indeksu boju tako da bojenje grafa G bude ispravno.

Kao što je prethodno pomenuto, ukoliko je problem NP-kompletan paralelizacijom se ne može prevesti u problem iz klase P, ali može u određenoj meri ubrzati izvršavanje algoritma na paralelnim arhitekturama računara. Poznato je da neki algoritmi nisu pogodni za paralelizaciju zbog svoje specifične konstrukcije. Za neke algoritme je izazovno postići paralelizaciju, dok je za druge to znatno lakše. Kada govorimo o pohlepnim sekvencijalnim heuristikama za bojenje, primetno je da njihova paralelizacija predstavlja poseban izazov, zbog načina upotrebe funkcije za bojenje čvora - $CalculateNodeColor(v, G)$. Kod ovakvih heuristika se (u svakoj iteraciji) prvo određuje najprioritetniji čvor za bojenje, nakon čega se on boji pomoću ove funkcije. Dakle, da bi naredni čvor v bio obojen moraju se sačekati rezultati bojenja svih čvorova koji imaju veći prioritet od čvora v. Ako bismo, umesto toga, želeli da postignemo paralelno bojenje grafa i sa tim ciljem pozivali funkciju $CalculateNodeColor(v, G)$ paralelno za svaki čvor, kao rezultat bismo najverovatnije dobili neispravno bojenje grafa. Rešavanje ovakvih problema predstavlja najizazovniji deo paralelizacije prethodno navedenih heuristika. U okviru ovog poglavlja nastojaćemo da neke sekvencijalne heuristike prilagodimo paralelnom načinu

izvršavanja. Takođe, u okviru ove glave biće predstavljene nove heuristike pogodne za paralelno bojenje grafova.

4.1 GM heuristika

Paralelnu heuristiku bojenja inspirisanu opštom pohlepnom heuristikom bojenja razvili su *Amanuel H. Gebremedhin* i *Fredrik Manne*. U literaturi ova heuristika se često spominje pod skraćenicom *GM heuristika* nastalom od prezimena autora. Heuristika se realizuje u dva paralelna koraka unutar glavne `while` petlje. Prvi korak obezbeđuje bojenje koje ne mora biti pravilno - kao rezultat rada ovog koraka moguće je dobiti bojenje kod koga su dva susedna čvora obojena istom bojom. Naredni korak podrazumeva detekciju nepravilnosti u bojenju - ukoliko zaključimo da postoje susedni čvorovi obojeni istom bojom onda se prelazi u narednu iteraciju petlje koja treba da otkloni pomenute anomalije. Petlja će iterirati sve dok bojenje ne bude pravilno.

Heuristika 10 Paralelna *GM* heuristika za bojenje grafa
Naziv heuristike: *ParallelGM(G)*
Ulaz: Neusmeren graf $G = (V, E)$

```
1: U = V
2: while U ≠ ∅ do
3:    for each v ∈ U in parallel do
4:       CalculateNodeColor(v)
5:    end for
6:    R = ∅
7:    for each v ∈ U in parallel do
8:       for each w ∈ Neighbours(v) do
9:          if color[v] == color[w] and v < w then
10:            R = R ∪ {v}
11:         end if
12:      end for
13:   end for
14:   U = R
15: end while
```

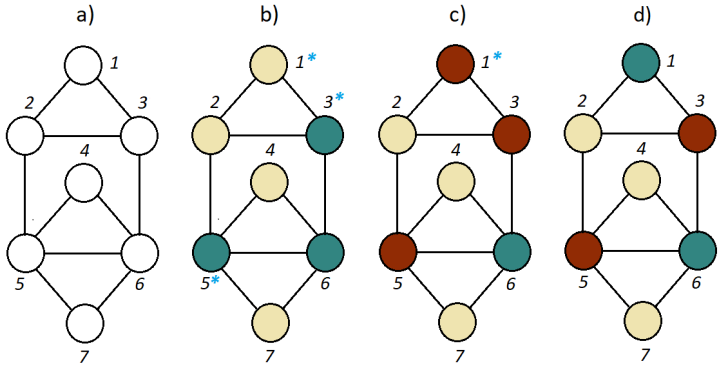
Skup U predstavlja skup čvorova koje je potrebno obraditi u trenutnoj iteraciji glavne `while` petlje. Njegova inicijalna vrednost je ceo skup čvorova V. Kao što je rečeno, heuristika se sastoji od dva glavna koraka prisutna u okviru spoljašnje `while`

petlje. Prvi korak podrazumeva bojenje svih neobrađenih čvorova paralelnim pozivanjem funkcije *CalculateNodeColor* za svaki čvor $v \in U$. Naredni korak heuristike detektuje konflikte u prethodnom bojenju. Ovaj korak određuje koji su to susedni čvorovi obojeni istom bojom u toku prethodnog koraka. Ukoliko se ispostavi da postoje dva susedna čvora obojena istom bojom - čvor manjeg indeksa se ubacuje u kolekciju čvorova za ponovno bojenje koja je označena sa R. Nakon završenog drugog koraka pomoćnom skupu čvorova U se dodeljuje vrednost R, nakon čega se prelazi na narednu iteraciju spoljašnje `while` petlje. Ova petlja iterira sve dok se ceo skup čvorova V ne oboji pravilno - sve dok je skup U neprazan. Razmotrimo uslov $v < w$ u liniji pod rednim brojem 9 algoritma. On nam garantuje da, pod uslovom da susedni čvorovi budu obojeni istom bojom, čvor višeg indeksa ima veći prioritet u procesu bojenja. Ovime je osigurano da će, ukoliko dođe do nepravilnosti u bojenju, kandidat za ponovno bojenje biti čvor nižeg indeksa i da će procesor koji odgovara čvoru v na osnovu toga odrediti kandidata za ponovno bojenje umesto da oba čvora (v i w) ubaci u skup R.

Razmotrimo zašto se heuristika zaustavlja i zašto je rezultat njenog rada pravilno bojenje grafa G. Posmatrajmo paralelnu petlju u liniji pod rednim brojem 3 heuristike. Ova petlja vrši paralelno pseudobojenje grafa, što znači da u opštem slučaju kao rezultat neće dati pravilno bojenje grafa G. U najgorem mogućem slučaju će sve čvorove obojiti istom bojom. Označimo tu boju (kojom su obojeni svi čvorovi) sa c. Nakon toga se detektuju konflikti u bojenju grafa i prelazi se u narednu iteraciju glavne `while` petlje. U narednoj iteraciji paralelna petlja u liniji pod rednim brojem 3 heuristike će čvorovima skupa R dodeliti nove boje. I u ovoj iteraciji je moguće da će svi čvorovi skupa R biti obojeni istom bojom. U najgorem slučaju će u svakoj narednoj iteraciji glavne `while` petlje svi čvorovi skupa R biti obojeni istom bojom. U tom slučaju se u svakoj narednoj iteraciji broj boja povećava za 1 sve dok bojenje ne bude pravilno.

Na slici 4.1 je ilustrovano bojenje grafa pomoću *GM* heuristike. U delu pod a) je prikazan neobojeni graf sa indeksiranim čvorovima. Na istoj slici, u delu pod b), je ilustrovan rezultat rada prve iteracije glavne `while` petlje. Primetimo da je u toku paralelnog pseudobojenja grafa nekim susednim čvorovima grafa dodeljena ista boja. Oni čvorovi koji su obojeni istom bojom kao neki njihov sused a imaju indeks manji od tog suseda su označeni zvezdicom i dodaju se u skup R. Na istoj slici, u delu pod c), je prikazan rezultat rada naredne iteracije glavne `while` petlje. Na kraju ove iteracije čvorovi sa indeksima 1 i 3 su obojeni istom bojom, pa čvor

sa indeksom 1, koji je označen zvezdicom, biva dodat u skup R. U delu pod d) je prikazan rezultat rada naredne iteracije glavne `while` petlje. U toku ove iteracije pseudobojenje skupa R je rezultovalo različito dodeljenim bojama svih susednih čvorova. Pošto je skup R na kraju ove iteracije prazan heuristika prestaje sa radom.



Slika 4.1: Primer bojenja grafa pomoću GM heuristike

4.2 Paralelne heuristike za bojenje zasnovani na odabiru nezavisnih skupova

4.2.1 Opšta paralelna heuristika za bojenje bazirana na odabiru nezavisnih skupova

Pretpostavimo da je data heuristika za nalaženje nezavisnog skupa u grafu $G = (V, E)$. Pretpostavimo, dalje, da smo pomoću date heuristike pronašli nezavisne skupove čvorova: $I_1, I_2, \ldots, I_k$ u grafu G i da važi da je $V = \bigsqcup_{j=1}^{k} I_j$. Tada imamo mogućnost da svaki od skupova I_j obradimo nezavisno i paralelno. Ovo postižemo pokretanjem k paralelnih procesa $P_1, P_2, \ldots, P_k$, pri čemu će proces P_j bojiti sve elemente skupa I_j istom bojom, odnosno bojom c_j (gde je $1 \leq j \leq k$). Pseudokod ove heuristike je dat u nastavku.

Heuristika 11 Opšta paralelna heuristika za bojenje grafova bazirana na odabiru nezavisnih skupova

Naziv heuristike: *ParallelColoringByMIS(G)*
Ulaz: Neusmeren graf $G = (V, E)$

1: Odrediti nezavisne skupove čvorova: $I_1, I_2, \ldots, I_k$ tako da važi: $V = \bigsqcup_{j=1}^{k} I_j$
2: **for** $j = 1$ to k in parallel **do**
3: $\quad ProcessNodeColoring(I_j, c_j)$
4: **end for**

Funkcija $ProcessNodeColoring(I_j, c_j)$ boji sve čvorove nezavisnog skupa I_j grafa $G = (V, E)$ istom bojom c_j. Ova funkcija može biti implementirana kao sekvencijalni ili kao paralelni algoritam. Njena implementacija u formi paralelnog algoritma je data u nastavku.

Heuristika 12 Paralelni algoritam za bojenje svih čvorova skupa I istom bojom c

Naziv algoritma: *ProcessNodeColoring(I, c)*
Ulaz: Podskup I skupa čvorova grafa $G = (V, E)$, boja c

1: **for** each $v \in I$ in parallel **do**
2: $\quad v.Color = c$
3: **end for**

Sekvencijalna verzija ovog algoritma je takođe jednostavna. Jedina razlika je u tome što bi se umesto paralelne `for` petlje koristila sekvencijalna `for` petlja. Razmotrimo sada vremensku složenost Heuristike 11. Ako je funkcija *ProcessNodeColoring*-(I_j, c_j) (Heuristika 12) implementirana kao paralelni algoritam, cela paralelna `for` petlja algoritma u liniji pod rednim brojem 3 se izvršava u vremenu $O(1)^*$. U slučaju da je funkcija $ProcessNodeColoring(I_j, c_j)$ implementirana kao sekvencijalni algoritam ona ima složenost $O(|I_j|)$, pa je u tom slučaju složenost `for` petlje unutar Heuristike 11 jednaka $O(\max_{1 \leq j \leq k} \{|I_j|\})$. S obzirom na težinu problema nalaženja maksimalnog nezavisnog skupa u grafu, ukupna složenost Heuristike 11 u najvećoj meri zavisi od heuristike koja generiše nezavisne skupove čvorova sa početka ove heuristike. U nastavku ćemo se baviti paralelnim heuristikama za nalaženje maksimalnih nezavisnih skupova u grafu koje možemo iskoristiti kao osnovu prethodno navedene heuristike.

*Ukoliko nam je na raspolaganju dovoljan broj procesora p.

4.2.2 Luby - MIS heuristika za nalaženje maksimalnog nezavisnog skupa u grafu

U nastavku je opisana heuristika konstrukcije nezavisnog skupa I koja se temelji na Monte Karlo strategiji. Monte Karlo strategija pronalaženja nezavisnog skupa I u neusmerenom grafu $G = (V, E)$ koju ćemo koristiti se temelji na sledećim pravilima:

1. Svakom čvoru skupa V treba dodeliti jedinstven slučajno generisani broj $\rho(v)$
2. $v \in I \iff (\forall w \in Neighbours(v))(\rho(v) > \rho(w))$

Broj $\rho(v)$ se može shvatiti kao prioritet čvora v - čvorovi sa većim prioritetom imaju veću šansu da budu ubačeni u nezavisni skup. Uslov 2 nam govori da čvor $v \in V$ može biti ubačen u nezavisan skup I samo ako ima veći prioritet od prioriteta svih njegovih suseda. Drugim rečima, opisana Monte Karlo strategija podrazumeva da se nakon dodeljivanja prioriteta svakom čvoru $v \in V$ skup I formira uniranjem onih čvorova kojima je dodeljen veći prioritet u odnosu na prioritet svih njihovih suseda. Na ovaj način dobija se nezavisni skup u grafu koji ne mora nužno biti maksimalni nezavisni skup.

Heuristika 13 Monte Karlo heuristika za nalaženje nezavisnog skupa I u grafu $G = (V, E)$

Naziv heuristike: *MonteCarloIndependentSet(G)*
Ulaz: Neusmeren graf $G = (V, E)$
Izlaz: Nezavisni skup čvorova I u grafu G

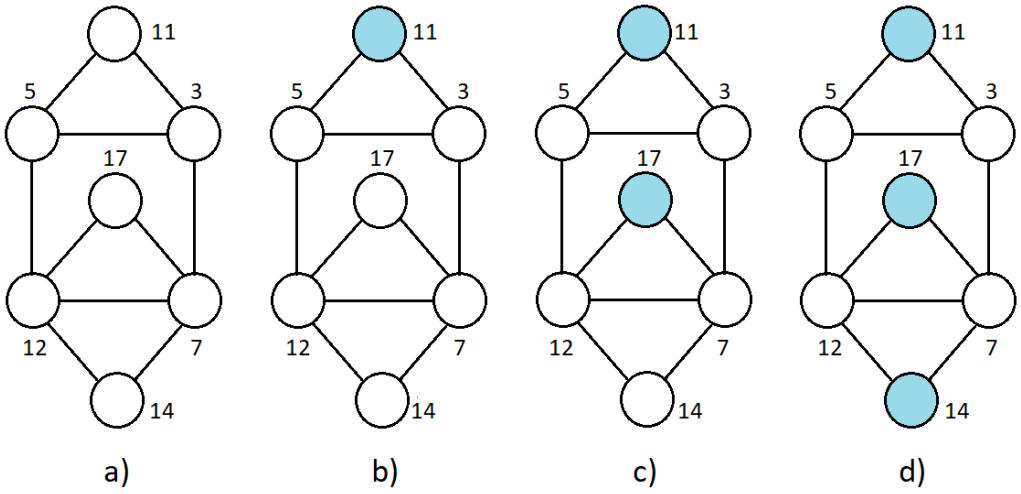
```
1: I = ∅
2: for each v ∈ V do
3:     ρ(v) = RandomNumber()
4: end for

5: for each v ∈ V do
6:     if ∀w ∈ Neighbours(v, G) ρ(v) > ρ(w) then
7:         I = I ∪ {v}
8:     end if
9: end for

10: return I
```

Na slici 4.2 je ilustrovano nalaženje nezavisnog skupa čvorova pomoću prethodno opisane heuristike. U delu pod a) je ilustrovan neusmereni graf čijim su čvorovima $v \in V$ dodeljeni slučajni brojevi $\rho(v)$. Nakon dodeljivanja prioriteta čvorovima u

drugoj po redu `for` petlji se vrši izračunavanje rezultujućeg nezavisnog skupa I. U svakoj iteraciji ove petlje se u skup I dodaje čvor koji se trenutno razmatra ako se ustanovi da mu je dodeljen prioritet koji je veći od prioriteta svih njegovih suseda. Na istoj slici, u delu pod b), je prikazan rezultat rada prve iteracije ove petlje. Odabran je čvor u za koga važi $\rho(u) = 11$ jer ima veći prioritet od prioriteta svih svojih suseda i kao takav će pripadati rezultujućem nezavisnom skupu I. U narednim iteracijama ove petlje (na istoj slici u delovima pod c) i d) redom) u skup I će biti dodati čvorovi v i w za koje važi: $\rho(v) = 17$ i $\rho(w) = 14$. Nakon završetka rada `for` petlje kao rezultat rada heuristike dobijamo nezavisni skup $I = \{u, v, w\}$, gde važi: $\rho(u) = 11$, $\rho(v) = 17$ i $\rho(w) = 14$. Primetimo da je skup I ujedno i maksimalni nezavisni skup u grafu.



Slika 4.2: Primer određivanja nezavisnog skupa grafa pomoću *Monte Karlo* algoritma

Neka je data funkcija $MonteCarloIndependentSet(G)$ koja koristi prethodno navedena heuristiku (Heuristika 13) za izračunavanje nezavisnog skupa u grafu G. U nastavku je data heuristika *Luby-MIS* za nalaženje maksimalnog nezavisnog skupa u grafu koji koristi $MonteCarloIndependentSet(G)$ funkciju. Heuristiku je konstruisao američki naučnik *Michael Luby* koji je poznat po izuzetnim dostignućima u oblasti paralelnih algoritama. *MIS* u nazivu heuristike označava englesku skraćenicu za *maksimalni nezavisni skup* (eng. *maximal independent set*).

Heuristika 14 *Luby - MIS* heuristika za nalaženje maksimalnog nezavisnog skupa u grafu

Naziv heuristike: *LubyMIS(G)*
Ulaz: Neusmeren graf $G = (V, E)$
Izlaz: Maksimalni nezavisni skup čvorova I u grafu G

1: $U = V$
2: $I = \emptyset$
3: $G' = G$

4: **while** $G' \neq \emptyset$ **do**
5: $\quad I' = MonteCarloIndependentSet(G')$
6: $\quad I = I \cup I'$
7: $\quad U = U \setminus (I' \cup Neighbours(I', G'))$
8: $\quad G' = G'[U]$
9: **end while**

10: **return** I

Skup U predstavlja skup čvorova koji se razmatra u aktuelnoj iteraciji `while` petlje. Skup I predstavlja maksimalni nezavisni skup čvorova u grafu, odnosno rezultat rada date heuristike. Skup I' predstavlja pomoćni skup koji koristimo za iterativno dopunjavanje skupa I. Pomoćna funkcija $Neighbours(I', G')$ izračunava skup suseda svih čvorova skupa I' u grafu G'. Nakon što odredimo nezavisni skup čvorova I', uniramo ga sa skupom I i rezultat smeštamo u skup I. Iz skupa U, zatim, izbacujemo skup čvorova I' zajedno sa skupom suseda $Neighbours(I', G')$, jer susedi čvorova skupa I' ne mogu pripadati rezultujućem skupu I. U odnosu na skup U se na kraju svake iteracije `while` petlje indukuje podgraf $G' = G[U]$ grafa G u odnosu na koji tražimo nezavisne skupove u narednoj iteraciji petlje.

4.2.3 Paralelna Luby - MIS heuristika za nalaženje maksimalnog nezavisnog skupa

Zbog načina na koji je konstruisana, *Luby - MIS* heuristiku je vrlo lako paralelizovati. Zbog svojih doprinosa, pronalazač ove paralelne heuristike, *Michael Luby*, nagrađen je prestižnom nagradom u oblasti distribuiranog računarstva *Edsger W. Dijkstra Prize in Distributed Computing*. U nastavku je dat pseudokod paralelne *Luby - MIS* heuristike.

Heuristika 15 Paralelna *Luby - MIS* heuristika za nalaženje maksimalnog nezavisnog skupa u grafu

Naziv heuristike: *ParallelLubyMIS(G)*
Ulaz: Neusmeren graf $G = (V, E)$
Izlaz: Maksimalni nezavisni skup čvorova I u grafu G

1: $U = V$
2: $I = \emptyset$
3: $G' = G$

4: **for** each $v \in U$ in parallel **do**
5: $\quad \rho(v) = RandomNumber(n^c)$
6: **end for**

7: **while** $U \neq \emptyset$ **do**
8: $\quad I' = \emptyset$
9: $\quad$ **for** each $v \in U$ in parallel **do**
10: $\quad\quad$ **if** $\forall u \in Neighbours(v, G')\ \rho(v) > \rho(u)$ **then**
11: $\quad\quad\quad I' = I' \cup \{v\}$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: $\quad I = I \cup I'$
15: $\quad U = U \setminus (I' \cup Neighbours(I', G'))$
16: $\quad G' = G[U]$
17: **end while**

18: **return** I

Funkcija $RandomNumber(k)$ čvoru v dodeljuje slučajan broj iz skupa $\{1, 2, \ldots, k\}$, $k \in N$. Pretpostavimo da ova funkcija koristi ravnomernu raspodelu za generisanje slučajnog broja iz skupa $\{1, 2, \ldots, k\}$. Ovo znači da važi:

$$P\{RandomNumber(k) = l\} = \frac{1}{k},\ l \in \{1, 2, \ldots, k\}.$$

Parametar c je odabran na osnovu zaključaka dobijenih pomoću leme 2, koja je prikazana u nastavku rada. Primetimo da se dodeljivanje prioriteta $\rho(v)$ čvorovima $v \in V$ može obaviti paralelno za vreme $O(1)^*$, što znači da se paralelna `foreach` petlja date heuristike može izvršiti za vreme $O(1)$. Isti zaključak važi i za korak pod rednim brojem 10 date heuristike: uz pomoć dovoljnog broja procesora, provera da li neki čvor ima veći prioritet od svih svojih suseda može da se obavi za vreme $O(1)$.

Na slici 4.3 je ilustrovan postupak nalaženja maksimalnog nezavisnog skupa u

*Ukoliko nam je na raspolaganju neograničen broj procesora p.

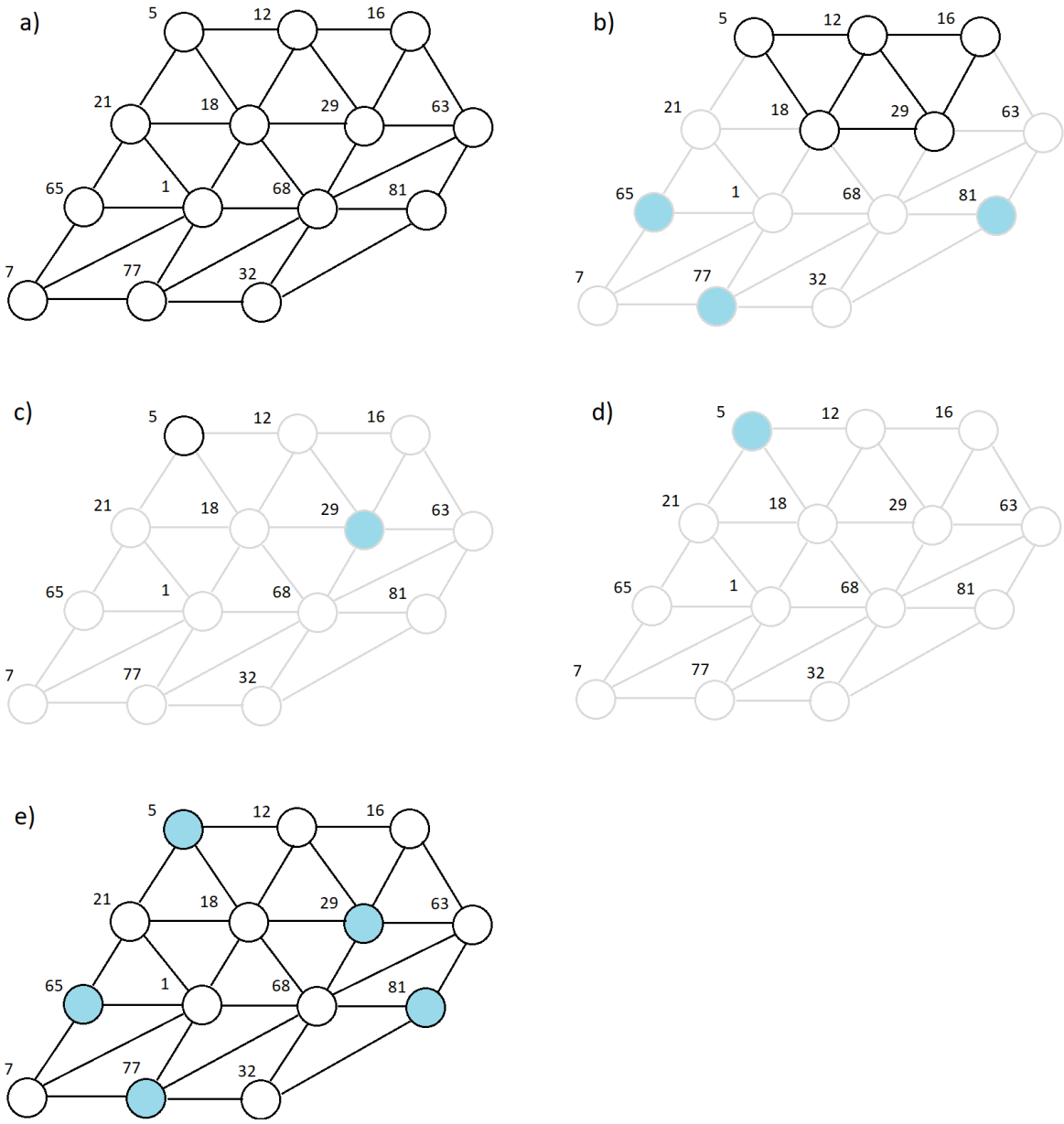
grafu pomoću paralelne *Luby - MIS* heuristike: u delu pod a), je prikazan neusmeren graf koji nije obojen, gde je svakom čvoru $v \in V$ dodeljen slučajan broj $\rho(v)$. Na istoj slici, u delu pod b), prikazan je rezultat rada prve iteracije spoljašnje `while` petlje. Pomoćni skup I', koji koristimo za nalaženje maksimalnog nezavisnog skupa I, nakon prve iteracije iznosi $I' = \{65, 77, 81\}$. Aktuelna iteracija petlje će iz skupa U izbaciti sve čvorove skupa I' i sve susede svih čvorova skupa I'. Na istoj slici, u delovima pod c) i d), ilustrovani su rezultati rada druge i treće iteracije spoljašnje `while` petlje. Čvorovi skupa I', koji predstavlja rezultat rada aktuelne iteracije petlje, su obojeni plavom bojom. Čvorovi i grane koje su izbačene iz grafa G' u okviru prve, druge i treće iteracije spoljašnje `while` petlje su (u delovima pod b), c) i d)) prikazani svetlijom bojom u odnosu na čvorove koji nisu izbačeni iz grafa G'. Nakon što se u toku aktuelne iteracije petlje pronađe nezavisni skup I', on se unira sa skupom koji je konstruisan kao rezultat rada prethodnih iteracija. Rezultat uniranja se upisuje u skup I. Konačni rezultat rada paralelne *Luby - MIS* heuristike je prikazan na istoj slici, u delu pod e). Maksimalni nezavisni skup određen ovom heuristikom je $I = \{5, 29, 65, 77, 81\}$.

U nastavku pod skupom I ćemo podrazumevati skup koji se formira iterativno u toku rada paralelne *Luby-MIS* heuristike (Heuristika 15). Na kraju rada heuristike skup I će biti maksimalni nezavisni skup u grafu G.

Prethodno opisana heuristika, iako konstruiše maksimalni nezavisni skup grafa, u opštem slučaju neće odrediti maksimalni nezavisni skup najveće veličine. U tom slučaju se ispostavlja da je umesto prioriteta čvorova bolje porediti stepene čvorova - ukoliko neki čvor grafa ima veći stepen od stepena svih svojih suseda on se dodaje u nezavisan skup I'. Ukoliko neka dva čvora imaju isti stepen, prioritet za ulazak u nezavisni skup I' dobija onaj čvor v koji ima veći prioritet $\rho(v)$. Za primenu ovakvog principa određivanja maksimalnog nezavisnog skupa dovoljno je u Heuristici 15 linije 10, 11 i 12 zameniti sledećim linijama:

1: **if** $\forall u \in Neighbours(v, G')$ $deg(v) > deg(u)$ or $(deg(v) == deg(u)$ and $\rho(v) > \rho(u))$ **then**
2: $\quad I' = I' \cup \{v\}$
3: **end if**

U nastavku ćemo uvesti definicije aktivnih čvorova i grana koje će nam pomoći u analizi očekivane vremenske slozenosti prethodno opisane heuristike.



Slika 4.3: Primer određivanja maksimalnog nezavisnog skupa grafa pomoću paralelne *Luby - MIS* heuristike

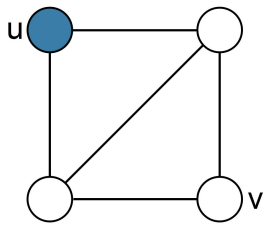
Definicija 4.2.1. *Aktivni čvor* (eng. *active vertex*) $v \in V$ je čvor koji se može dodati u nezavisan skup I u grafu $G = (V, E)$, tako da skup čvorova I ostane nezavisan.

Drugim rečima, čvor v je aktivni čvor ako ni jedan čvor skupa $Neighbours(v, G)$ nije sadržan u skupu I. Čvor v gubi svojstvo aktivnog čvora onda kada bude do-

dat u skup I ili kada neki njegov sused bude dodat u skup I. Primetimo da je na početku *ParallelLubyMIS* heuristike (Heuristika 15) svaki čvor aktivni čvor. Neka je data funkcija $ActiveNeighbours(v)$ koja računa skup svih aktivnih čvorova koji su susedni čvoru v. Neka je data funkcija $ActiveDegree(v)$ koja računa $|ActiveNeighbours(v)|$.

Definicija 4.2.2. Kažemo da aktivni čvor v *gasi* aktivni čvor $u \in$ *ActiveNeighbours(v)* ako v pripada maksimalnom nezavisnom skupu I (ako je dodat u skup I u nekoj iteraciji). Tada kažemo da je čvor u *ugašen*.

Na slici 4.4 je prikazan primer grafa G čiji je čvor u dodat u nezavisni skup I. Skup I u trenutnoj iteraciji sadrži samo čvor u. Na istoj slici je prikazan jedan aktivni čvor, obeležen oznakom v. Čvor v je aktivni čvor jer se može dodati u skup I, a da pri tome skup I ostane nezavisni skup čvorova u grafu G. Primetimo da je čvor v jedini aktivni čvor u prikazanom grafu.



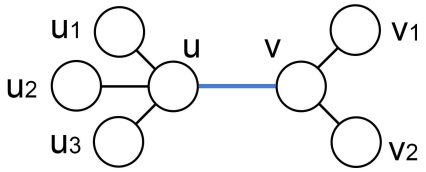
Slika 4.4: Primer aktivnog čvora v

Definicija 4.2.3. *Aktivna grana* (eng. *active edge*) je grana $e = (u, v) \in E$ za koju važi da su čvorovi u i v aktivni čvorovi

Drugim rečima, aktivna grana je ona grana grafa za koju važi da ni jedan od njenih čvorova nije dodat u skup I i da se bilo koji od njih može dodati u nezavisni skup I.

Definicija 4.2.4. Kažemo da je grana $e = (u, v) \in E$ *ugašena* ako je ugašen barem jedan od njenih čvorova u ili v.

Na slici 4.5 je ilustrovan podgraf nekog grafa G koji sadrži aktivnu granu (u, v). Da bi grana (u, v) bila aktivna u grafu G ni jedan od čvorova: $u, u_1, u_2, u_3, v, v_1, v_2$ ne sme biti dodat u nezavisni skup I. Dodavanjem bilo kog od ovih čvorova u nezavisni skup čvorova I ova grana gubi svojstvo aktivne grane.



Slika 4.5: Primer aktivne grane (u, v)

Lema 2. *Neka je* $\rho(v) \in \{1, 2, \ldots, n^c\}, v \in V, c \in N$. *Tada će verovatnoća da će nekom aktivnom čvoru* $v \in V$ *na početku paralelne* Luby - MIS *heuristike (Heuristika 15) biti dodeljena ista slučajna vrednost* $\rho(v) = \rho(u)$ *koja je dodeljena nekom njegovom susednom čvoru* $u \in V, u \neq v$ *biti odozgo ograničena brojem* $\frac{m}{n^c}$.

Dokaz: Neka je $(u, v) \in E$ fiksirano. Verovatnoća da će čvorovima u i v biti dodeljena ista slučajna vrednost ρ je jednaka:

$$P\{\rho(u) = \rho(v) = \rho\} = \frac{1}{n^c}.$$

Verovatnoća da u grafu postoje dva čvora u i v kojima će biti dodeljena ista slučajna vrednost ρ je jednaka:

$$P\{\exists(u, v) \in E \mid \rho(u) = \rho(v) = \rho\} \leq \sum_{(u,v)\in E} P\{\rho(u) = \rho(v)\} \leq \frac{m}{n^c}.$$

Primetimo da je ova verovatnoća ograničena malim brojem $\frac{m}{n^c}$ za dovoljno veliku vrednost parametra c, što znači da će verovatnoća dodeljivanja jedinstvene vrednosti $\rho(v)$ svakom čvoru $v \in V$ biti jako velika. □

Primetimo da odabirom parametra c možemo kontrolisati gornju granicu navedene verovatnoće. Za dovoljno veliko c verovatnoća da u grafu G postoje dva čvora kojima je dodeljen isti prioritet postaje jako mala: potrebno je odrediti takvo c tako da važi $n^c >> m$, odnosno: $c >> \log_n m$, gde simbol $>>$ označava relaciju *dosta veće*.

Definicija 4.2.5. Kažemo da čvor v *preventivno gasi* (eng. *preemptively kills*) čvor u ako važe sledeći uslovi:

1. $(u, v) \in E$ je aktivna grana u $G[U]$
2. $(\forall w \in ActiveNeighbours(v))(\rho(v) > \rho(w))$

3. $(\forall w \in ActiveNeighbours(u) \setminus \{v\})(\rho(v) > \rho(w))$

Drugim rečima, čvor v preventivno gasi čvor u ako čvor v gasi čvor u i važi da od svih čvorova koji gase čvor u čvor v ima najveći prioritet $\rho(v)$. Primetimo da aktivni čvor u može biti preventivno ugašen najviše jednom. Takođe, ako čvor v preventivno ugasi čvor u tada kažemo da on preventivno gasi sve aktivne grane koje sadrže u. Svaka aktivna grana $e = (u, v) \in E$ se može preventivno ugasiti najviše dva puta: jednom kada je čvor u preventivno ugašen i drugi put kada je čvor v preventivno ugašen. Primetimo da je uslov da čvor v preventivno gasi čvor u strožiji od uslova da čvor v gasi čvor u, zbog prvog uslova u prethodno navedenoj definiciji.

Lema 3. *U svakom koraku paralelne* Luby - MIS *heuristike (Heuristika 15) očekivani broj grana u grafu $G[U]$ na kraju tog koraka iznosi najviše polovinu veličine skupa grana u grafu $G[U]$ sa početka ovog koraka.*

Dokaz: Radi preglednosti uvedimo skraćene oznake: *AN(v) = ActiveNeighbours(v)* i $AD(v) = ActiveDegree(v)$. Analizirajmo i-tu iteraciju spoljašnje `while` petlje paralelne *Luby - MIS* heuristike. Procenimo donju granicu broja aktivnih grana koje su ugašene u ovoj iteraciji. Neka slučajna veličina Y predstavlja broj preventivno ugašenih grana u i - toj iteraciji spoljašnje `while` petlje. Neka je broj grana u grafu $G[U]$ označen sa $E(G[U])$. Tada važi:

$$E[Y] = \sum_{j=1}^{k} p_j x_j, \qquad k = |E(G[U])|,$$

gde je p_j verovatnoća da je u i-toj iteraciji spoljašnje `while` petlje ugašeno x_j grana. Definišimo slučajnu veličinu kao indikator $X_{(u,v)}$ koji ima vrednost 1 ako v preventivno gasi u, a u suprotnom ima vrednost 0.

$$E[Y] = \sum_{(u,v)\in E(G[U])} P\{X_{(u,v)} = 1\}AD(u) + \sum_{(u,v)\in E(G[U])} P\{X_{(v,u)} = 1\}AD(v),$$

U prvoj sumi se koristi prebrojavanje po onim aktivnim granama koje su preventivno ugašene onda kada čvor v preventivno gasi čvor u. Broj takvih grana iznosi $AD(u)$. Sličan zaključak važi za drugu sumu. Označimo sa X slučajnu veličinu koja predstavlja broj ugašenih grana u tekućoj iteraciji petlje. Pošto svaka grana može biti preventivno ugašena najviše dva puta dok ista može biti ugašena najviše jednom zaključujemo da važi da je $2E[X] \geq E[Y]$, odnosno:

$$2E[X] \geq \sum_{(u,v)\in E(G[U])} (P\{X_{(u,v)} = 1\}AD(u) + P\{X_{(v,u)} = 1\}AD(v)) = E[Y]$$

Da bi čvor v preventivno ugasio čvor u njemu mora biti dodeljena vrednost $\rho(v)$ koja je veća od svih vrednosti $\rho(u'), (u', u) \in E$ i svih vrednosti $\rho(v'), (v', v) \in E$. Iz toga izvodimo zaključak da važi:

$$P\{X_{(u,v)} = 1\} \geq \frac{1}{AD(u) + AD(v)}$$

odakle zaključujemo:

$$2E[X] \geq \sum_{(u,v)\in E(G[U])} (\frac{AD(u)}{AD(u) + AD(v)} + \frac{AD(v)}{AD(u) + AD(v)}) = |E(G[U])|$$

Odavde sledi:

$$E[X] \geq \frac{|E(G[U])|}{2}$$

što znači da očekivani broj grana u grafu $G[U]$ nakon ovog koraka heuristike iznosi maksimalno polovinu broja grana u grafu $G[U]$ sa početka iteracije.

□

Konačno, dolazimo do ključne leme koja daje odgovor na pitanje složenosti *Luby - MIS* heuristike.

Lema 4. O*značimo sa* p *broj procesora koji računar sa paralelnom arhitekturom koristi. Tada za* Luby - MIS *heuristiku (Heuristika 15) važi:*

1. $E[T(n, 1)] = O(m)$
2. $E[T(n, p)] = O(\log(m))$, *kada* $p \to \infty$

Dokaz:

1. Bez gubitka opštosti pretpostavimo da je $|E| = 2^k$, za neko $k \in N$. U svakom koraku *Luby - MIS* heuristike obavi se $|E(G[U])|$ operacija. Broj $|E(G[U])|$ inicijalno iznosi $|E|$ a nakon aktuelne iteracije iznosi najviše pola svoje prvobitne vrednosti. Takođe, znamo da spoljašnja `while` petlja iterira sve dok skup U ne postane prazan (dok graf $G[U]$ ne postane prazan). Prema tome važi:

   $$E[T(n, 1)] \leq |E| + \frac{|E|}{2} + \frac{|E|}{4} + \ldots + 1.$$

   Ova suma konvergira ka broju $2|E| = O(m)$, odakle sledi da je $E[T(n, 1)] = O(m)$.

2. Pod pretpostavkom da nam je na raspolaganju neograničen broj procesora svaki korak *Luby-MIS* heuristike se obavlja u konstantnoj vremenskoj složenosti (za razliku od slučaja kada je $p = 1$ kada taj broj iznosi $|E(G[U])|$). Pošto nakon svakog koraka heuristike broj $|E[G[U]]|$ iznosi najviše pola svoje inicijalne vrednosti, ukupan broj iteracija `while` petlje biće $O(\log(m))$ odakle sledi da je $E[T(n,p)] = O(\log(m))$, kada $p \to \infty$. □

4.2.4 Paralelna Luby heuristika za bojenje grafa

Pretpostavimo da funkcija $ParallelLubyMIS(G)$ koristi prethodno opisanu heuristiku (Heuristika 15) za izračunavanje maksimalnog nezavisnog skupa u neusmerenom grafu $G = (V, E)$. U nastavku je data paralelna heuristika za bojenje grafa koja koristi prethodno navedenu funkciju. Data heuristika koristi opštu ideju heuristike zasnovane na odabiru nezavisnih skupova.

Heuristika 16 Paralelna Luby heuristika za bojenje grafa

Naziv heuristike: *ParallelLuby(G)*
Ulaz: Neusmeren graf $G = (V, E)$

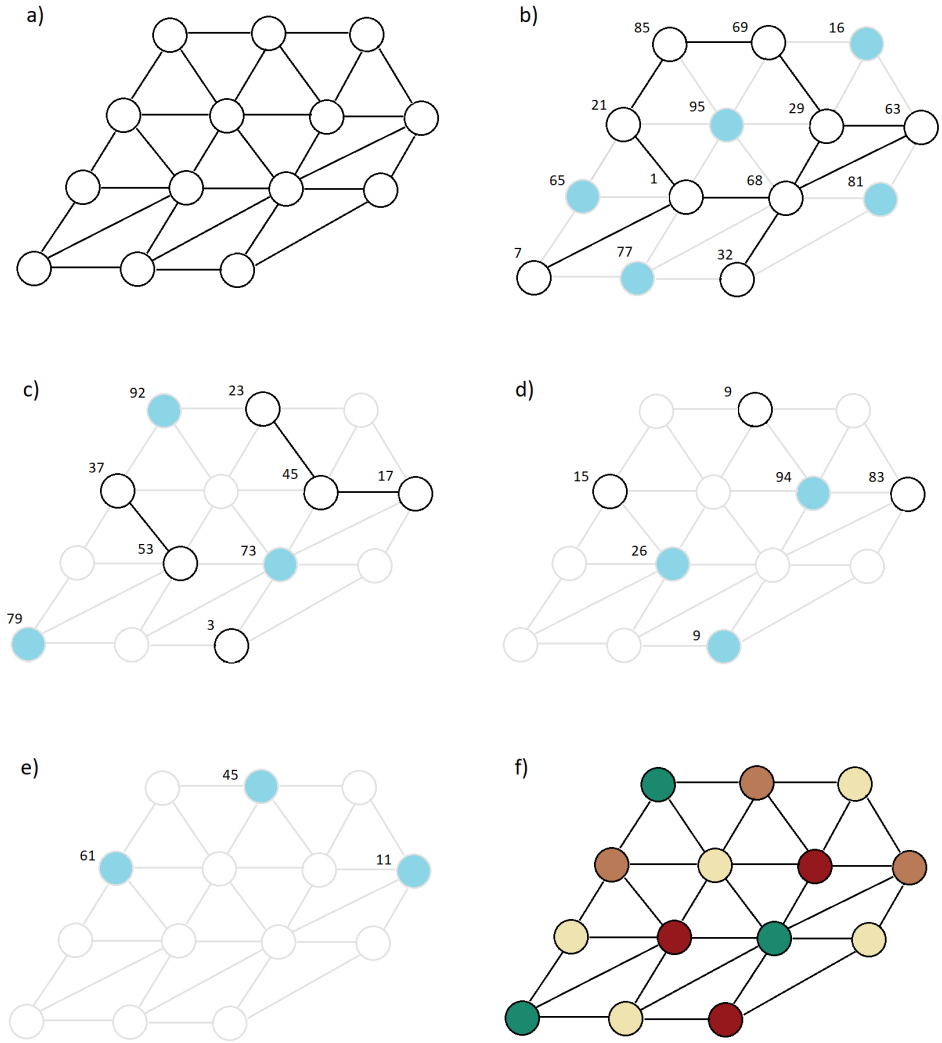
1: $U = V$
2: $G' = G$
3: $\mathcal{F} = \emptyset$

4: **while** $U \neq \emptyset$ **do**
5: $\quad I = ParallelLubyMIS(G')$
6: $\quad \mathcal{F} = \mathcal{F} \cup \{I\}$
7: $\quad U = U \setminus I$
8: $\quad G' = G[U]$
9: **end while**

10: **for** each I_j in $\mathcal{F}$ in parallel **do**
11: $\quad ProcessNodeColoring(I_j, c_j)$
12: **end for**

Analizirajmo vremensku složenost date heuristike. Pod pretpostavkom da pri konstrukciji heuristike postoji mogućnost korišćenja neograničenog broja procesora p, paralelna `for` petlja će se izvršiti u vremenu $O(1)$. Najveći uticaj na vremensku složenost heuristike imaju broj iteracija `while` petlje i složenost paralelne *Luby - MIS* heuristike. S obzirom na to da broj iteracija `while` petlje iznosi $O(\Delta)$ i da očekivana složenost paralelne *Luby - MIS* heuristike iznosi $O(\log(m))$, očekivana složenost date heuristike za bojenje iznosi $O(\Delta \cdot \log(m))$.

Na slici 4.6 je prikazan postupak bojenja grafa G pomoću prethodno opisane heuristike. U delu pod a) je prikazan neusmeren graf koji je potrebno obojiti. Na istoj slici, u delovima pod b), c), d), i e) su ilustrovani rezultati rada i-te iteracije `while` petlje. U i-toj iteraciji ove petlje heuristika pronalazi maksimalni nezavisni skup I čiji su elementi, radi ilustracije, obojeni plavom bojom. Nakon toga, skup I se ubacuje u familiju nezavisnih skupova $\mathcal{F}$. U istoj iteraciji se formira novi graf G' za narednu iteraciju petlje. Grane grafa i čvorovi grafa koji su izbačeni iz grafa G' u prethodnim iteracijama (uključujući i trenutnu iteraciju) `while` petlje su prikazani svetlijom bojom u odnosu na one čvorove i grane koji nisu izbačeni iz grafa G'. Na istoj slici, u delu pod f), je prikazan rezultat rada paralelne `foreach` petlje koja se izvršava nakon `while` petlje - dat je obojeni graf G.



Slika 4.6: Primer bojenja grafa pomoću paralelne *Luby* heuristike

4.3 Jones - Plassman heuristika

Primetimo da *Luby* heuristika (Heuristika 16) za bojenje grafa zahteva da se izvrši dodeljivanje slučajnih brojeva $\rho(v)$ čvorovima svaki put kada se izračunava novi skup nezavisnih čvorova. Razlog za razvoj naredne heuristike proizlazi iz dvostruke potrebe. Prvo, želimo smanjiti očekivanu vremensku složenost i unaprediti kvalitet dobijenog bojenja. Drugo, želimo da dodela prioriteta čvorovima bude izvršena samo na početku heuristike. Činjenica koja ide u prilog izbegavanju prečestog dodeljivanja prioriteta čvorovima počiva u tome da u praksi izračunavanje slučajnih brojeva nosi sa sobom značajno računarsko opterećenje, jer se strukture podataka povezane sa slučajnim brojevima moraju ponovo izračunavati.

Jones - Plassman (skraćeno *JP*) heuristika se zasniva na ideji nalik onoj koju koriste heuristike zasnovane na odabiru nezavisnih skupova. Kao i kod heuristika zasnovanih na odabiru nezavisnih skupova, kod *JP* heuristike se glavni deo koda izvršava u okviru glavne `while` petlje. Glavna razlika u odnosu na pomenute heuristike je u tome što se u okviru glavne `while` petlje *JP* heuristike konstruiše nezavisan skup čvorova I koji ne mora biti maksimalni nezavisni skup čvorova u grafu koji se razmatra u aktuelnoj iteraciji petlje. Dodatna razlika je u tome što čvorovi nezavisnog skupa I ne moraju biti obojeni istom bojom. Na početku *JP* heuristike se svakom čvoru paralelno dodeljuje slučajno izabran broj $\rho(v)$ koji izračunava funkcija $RandomNumber()$. U svakoj iteraciji spoljašnje `while` petlje se izračunava po jedan nezavisni skup I. Kao i ranije, broj $\rho(v)$ određuje prioritet čvora - čvorovi sa većim prioritetom u odnosu na svoje susede imaju veću šansu da budu ubačeni u nezavisni skup I u okviru trenutne iteracije `while` petlje. Skup U je pomoćni skup čvorova koji sadrži one čvorove koji se obrađuju u aktuelnoj iteraciji spoljašnje `while` petlje. Promenljiva *addVertex* predstavlja indikator da li čvor v treba da uđe u nezavisni skup čvorova I. Ona je na početku postavljena na vrednost *true*. U paralelnoj `for` petlji proveravamo da li postoji neki sused čvora v koji je većeg prioriteta od čvora v. Ako je taj uslov ispunjen, ovoj promenljivoj se dodeljuje vrednost $false$ i na taj način se čvoru v ukida mogućnost da uđe u nezavisni skup I. Nakon što skup I bude izračunat svi njegovi čvorovi se paralelno boje tako što se svakom čvoru $v \in I$ dodeli najmanja po indeksu boja tako da bojenje bude ispravno. Posledica ovakvog načina dodeljivanja boja čvorovima skupa I je postojanje mogućnosti da čvorovi skupa I ne budu obojeni istom bojom.

Pokazano je da očekivana vremenska složenost ove heuristike iznosi $O(\Delta \cdot \frac{\log m}{\log \log m})$

(pogledati rad [9]), što je poboljšanje u odnosu na očekivanu složenost *Luby* heuristike (Heuristika 16) za bojenje grafa (koja iznosi $O(\Delta \cdot \log m)$).

Heuristika 17 Paralelna *JP* heuristika za bojenje grafa

Naziv heuristike: *ParallelJP(G)*
Ulaz: Neusmeren graf $G = (V, E)$

1: $U = V$
2: **for** each $v \in U$ in parallel **do**
3: $\quad \rho(v) = RandomNumber()$
4: **end for**

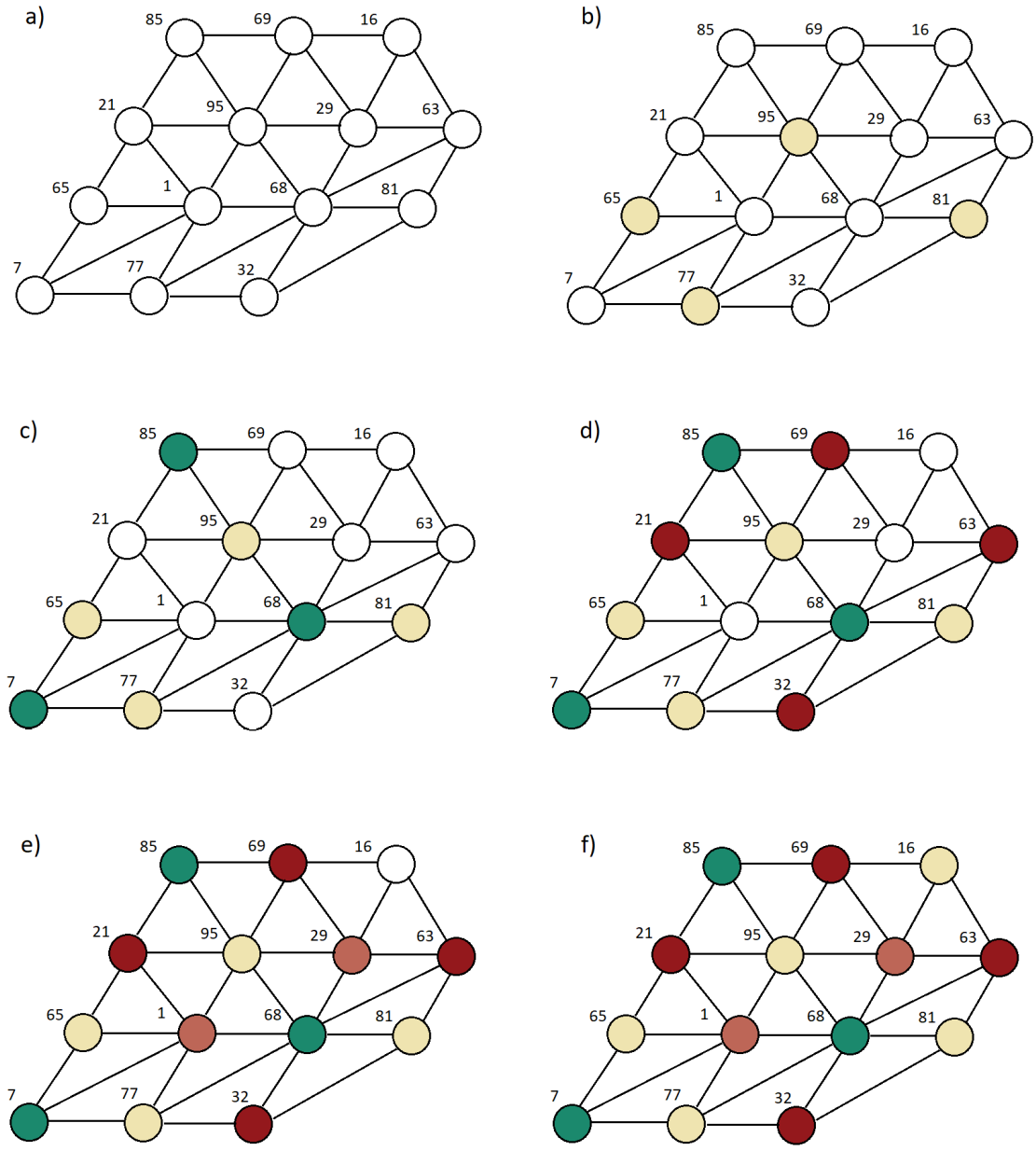
5: **while** $U \neq \emptyset$ **do**
6: $\quad G' = G[U]$
7: $\quad I = \emptyset$
8: $\quad addVertex = true$

9: $\quad$ **for** each $v \in U$ in parallel **do**
10: $\quad\quad$ **for** each $w \in Neighbours(v, G')$ **do**
11: $\quad\quad\quad$ **if** $\rho(v) < \rho(w)$ **then**
12: $\quad\quad\quad\quad addVertex = false$
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end for**
15: $\quad\quad$ **if** $addVertex == true$ **then**
16: $\quad\quad\quad I = I \cup \{v\}$
17: $\quad\quad$ **end if**
18: $\quad$ **end for**

19: $\quad$ **for** each $v \in I$ in parallel **do**
20: $\quad\quad C = \{u.Color \mid u \in Neighbours(v, G)\}$
21: $\quad\quad v.Color = min\{color > 0 \mid color \notin C\}$
22: $\quad$ **end for**
23: $\quad U = U \setminus I$
24: **end while**

Na slici 4.7 je ilustrovano bojenje grafa po iteracijama pomoću JP heuristike. U delu pod a), je prikazan neusmeren i neobojen graf. Svakom čvoru v grafa dodeljen je slučajni broj $\rho(v)$ čime je završen prvi korak JP heuristike. Na istoj slici, u delovima pod b), c), d), e) i f) redom su ilustrovani rezultati rada i-te iteracije spoljašnje `while` petlje gde važi: $i \in \{1, 2, 3, 4, 5\}$. Primetimo da redni broj iteracije bojenja ne određuje boju kojom će čvorovi biti obojeni, pa je moguće da neki čvorovi budu obojeni bojama koje su korišćene u prethodnim iteracijama glavne `while` petlje.

Naime, u delu pod b) čvorovi su obojeni žutom bojom, dok u delu pod f) vidimo da je čvor pod rednim brojem 16 obojen takođe žutom bojom u toj iteraciji.

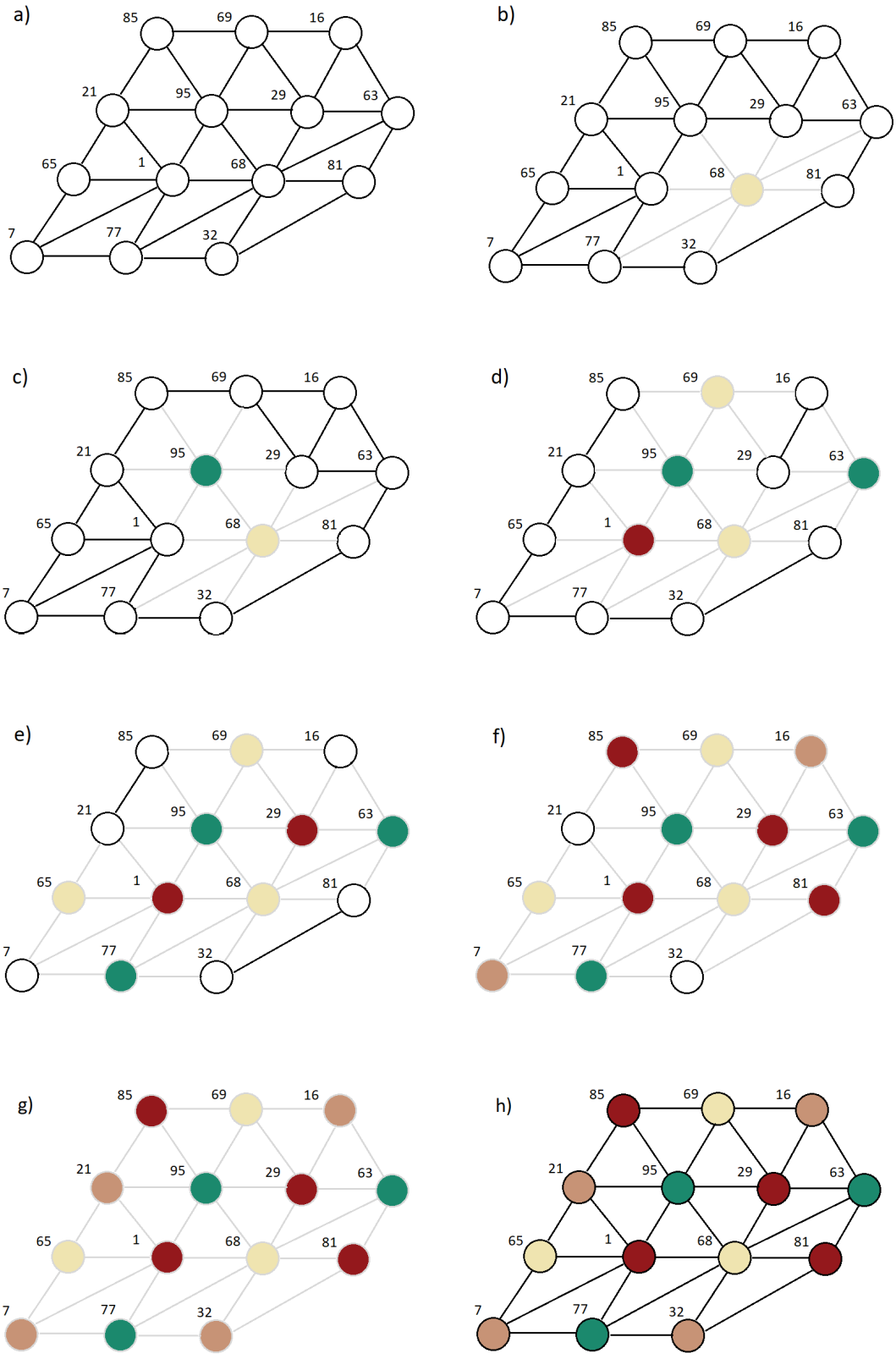


Slika 4.7: Primer bojenja grafa po iteracijama pomoću *JP* heuristike

4.4 Largest Degree First heuristika

Largest Degree First (LDF) heuristika predstavlja modifikaciju JP heuristike. Razlika u odnosu na JP heuristiku je u tome što se umesto slučajno generisanih prioriteta $\rho(v)$ za kreiranje nezavisnih skupova koriste osobine stepena čvorova. Preciznije, prilikom kreiranja nezavisnih skupova (koji se u narednom koraku heuristike boje), prednost se daje onim čvorovima $v \in V$ koji imaju veći stepen $deg(v)$ od svih svojih suseda. Ukoliko se ispostavi da čvor v ima veći stepen od svih svojih suseda on se dodaje u nezavisni skup I. U slučaju da postoje dva susedna čvora koja imaju isti stepen prioritet se daje onom čvoru v kome je dodeljen veći slučajni broj $\rho(v)$. Ispostavlja se da u proseku LDF heuristika koristi manje boja za bojenje grafa od JP algoritma. LDF heuristika se neznatno razlikuje od JP heuristike što za posledicu ima to da je asimptotska složenost LDF heuristike ista kao i asimptotska složenost JP heuristike.

Na slici 4.8 je ilustrovan primer bojenja grafa po iteracijama pomoću LDF heuristike. U delu pod a), dat je neusmeren i neobojen graf. Svakom čvoru v grafa dodeljen je slučajni broj $\rho(v)$ čime je završen prvi korak LDF heuristike. Na istoj slici, u delovima pod b), c), d), e), f) i g) redom su ilustrovani rezultati rada i-te iteracije spoljašnje `while` petlje i izgled grafa G' nakon ove iteracije, gde važi: $i \in \{1, 2, 3, 4, 5, 6\}$. Na primer, u delu pod b) vidimo da je u okviru prve iteracije odabran čvor v kao čvor najvećeg stepena u grafu G'. Za njega važi da je $\rho(v) = 68$ i $deg(v) = 7$. Pošto u okviru ove iteracije ni jedan drugi čvor u grafu nema veći stepen od svih svojih suseda, čvor v će biti jedini kandidat za bojenje u datoj iteraciji. Nakon bojenja čvora v novi graf G' za narednu iteraciju se formira izbacivanjem čvora v i svih njegovih grana iz tekućeg grafa G'. Grane i čvorovi koji su izbačeni iz grafa G' u prethodnim iteracijama i u tekućoj iteraciji `while` petlje su prikazani svetlijom bojom u odnosu na one grane i čvorove koji nisu izbačeni iz grafa G'. Na istoj slici, u delu pod h), prikazan je obojen graf.



Slika 4.8: Primer bojenja grafa po iteracijama pomoću *LDF* heuristike

Heuristika 18 Paralelna *LDF* heuristika za bojenje grafa

Naziv heuristike: *ParallelLDF(G)*

Ulaz: Neusmeren graf $G = (V, E)$

1: $U = V$

2: **for** each $v \in U$ in parallel **do**
3: $\rho(v) = RandomNumber()$
4: **end for**

5: **while** $U \neq \emptyset$ **do**
6: $G' = G[U]$
7: $I = \emptyset$
8: $addVertex = true$

9: **for** each $v \in U$ in parallel **do**
10: **for** each $w \in Neighbours(v, G')$ **do**
11: **if** $deg(v) < deg(w)$ or $(deg(v) == deg(w)$ and $\rho(v) < \rho(w))$ **then**
12: $addVertex = false$
13: **end if**
14: **end for**
15: **if** $addVertex == true$ **then**
16: $I = I \cup \{v\}$
17: **end if**
18: **end for**

19: **for** each $v \in I$ in parallel **do**
20: $C = \{u.Color \mid u \in Neighbours(v, G)\}$
21: $v.Color = min\{color > 0 \mid color \notin C\}$
22: **end for**
23: $U = U \setminus I$
24: **end while**

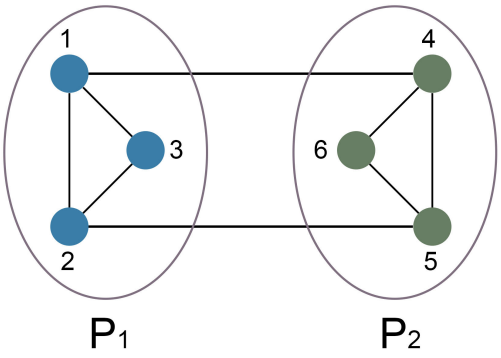
4.5 Heuristike zasnovane na particionisanju grafa

4.5.1 Problem particionisanja grafa

Problem particionisanja grafa (eng. *graph partitioning problem*) podrazumeva podelu skupa čvorova V neusmerenog grafa $G = (V, E)$ u particije $\{V_1, V_2, \ldots, V_k\}$ tako da važe sledeća tri uslova:

1. $V = \bigsqcup_{i=1}^{k} V_i$
2. Particije V_i sadrže isti ili skoro isti broj čvorova.
3. Broj grana koje povezuju čvorove različitih particija je minimalan.

Poznato je da je navedeni problem NP-težak, međutim postoje različite heuristike koji se koriste za pronalaženje približnih rešenja ovog problema. U okviru narednih poglavlja biće razmotreno kako particionisanje grafa može da se iskoristi u rešavanju problema bojenja grafa nakon čega će biti predstavljene alternative datom pristupu koje koriste ideju sličnu particionisanju. Na slici 4.9 je prikazano particionisanje neusmerenog grafa sa 6 čvorova u dve particije.



Slika 4.9: Primer particionisanja grafa u dve particije

Definicija 4.5.1. *Pseudobojenje grafa* (eng. *pseudocoloring of graph*) predstavlja bilo kakvo dodeljivanje boja čvorovima grafa $G = (V, E)$.

Nakon pseudobojenja grafa mogu postojati čvorovi $u \in V$ i $v \in V$ koji su susedni u grafu G i koji su obojeni istom bojom.

Definicija 4.5.2. Kažemo da je u toku pseudobojenja grafa $G = (V, E)$ došlo do *konflikta u bojenju* (eng. *conflict in coloring*) čvorova $u \in V$ i $v \in V$ ako su čvorovi u i v susedni u grafu G i ako su obojeni istom bojom.

Intuitivno je jasno da će pseudobojenje grafa biti upotrebljivo samo ako je broj konflikata u bojenju mali.

Definicija 4.5.3. *Parcijalno bojenje grafa* (eng. *partial graph coloring*) je bojenje podskupa $V' \subseteq V$ tako da nikoja dva susedna čvora skupa V' ne budu obojena istom bojom.

Primetimo da se bilo koje bojenje grafa može smatrati parcijalnim bojenjem istog tog grafa, što važi u slučaju da je $V' = V$.

4.5.2 Opšta heuristika za bojenje grafa bazirana na particionisanju

Pretpostavimo da je skup čvorova V neusmerenog grafa $G = (V, E)$ particionisan u k particija $\{V_1, V_2, \ldots, V_k\}$. Neka je data funkcija $\pi : V \to \{1, 2, \ldots, k\}$ koja čvoru $v \in V$ dodeljuje redni broj particije V_i kojoj pripada tj. važi $\pi(v) = i$, za $v \in V_i$.

Definišimo skup grana E_S kao skup $E_S \subseteq E$ za koji važi:

$$(u, v) \in E_S \Leftrightarrow \pi(u) \neq \pi(v)$$

Drugim rečima skup E_S je skup grana čiji čvorovi pripadaju različitim particijama. S u indeksu skupa E_S označava skraćenicu za engleski reč *shared* (srp. *deljen*). Na sličan način definišemo skup deljenih čvorova V_S kao skup $V_S \subset V$ za koji važi:

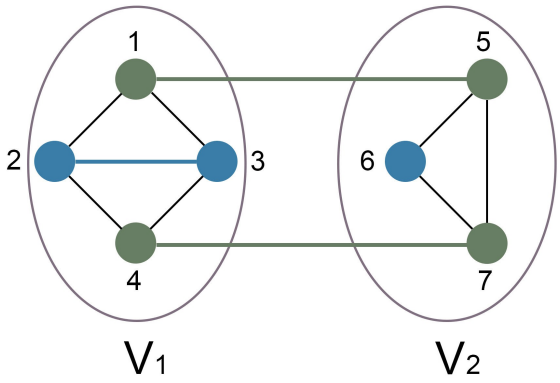
$$v \in V_S \Leftrightarrow (\exists u \in V)((u, v) \in E_S)$$

Drugim rečima, svaki čvor $u \in V_S$ je povezan granom sa nekim čvorom $v \in V_S$ koji pripada različitoj particiji u odnosu na čvor u.

Uvedimo i sledeće oznake: $V_L = V \setminus V_S$ i $V_i^L = V_L \cap V_i$. L u indeksu skupa V_L označava skraćenicu za engleski reč *local* (srp. *lokalni*), jer dva čvora skupa V_L jedino lokalno mogu biti povezana granom - samo ako pripadaju istoj particiji. Uočimo da važi: $V_L = \bigsqcup_i V_i^L$. Označimo sa $G_i = G[V_i^L]$, $G_L = G[V_L]$ i $G_S = G[V_S]$ podgrafove grafa G koji su indukovani skupovima V_i i V_L i V_S redom.

Na slici 4.10 je prikazan primer neusmerenog grafa G sa njegovim podgrafovima

G_L i G_S. Graf G je particionisan u dve particije: $V_1 = \{1, 2, 3, 4\}$ i $V_2 = \{5, 6, 7\}$. Skup deljenih grana (koje su na slici obojene zelenom bojom) E_S u ovom slučaju podrazumeva skup $\{(1, 5), (4, 7)\}$. Čvorovi V_L podgrafa G_S (zajedno sa granama podgrafa G_S) su obojeni plavom bojom dok su čvorovi V_S podgrafa G_S (zajedno sa granama podgrafa G_S) obojeni zelenom bojom.



Slika 4.10: Primeri podgrafova G_L (čiji su čvorovi i grane na slici obojeni plavom bojom) i G_S (čiji su čvorovi i grane na slici obojeni zelenom bojom) u neusmerenom grafu G

Lema 5. *Neka je C_i funkcija bojenja grafa G_i. Bojenje grafa G_L se može zadati na sledeći način: $C(v) = C_i(v)$, za $v \in V_i^L$ gde je $i = 1, 2, \ldots, k$.*

Dokaz: Znamo da važi $G_i = G[V_i^L]$. Svaki graf G_i je jedna komponenta povezanosti grafa G_L. Iz ovoga sledi da bojenje grafa G_i neće poremetiti ispravnost bojenja grafa G_j, za $i \neq j$, pošto čvorovi grafa G_i nisu povezani sa čvorovima grafa G_j. Iz toga sledi da je sa $C(v) = C_i(v)$, za $v \in V_i^L$ jedno dobro definisano bojenje grafa G_L. □

Lema 6. *Bojenje grafa G_L je jedno parcijalno bojenje grafa G i može se proširiti na bojenje celog grafa G.*

Dokaz: Neka je data funkcija bojenja $C : V_L \to Colors$ grafa $G_L = G[V_L]$. Znamo da važi $V = V_S \cup V_L$. Čvorovi koji pripadaju skupu V_L su obojeni pomoću funkcije C. Bojenje C se može produžiti na bojenje celog grafa G ukoliko se na skupu V_S definiše konzistentno bojenju skupa V_L. □

Heuristika 19 Opšta verzija heuristike za bojenje bazirana na particionisanju skupa čvorova

Naziv heuristike: *ParallelColoringWithPartitioning(G, k)*
Ulaz: Neusmeren graf $G = (V, E)$, prirodan broj k

1: Izvršiti particionisanje skupa čvorova V u k particija i pronaći V_S, G_i, V_i^L, za $i \in \{1, 2, \ldots, k\}$

2: **for** $i = 1$ to k in parallel **do**
3: Izvršiti bojenje grafa G_i koristeći sekvencijalnu heuristiku bojenja
4: **end for**

5: Izvršiti bojenje preostalih čvorova $v \in V_S$ konzistentno bojenju skupa V_L koristeći paralelnu heuristiku bojenja

Performanse navedene heuristike najviše zavise od heuristike za particionisanje skupa čvorova grafa. Postoji veliki broj različitih heuristika za particionisanje skupa čvorova V grafa. Ukoliko heuristika za particionisanje ima manju vremensku složenost od heuristike koja se koristi za bojenje grafova G_i onda je asimptotska složenost heuristike jednaka zbiru vremenskih složenosti heuristike za bojenje grafova G_i i paralelne heuristike koja boji čvorove V_S. U suprotnom, ukupna vremenska složenost heuristike je asimptotski jednaka vremenskoj složenosti heuristike za particionisanje skupa čvorova grafa.

4.5.3 Heuristika bojenja zasnovana na blokovskom particionisanju

Motivacija za konstukciju narednih heuristika se javlja iz potrebe da se izbegne rad sa NP-teškim problemom particionisanja koji se javlja kod prethodno opisanih heuristika zasnovanih na particionisanju grafa. Za neke klase grafova veličina skupa E_S može biti jako bliska veličini skupa E, što može predstavljati problem - tada će grafovi G_i biti u proseku male veličine što narušava potencijal da se heuristika ubrza paralelizacijom.

U nastavku, pod terminom *blok* (eng. *block*) ćemo podrazumevati proizvoljni podskup skupa čvorova V. U ovom slučaju ćemo, umesto dosadašnjeg particionisanja skupa čvorova grafa, primeniti strategiju deljenja skupa čvorova V u p blokova: $\beta_1, \beta_2, \ldots, \beta_p$ iste (ili što sličnije) veličine gde važi: $V = \bigsqcup_{i=1}^{p} \beta_i$ i za čiju će nam konstrukciju biti potrebno znatno manje vremena u odnosu na vreme potrebno za prethodno opisano particionisanje skupa V. U ovom slučaju nećemo ulagati dodatni trud da minimizujemo broj grana čiji čvorovi pripadaju različitim blokovima. Tada ne postoji garancija da će se paralelnim bojenjem prethodno dobijenih blokova dobiti ispravno bojenje grafa. Takođe, ovakvo deljenje skupa V na blokove može imati za posledicu to da je prosečan broj grana koje dele neka dva bloka veliki, što može dovesti do velikog broja konflikata u bojenju koje kasnije treba ispraviti.

Ulazni parametri heuristike su neusmereni graf $G = (V, E)$ i broj p koji nam ukazuje na koliko blokova je potrebno podeliti skup čvorova V. Heuristika se sastoji iz dve faze. Na početku prve faze heuristike se vrši particionisanje skupa V u p jednakih blokova: $\beta_1, \beta_2, \ldots, \beta_p$. Nakon ovog koraka vrši se pseudobojenje čvorova svakog bloka β_i paralelno pomoću funkcije $CalculateNodeColor(v, G)$, čime je završena prva faza heuristike. Pošto se bojenje blokova vrši paralelno često se podrazumeva da je broj blokova p jednak broju dostupnih procesora, kako bi se u potpunosti iskoristile prednosti paralelizacije.

Nakon završetka prve faze heuristike čvorovi unutar svakog bloka β_i će biti obojeni pravilno - ukoliko su neka dva čvora bloka β_i povezana granom oni će biti obojeni različitom bojom. Međutim, pošto se u prvoj fazi heuristike svaki blok paralelno boji, a pritom se ne obraća pažnja na to da li postoji grana između neka dva čvora $u \in \beta_i$ i $v \in \beta_j$ koji se paralelno boje, može doći do konflikata u bojenju čvorova koji pripadaju različitim blokovima. Druga faza heuristike detektuje konflikte u bojenju i ispravlja ih. Ukoliko se ispostavi da postoje dva čvora u i v koji su susedni u grafu

G a pritom su obojeni istom bojom, čvor nižeg indeksa se ubacuje u skup S, koji je inicijalno prazan. Nakon detekcije takvih čvorova skup S se boji pomoću proizvoljne sekvencijalne heuristike za bojenje. U nastavku je data heuristika za bojenje grafa zasnovana na blokovskom particionisanju skupa čvorova.

Heuristika 20 Heuristika bojenja zasnovana na blokovskom particionisanju skupa čvorova

Naziv heuristike: *ParallelColoringWithBlockPartitioning(G, p)*
Ulaz: Neusmeren graf $G = (V, E)$, prirodan broj p

1: //Prva faza heuristike:
2: Podeliti skup čvorova V u p blokova iste veličine: $\beta_1, \beta_2, \ldots, \beta_p$ tako da važi: $|\beta_i| = \frac{|V|}{p} = \frac{n}{p}$

3: **for** $i = 1$ to p in parallel **do**
4: **for** each v in β_i **do**
5: $CalculateNodeColor(v, G)$
6: **end for**
7: **end for**

8: //Druga faza heuristike:
9: $S = \{\}$
10: **for** i =1 to p in parallel **do**
11: **for** each v in β_i **do**
12: **for** each w in $Neighbours(v, G)$ **do**
13: **if** $v.Color == w.Color$ **then**
14: $x = min\{v, w\}$
15: $S = S \cup \{x\}$
16: **end if**
17: **end for**
18: **end for**
19: **end for**

20: Obojiti čvorove skupa S pomoću sekvencijalne heuristike

Analizirajmo sada vremensku složenost heuristike.

Lema 7. *Pod pretpostavkom korišćenja CRCW modela, prva faza prethodno navedene heuristike (Heuristika 20) vrši pseudobojenje grafa za vreme* $O(\frac{\Delta \cdot n}{p})$ *koristeći ukupno* $O(m)$ *operacija*[3].

[3]Ukupan broj operacija podrazumeva zbir broja svih operacija koje se izvrše na svim korišćenim

Dokaz: Prva faza prethodno navedene heuristike se sastoji od p paralelnih koraka (p iteracija paralelne `for` petlje). Ovo znači da je ukupna složenost heuristike proporcionalna složenosti jedne iteracije date petlje. U svakom paralelnom koraku se iterira kroz skup β_i koji ima $\frac{n}{p}$ elemenata i u svakoj iteraciji se poziva $CalculateNodeColor(v, G)$ metoda koja ima složenost $O(deg(v))$. Ovo znači da je, za fiksirano $v \in V$, složenost sinhrone petlje $O(\frac{n}{p}deg(v)) = O(\frac{n}{p}\Delta)$, što je i ukupna složenost prve faze heuristike. Da bismo odredili ukupan broj operacija prve faze heuristike posmatraćemo paralelnu `for` petlju kao sinhronu `for` petlju. Tada je ukupan broj operacija jednak $T(n,1) = \sum_{i=1}^{p} \sum_{v_j \in \beta_i} deg(v_j) = 2m = O(m)$.□

Lema 8. *Pod pretpostavkom korišćenja CRCW modela, očekivani broj konflikata boja čvorova na kraju prve faze prethodne heuristike (Heuristika 20) iznosi $O(\mu p)$, gde je $\mu = \frac{m}{n}$ prosečni stepen čvora u grafu.*

Dokaz: Označimo sa $t_1, t_2, \ldots, t_{\frac{n}{p}}$ vremenske intervale u okviru kojih se vrši bojenje čvorova u prvoj fazi heuristike (`for` petlja u liniji 4 ima ukupno $\frac{n}{p}$ iteracija pa toliko ima i vremenskih trenutaka u okviru kojih se vrši bojenje). Neka je $e = (u, v) \in E$ fiksirano. Označimo sa X_{S,t_j} indikatorsku slučajnu veličinu koja uzima vrednost 1 ako se svi čvorovi skupa S boje u trenutku t_j u prvoj fazi heuristike, a u suprotnom uzima vrednost 0, gde važi $1 \leq j \leq \frac{n}{p}$. Za svako $v \in V$ važi:

$$P\{X_{\{v\},t_j} = 1\} = \frac{1}{n/p} = \frac{p}{n},\ 1 \leq j \leq \frac{n}{p}$$

Pošto važi $X_{\{u,v\},t_j} = X_{\{u\},t_j} X_{\{v\},t_j}$ tada važi i:

$$P\{X_{\{u,v\},t_j} = 1\} = P\{X_{\{u\},t_j} X_{\{v\},t_j} = 1\} = P\{X_{\{u\},t_j} = 1\} P\{X_{\{v\},t_j} = 1\} = \frac{p}{n}\frac{p}{n} = \frac{p^2}{n^2}$$

Tada je verovatnoća da su čvorovi u i v obojeni u nekom zajedničkom vremenskom trenutku t_j jednaka:

$$P\{\exists t_j \mid X_{\{u,v\},t_j} = 1\} = P\{\bigsqcup_{j=1}^{\frac{n}{p}} \{X_{\{u,v\},t_j} = 1\}\} = \sum_{j=1}^{\frac{n}{p}} P\{X_{\{u,v\},t_j} = 1\} = \sum_{j=1}^{\frac{n}{p}} \frac{p^2}{n^2} = \frac{p}{n}$$

Označimo sa N slučajnu veličinu koja predstavlja broj grana $e = (u_1, u_2)$ čiji se čvorovi u_1 i u_2 boje konkurentno. Neka je data indikatorska slučajna veličina

procesorima.

$I_{(u,v)}$ koja uzima vrednost 1 ukoliko se oba čvora grane (u, v) boje konkurentno u nekom vremenskom intervalu t_j i 0 u suprotnom. Tada slučajna veličina $I_{(u,v)}$ ima raspodelu:

$$\begin{pmatrix} 0 & 1 \\ 1 - \frac{p}{n} & \frac{p}{n} \end{pmatrix}$$

Tada važi:

$$E[N] = E[\sum_{(u,v)\in E} I_{(u,v)}] = \sum_{(u,v)\in E} E[I_{(u,v)}] = \sum_{(u,v)\in E} \frac{p}{n} = \frac{mp}{n} = \mu p$$

Neka je sa C označena slučajna veličina koja predstavlja broj grana koje pripadaju nekim od skupova β_i i β_j a čiji se čvorovi boje istom bojom. Tada važi da je $C \leq N$ pa zbog toga važi i da je $E[C] = O(\mu p)$ □

Lema 9. *Ukoliko važi $p = O(\sqrt{\frac{n^2}{m}})$, tada prethodno navedena heuristika (Heuristika 20) ima očekivanu vremensku složenost $O(\frac{\Delta \cdot n}{p})$, pod pretpostavkom korišćenja CRCW modela.*

Dokaz: Očekivano vreme izvršavanja prve faze heuristike iznosi $O(\frac{\Delta \cdot n}{p})$. U drugoj fazi heuristike se vrši detektovanje onih čvorova koje je potrebno ponovo obojiti nakon čega se vrši njihovo bojenje pomoću sekvencijalne heuristike. Analizirajmo očekivanu vremensku složenost druge faze heuristike. Najpre se izvršava paralelna `for` petlja koja formira skup S u vremenu $O(\frac{\Delta \cdot n}{p})$. Zatim se vrši bojenje skupa S. Skup S ima onoliko elemenata koliko je bilo konflikata u fazi pseudobojenja. Bojenje skupa S sekvencijalnom heuristikom se izvršava za vreme $O(\Delta \cdot K)$, gde je K broj konflikata u pseudobojenju. Iz ovoga zaključujemo da je očekivana složenost druge faze $O(\frac{\Delta \cdot n}{p}) + O(\Delta \cdot K)$. Iz prethodne leme imamo da je $E[K] = O(\mu p)$ pa je ukupna očekivana složenost druge faze $O(\frac{\Delta \cdot n}{p}) + O(\Delta \cdot \mu p)$. Izvedimo sada asimptotsko poređenje veličina $\frac{\Delta \cdot n}{p}$ i $\Delta \cdot \mu p$. Primetimo da je dovoljno uporediti veličine $\frac{n}{p}$ i μp.
Prvi slučaj ($\frac{n}{p} >> \mu p$):

$$\frac{n}{p} >> \mu p \Leftrightarrow \frac{n}{p} >> \frac{mp}{n} \Leftrightarrow \frac{n^2}{m} >> p^2 \Leftrightarrow \sqrt{\frac{n^2}{m}} >> p \Leftrightarrow p = O(\sqrt{\frac{n^2}{m}})$$

Iz uslova teoreme vidimo da je poslednji uslov ispunjen pa važi pa je ukupna složenost druge faze u ovom slučaju $O(\frac{\Delta \cdot n}{p})$.

Drugi slučaj ($\frac{n}{p} << \mu p$):

$$\frac{n}{p} << \mu p \Leftrightarrow \sqrt{\frac{n^2}{m}} << p \Leftrightarrow p = \Omega(\sqrt{\frac{n^2}{m}})$$

Međutim, pošto je $p = O(\sqrt{\frac{n^2}{m}})$ onda važi da je: $p = \Theta(\sqrt{\frac{n^2}{m}}) = \Theta(\frac{n}{\sqrt{m}})$. Tada važi: $\sqrt{m} = \Theta(\frac{n}{p})$. Dalje zaključujemo da važi:

$$O(\frac{\Delta \cdot n}{p}) \ + \ O(\Delta \cdot \mu p) = O(\frac{\Delta \cdot n}{\frac{n}{\sqrt{m}}}) \ + \ O(\Delta \cdot \frac{m}{n}\frac{n}{\sqrt{m}}) = O(\Delta \cdot \sqrt{m}) \ + \ O(\Delta \cdot \sqrt{m})$$

$$= O(\Delta \cdot \sqrt{m}) = O(\frac{\Delta \cdot n}{p})$$

Analizirajući prethodna dva slučaja zaključujemo da je u oba očekivana složenost druge faze jednaka $O(\frac{\Delta \cdot n}{p})$. Ukupna očekivana složenost heuristike je zbir očekivanih složenosti prve i druge faze i iznosi $O(\frac{\Delta \cdot n}{p})$, sto predstavlja očekivani broj konflikata u bojenju nakon prve faze heuristike. □

4.5.4 Podela skupa čvorova u blokove

Razmotrimo načine na koje se skup čvorova V grafa $G = (V, E)$ može podeliti u p blokova $\beta_1, \beta_2, \ldots, \beta_p$ tako da važi: $|\beta_i| = \frac{n}{p}$ i $V = \bigsqcup_{i=1}^{\frac{n}{p}} \beta_i$. Postoji veliki broj različitih načina za deljenje skupa V u blokove. Najbolje bi bilo da podela u blokove bude takva da je broj grana koji je deljen među blokovima β_i i β_j, gde je $1 \leq i < j \leq p$, minimalan. Ovakvom podelom u blokove bi se minimizovao broj konflikata u bojenju i vreme potrebno za rešavanje problema koje donose konflikti. Međutim, dobiti takvo jedno particionisanje je teško. Umesto toga razmotrićemo jednostavniju strategiju. U nastavku će biti predstavljena strategija koja čvor $v_i \in V$ dodeljuje bloku čiji se indeks dobija kao ostatak pri deljenju broja i sa p. Na taj način će svaki blok β_i imati jednak broj čvorova.

Heuristika se sastoji iz dve paralelne `for` petlje. Uz pretpostavku mogućnosti korišćenja dovoljnog broja procesora p, obe `for` petlje se izvršavaju za vreme $O(1)$ pa je ukupna složenost heuristike $O(1)$. Iako je prednost date heuristike u dobroj vremenskoj složenosti ona ne daje garanciju za broj grana koje će biti deljene između blokova β_i i β_j. Međutim, za potrebe brzog particionisanja u proseku daje dobre rezultate.

Heuristika 21 Heuristika za blokovsko particionisanje skupa čvorova u grafu

Naziv heuristike: *EvaluateBlocks(G, p)*
Ulaz: Neusmeren graf $G = (V, E)$ i prirodan broj p
Izlaz: Skup blokova $\{\beta_1, \beta_2, \ldots, \beta_p\}$ u grafu G

1: **for** $i = 1$ to p in parallel **do**
2: $\quad \beta_i = \emptyset$
3: **end for**
4: **for** $i = 1$ to n in parallel **do**
5: $\quad i' =$ mod (i, p)
6: $\quad \beta_{i'} = \beta_{i'} \cup \{v_i\}$
7: **end for**
8: **return** $\{\beta_1, \beta_2, \ldots, \beta_p\}$

4.5.5 Napredna heuristika blok particionisanja

Heuristika se sastoji od dve faze. Prva faza je u velikoj meri slična prvoj fazi prethodno opisane heuristike: na početku se vrši deljenje skupa čvorova u p blokova iste veličine, nakon čega se vrši njihovo paralelno pseudobojenje. Nakon toga se pomoću skupa $Colors$ vrši prebrojavanje boja koje su iskorišćene u prethodnom koraku pseudobojenja. Rezultat prebrojavanja se smešta u brojačku promenljivu k. Nakon toga se čvorovi grupišu u k grupa: $g_1, g_2, \ldots, g_k$ tako da se u grupi g_i nalaze samo čvorovi obojeni bojom indeksa i. Za takve grupe kažemo da su *pseudonezavisne* (eng. *pseudoindependent*) zato što one ne moraju predstavljati nezavisne skupove kao što je to bio slučaj kod modifikovane *First Fit* heuristike.

Na početku druge faze heuristike se svaka grupa g_i deli u p jednakih blokova $\beta'_1, \beta'_2, \ldots, \beta'_p$, nakon čega se vrši paralelno bojenje svakog od datih blokova. Primetimo da nakon ovog koraka može doći do konflikata u bojenju, ali se očekuje da broj konflikata bude manji neko što je bio nakon izvršavanja prve faze heuristike. Nakon koraka pseudobojenja pseudogrupa $g_1, g_2, \ldots, g_k$ vrši se detekcija konflikata u bojenju nakon čega se vrši ispravka konflikata. Ovi koraci se vrše na isti način kao i kod prethodno opisane heuristike. Heuristika je data u nastavku.

Heuristika 22 Napredna heuristika bojenja zasnovana na blokovskom particionisanju

Naziv heuristike: *ParallelAdvancedColoringWithBlockPartitioning(G, p)*
Ulaz: Neusmeren graf $G = (V, E)$, prirodan broj p

1: //Prva faza heuristike:
2: Izvršiti particionisanje skupa čvorova V u p blokova: $\beta_1, \beta_2, \ldots, \beta_p$ tako da važi: $|\beta_i| = \frac{|V|}{p} = \frac{n}{p}$

3: **for** $i = 1$ to p in parallel **do**
4: **for** each v in β_i **do**
5: $CalculateNodeColor(v, G)$
6: **end for**
7: **end for**

8: $Colors = \bigcup_{v \in V} \{v.Color\}$
9: $k = Length(Colors)$
10: Odrediti pseudonezavisne grupe čvorova: $g_1, g_2, \ldots, g_k$, gde važi: $g_i = \{v \in V \mid v.Color = i\}$

11: //Druga faza heuristike:
12: $S = \{\}$
13: **for** i = k down to 1 **do**
14: Izvrši particionisanje grupe g_i u p jednakih blokova: $\beta'_1, \beta'_2, \ldots, \beta'_p$
15: **for** $j = 1$ to p in parallel **do**
16: **for** each w in β'_i **do**
17: $CalculateNodeColor(w, G)$
18: **end for**
19: **end for**
20: **end for**

21: Izvršiti naredbe definisane u drugoj fazi prethodne heuristike:

- Detektovati konflikte u bojenju i rezultat smestiti u skup S
- Obojiti čvorove skupa S pomoću sekvencijalne heuristike

Neka je broj konflikata nakon izvršavanja prve faze prethodno navedene heuristike jednak C. Može se pokazati da je nakon izvršavanja druge faze prethodno navedene heuristike (Heuristika 22) očekivani broj konflikata u bojenju jednak $O(C\frac{\Delta \cdot \mu p}{m})$. Detalji dokaza se mogu naći u radu [5].

4.6 Metaheuristike za bojenje grafova

Na kraju napomenimo da postoji dosta metaheuristika za rešavanje problema bojenja grafova. Ispostavlja se da metaheuristike za rešavanje problema bojenja grafa u velikom broju slučajeva daju jako dobre rezultate. U ovom radu se nećemo baviti metaheuristikama, ali ćemo navesti popularne klase metaheuristika koje su se pokazale dobro za rešavanje pomenutog problema. To su:

- *Evolutivni algoritmi* (eng. *evolutionary algorithms*),
- *Celobrojno programiranje* (eng. *integer programming*),
- *Metoda promenljivih okolina* (eng. *variable neighborhood search*),
- *Metoda tabu pretrage* (eng. *tabu search*).

Za više informacija o metaheuristikama za bojenje grafova pogledati [4].

Glava 5

Programska realizacija algoritama i evaluacija

U okviru ove glave biće opisana programska realizacija aplikacije koja omogućava odabir heuritike bojenja, vizuelnu reprezentaciju obojenog grafa kao i komparativnu analizu broja boja i vremena izvršavanja različitih heuristika. Na kraju glave je opisana evaluacija rezultata rada različitih heuristika, kao i njihovo poređenje na istim instancama slučajno generisanih grafova.

5.1 Aplikacija za bojenje grafova

Za implementaciju aplikacije je korišćen programski jezik *C#* kao i radno okruženje *.NET 6*. Za implementaciju korisničkog interfejsa su korišćene *windows forme*. Aplikacija ima sledeće mogućnosti:

1. Generisanje slučajnog grafa sa n čvorova (unošenjem broja čvorova grafa u tekstualno polje i klikom na dugme *Genarate random graph*).

2. Bojenje slučajno generisanog grafa pomoću odabrane heuristike iz padajuće liste algoritama (klikom na dugme *Process Coloring*).

3. Prikaz slučajno generisanog grafa u glavnom prozoru (klikom na dugme *Visualize Results*).

4. Merenje vremena potrebnog za bojenje grafa kao i prikaz informacije o broju boja koje je odabrana heuristika iskoristila - nakon pritiska na dugme *Proc-*

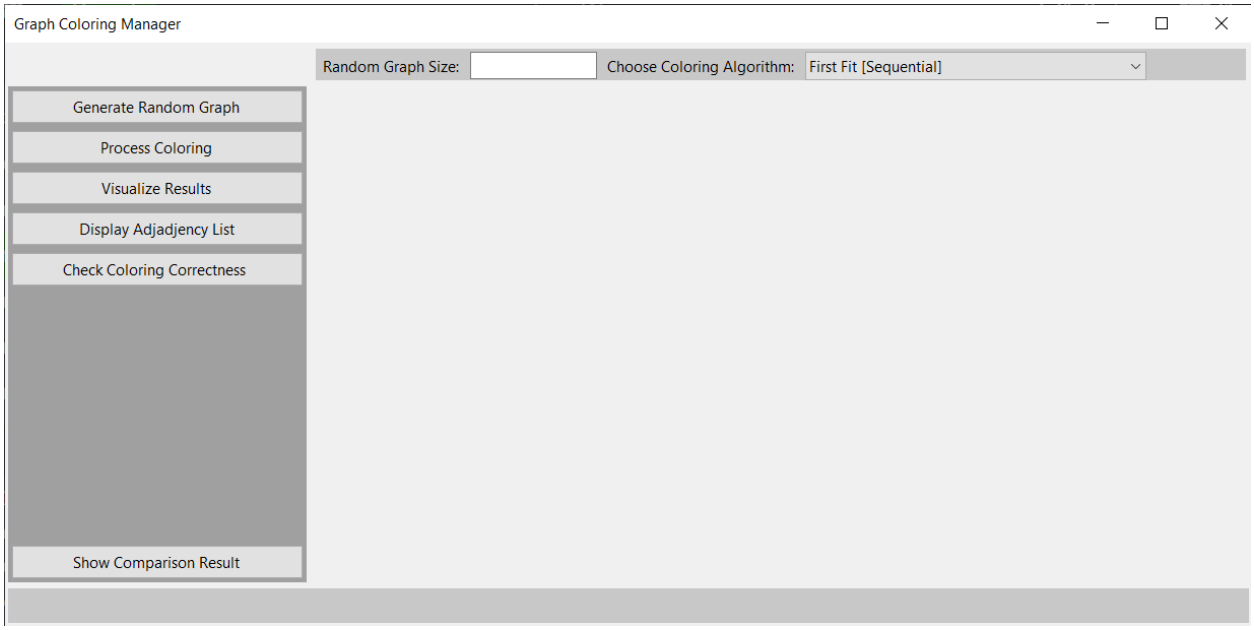
cess Coloring na dnu forme biće prikazane informacije o vremenu izvršavanja heuristike i broju boja koje je heuristika iskoristila.

5. Provera ispravnosti bojenja grafa - nakon generisanja slučajnog grafa i njegovog bojenja pritiskom na dugme *Check Coloring Correctness* prikazuje se polje sa porukom da li je bojenje ispravno ili da postoje susedni čvorovi koji su obojeni istom bojom.

6. Prikaz liste povezanosti slučajno generisanog grafa i informacija o bojama (u ARGB formatu) koje su dodeljene čvorovima (pritiskom na dugme *Display Adjacency List*).

7. Prikaz poređenja vremena izvršavanja i broja korišćenih boja za različite heuristike pomoću kojih je vršeno bojenje poslednjeg slučajno generisanog grafa (klikom na dugme *Show Comparison Results*).

Za vizuelni prikaz grafa se koristi Viewer kontrola dobijena instaliranjem *Microsoft.Msagl.GraphViewerGDI* NuGet paketa. Nakon pokretanja aplikacije otvara se forma koja je prikazana na slici 5.1.



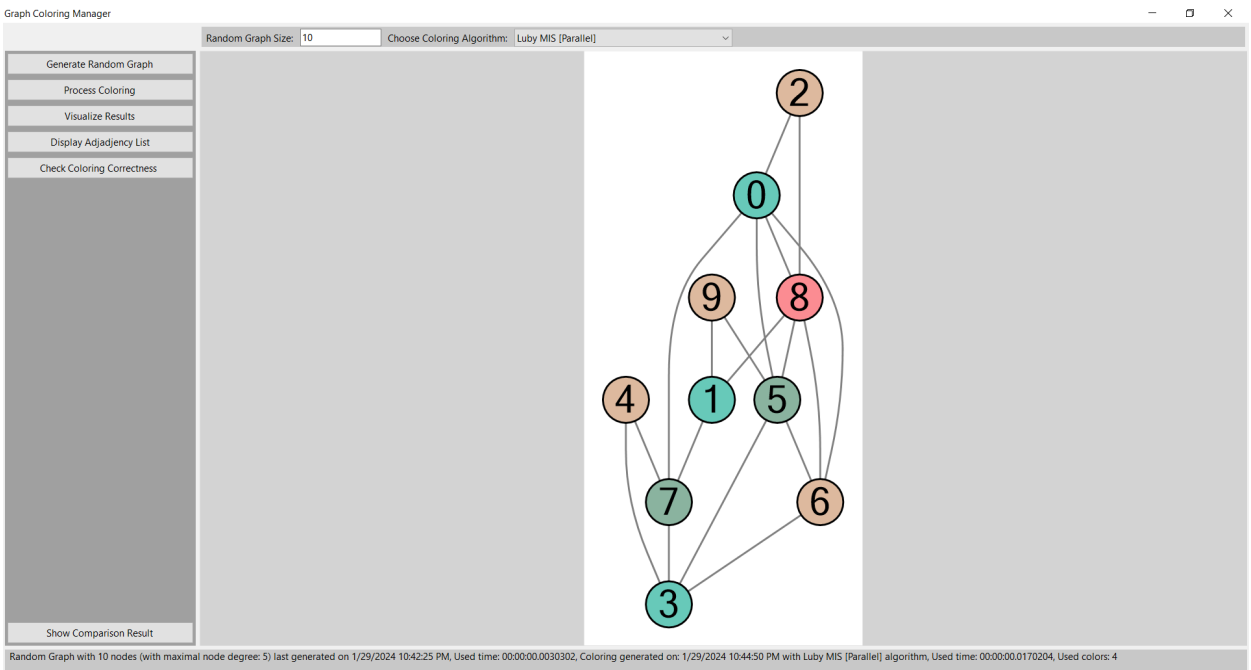
Slika 5.1: Izgled glavne forme nakon pokretanja aplikacije

Nakon generisanja slučajnog grafa u liniji na dnu glavne forme biće prikazane sledeće informacije: kada je poslednji put generisan slučajni graf, koliki je maksimalni stepen čvora u grafu i vreme potrebno za njegovo generisanje.

Nakon uspešnog bojenja grafa linija na dnu glavne forme će biti dopunjena sledećim informacijama:

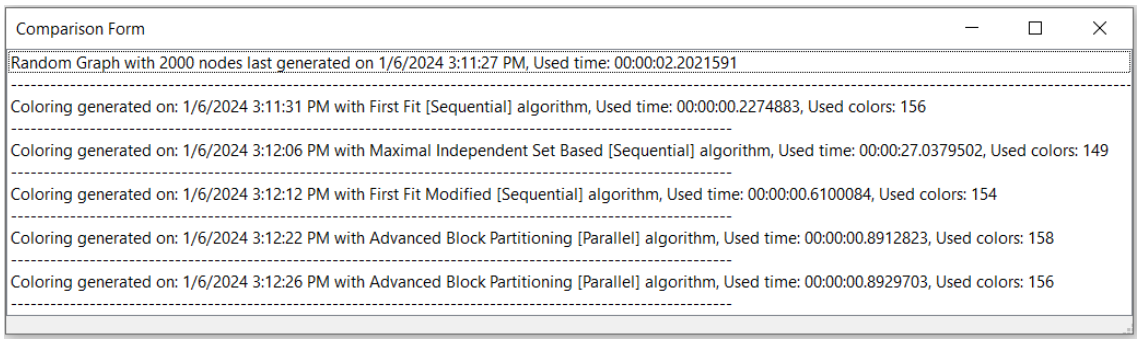
1. kada je poslednji put izvršeno bojenje grafa,
2. algoritam koji je korišćen za bojenje,
3. vreme izvršavanja heuristike za bojenje,
4. broj boja koje je heuristika iskoristila.

Pod pretpostavkom da je korisnik prethodno generisao slučajni graf, nakon pritiska na dugme *Visualize Results* u glavnom prozoru biće prikazan slučajno generisani graf. Ukoliko je izvršeno bojenje grafa čvorovi grafa u glavnom prozoru biće obojeni saglasno tom bojenju. Na slici 5.2 je prikazan izgled glavne forme nakon generisanja slučajnog grafa sa 10 čvorova i njegovog bojenja.



Slika 5.2: Prikaz obojenog grafa

Za potrebe poređenja performansi heuristika na istom slučajno generisanom grafu aplikaciji je dodato dugme *Show Comparison Result*. Prilikom generisanja novog slučajnog grafa, poruka koja se ispisuje na klik ovog dugmeta se postavlja tako da sadrži samo informacije kada je graf generisan i koliko je bilo potrebno vremena za njegovo generisanje. Svakim narednim bojenjem grafa ova poruka se dopunjuje informacijama o bojenju koje je heuristika izvršila (kada je generisano bojenje, utrošeno vreme heuristike, broj iskorišćenih boja). Na slici 5.3 je prikazan primer sadržaja forme koja se dobija pritiskom na dugme *Show Comparison Result*.



Slika 5.3: Prikaz forme za poređenje heuristika

5.2 Zaštita deljenih resursa

Kako bi se očuvala konzistentnost podataka i omogućila konkurentna brisanja i pisanja u kolekciju podataka, prilikom pisanja aplikacije su korišćene takozvane *Thread Safe* kolekcije iz prostora imena *System.Collections.Concurrent*. Pomenute kolekcije predstavljaju proširenja nekih standardnih kolekcija iz prostora imena *System.Collections.Generics*. Naime, kolekcije iz prostora imena *System.Collection-.Generics* (u koje se ubrajaju *List*, *HashSet*, *Dictionary*,...) predstavljaju kolekcije koje zahtevaju dodatnu sinhronizaciju niti koje konkurentno dodaju ili brišu sadržaj iz kolekcije, jer takve kolekcije u sebi ne sadrže mehanizme sinhronizacije. Iako je za konkurentna dodavanja i brisanja kod ovakvih kolekcija potrebna sinhronizacija, one se mogu koristiti za konkurentna čitanja vrednosti. S druge strane, o sinhronizaciji niti ne moramo voditi računa kada se koriste *Thread Safe* kolekcije jer takve kolekcije implicitno u sebi sadrže mehanizme za zaštitu konzistetnosti podatka. *Thread Safe* kolekcije koje su korišćene u implementacijama heuristika su:

1. *ConcurrentDictionary - Thread Safe* kolekcija koja je pandan kolekciji *Dictionary* iz *System.Collections.Generic* prostora imena.

2. *ConcurrentBag* koja predstavlja neuređenu *Thread Safe* kolekciju. Ova kolekcija ne garantuje očuvanje redosleda elemenata koji se dodaju u kolekciju.

Pored navedenih kolekcija za omogućavanje ekskluzivnog pristupa podatku su korišćeni `lock` blokovi. Oni omogućavaju da se resursi zaključaju za sve niti sem za trenutnu nit (koja ima ekskluzivan pristup resursu) i otključaju kada trenutna nit završi obradu resursa. Primer korišćenja `lock` bloka za bezbedno dodeljivanje boje čvoru grafa kada više niti ima mogućnost da konkurentno boji čvorove grafa je prikazan na slici 5.4. Objekat `lockObject` koristimo za zaključavanje dela koda kojem više niti može imati istovremeni pristup. Ovaj objekat može biti definisan na početku klase na sledeći način:

```
private readonly object lockObject = new object();
```

```
lock (lockObject)
{
    node.Color = color;
}
```

Slika 5.4: Primer upotrebe lock bloka

5.3 Evaluacija

U nastavku su predstavljeni rezultati poređenja heuristika za bojenje grafova na istim instancama slučajno generisanih grafova. Poređenje heuristika je izvedeno uzimajući u obzir dva aspekta:

1. vreme izvršavanja heuristike,

2. broj boja koji je heuristika koristila za bojenje grafa.

Eksperimentalna poređenja su vršena na računaru koji ima sledeće specifikacije:

1. Procesor - Intel Core i5-8250U

2. RAM memorija - 16.0GB 664MHz DDR4

3. Matična ploča - LENOVO 20M8S0YQ00 (U3E1)

Prilikom merenja parametara heuristike za poređenje nije bio pokrenut ni jedan program koji bi mogao značajno uticati na rezultate istih. Treba uzeti u obzir da svi sekvencijalne heuristike kao rezultat rada uvek daju bojenje istog kvaliteta, bez obzira na to koliko puta je sekvencijalna heuristika pokrenuta. Kod paralelnih heuristika koje su razmatrane u ovom radu to ne važi - ponovnim pokretanjem bilo koje paralelne heuristike se može dobiti bojenje drugačijeg kvaliteta u odnosu na kvalitet bojenja dobijen prethodnim pokretanjem iste heuristike.

Rezultati eksperimenata

Svaka heuristika je pokrenuta nad pet istih slučajno generisanih grafova koji imaju redom 2000, 3500, 5000, 8000 i 10000 čvorova i maksimalne stepene 785, 1363, 1921, 3030 i 3796 respektivno. U tabelama 5.1 i 5.2 su prikazani rezultati poređenja utrošenog vremena i broja boja koje su koristile heuristike. Svaka sekvencijalna heuristika je nad prethodno opisanim grafovima pokrenuta tačno jednom nakon čega je za svaki graf u tabelu 5.1 upisan rezultat. Paralelne heuristike su pokretane po 3 puta nad svakim grafom nakon čega je za svaki graf u tabelu 5.2 upisan rezultat koji ima najbolji kvalitet bojenja. U okviru liste koja je data u nastavku su dati detalji nekih heuristika čije su performanse merene.

1. *MIS* - sekvencijalna heuristika bazirana na odabiru nezavisnih skupova grafa. Za konstrukciju nezavisnih skupova grafa je korišćena pohlepna heuristika koja prioritet daje čvorovima najvećeg stepena. Ukoliko se ispostavi da dva čvora koja se razmatraju imaju isti stepen, prioritet za ulazak u maksimalni nezavisni skup dobija onaj čvor v koji ima veći sučajno dodeljeni prioritet $\rho(v)$.

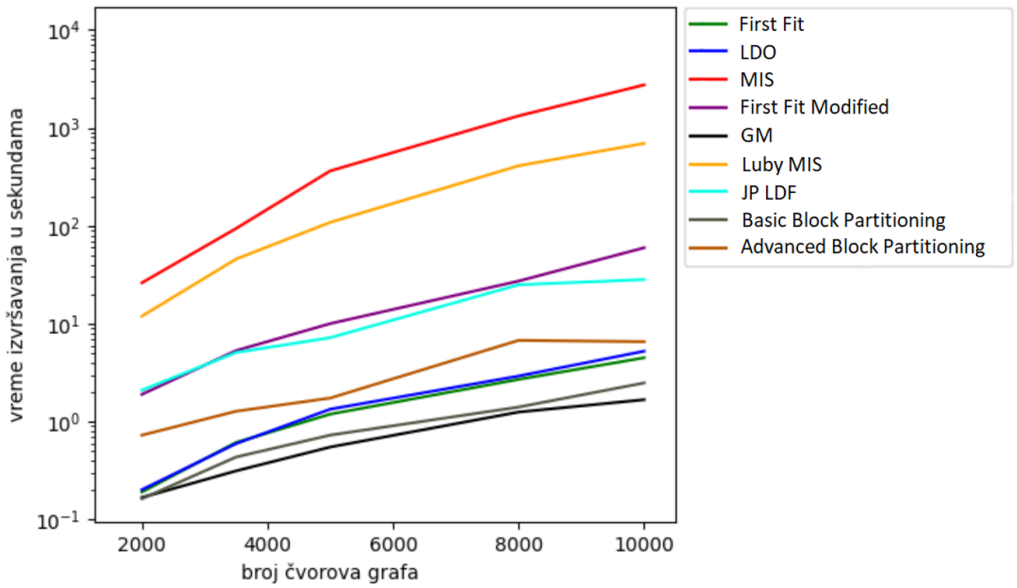
2. *First Fit Modified*. Za registrovanje progresa bojenja je korišćena funkcija cilja koja predstavlja broj različitih boja čvorova u aktuelnoj iteraciji glavne `while` petlje. Radi poboljšanja kvaliteta bojenja preduzet je sledeći korak: nakon što se pomoću funkcije cilja registruje da ne postoji progres u bojenju grafa, glavna `while` petlja će imati dodatne tri iteracije.

3. *Luby MIS* - poznata paralelna heuristika za bojenje zasnovana na odabiru nezavisnih skupova sa malom izmenom. Pri konstrukciji maksimalnih nezavisnih skupova prioritet za dodavanje čvora u rezultujući skup dobijaju oni čvorovi koji imaju najveći stepen. U slučaju da dva čvora imaju isti stepen prioritet dobija onaj čvor v kome je dodeljen veći broj $\rho(v)$. Ovaj princip je detaljnije opisan u opisu *Luby MIS* heuristike (Heuristika 15) u okviru prethodne glave.

4. *JP LDF* je poznata modifikacija *Jones Plassman* heuristike koja pri konstrukciji maksimalnog nezavisnog skupa prednost daje onim čvorovima koji imaju veći stepen od stepena svih svojih suseda.

5. *Block Partitioning* - osnovna heuristika zasnovana na blokovskom particionisanju koja koristi heuristiku *EvaluateBlocks(G, p)* (Heuristika 21) za konstrukciju blokova $\beta_1, \beta_2, ..., \beta_p$. Za bojenje čvorova u drugoj fazi heuristike (nakon detekcije konflikata u bojenju) je korišćena modifikovana *First Fit* heuristika.

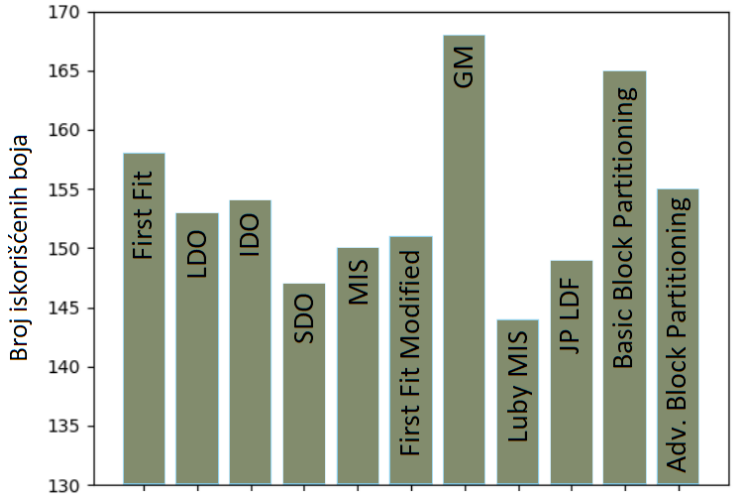
6. *Advanced Block Partitioning* - paralelna heuristika koja koristi ideju naprednog blokovskog particionisanja. Za bojenje čvorova u drugoj fazi heuristike (nakon detekcije konflikata u bojenju) je korišćena modifikovana *First Fit* heuristika. Ova heuristika takođe koristi heuristiku *EvaluateBlocks(G, p)* (Heuristika 21).



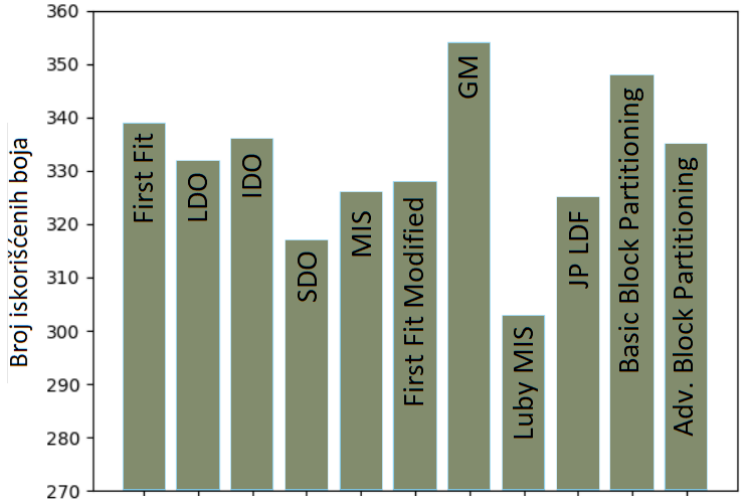
Slika 5.5: Poređenje vremena izvršavanja

Poređenja vremena izvršavanja heuristika su prikaza na slici 5.5. Grafik prikazan na ovoj slici koristi logaritamsku skalu po y-osi i prikazuje zavisnost vremena izvršavanja heuristika od veličine grafa, odnosno broja čvorova slučajno generisanog grafa. S obzirom na to da *IDO* i *SDO* heuristike imaju znatno duže vreme izvršavanja od svih ostalih heuristika, analize njihovih vremena izvršavanja nisu prikazane.

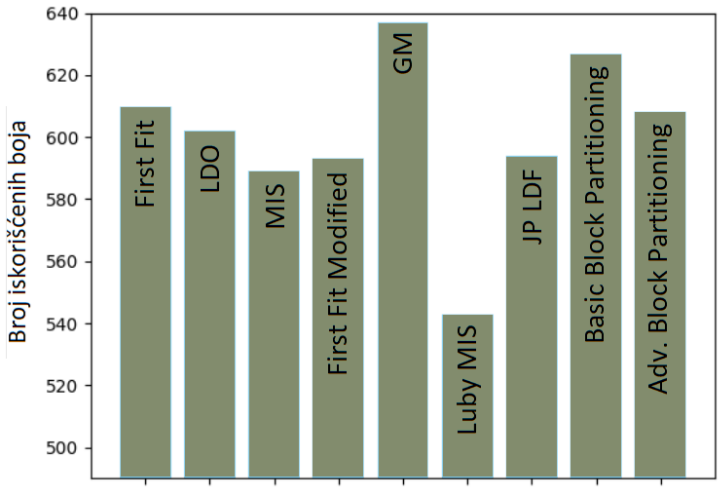
U nastavku, na slikama 5.6, 5.7 i 5.8 su predstavljena poređenja broja iskorišćenih boja kada se na ulaz heuristika dovedu tri prethodno opisana slučajna grafa sa 2000, 5000 i 10000 čvorova.



Slika 5.6: Poređenje broja iskorišćenih boja za graf sa 2000 čvorova



Slika 5.7: Poređenje broja iskorišćenih boja za graf sa 5000 čvorova



Slika 5.8: Poređenje broja iskorišćenih boja za graf sa 10000 čvorova

Na osnovu rezultata prikazanih u tabeli 5.1 doneti su sledeći zaključci[1]:

1. *First Fit* heuristika je najbrža sekvencijalna heuristika za bojenje grafa. Ova heuristika je brža od većine paralelnih heuristika. Jedine paralelne heuristike koje mogu biti brže od ove heuristike su *Basic Block Partitioning* i *GM* heuristike. Kada je u pitanju kvalitet bojenja, ova heuristika ne daje dobre rezultate.

2. *SDO* heuristika kao rezultat daje bojenje jako dobrog kvaliteta, međutim izvršava se jako sporo - pokazala se kao najsporija od svih heuristika.

3. *MIS* heuristika kao rezultat daje bojenje dosta dobrog kvaliteta i može predstavljati dobru sekvencijalnu alternativu *SDO* heuristici. Međutim, iako se primećuje ubrzanje u odnosu na *SDO* heuristiku, njeno vreme izvršavanja je veliko pa se preporučuje razmatranje drugih heuristika koje su bazirani na odabiru nezavisnih skupova kao što su *Luby MIS* i *Jones Plassman* heuristike.

4. *First Fit Modified* pruža bojenje dobrog kvaliteta za relativno kratko vreme pa iz tih razloga može predstaviti dobar izbor pri odabiru heuristike za bojenje. Primetno je znatno ubrzanje u odnosu na *SDO* i *MIS* heuristike.

5. *GM* heuristika u prosečnom slučaju ima najkraće vreme izvršavanja u poređenju sa ostalim heuristikama. Međutim, u prosečnom slučaju ova heuristika daje

[1]Prokomentarisane su samo heuristike koje se ističu svojim rezultatima

bojenje najlošijeg kvaliteta u poređenju sa kvalitetom bojenja koje pokazuju rezultati ostalih heuristika.

6. *Luby MIS* heuristika pruža znatno ubrzanje u odnosu na sekvencijalnu heuristiku koja je bazirana na odabiru nezavisnih skupova. Velika prednost ove heuristike je u kvalitetu bojenja koje dobijemo njenom upotrebom - u slučaju svih pet grafova ova heuristika je dala najbolje rezultate kada je u pitanju kvalitet bojenja. Kada je vreme izvršavanja paralelnih heuristika u pitanju, ova heuristika je sporija od svih paralelnih algoritama.

7. *Jones Plassman LDF* daje slične rezultate kao i *Luby MIS* heuristika kada je u pitanju kvalitet bojenja. Velika prednost ove heuristike počiva u njenoj brzini - ona pruža znatno ubrzanje u odnosu na *Luby MIS* heuristiku a pritom se dobija bojenje koje koristi sličan broj boja kao *Luby MIS* heuristika.

8. *Basic Block Partitioning* može da bude dobra alternativa *GM* heuristici jer ima jako kratko vreme izvršavanja a u proseku daje bolji kvalitet bojenja u odnosu na *GM* heuristiku. Međutim, problem može da predstavlja to što ova heuristika često rezultuje bojenjem koje je lošijeg kvaliteta čak i od *First Fit* heuristike.

9. *Advanced Block Partitioning* može rezultovati boljim bojenjem u odnosu na bojenje koje daje *First Fit* heuristika. Međutim, većina sekvencijalnih heuristika kao i *Luby MIS* i *Jones Plassman* heuristike u proseku rezultuju kvalitetnijim bojenjem. S druge strane, ova heuristika predstavlja jednu od najbržih heuristika u poređenju sa ostalim razmatranim heuristikama pa iz tih razloga može predstavljati dobar izbor. Međutim, da bi se iskoristili svi benefiti ove heuristike treba razmotriti bolji način konstrukcije blokova $\beta_1, \beta_2, ..., \beta_p$ kako bi se dobilo kvalitetnije bojenje grafa.

U tabeli 5.1 nisu prikazani razultati eksperimenta za *IDO* i *SDO* heuristika u slučaju kada je ulaz slučajni graf od 10000 čvorova, zbog veoma velikog vremena izvršavanja. Veme izvršavanja pomenutih heuristika u tom slučaju je bilo preko tri sata, zbog čega su one heuristike pre kraja njihovog rada.

Analizirajmo, na kraju, u kojim slučajevima se preporučuje upotreba najistaknutijih heuristika. Ukoliko se pri rešavanju praktičnog problema zahteva da vreme izvršavanja heuristike bude minimalno, onda treba upotrebiti *GM* heuristiku ili osnovnu heuristiku baziranu na blokovskom particionisanju. Ukoliko se zahteva da

rezultujuće bojenje bude što boljeg kvaliteta, a da pritom vreme izvršavanja heuristike nije u prvom planu, onda treba upotrebiti *Luby MIS* heuristiku. Ukoliko je potrebno ostvariti što bolji kvalitet bojenja za što kraće vreme, onda treba upotrebiti *Jonnes Plassman LDF* heuristiku.

Tabela 5.1: Eksperimentalno poređenje sekvencijalnih heuristika

Naziv heuristike	n	**Vreme izvršavanja [s]**	**Broj boja**
First Fit	2000	0.191	158
	3500	0.613	253
	5000	1.192	339
	8000	2.698	504
	10000	4.489	610
LDO	2000	0.202	153
	3500	0.596	248
	5000	1.34	332
	8000	2.919	497
	10000	5.241	602
SDO	2000	186.201	147
	3500	954.544	234
	5000	3359.38	317
	8000	11124.064	477
	10000	-	-
IDO	2000	116.014	154
	3500	593	247
	5000	1718.171	336
	8000	2727.799	500
	10000	-	-
MIS	2000	26.148	150
	3500	93.854	238
	5000	363.487	326
	8000	1324.262	488
	10000	2744.244	589
First Fit Modified	2000	0.661	154
	3500	5.313	242
	5000	10.019	328
	8000	27.246	490
	10000	59.593	593

Tabela 5.2: Eksperimentalno poređenje paralelnih heuristika

Naziv heuristike	n	**Vreme izvršavanja [s]**	**Broj boja**
GM	2000	0.168	168
	3500	**0.314**	267
	5000	**0.550**	354
	8000	**1.252**	526
	10000	**1.674**	637
Luby MIS	2000	11.934	**144**
	3500	45.7153	**226**
	5000	108.374	**303**
	8000	409.941	**447**
	10000	693.741	**543**
Jones Plassman LDF	2000	2.095	149
	3500	5.08	240
	5000	7.202	325
	8000	24.968	490
	10000	28.210	594
Basic Block Partitioning	2000	**0.163**	165
	3500	0.4356	260
	5000	0.728	348
	8000	1.408	516
	10000	2.484	627
Advanced Block Partitioning	2000	0.727	155
	3500	1.276	248
	5000	1.740	335
	8000	6.761	502
	10000	6.553	608

Glava 6

Zaključak

Problem bojenja grafova predstavlja temu koja iziskuje posebnu pažnju i analizu različitih tehnika za rešavanje ovog problema. Uverili smo se da postoje razne sekvencijalne heuristike za približno rešavanje ovog problema. U većini slučajeva pomenute heuristike su zahtevale poboljšanja, kako po pitanju njihovog vremena izvršavanja tako i po pitanju kvaliteta bojenja koje one pružaju. Pokazano je da paralelne heuristike za bojenje grafova mogu doprineti i ubrzanju izvršavanja i boljem kvalitetu bojenja.

Prilikom odabira heuristike za bojenje posebna pažnja se mora obratiti na to da li je potrebno brzo obojiti graf, dobiti bojenje najboljeg mogućeg kvaliteta ili postići oboje. U slučaju da je potrebno obojiti graf za što kraće vreme, a pritom kvalitet bojenja nije u prvom planu - preporučuje se upotreba *GM* heuristike ili osnovne heuristike zasnovane na blokovskom particionisanju. Ukoliko je potrebno ostvariti bojenje najboljeg mogućeg kvaliteta preporučuje se upotreba *Luby MIS* paralelne heuristike. Ukoliko dužina vremena izvršavanja *Luby MIS* heuristike predstavlja problem, a da je pritom potrebno ostvariti bojenje što boljeg kvaliteta, umesto ove heuristike se može upotrebiti *Jones Plassman LDF* heuristika. Zbog svog relativno kratkog vremena izvršavanja i dosta dobrog kvaliteta bojenja u većini slučajeva ova heuristika predstavlja najbolji mogući izbor između svih opisanih heuristika.

Heuristike bazirane na blokovskom particionisanju grafa predstavljaju nov i vrlo zanimljiv pristup bojenju grafa. Napredna heuristika blokovskog particionisanja ima potencijal za ubrzanje izvršavanja i dobijanje dobrog kvaliteta bojenja, ali za efikasno rešavanje problema bojenja grafa pomoću ovih heuristika je potrebno razmatranje heuristika koje se bave drugim NP-teškim problemom - problemom particionisanja grafa.

Zanimljiva je činjenica da od svih razmatranih heuristika u radu, kako sekvencijalnih tako i paralelnih, upravo paralelna *Luby MIS* heuristika na svim test primerima daje bojenje najboljeg kvaliteta. Upravo je način konstrukcije nezavisnih skupova kod paralelne *Luby MIS* heuristike zaslužan za kvalitet dobijenog bojenja - prioritet za ulazak u maksimalni nezavisni skup dobijaju čvorovi niskog stepena, a ukoliko se prilikom poređenja ispostavi da dva susedna čvora imaju isti stepen onda se porede njima slučajno dodeljeni prioriteti.

Pored svih navedenih tehnika za rešavanje problema u radu su prodiskutovane i heuristike zasnovane na particionisanju skupa čvorova grafa. Poznato je da je prethodno pomenuti problem particionisanja skupa čvorova grafa NP-težak problem. Pošto ovakve heuristika zahtevaju poznavanje heuristika za particionisanje skupa čvorova grafa, koje nisu opisane u okviru ovog rada, ovakve heuristike mogu biti vrlo zanimljiva tema za dalji rad.

Bibliografija

[1] Karp Richard, *Reducibility Among Combinatorial Problems*, University of California, Berkeley, 1972.

[2] Georges Gonthier, *A computer-checked proof of the Four Colour Theorem*, Microsoft Research Cambridge

[3] Pingfan Li, Xuhao Chen, Zhe Quan, Jianbin Fang, Huayou Su, Tao Tang and Canqun Yang, *High Performance Parallel Graph Coloring on GPGPUs*, 2016.

[4] Rhyd Lewis, *A Guide to Graph Coloring*, Springer 2016.

[5] Assefaw Hadish Gebremedhin, *Parallel Graph Coloring*, University of Bergen, Norway, 1999.

[6] Geoffrey Grimmett, Colin McDiarmid, *On coloring random graphs*, Mathematical proceedings of the the Cambridge philsophical Society, 1975.

[7] Miodrag Živković, *Algoritmi*, Univerzitet u Beogradu, Matematički fakultet, 2000.

[8] Jessica Shi, Yiqiu Wang, Zeyuan Shang, *Parallel maximal independent set algorithms for graphs and hypergraphs*, 6.854 Advanced Algorithms, 2018.

[9] Mark Jones and Paul Plassmann, *A parallel graph coloring heuristic*, SIAM journal of scientfic computing, 1992.

[10] *Microsoft* dokumentacija za lock blok. URL: `https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/statements/lock`

[11] *Microsoft* dokumentacija za thread safe kolekcije. URL: `https://learn.microsoft.com/en-us/dotnet/standard/collections/thread-safe/`