SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 656

Bojanje grafova prilagodljivim metaheurističkim postupcima

Dino Šantl

Zagreb, lipanj 2014.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod **1**

1.1. Sažetci korištenih znanstvenih članaka 1

2. Formalni opis problema **4**

2.1. Tehnički opis problema 4

2.1.1. Ulazni i izlazni podaci 5

2.2. Matematički opis problema 5

2.2.1. Definicije za klasičan problem bojanja grafova 6

2.2.2. Bojanje težinskih grafova 10

2.3. Analiza problema bojanja težinskih grafova 12

2.3.1. Prostor pretrage stanja 12

2.3.2. Pozadina problema iz perspektive računske teorije složenosti . 16

2.3.3. Posljedica "*No free lunch*" teorema 17

3. Algoritmi **19**

3.1. Uvod 19

3.1.1. Mogući problemi 19

3.2. Pohlepni algoritam 20

3.2.1. Neispravnost pohlepnog algoritma 21

3.3. Agentski algoritam 22

3.3.1. Parametri algoritma 23

3.3.2. Detalji algoritma 23

3.3.3. Karakteristike algoritma 25

3.4. Algoritam evolucijske strategije 25

3.4.1. Genetski algoritam 26

3.4.2. Evolucijska strategija 26

3.4.3. Parametri algoritma 27

3.4.4. Detalji algoritma 27

3.4.5. Karakteristike algoritma 28
3.5. Simulirano kaljenje 29
3.5.1. Parametri algoritma 29
3.5.2. Detalji algoritma 29
3.5.3. Karakteristike algoritma 30
3.6. Genetsko kaljenje 31
3.6.1. Parametri algoritma 31
3.6.2. Detalji algoritma 32
3.6.3. Karakteristike algoritma 33
3.7. Pronalaženje najpogodnijeg pohlepnog algoritma 33
3.8. Tuneliranje 34
3.9. Upravljanje postotkom promijenjenih čvorova 35

4. Infrastruktura **36**
4.1. Uvod 36
4.1.1. Modul *input* 37
4.1.2. Modul "*output*" 37
4.1.3. Modul "*algorithm*" 37
4.1.4. Modul "*unit*" 39

5. Rezultati **40**
5.1. Metode testiranja algoritama 40
5.1.1. Testiranje metaheuristika 40
5.2. Slučajno pretraživanje 41
5.3. Agentski algoritam 42
5.4. Simulirano kaljenje 42
5.5. Genetski algoritam 42
5.6. Genetsko kaljenje 42
5.7. Metode strojnog učenja za odair pohlepnog algoritma 42

6. Zaključak **43**

Literatura **44**

1. Uvod

Uvod rada. Nakon uvoda dolaze poglavlja u kojima se obrađuje tema.

1.1. Sažetci korištenih znanstvenih članaka

An Efficient GA with Multipoint Guided Mutation for Graph Coloring Problems [12]

Rad opisuje rješavanje problema bojanja grafova pomoću genetskog algoritma. Posebna pažnja posvećena je operatoru mutacije. Prikaz jedinki je pomoću niza cijelih brojeva. Svakom čvoru u grafu pridružen je jedinstveni cijeli broj - identifikacija. Jedinka je prikazana pomoću niza u kojem je svakoj identifikaciji (čvoru) pridružen broj boje. Funkcija cilja je broj korištenih boja. Inicijalna populacija generira se slučajnim dodjeljivanjem boja svakom čvoru u grafu. Dodatno koristi se operator ispravaka koji popravlja jedinku tako da izabere dva susjedna čvora koji imaju istu boju i zatim jednome od njih promjeni boju. Mutacija se vrši nad jedinkom tako da se prvo reducira broj korištenih boja. To se može napraviti tako da se odaberu dva čvora i pridruži im se boja koja nije jedna od dvije odabrane. Zatim se primijeni operator ispravaka nad tom jedinkom. To se sve ponavlja sve dok se ne generira legalno pobojani graf. Rezultati su uspoređivani s klasičnim implementacijama genetskih algoritama i dobiveni su bolji rezultati ali u zaključku se navodi kako bi algoritam trebalo poboljšati da bi vrijeme izvođenja na većim i kompleksnijim grafovima bilo bolje. Testiranje se obavljalo nad nekim primjerima iz DIMACS baze.

Efficient Graph Coloring With Parallel Genetic Algorithms [7]

Koristi se paralelni genetski algoritam i to migracijski model. Osim toga definirani su i novi genetski operatori. Jedinka je prikazana pomoću particije skupa.

Ispitivanje je vršeno nad DIMACS bazom. Dobiveni su rezultati:

1. Uvijek je bolje koristiti migraciju a ne izolaciju. Za neke operatore može se odrediti najbolji parametar migracije (Transposition mutacija), dok za većinu drugih to nije pronađeno.

2. U migraciji je bolje koristiti najbolje jedinke. Samo u nekim slučajevima kod CEX operatora je bolje koristiti slučajno odabrane jedinke.

3. First Fit je uvijek bolja mutacija od Transposition mutacije. First Fit najbolje se ponaša uz CEX i GPX operator. Transposition mutacija relativno dobro radi sa SPPX operatorom.

4. CEX i GPX operatori su bili najbolji uz korištenje First Fit mutacije. Nakon toga slijedi USIX. SPPX je jako brz operator ali da bi bio uspješan potrebno ga je izvršiti u više iteracija.

5. Najbrži operator križanja je CEX s First Fit mutacijom. Najsporji je UISX.

An ACO Algorithm for the Graph Coloring Problem [13]

Opisuje se jedna inačica ACO (engl. *Ant colony optimization*) algoritma. Ideja je u svakom koraku bojati jednom bojom (odabiru se čvorovi koji će se bojati u zadanu boju). Radi se o proširenju ANTCOL-a. Uz kemijske tragove dodaje se heuristika (npr. broj čvorova koji su iste boje kao i trenutni korak). Graf koji mravi rješavaju nije orginalni graf koji je zadan, već je to graf gdje su povezana dva čvora koja u originalnom grafu nisu susjedna. Prema tome računa se vjerojatnost da se nekom čvoru pridoda neka boja.

Breaking the Symmetry of the Graph Colouring Problem with Genetic Algorithms [9]

Razbija se simetrija u klasičnom problemu bojanja grafova. Tj. različite jedinke mogu predstavljati isto pobojane grafove. Problem se pokušava rješitit tako što se na početku fiksiraju boje za određene čvorove. U našem problemu članak se može koristiti kod pristupa pri pohlepnom algoritmu.

New Graph Coloring Algorithms [1]

Članak koji opisuje heuristike koje se koriste u pohlepnim algoritmima. Članak donosi ideje koje se lako implementiraju i daju dobre rezultate. Neke vrste bojanja i neke vrste

sortiranja ideje vuku iz ovog članka (npr. SDO-LDO).

Graph Coloring Algorithms for Assignment Problems in Radio Networks [3]

Problem koji je veoma sličan praktičnoj primjeni problema koji se rješava. Koriste se različiti algoritmi i uspoređuju se rezultati. Cilj rada je istražiti problem i pokazati koji algoritmi tj. heuristike su efikasne.

Evolutionary Algorithms for Real-World Instances of the Automatic Frequency Planning Problem in GSM Networks [8]

Rad koji je vrlo sličan problematici koja se rješava u diplomskom radu. U radu je primjećeno da operator križanja mora biti vrlo napredan ili ga uopće ne koristiti, te se predlaže korištenje evolucijske strategije (μ, λ)

Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning [6]

Opisuje se korištenje simuliranog kaljenja u optimizacijskim algoritmima. Pokazuje se osnovna struktura simuliranog kaljenja i predlažu se nadogradnje kako bi metaheuristika dala bolje rezultate.

Genetic Annealing Optimization: Design and Real World Applications [4]

Jedan od rjetkih članaka koji opisuje metaheuristiku - genetsko kaljenje. Analizira se metaheuristika u kojoj su uzete dobre strane genetskog algoritma i dobre strane simuliranog kaljenja. Algoritam pokazuje dobre rezultate, ali nedostatak može predstavljati velika populacija za koju je potrebno računati funkcije cilja čija je vremenska složenost velika.

2. Formalni opis problema

U ovom poglavlju opisuje se problem bojanja grafova i njegova primjena u području telekomunikacija. Opisuje se konkretan problem u telekomunikacijama uz pretpostavke i ograničenja. Zatim se isti problem opisuje pomoću matematičkog modela tj. problema bojanja grafova. Kreće se od općenitog problem bojanja grafova, a zatim se matematički model prilagodi konkretnom telekomunikacijskom problemu.

2.1. Tehnički opis problema

Za opis problema dovoljno je pretpostaviti da postoji korisnikova oprema (što je najčešće mobilni uređaj) i bazna stanica. Bazne stanice geografski su statične, za razliku od mobilnih uređaja koji to nisu. Ono što se u ovom radu naziva baznom stanicom je radio pristupna mreža (engl. *Radio Access Network*) (*RAN*). *RAN* dio je telekomunikacijskog sustava i nalazi se između opreme korisnika i jezgre mreže. To je sloj u mreži koji je zaslužan za prenošenje komunikacije između mobilnog uređaja do antene i od antene do drugog mobilnog uređaja.

Za što kvalitetniji rad mreže potrebno je optimirati parametre *RAN*-a. Jedan od parametra je *scrambling* kôd. *Scrambling* kôd među ostalim služi kako bi mobilni uređaj mogao razlikovati bazne stanice. Svakoj baznoj stanici potrebno je pridružiti različit kôd. Problem je u tome što je dostupno samo 512 različitih kodova. To znači da neke bazne stanice moraju imati isti kôd (ako se u sustavu nalazi više od 512 baznih stanica). Ako neke dvije bazne stanice imaju isti kôd i mobilni uređaj se nalazi u dosegu signala od obje stanice, tada uređaj ne može odrediti s kojom baznom stanicom komunicira. Tada se može dogoditi prekid poziva, što uzrokuje pad kvalitete usluge.

Potrebno je dodijeliti *scrambling* kôdove tako da dvije bazne stanice koje mogu biti istovremeno vezane za jedan mobilni uređaj nemaju isti kôd ili ako je to nemoguće, dodijeliti kôdove tako da je negativan utjecaj na kvalitetu usluge što manji. Postoje još neka tehnička ograničenja koja je potrebno uvažiti, a biti će navedena u nastavku.

Iako postoji 512 različitih kôdova, svaka bazna stanica ima ograničen skup kôdova

koje može koristiti. Za svaku baznu stanicu zadan je skup kôdova. Za neke bazne stanice postoji svojstvo nepromjenjivost, što znači da kôd koji je trenutno zadan za tu baznu stanicu mora ostati takav. Takve se vrste baznih stanica nazivaju **nepromjenjivima**. Početno stanje mreže definirano je kôdovima koji su pridijeljeni nekoj baznoj stanici. Osim toga, za svaku baznu stanicu poznata je njezina vrsta. Vrsta može biti označena sa slovima: *A, B ili C*. Bazne stanice različitih vrsta ne utječu jedna na drugu.

2.1.1. Ulazni i izlazni podaci

Ulaz algoritma je inicijalno stanje mreže. Za svaku baznu stanicu poznat je trenutni kôd koji koristi i sva pravila koja moraju biti zadovoljena. Izlaz algoritma je skup uređenih parova, gdje je prvi element identifikator bazne stanice, a drugi element pripadajući kôd. U navedenom popisu umjesto pojma bazne stanice koristi se čvor, a naziv će biti opravdan u nastavku.

Ulazni podaci

1. Popis i definicija domene boja (skupova boja)
2. Popis čvorova (baznih stanica).
   (a) Oznaka čvora
   (b) Vrsta čvora (grupa) - A, B ili C
   (c) Oznaka domene za boju čvora (koji skup boja koristi)
   (d) Početna boja za čvor
   (e) Oznaka da li je čvor nepromjenjiv

Izlazni podaci

Kao izlaz algoritma koristi se niz parova brojeva (i, c) gdje je i oznaka za baznu stanicu, a c je kôd (boja) čvora.

2.2. Matematički opis problema

Potrebno je modelirati problem u kojem postoji *bazna stanica* i veze između istih. Veze predstavljaju mjeru u kojoj jedna bazna stanica utječe na drugu. Jedan od mogućih modela je graf. Čvorovi će predstavljati bazne stanice, a jakost između dvije bazne

stanice biti će modelirano pomoću težine brida. U nastavku bazna stanica nazivat će se čvorom. Nad ovako postavljenim grafom problem je dodijeliti kôdove tako da nema bridova koji na svojim krajevima imaju čvor s istim kôdom. Problem je poznat pod nazivom *bojanje grafa*. Kako je uobičajeno pričati o bojama čvora (a ne o kôdovima) od sad pa nadalje *scrambling* kôd nazivat će se bojom čvora. U nastavku će prvo biti opisan klasičan problem bojanja grafova, gdje se promatraju bestežinski grafovi (oni čiji bridovi nemaju težine). Nakon toga problem se poopćuje na težinske grafove, čime se modelira prethodno opisan problem.

2.2.1. Definicije za klasičan problem bojanja grafova

Najprije se definiraju matematički pojmovi. Zatim se formalno opisuje problem bojanja grafova. Uz to se nadovezuje teorija izračunljivosti.

Pojam grafa

Definicija 1. *Jednostavni graf **G** sastoji se od nepraznog konačnog skupa **V(G)**, čije elemente nazivamo čvorovi grafa **G** i konačnog skupa **E(G)** različitih dvočlanih podskupova **V(G)** koje zovemo bridovi.*

Definicija 2. *Skup vrhova koji su susjedni vrhu **v** zovemo susjedstvo vrha **v** i označavamo s oznakom **H(v)**.*

Definicija 3. *Stupanj vrha **v** grafa **G** jednak je broju bridova koji su incidentni s **v**. Označavamo ga s **d(v)**.*

Definicija 4. *Neka je **G** jednostavan graf i ω funkcija $\omega : \mathbf{E} \to \mathbb{R}$. Par **(G,** ω**)** naziva se težinski graf. Pri čemu funkcija ω svakom bridu iz **G** dodjeljuje jedan element iz skupa realnih brojeva. Neka je **e** neki brid grafa **G**, njegova težina označena je s $\omega(e)$.*

Definicija bojanja grafova

Definicija 5. *Definiramo funkciju $\phi : V(G) \to \mathbb{N}$, koja svakom čvoru u grafu pridružuje jedan prirodan broj koji $\phi(\mathbf{v})$, gdje je $\mathbf{v}$ čvor u grafu $\mathbf{G}$. Broj $\phi(\mathbf{v})$ nazivamo boja čvora, a funkciju ϕ nazivamo **bojanje grafa**.*

Definicija 6. *Bojanje grafa s najviše **k** boja nazivamo **k-bojanje** grafa.*

Definicija 7. *Ako se graf može obojiti s najviše **k** boja tada takvo bojanje nazivamo **legalno k-bojanje** grafa.*

Definicija 8. *Graf je **k-obojiv** kako postoji legalno k-bojanje grafa.*

Definicija 9. *Ako je graf G **k**-obojiv, ali nije **(k-1)**-obojiv tada kažemo da je **k** kromatski broj grafa G, gdje se koristi oznaka $\chi(G) = k$*

Definicija 10. *Podskup skupa V(G) nazivamo **nezavisni skup** ako u njemu ne postoje dva čvora koja su susjedna.*

Definicija 11. *Brid koji spaja dva čvora iste boje nazivamo **konfliktni brid**.*

Definicija 12. *Dva čvora koja spaja konfliktni brid nazivamo **konfliktnim čvorovima**.*

Definicija 13. *Particiju skupa V(G) na k disjunktnih nepraznih podskupova $V_1, ..., V_k$ tako da vrijedi $V(G) = \bigcup_{j=1}^{k} V_j$ zovemo **k-dioba** grafa G. Ako su podskupovi $V_1, ..., V_k$ ujedno i nezavisni skupovi onda se to naziva **legalna k-dioba** grafa G.*

Teorem 1. *Graf **G** je **k-obojiv** kako postoji **legalna k-dioba** grafa **G**.*

Dokaz. Pretpostavimo da je graf *G* **k-obojiv**. Definiramo skupove S_i tako da čvor grafa **v** pripada skupu S_i ako je obojen bojom **i**. Takvi skupovi su neprazni i disjunktni, te unija skupova S_i čini skup svih vrhova grafa *G*. Kako je G **k-obojiv** tada ne postoji boja zbog koje bi neki brid bio konfliktan pa sledi da je podjela na skupove S_i **legalna k-dioba** grafa G jer su skupovi S_i nezavisni.

Drugi smjer dokazuje se tako što se pretpostavi da postoji **legalna k-dioba** grafa **G** na skupove S_i. Kako je svaki skup indeksiran s indeksom **i**, tada svakom čvoru koji se nalazi u skupu S_i dodijelimo boju **i**. Kako su skupovi S_i prema pretpostavci disjunktni, neprazni i nezavisni tada ne postoje dva čvora koja bi imala istu boju. □

Definicije računske teorije složenosti

Definicija 14. ***Problem odluke** je problem koji uvijek ima odgovor **da** ili **ne**. Primjerice, problem da li je graf točno **k-obojiv** je problem odluke. Problem traženja kromatskog broja grafa nije problem odluke, jer tražimo točan broj **k**.*

Definicija 15. *Problem odluke za koje postoje algoritmi koji daju odgovor, a čije vrijeme izvršavanje ovisi polinomno o veličini ulaznih podataka spadaju u **klasu P**.*

Definicija 16. ***Problem odluke** spada u klasu **NP problema** ako se točnost njegovog rješenja može ispitati u polinomnom vremenu.*

Definicija 17. ***Problem odluke** spada u klasu **NP-potpunih problema** ako spada u **klasu NP** problema i koji ima svojstvo da se svaki drugi problem iz **klase NP** može polinomno reducirati na njega.*

Definicija 18. *Kažemo da je problem **NP-težak** akko postoji **NP-potpun problem** koji se može polinomno reducirati na njega.*

Problem odluke za koji postoji algoritam čije trajanje ovisi polinomno o veličini ulaznih podataka tada je to **P problem**. Ako za neki problem odluke ne možemo pronaći algoritam čije vrijeme izvođenja ovisi polinomno o veličini ulaznih podataka, ali točnost rješenja možemo provjeriti u polinomnom vremenu tada je to **NP problem**. **NP-potpun problem** je problem odluke čiji algoritam možemo iskoristiti da bi riješili sve **NP probleme** tako da koristimo polinomijalan broj poziva tog algoritma. Općeniti problemi (koji ne moraju biti problemi odluke) nazivaju se **NP-teški problemi** ako postoji problem u klasi **NP-potpunih problema** koji se može riješiti pomoću polinomnog broja poziva algoritma za promatrani **NP-teški** problem. Odnos **P** i **NP** klasa još je uvijek otvoren problem ($P = NP$ ili $P \subset NP$). Kada bi se moglo dokazati da je neki **NP-potpun** problem moguće riješiti u polinomnom vremenu, tada bi se svi **NP problemi** mogli riješiti u polinomnom vremenu, pa bi klase **P** i klase **NP** bile jednake. Ako se pak pokaže da za neki **NP** problem ne postoji algoritam čije izvršavanje ovisi polinomno o ulaznim podacima, tada bi klasa **P** bila pravi podskup od klase **NP**.

No free lunch teorem za optimizacijske algoritme

Definicija 19. ***Optimizacija** je grana matematike koja proučava pronalaženje ekstrema funkcija.*

Definicija 20. ***Kombinatorna optimizacija** je grana optimizacije, gdje je domena funkcije skup s konačnim brojem elemenata.*

Definicija 21. *Konkretna funkcija koja se proučava naziva se **funkcija cilja**. U radu se zbog semantike umjesto funkcije cilja često koristi pojam greške.*

Definicija 22. *Funkcije cilja koje se promatraju imaju diskretnu i konačnu domenu i kodomenu. Iako se kao elementi kodomene mogu pojaviti realni brojevi, zbog toga što su računala diskretni strojevi s konačno mnogo memorije, to je samo konačni podskup realnih brojeva.*

Teorem 2. *Funkcija cilja definira se kao: $f : \mathbb{X} \to \mathbb{Y}$, gdje su skupovi $\mathbb{X}$ i $\mathbb{Y}$ diskretni i konačni. Tada je broj svih mogućih funkcija jednak $|\mathbb{F}| = |\mathbb{Y}|^{|\mathbb{X}|}$.*

Dokaz. Za svaki element domene možemo odabrati točno $|\mathbb{Y}|$ elemenata kodomene.

$$|\mathbb{F}| = \prod_{j=1}^{|\mathbb{X}|} |\mathbb{Y}| = |\mathbb{Y}|^{|\mathbb{X}|} \tag{2.1}$$

□

Definicija 23. *Vremenski niz d_m je niz parova domene i kodomene.*

$$d_m = \{(d_m^x(1), d_m^y(1)), (d_m^x(2), d_m^y(2)), ..., (d_m^x(m), d_m^y(m))\} \tag{2.2}$$

Iz nekog niza d_m može se izvući samo niz elemenata domene i to se označava s $\mathbf{d_m^x}$ ili samo niz elemenata kodomene što se označava s $\mathbf{d_m^y}$.

Definicija 24. *Prostor svih vremenskih nizova veličine m: $\mathbb{D}_m = (\mathbb{X} \times \mathbb{Y})^m$, a prostor svih vremenskih nizova maksimalne veličine m je: $\mathbb{D} = \bigcup_{m \geq 0} \mathbb{D}_m$.*

Definicija 25. *Optimizacijski algoritam a definira se kao: $a : d \in \mathbb{D} \rightarrow \{x | x \notin d_m^x\}$*

Optimizacijski algoritam preslikava neki vremenski niz (podatke iz prethodnih koraka izvođenja) u novu vrijednost domene. S time da ta vrijednost domene ne smije biti već viđena. Ovo ograničenje koristi se u dokazu teorema, ali u praksi se često mogu javiti slučajevi gdje se više puta računa vrijednost funkcije za isti element domene.

Definicija 26. ***Heuristikom*** *se naziva optimizacijski algoritam koji ne mora kao rezultat dati globalni ekstrem **funkcije cilja**, ali daje dovoljno dobre rezultate u svrhu bržeg kraćeg vremenskog izvođenja.*

Definicija 27. ***Metaheuristikom*** *naziva se familija optimizacijskih algoritama, čije konkretno izvođenje ovisi o njegovim parametrima.*

Definicija 28. *Skup svih dozvoljenih rješenja u optimizaciji nazivamo **prostor pretrage**.*

Teorem 3. ***"No free lunch" teorem za optimizaciju*** *- Neka su a_1 i a_2 dva različita heuristička algoritma koji traže ekstrem funkcije. Funkcije su predstavljene crnom kutijom. Tada vrijedi ova jednakost:*

$$\sum_f P(d_m^y | f, m, a_1) = \sum_f P(d_m^y | f, m, a_2) \tag{2.3}$$

Dokaz teorema može se pronaći u [15].

Kako se ne zna ništa više o funkcijama (npr. simbolički zapis) već samo ulazni i izlazni parovi, ne može se koristiti nikakvo unutarnje znanje koje bi nekom algoritmu dalo prednost. Ako je poznat broj koraka m tada za bilo koji algoritam, sume vjerojatnosti su međusobno jednake, pri čemu se gleda vjerojatnost da se pojavio neki niz

izlaznih vrijednosti funkcije d_m^y. Iz niza izlaznih vrijednost d_m^y može se lako izvući minimalna ili maksimalna vrijednost koja je tada konačan izlaz algoritma. Dakle ne postoji algoritam koji bi bio dominantniji za sve funkcije cilja f. Zbog toga je potrebno svaki optimizacijski problem promatrati veoma usko i koristiti dodatne informacije o samoj funkciji f.

Prethodne definicije i teoremi služe kako bi se uskladili nazivi za matematičke pojmove koji će se koristiti u radu. Sve definicije uzete su iz [5, 15, 14], gdje se među ostalim mogu naći i druge definicije i teoremi vezani za teoriju grafova, računsku teoriju složenosti i "*No free lunch*" teorem.

2.2.2. Bojanje težinskih grafova

Za navedeni telekomunikacijski problem potrebno je odabrati prikladan matematički model. Već je spomenuto da se bazne stanice modeliraju čvorovima u grafu. Postavlja se pitanje što predstavlja brid u tome grafu. Očigledno je da jedna bazna stanice utječe na drugu baznu stanicu, tj. mobilni uređaj istovremeno vidi više baznih stanica. To znači da bi brid mogao predstavljati relaciju da li neka bazna stanica vidi drugu. Problem se može razviti i korak dalje. Nije svejedno na kojoj su udaljenosti bazne stanice, te koja je njihova snaga. To znači da dvije bliže (ili jače vezane) bazne stanice više utječu jedna na drugo nego što to čine dvije udaljenije. Ideja je svakom bridu dodijeliti težinu koja predstavlja neku mjeru koliko jedna bazna stanica utječe na drugu. To znači da brid između dva čvora daje mjeru međusobnog utjecaja čvorova. Ako je utjecaj premali (tehnički nevidljiv), tada brid između ta dva čvora ne postoji.

Na tako zadanom težinskom grafu potrebno je napraviti bojanje grafa uz poštivanje svih tehničkih uvjeta. Problem sad nije dobro definiran. Zato se klasični problem bojanja grafova poopćuje:

Definicija 29. *Bojanje težinskog grafa je optimizacijski problem u kojem se minimizira funkcija cilja:*

$$f(\phi) = \sum_{i=1}^{E(G)} \omega(e_i) \cdot R(e_i) \tag{2.4}$$

, gdje je $\omega(e_i)$ težina brida, a R funkcija koja ima vrijednost 1 ako je e_i konfliktan brid ili 0 inače. Funkcija f ovisi o bojanju grafa, tj. funkcija cilja jednaka je sumi težina konfliktnih bridova.

Zbog tehničkih ograničenja potrebno je uvesti još malo matematičkih detalja koji pokrivaju te slučajeve.

Definicija 30. *Funkcija ϕ definira se kao funkcija koja svakom ne nepromjenjivom čvoru v grafa **G** pridružuje prirodan broj iz skupa dopuštenih boja za taj čvor.*

Definicija 31. *Ako su dva čvora **nepromjenjiva** spojena bridom i imaju iste boje, tada smatramo da taj brid nije **konfliktan**.*

Razlog ovakvoj definiciji je to što bridovi koji spajaju dva nepromjenjiva čvora različitih boja ne mogu se nikada poboljšati, pa shodno tome neovisni su o funkciji bojanja grafa ϕ.

Definicija 32. *Ako su dva čvora koja pripadaju u suprotne vrste spojena bridom, tada takav brid ne smatramo **konfliktnim**.*

Tehnički, takva dva čvora ne utječu jedan na drugog pa nema potrebe ugrađivati u funkciju cilja te bridove.

U slučaju da je čvor obojen bojom koja nije u njegovoj domeni tada je potrebno dodati u funkciju cilja kaznu za takav slučaj. Iako po definiciji funkcije bojanja grafa to nije moguće, zbog inicijalnog stanja boja koje algoritam primi kao ulaz može se dogoditi da boja čvora nije u njegovoj domeni. Stoga se definira proširena funkcija cilja kao:

Definicija 33.

$$f(\phi) = \sum_{i=1}^{E(G)} \omega(e_i) \cdot R(e_i) + \sum_{i=1}^{V(G)} C(v_i) \tag{2.5}$$

Prva suma jednaka je već definiranoj funkciji cilja. Drugi član je suma funkcije C po čvorovima, gdje funkcija C ima vrijednost ε ako je čvor v_i krivo obojen ili 0 ako je obojen bojom iz svoje domene.

Za epsilon se odabire neki veliki pozitivan broj. U konkretnoj implementaciji koja se koristi u radu $\varepsilon = 10000000$.

Definicija 34. *Smatra se da je bojanje **valjano** ako postotak promjene boja inicijalnog stanja ne prelazi prag od α posto.*

U konkretnom problemu $\alpha = 66\%$.

Sažetak problema kojeg modeliramo pomoću težinskog grafa glasi:

- Svaki čvor može poprimiti jednu boju iz skupa dopuštenih boja za taj čvor
- Svaki čvor spada u jednu grupu tj. vrstu (A, B ili C)
- Neki čvorovi su definirani kao nepromjenjivi, njima se boja nikad ne smije mijenjati

– Dodatan uvjet je da postotak promijenjenih čvorova naprema inicijalnom stanju ne smije biti veća od 66%

– Optimizacija se provodi nad funkcijom cilja definiranom formulom (2.5).

2.3. Analiza problema bojanja težinskih grafova

U ovom odjeljku biti će analiziran zadani matematički problem optimizacije. U svakom koraku analize uspoređuju se klasičan problem bojanja grafova i bojanje težinskih grafova. Kreće se od analize prostora pretrage. Zatim se dokazuje u kojoj se klasi problemi nalaze. Na kraju se pokazuje da "*No free lunch*" teorem vrijedi i za bojanje grafova i kakve posljedice donosi.

2.3.1. Prostor pretrage stanja

Kod klasičnog problema bojanja grafova analiziraju se dva problema. Prvi je odrediti da li se graf može obojiti s najviše k boja, a drugi problem je odrediti najmanji takav k. Pretpostavimo za početak da provjeravano da li je graf **k-obojiv**. Pitanje je na koliko načina se graf može obojiti ako koristimo najviše **k** boja. Svaki čvor može poprimiti **k** boja. To znači da je ukupan broj bojanja N jednak:

$$N = k^{V(G)} \tag{2.6}$$

Valja primijetiti da u ovom brojanju veličine prostora stanja postoji više jednako vrijednih bojanja. U bojanju nije bitna točna boja, već je bitno da različiti skupovi čvorova imaju različite boje. Točno pitanje je na koliko načina možemo čvorove podijeliti u k skupova (teorem 1).

$$N = \sum_{i=1}^{k} S(V(G), i) \tag{2.7}$$

Oznaka $S(m, n)$ je za *Stirlingov* broj druge vrste, gdje je m broj različitih elemenata (čvorovi) koje smještamo u n istovrsnih skupova boja, tako da svaka boja ima barem jedan čvor. Kako je dopušteno da kutije mogu biti prazne, tada se problem pretvori u disjunktne probleme gdje se koristi samo i kutija.

Ako se zahtjeva da svaka od k boja mora biti iskorištena, ekvivalentno da ni jedan od k skupova ne smije biti prazan, tada je ukupan broj stanja nešto manji:

$$N = S(V(G), k) = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k-i)^{V(G)} \tag{2.8}$$

Ako se čvrsto zahtjeva da svaka boja mora biti barem na jednome čvoru, tada je to *Stirlingov* broj druge vrste, gdje je i navedena formula za izračun $S(m, n)$.

Neka se promatra problem traženja kromatskog broja grafa $\chi(G)$, tj. minimalni broj boja k, a da se graf može legalno obojiti. Maksimalni broj za koji je to potrebno provjeriti je broj čvorova u grafu, jer u tom slučaju svaki čvor povezan je sa svakim drugim i svaki čvor mora imati svoju boju. Kromatski broj grafa je dakle ograničen odozgo s brojem čvorova. Za svaki broj k do $V(G)$ treba provjeriti da li je graf **k-obojiv**. Ukupan broj stanja je:

$$N = B(V(G)) = \sum_{i=0}^{V(G)} S(V(G), i) \tag{2.9}$$

Broj $B(n)$ naziva se *Bellov* broj i on predstavlja broj načina na koje se skup od n članova može podijeliti u neprazne podskupove. Za bolji uvid koliko brzo *Bellov* niz raste dane su nejednakosti (za $n \geq 8$):

$$2^n \leq B(n) \leq n! \tag{2.10}$$

Bellov broj brže raste od eksponencijalne funkcije, ali sporije od faktorijela.

Dokaz. Prvo se pokazuje nejednakost: $2^n \leq B(n), n \geq 5$. Dokaz se provodi indukcijom. Koriste se razvoj binoma:

$$2^n = (1+1)^n = \sum_{k=0}^{n} \binom{n}{k}$$

i svojstvo Bellovog broja:

$$B(n+1) = \sum_{k=0}^{n} \binom{n}{k} B(k)$$

$$2^n \leq B(n) \tag{2.11}$$

$$\text{Baza indukcije: } n = 5, 2^5 \leq B(5) \rightarrow 32 \leq 52 \tag{2.12}$$

$$\text{Pretpostavka: } 2^n \leq B(n) \tag{2.13}$$

$$2^{(n+1)} \leq B(n+1) \tag{2.14}$$

$$2 \cdot 2^n \leq B(n+1) \tag{2.15}$$

$$\sum_{k=0}^{n} 2 \cdot \binom{n}{k} \leq \sum_{k=0}^{n} \binom{n}{k} B(k) \tag{2.16}$$

$$\sum_{k=0}^{3} 2 \cdot \binom{n}{k} + \sum_{k=4}^{n} 2 \cdot \binom{n}{k} \leq \sum_{k=0}^{3} \binom{n}{k} B(k) + \sum_{k=4}^{n} \binom{n}{k} B(k) \tag{2.17}$$

$$\text{Prvo se dokazuje: } \sum_{k=0}^{3} 2 \cdot \binom{n}{k} \leq \sum_{k=0}^{3} \binom{n}{k} B(k) \tag{2.18}$$

$$2\binom{n}{0} + 2\binom{n}{1} + 2\binom{n}{2} + 2\binom{n}{3} \leq \tag{2.19}$$

$$\binom{n}{0} B(0) + \binom{n}{1} B(1) + \binom{n}{2} B(2) + \binom{n}{3} B(3) \tag{2.20}$$

$$2 + 2n + \frac{n(n-1)}{2} 2 + \frac{1}{3} n(n-1)(n-2) \leq \tag{2.21}$$

$$1 + n + \frac{n(n-1)}{2} 2 + \frac{n(n-1)(n-2)}{6} 5 \tag{2.22}$$

$$1 + n \leq n(n-1)(n-2)(\frac{5}{6} - \frac{1}{3}) \tag{2.23}$$

$$1 + n \leq \frac{n(n-1)(n-2)}{3} \tag{2.24}$$

$$3 + 3n \leq n^3 - 3n^2 + 3n \tag{2.25}$$

$$3n^2 + 3 \leq n^3 \tag{2.26}$$

$$3 \leq n^3 - 3n^2 \tag{2.27}$$

$$3 \leq n^2(n-3), \text{istina za } n \geq 5 \tag{2.28}$$

$$\text{Druge dvije sume: } \sum_{k=4}^{n} 2 \cdot \binom{n}{k} \leq \sum_{k=4}^{n} \binom{n}{k} B(k) \tag{2.29}$$

$$\text{Član po član sume: } \binom{n}{k} 2 \leq \binom{n}{k} B(k) \tag{2.30}$$

$$2 \leq B(k), \text{za } k \geq 4 \tag{2.31}$$

□

Dokaz. Potrebno je dokazati $B(n) \leq n!$. Dokaz se provodi iz svojstava za Bellov broj

[2]:

$$B(n) \leq \left\lceil \frac{0.792n}{ln(1+n)} \right\rceil^n$$

i svojstvo faktorijela koja slijedi iz analize Stirlingove aproksimacije:

$$n! \geq \left(\frac{n}{e}\right)^n$$

$$B(n) \leq \left\lceil \frac{0.792n}{ln(1+n)} \right\rceil^n \leq \left(\frac{n}{e}\right)^n \leq n! \tag{2.32}$$

$$\text{Dovoljno je pokazati: } \left\lceil \frac{0.792n}{ln(1+n)} \right\rceil^n \leq \left(\frac{n}{e}\right)^n \tag{2.33}$$

$$\left\lceil \frac{0.792n}{ln(1+n)} \right\rceil \leq \left(\frac{n}{e}\right) \tag{2.34}$$

$$0.792n \cdot e \leq n \cdot ln(n+1) \tag{2.35}$$

$$0.792 \cdot e \leq ln(n+1) \tag{2.36}$$

$$e^{0.792 \cdot e} \leq n+1 \tag{2.37}$$

$$8.61 - 1 \leq n \tag{2.38}$$

$$7.61 \leq n \tag{2.39}$$

□

Zaključak Prethodne analize je da za relativno veliki graf postoji velik broj stanja koje je nemoguće pretražiti pomoću *brute force* algoritama. Analiza prostora stanja za zadani problem malo je drugačija. Kako svaki čvor ima konačan broj boja koje može poprimiti ukupan broj stanja je:

$$N = \prod_{i=1}^{V(G)} K_i \tag{2.40}$$

, gdje je oznaka K_i ukupan broj boja koje može poprimiti čvor s indeksom i. Kako je moguće dodijeliti maksimalno samo 512 boja, možemo ograničiti K_i:

$$K_i \leq M = 512 \tag{2.41}$$

$$N = \prod_{i=1}^{V(G)} K_i \leq \prod_{i=1}^{V(G)} M = M^{V(G)} \tag{2.42}$$

Broj stanja eksponencijalno ovisi o broju čvorova, što je isto kao i za klasičan

problem bojanja grafa jako velik broj stanja. I u ovom slučaju važno je primijetiti da su neka stanja ekvivalentna. Broj jedinstvenih stanja je broj particija skupa čvorova u skupove boja, ali uz ograničenje da barem jedan čvor ima neku od zadanih M boja, tada vrijedi ova nejednakost za broj različitih stanja:

$$N \leq \sum_{i=1}^{M} S(V(G), i) \tag{2.43}$$

Broj stanja je manji jer se poštuje ograničenje da čvorovi mogu poprimiti neku od boja u podskupu od ukupnog konstantnog broja boja M. Kada je $k \geq M$ tada broj stanja optimizacijskog problema je manji od broja stanja u problemu gdje se traži odgovor na pitanje da li je graf **k-obojiv**. Ovo razmatranje biti će zanimljivo i u sljedećem poglavlju gdje se problem svrstava u klase računske teorije složenosti.

2.3.2. Pozadina problema iz perspektive računske teorije složenosti

Problem odluke da li je graf **k-obojiv** spada u klasu **NP-potpunih** problema. Traženje kromatskog broja grafa $\chi(G)$ spada u klasu **NP-teških** problema. Dokazi se mogu pronaći u [**?**]. Potrebno je pokazati da postavljen optimizacijski problem bojanja grafa spada u klasu **NP-teških** problema. Problem sigurno ne može spadati u druge spomenute (P, NP, NP-potpun) klase jer to nije problem odluke. Postupak dokazivanja da problem spada u klasu **NP-teških** problema ima nekoliko koraka. Prvi korak je pronalaženje nekog **NP-potpunog** ili **NP-teškog** problema koji će se koristiti u dokazu. Zatim se pokaže da je taj *stari* problem moguće polinomno reducirati na problem za koji se dokazuje da je **NP-težak**.

Dokaz. Odaberemo **NP-potpun problem** koji ispituje da li je graf G **k-obojiv**. Da bismo riješili taj problem koristimo zadani optimizacijski problem. Ulaz u optimizacijski algoritam među ostalim su graf G i ograničenja boja za svaki od čvorova. Svakom čvoru daje se skup $\{1, 2, ..., k\}$ s bojama. Svim bridovima daje se težina 1. Sada se pokreće crna kutija s optimizacijskim problemom koja će riješiti taj problem. Ako je rezultat optimizacije tj. funkcije cilja $f = 0$, tada je graf **k-obojiv**, inače nije. U konstantnom vremenu možemo reducirati problem, iz čega slijedi da je optimizacijski problem **NP-težak**. □

Zanimljivo je primijetiti da **NP-potpun** problem k-obojivosti može imati više stanja od **NP-teškog** problema optimizacije. Što znači mogućnost manjeg vremena izvođenja od vremena provjere k-obojivosti. Kako nije poznat odnos *NP* i *P* klase, za ovaj

optimizacijski problem nameće se korištenje heuristika. U ovom tehničkom problemu nije potrebno do kraja minimizirati funkciju cilja, nego dobiti dovoljno malu vrijednost f što će uzrokovati bolju kvalitetu mobilne mreže.

2.3.3. Posljedica "*No free lunch*" teorema

U originalnom članku, gdje se dokazuje "*No free lunch*" teorem [15], pretpostavlja se da sve funkcije cilja imaju iste domene i kodomene. Za problem bojanja grafova to nije tako. Funkcija cilja definirana je kao: $f : V(G) \to \mathbb{R}$, ali kako računalo ima konačnu memoriju tada je funkcija zapravo definirana kao: $f : V(G) \to \mathbb{Y}$, gdje je $\mathbb{Y}$ konačni podskup realnih brojeva koji se mogu prikazati na računalu. Problem za "*No free lunch*" teorem je to da veličina domene funkcije cilja ovisi o broju čvorova grafa. Broj grafova je beskonačan, jer možemo uzeti proizvoljan broj čvorova, što znači da postoji i beskonačan broj mogućih funkcija cilja. Teorem radi samo ako je broj funkcija cilja ograničen (teorem 3). Kako je na nekom računalu nemoguće prikazati proizvoljno veliki graf, broj čvorova može se ograničiti s nekom granicom M. Sada je moguće prebrojati sve funkcije cilja f.

Dokaz. Pitanje je koliko funkcija postoji, ako je broj članova domene ograničen s M. Broj funkcija koje imaju točno k elemenata u domeni ima:

$$|\mathbb{F}_k| = |\mathbb{Y}|^k \tag{2.44}$$

Ukupan broj funkcija je suma po svim k do granice M (koristi se suma za geometrijski niz):

$$|\mathbb{F}| = \sum_{k=1}^{M} |\mathbb{F}_k| = \sum_{k=1}^{M} |\mathbb{Y}|^k = \frac{|\mathbb{Y}|^{M+1} - |\mathbb{Y}|}{|\mathbb{Y}| - 1} \tag{2.45}$$

□

Ako se ograniči broj čvorova, dobije se konačan broj funkcija te je to lako ugraditi u teorem 3. Teorem govori o tome kako ne postoji nikakvo znanje o funkciji. U ovom problemu funkcija cilja direktno ovisi o grafu tj. strukturi i to je važna informacija koja treba biti iskorištena da bi se dobili bolji algoritmi. Glavna Posljedica ovog teorema je razvoj pohlepnih algoritama koji koriste znanje o strukturi (grafu) i primjena tih algoritama u metaheurističkim postupcima. Važno je uočiti da je svaki graf posjeduje vlastitu funkciju cilja. To znači da korištenjem algoritama na slijepo koji rade na nekim grafovima ne mogu jamčiti efikasnost na drugim.

U članku [11] opisuje se kako postoji slučajevi u kojima hiper-heuristike mogu imati "*Free lunch*" svojstvo. Hiper-heuristike su algoritmi koji traže najbolji algoritam koji može riješiti neki problem, tj. pronaći ekstrem funkcije cilja. Kako je gore opisano, svaki graf ima svoju funkciju cilja. Ako je skup funkcija cilja dovoljno malen, moguće je da hiper-heuristika uvijek pronađe neki algoritam koji je bolji od svih ostalih za konkretnu funkciju cilja. Ovo nije dokazano za bojanje grafova (tako ni za optimizacijski problem bojanja), ali daje motivaciju u razvoju algoritama. U zadnjem dijelu opisa algoritama u ovom radu koriste se metode strojnog učenja kako bi se izvuklo znanje za primjenu u nekim pohlepnim algoritmima, tj. kako bi se odabrao najbolji pohlepan algoritam, što je zapravo jednostavan oblik hiper-heuristike. Osim toga u implementaciji je otvoren prostor gdje se lako mogu donositi odluke i dinamički birati algoritmi koji u trenutnoj točci u prostoru stanja daju bolje rezultate od drugih algoritama.

3. Algoritmi

3.1. Uvod

Algoritmi koji su implementirani za ovaj problem podijeljeni su u dva dijela. Prva skupina su pohlepni algoritmi, a drugi algoritmi s kojima se i rad bavi su meta heuristike. Proučavaju se četiri metaheuristike: agentski algoritam, simulirano kaljenje, genetski algoritam i genetsko kaljenje. Započinje se s konstruiranjem hiper-heuristike nad pohlepnim algoritmima, gdje se želi pokazati koji pohlepni algoritam je najbolji u raznim ovisnostima. Pri testiranju algoritama treba biti veoma oprezan. Prethodna razmatranja upućuju na to da neki algoritam (metaheuristika s određenim parametrima) može dati bolje rezultate od drugih algoritama na nekim grafovima, dok je na nekim grafovima obrnuta situacija. To se može usporediti s pojmom *prenaučenosti* iz strojnog učenja. Zbog toga je za ovaj problem potrebno graditi algoritme koji su prilagodljivi.

3.1.1. Mogući problemi

Mogući problemi mogu se podijeliti na dva dijela: problemi vezani uz ograničenja i problemi vezani uz implementaciju. Problemi vezani uz ograničenja su velik broj čvorova i bridova koji se mogu pojaviti, velika razlika između težina bridova (razlika za više redova veličina), nepromjenjivi čvorovi koji su okruženi promjenjivim čvorovima. Velik broj čvorova i bridova uzrukuje i sporji rad algoritma. Razlika između težina bridova može kod algoritama koji koriste tu razliku uzrokovati neočekivano ponašanje (npr. dugo zadržavanje na manjim vrijednostima funkcije cilja). Neopromjenjivi čvorovi koji za susjede imaju promjenjive čvorove mogu uzrokovati sporu konvergenciju, a čest zbog takvh čvorova minimum funkcije cilja je relativno velika vrijednost, pa je teško razlikovati lokalni od globalnog minimuma.

Problemi vezani uz implementaciju su zapravo problemi dizajna algoritama i struktura podataka. S jedne strane potrebno je osigurati brze operacije nad grafom, a s druge strane grafovi mogu zauzimati velik dio memorijskog prostora. Zato je potrebno di-

zajnirati strukturu grafa koja ne zauzima previše memorijskog prostora, a omogućuje brze operacije. Mogući problem su i populacijski algoritmi kod kojih je potrebno čuvati više rješenja. Zbog toga se jedinke ne čuvaju kao grafovi, već je svaka jedinka funkcija bojanja grafa (za svaki čvor zna se koja je njegova boja). A sama struktura (bridovi) sadržanja je u strukturi grafa, a jedinka posjeduje vezu na taj graf.

3.2. Pohlepni algoritam

Pojam pohlepnog algoritma u ovom radu znači sortirati čvorove grafa po nekom svojstvu. Zatim obilaziti čvorove tim redoslijedom i na svaki od njih primijeniti odabranu vrstu bojanja čvora. Vrsta bojanja čvora je način odabira boje za neki čvor. Pohlepni algoritam se koristi kao operator kod meta heuristika, tj. kao metoda koja usmjeruje pretragu. U radu se koriste sljedeće metode za sortiranje i bojanje čvorova (za svaku od metoda koristi se specijalna oznaka):

Vrste sortiranja čvorova:

- *COL* - sortiranje čvorova prema koliziji (broj susjednih čvorova koji imaju drugačiju boju od promatranog čvora)
- *FIT* - sortiranje čvorova prema pogrešci koju generiraju (suma težina konfliktnih bridova iz tog čvora)
- *LDO* - sortiranje prema stupnju čvora $H(v)$
- *RND* - slučajan raspored čvorova
- *SDO* - sortiranje prema zasićenju (broj različitih boja susjednih čvorova)
- *SDOLDO* - sortiranje prema zasićenju, a ako dva čvora imaju isto zasićenje drugi kriterij je stupanj čvorova [članak]
- *STDORD* - sortiranje prema oznaci čvora (čvorovi su numerirani)

Vrste bojanja čvorova su:

- *ABW* - susjedi čvora sortiraju se prema težini brida, obilazi se svaki susjed i uzima se njegova boja. Ako je ta boja jedna od dopuštenih za promatrani čvor, ta boja više ne može biti odabrana. Kada na raspolaganju ostane samo jedna boja ili su svi susjedi posjećeni, uzima se jedna od boja koja je još uvijek na raspolaganju.
- *MC* - traži se dopuštena boja koja trenutnom čvoru daje minimalnu vrijednost kolizije

– *MF* - traži se dopuštena boja koja trenutnom čvoru daje minimalnu sumu konfliktnih bridova

– *RND* - daje se slučajna dopuštena boja

– *START* - čvoru se daje početna boja (ako je dopuštena) ili se boja ne mijenja ako u inicijalnom bojanju boja nije bila dopuštena

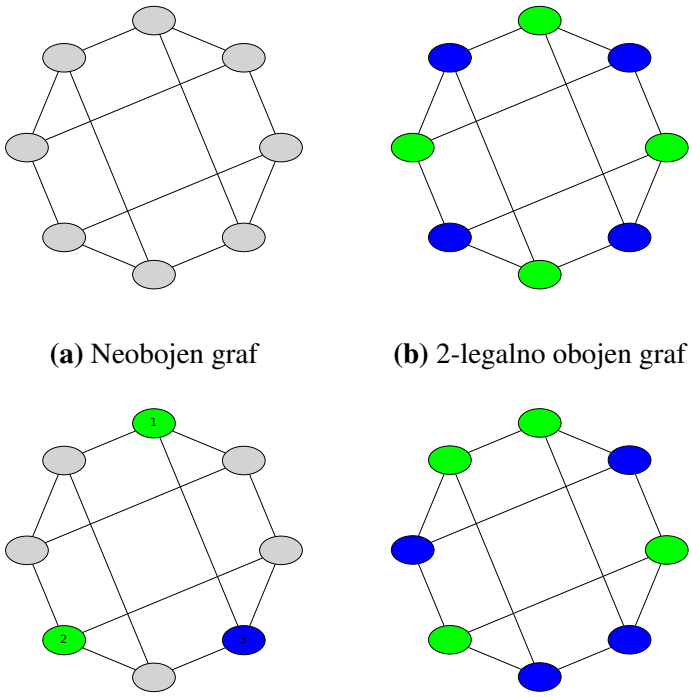
– *SWAP* - čvoru se daje neka boja gdje apsolutno odstupanje od greške (sume konfliktnih bridova) naspram trenutne greške nije veće od nekog praga

– *TRG* - čvoru se daje neka boja čija nova greška (suma konfliktnih bridova) ne prelazi neku fiksnu vrijednost.

3.2.1. Neispravnost pohlepnog algoritma

Važno je pokazati kako pohlepni algoritam neće pronaći globalni minimum funkcije cilja. Neka je zadan graf na slici 3.1. Slika 3.1a prikazuje graf koji je potrebno bojati korištenjem samo dvije boje. Neka su težine svih bridova jednake 1. Slika 3.1b prikazuje 2-legalno obojen graf čija je vrijednost funkcije cilja u optimizaciji jednaka 0. Ako se u postupku sortiranja čvorova dogodi poredak prva tri čvora kako je to prikazano na slici 3.1c, bez obzira na to kako je drugi dio čvorova sortiran ni jedan algoritam više ne može doći do globalnog minimum funkcije cilja, jer prva dva koraka su čvorove koji sigurno moraju biti u različitim bojama svrstao u isti skup, a Posljedica je bojanje trećeg čvora drugom bojom (jer je susjedan drugo odabranom), a nakon toga cijelo rješenje je pogrešno.

Moguće rješenje navedenog problema moglo bi biti u pretraživanju susjednih čvorova. Neka su boje označene brojevima, gdje je prva boja označena s 1. Prvi čvor odabere se slučajno. Za odabrani čvor odabere se boja s minimalnom oznakom, a nema je u susjednim čvorovima. Sljedeći čvor biti će neki od susjeda trenutnog čvora. Ako čvor nema susjeda za odabir onda se traži čvor koji nije obojen ali ima obojenog susjeda (pretpostavka je da je graf povezan). Ovako opisan algoritam uvijek će dobro bojati graf na slici 3.1a. Potrebno je ako ima, pronaći graf na kojem ovakav pohlepni algoritam koji se još i naziva *uspinjanje na vrh* ne radi dobro. Primjer takvog grafa i odabir čvorova može se vidjeti na slici 3.2. Brojevi u čvorovima predstavljaju poredak obilaženja grafa. Na slici 3.2a odabir čvorova je takav da je ukupan broj boja, a da funkcija cilja bude 0 jednak 3. To je ujedno i minimalni broj boja, jer u grafu postoje tri čvora koja su međusobno povezana, dakle s manjim brojem boja nije moguće legalno bojati graf. Na slici 3.2b odabir čvorova je bio takav da u zadnjem posjećenom čvoru



(a) Neobojen graf **(b)** 2-legalno obojen graf

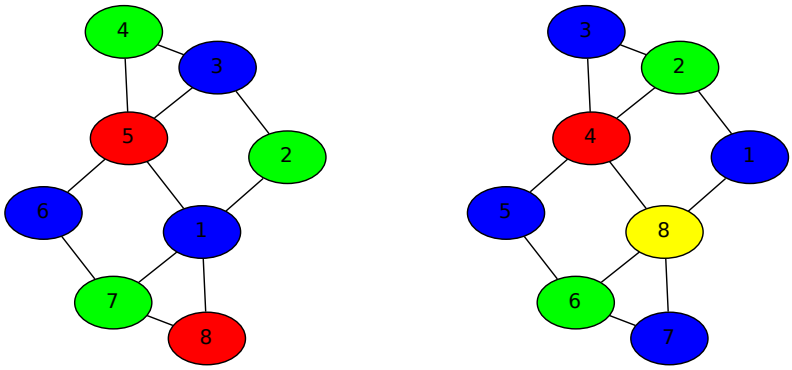
(c) Odabir prva tri čvora **(d)** Pogrešno obojen graf

Slika 3.1: Prikaz pravilno obojenog grafa i primjer pohlepnog algoritma

8 bilo potrebno odabrati novu boju da bi graf bio legalno obojen. Kako se pokazalo da je graf moguće bojati s 3 boje, zaključak je da algoritam ne daje uvijek točan rezultat.

3.3. Agentski algoritam

Agentski algoritam inspiriran je dobrim rezultatima pohlepnog algoritma. Glavna jedinica algoritma je agent. Agenti se rasporede po grafu, gdje se neki agent postavi na neki čvor grafa. *Globalna iteracija* je jedna iteracija algoritma u kojoj svaki agent svome čvoru pridruži neku boju. Agenti se boduju i to tako da na početku svi imaju 0 bodova, a nakon globalne iteracije, broj bodova za nekog agenta je jednak maksimalnoj pogrešci nekog susjednog čvora, na koji se i agent pomakne. Zatim se agenti sortiraju prema bodovima i u sljedećoj globalnoj iteraciji pravo na bojanje prvo dobiva agent s najviše bodova. Slično kao i kod pohlepnog algoritma, prvo se obrađuju *teški čvorovi*, a to je u ovom algoritmu modelirano pomoću bodovanja. Agenti osim akcije pomaka na *najbolji* čvor, mogu se pomaknuti na slučajan susjedni čvor ili pak ostati na čvoru na kojem jesu, tada bodovi za agenta ostaju isti.



(a) Dobar odabir čvorova **(b)** Loš odabir čvorova

Slika 3.2: Isti algoritam uz drugačiji odabir čvorova daje različita k-legalna bojanja

3.3.1. Parametri algoritma

1. Broj agenata - N
2. Broj globalnih iteracija - G
3. Način na koji agenti bojaju čvorove (vrsta bojanja) - CS
4. Vjerojatnost ostajanja agenta na trenutnom čvoru - PR_NO_MOVE
5. Vjerojatnost pomaka agenta na slučajni čvor u susjedstvu - PR_RND_MOVE

3.3.2. Detalji algoritma

Koristi se pseudokôd kako bi se algoritam opisao formalno. U sustavu se nalazi N agenata koje je prvo potrebno inicijalizirati, odnosno postaviti svakog agenta na neki od čvorova grafa. U svakoj od G globalnih iteracija, prolazi se kroz populaciju agenata i svaki od njih prvo pridruži svome čvoru boju, a nakon toga radi funkciju pomaka. Nakon što se to obavi za svakog agenta, agenti se sortiraju prema bodovima koje su dobili u funkciji pomaka "*MOVE*". U sljedećoj iteraciji agenti se obilaze po bodovima, od onog agenta koji ima najviše bodova do onog s najmanje bodova.

Algoritam 3.1: Pseudokôd agentskog algoritma

```
1   ALGORITHM(N, G, CS, PR_NO_MOVE, PR_RND_MOVE):
2     AGENT[N]
3     FOR i = 1 TO N:
4       init(AGENT[i])
```

```
    END FOR
    FOR j = 1 TO G:
      FOR i = 1 TO N:
        SETCOLOR(AGENT[i], CS)
        MOVE(AGENT[i], PR_NO_MOVE, PR_RND_MOVE)
      END FOR
      SORT(AGENT[N])
    END FOR
  END ALGORITHM
```

Funkcija pomaka ovisi o promatranom agentu i parametrima tj. vjerojatnostima "PR_NO_MOVE" i "PR_RND_MOVE". Prvo se radi odabir da li agent ostaje na istom čvoru ili ne. Ako ne ostaje opet se stohastički određuje akcija da li agent ide na čvor koji u susjedstvu radi najveću grešku ili ide na slučajnog susjeda. Ako agent odabire pomak prema najboljem susjedu, sljedeći čvor ne može biti isti onaj s kojeg je agent došao na trenutno promatrani, čime se sprječava titranje agenta.

Algoritam 3.2: Detalji funkcije pomaka - MOVE

```
  MOVE(AGENT, PR_NO_MOVE, PR_RND_MOVE):
    IF random() < PR_NO_MOVE THEN
      RETURN
    END IF
    NEXT_NODE = AGENT.NODE
    IF random() < PR_RND_MOVE THEN
      NEXT_NODE = AGENT.NODE.NEIGHBOUR[random()]
    ELSE
      FOR k = 1 TO AGENT.NODE.DEGREE:
        IF AGENT.NODE.NEIGHBOUR[k] = AGENT.BLOCK_NODE THEN
          CONTINUE
        ERROR = AGENT.NODE.NEIGHBOUR[k].ERROR
        IF ERROR > MAX_ERROR THEN
          MAX_ERROR = ERROR
          NEXT_NODE = AGENT.NODE.NEIGHBOUR[k]
        END IF
      END FOR
        AGENT.SCORE = MAX_ERROR
    END IF
    AGENT.BLOCK_NODE = AGENT.NODE
    AGENT.NODE = NEXT_NODE
  END MOVE
```

3.3.3. Karakteristike algoritma

Kako je algoritam temeljen na pohlepnom algoritmu ima nedostatak što lako zapinje u lokalnom minimumu. Dobra strana algoritma je što jedna globalna iteracija može vremenski trajati kraće od jedne iteracije pohlepnog algoritma, jer može posjedovati broj agenata manji od broja čvorova. Algoritam ima ugrađeno određeno znanje, tj. agent cilja na čvorove koje mora popraviti. Prednost dobivaju agenti koji moraju popraviti čvorove s većim greškama. Da bi se izbjegla brza konvergencija u legalni minimum, stohastički način odabira pomicanja omogućuje da agenti šetnjom dođu u područje koje bi trebalo popraviti. Ako je broj agenata veći od broja čvorova to omogućuje fino traženje minimuma. U tom slučaju jedan čvor biti će više puta obojen (minimiziran), a to znači da mu se prvo može dati neka boja, zatim se bojaju susjedi i onda opet taj isti čvor dolazi na red. Taj efekt propitkivanja starog poteza (koji se u ovom slučaju implicitno događa) omogućuje bolju odluku za pridruživanje boje. Nedostatak je povećanje vremena trajanja globalne iteracije.

Vremenska složenost algoritma ovisi o broju agenata i broju globalnih iteracija. Vremenska složenost je: $O(G \cdot (N + Nlog(N)) = O(G \cdot N + G \cdot Nlog(N)) = O(G \cdot Nlog(N))$. Kao što je vidljivo algoritam polinomno ovisi o G i N. U ovoj analizi složenosti, kao i ostalim koje sljede zanemarena je složenost izračuna funkcije cilja. Svaki algoritam za svako rješenje koje pamti dodatno koristi broj koraka koji je proporcionalan broju bridova u grafu, jer je vremenska složenost funkcije cilja $O(E(G))$.

3.4. Algoritam evolucijske strategije

Skupina evolucijskih algoritama veoma je popularna u rješavanju optimizacijskih problema. Takvi algoritmi spadaju u još veću skupini algoritama koji se nazivaju *metaheuristike*, što znači da konkretno ponašanje algoritma ovisi o postavljenim parametrima. Svim evolucijskim algoritmima zajedničko je što posjeduju populaciju rješenja, te iz koraka u korak evoluiraju rješenje, što nužno ne mora značiti dobivanje boljeg rješenja, već algoritmi provode usmjereno pretraživanje prostora stanja. Evolucijska strategija vrsta je evolucijskog algoritma kod kojeg je dominantan operator mutacije. Genetski algoritam je poopćenje evolucijskih strategija jer se koristi operator selekcije, te se operator križanja koristi češće od operatora mutacije. Operator selekcije koristi se za odabir roditelja. Operator križanja je operator koji kombinira roditelje (rješenja) i njihovom kombinacijom (nije definirano kako) generira djecu (nova rješenja). Križa-

nje se koristi kako bi se prostor stanja finije pretraživao, tj. vršila lokalna pretraga. Operator mutacije djeluje nad jednim djetetom i mijenja izgled te jedinke. Mutacija se koristi kako bi se pretraga pomakla u neki drugi dio prostora stanja i zadužena je za sprečavanje konvergencije u lokalne ekstreme.

3.4.1. Genetski algoritam

Kao preliminarni dio ovog rada nekoliko vrsta genetskih algoritama je isprobano. Jednostavni genetski algoritam koristi *3K* turnirsku selekciju, operator križanja s jednom točkom prekida, te mutacijom koja slučajno promijeni nekoliko gena u zapisu rješenja. Algoritam se pokazao veoma lošim na problemu optimizacijskog problema bojanja grafova. Ako se kao rješenje koristi niz brojeva, gdje svaki element predstavlja jedan čvor u grafu, a vrijednost elementa boju, tada operator križanja nema previše smisla, zbog toga što su čvorovi povezani u određenu strukturu, a križanje s jednom točkom prekida nema znanje o tome. Drugi problem je slučajna mutacija, koja može rješenje staviti u neki drugi dio prostora stanja, ali teško je odrediti njen utjecaj i jako ovisi o konkretnom grafu. Drugi korak je bilo korištenje hibridnog genetskog algoritma u kojem je za operator mutacije koristi pohlepni algoritam. Rezultati u ovom slučaju također nisu bili dobri, tj. algoritam se ponašao kao slučajna pretraga prostora. Najveći problem je podešavanje parametara i njihova osjetljivost. Jedno od mogućih pristupa je kreiranje operatora križanja koji je usko povezan uz domenu problema, jedan od primjera naveden je u [10]. Drugi pristup je izbacivanje operatora križanja, čime se efektivno dobiva evolucijska strategija. Takav pristup koristi se u članku [8], gdje je zamijećeno loše ponašanje genetskog algoritma tj. jednostavnih operatora križanja. U ovom radu koristi se evolucijska strategija. Ovaj izbor načinjen je zbog očekivanog manjeg vremena izvođenja evolucijske strategije, jer se koristi jedan operator manje, za razliku od genetskog algoritma. Mutacija u tom slučaju poprima dvojaku ulogu, tj. potrebno je implementirati više vrsta mutacija - koje će obavljati lokalnu pretragu i one koje će obavljati globalnu pretragu.

3.4.2. Evolucijska strategija

Odabrana je evolucijska strategija koja se koristi u članku [10]. Koristi se (μ, λ) evolucijska strategija. Oznake μ i λ služe za oznake skupova jedinki (rješenja). Skup μ je skup roditeljskih jedinki iz kojih se stvara skup djece λ. Iz toga proizlazi da broj elemenata u skupu μ mora biti veći ili jednak broju elemenata u skupu λ. Jedinka je predstavljena kao niz brojeva, gdje svaki element predstavlja neki čvor, a vrijednost

elementa je boja čvora. U svakoj iteraciji algoritam uniformno bira $|\lambda|$ jedinki iz skupa roditelja. Može se dogoditi da je neka jedinka više puta odabrana. Za svaku odabranu jedinku koristi se operator mutacije. Slučajnim odabirom bira se između mutacije koja ima lokalni karakter tj. trenutno rješenje nastoji poboljšati lokalnom pretragom prostora stanja i mutacije koja skače iz okoline trenutne točke prostora stanja. Mutacija se implementira pomoću vrsta bojanja. Sve takve mutiranje jedinke stavljaju se u skup λ. Na kraju postupka najboljih $|\mu|$ jedinki kopira se u μ skup. Poželjno je koristiti elitizam kako se najbolje rješenje kroz iteracije ne bi izgubilo.

3.4.3. Parametri algoritma

1. Veličina skupa μ - MI_SIZE
2. Veličina skupa λ - $LAMBDA\_SIZE$
3. Ukupan broj iteracija - N
4. Broj mutacija nad jednom jedinkom - M
5. Vjerojatnost mutacije gena pomoću lokalne pretrage - $PROP$
6. Način bojanja za lokalnu pretragu - CS_GLOBAL
7. Način bojanja za globalnu pretragu - CS_LOCAL

3.4.4. Detalji algoritma

Kostur algoritma izgleda veoma jednostavno. Kao što je opisano, iz skupa μ bira se jedinka koja se mutira i stavlja u λ skup. Najbolji se kopiraju u skup μ i brišu se sve stare jedinke.

Algoritam 3.3: Pseudokôd evolucijske strategije

```
1   ALGORITHM(MI_SIZE, LAMBDA_SIZE, N, M, PROP, CS_GLOBAL,
      CS_LOCAL):
2     MI[MI_SIZE]
3     LAMBDA[LAMBDA_SIZE]
4     FOR i = 1 TO MI_SIZE:
5       init(MI(i))
6     END FOR
7     FOR j = 1 TO N:
8       LAMBDA.clear()
9       FOR i = 1 TO LAMBDA_SIZE:
10        unit = SELECT(MI)
```

```
11          MUTATION(unit, M, PROP, CS_GLOBAL, CS_LOCAL)
12          LAMBDA.insert(unit)
13        END FOR
14        SORT(LAMBDA)
15        MI.clear()
16        FOR i = 1 TO MI_SIZE:
17          MI.insert(LAMBDA[i])
18        END FOR
19      END FOR
20    END ALGORITHM
```

Najvažnija stvar u algoritmu je operator mutacije. Operator mutacije kao argumente prima jedinku koja će se promijeniti, broj gena koji se mijenjaju (čvorova), vjerojatnost lokalne pretrage, vrstu bojanja za lokalnu pretragu i vrstu bojanja za globalnu pretragu.

Algoritam 3.4: Pseudokôd mutacije evolucijske strategije

```
1   MUTATION(unit, M, PROP, CS_GLOBAL, CS_LOCAL):
2       choice = random()
3       FOR i = 1 TO M:
4         index = random()
5         IF choice < PROP THEN
6           SETCOLOR(unit[i], CS_LOCAL)
7         ELSE
8           SETCOLOR(unit[i], CS_GLOBAL)
9         END IF
10  END MUTATION
```

3.4.5. Karakteristike algoritma

Algoritam ima relativno velik broj parametara, što ga čini teškim za prilagodbu konkretnom problemu (grafu). Veličine skupova za roditelje i djecu trebali bi biti relativno mali zbog vremena izvođenja. Većina parametara je vezana za mutaciju i to je najosjetljiviji dio algoritma. Potrebno je osigurati čestu lokalnu pretragu, ali s druge strane osigurati skakanje iz lokalnih minimuma. Algoritam često zapinje na platoima. Iako konvergencija može biti brža od agentskog algoritma vrijeme izvođenja je duže. Vremenska složenost algoritma je (LS je skraćeno za $LAMBDA\_SIZE$: $O(N \cdot (LS \cdot M + LS \cdot log(LS))) = O(N \cdot LS \cdot M + N \cdot LS \cdot log(LS))$. U praksi je čest slučaj: $LS \leq M$, pa je vremenska složenost: $O(N \cdot LS \cdot M)$. Algoritam polinomno ovisi o tri parametra, što predstavlja problem ako su oni relativno veliki. S jedne strane potrebno je osigurati dovoljan broj koraka algoritma kako bi se došlo do željene vri-

jednost funkcije cilja, broj djece LS ne smije biti premali - kako bi se zadržao genetski kod, a veličina mutacije M ne smije biti prevelika zbog velikog odstupanja, a s druge strane ne smije biti premala zbog spore konvergencije.

3.5. Simulirano kaljenje

Simulirano kaljenje također spada u skupinu evolucijskih algoritama. Njegova specifičnost je da se koristi samo jedna jedinka koja iz koraka u korak evoluira. Algoritam je inspiriran kaljenjem metala. Rješenje se nužno ne poboljšava iz koraka u korak, ali je vjerojatnost pogoršanja rješenja kako algoritam napreduje sve manja. Dopuštanjem lošijih rješenja nastoji se izbjeći lokalni optimum. Pretpostavka je da s vremenom algoritam dolazi u željeno područje globalnog optimuma. Na početku algoritam vrši globalnu pretragu, da bi s vremenom prostor stanja počeo pretraživati na finiji način. Simulirano kaljenje se često koristi kao alat u rješavanju problema bojanja grafova kao što se to prikazuje u radu [6].

3.5.1. Parametri algoritma

1. Početna temperatura - T_START
2. Globalni broj koraka algoritma - G
3. Broj iteracija s istom temperaturom - E
4. Faktor smanjenja temperature - α
5. Vjerojatnost lokalne vrste bojanja - $PROP$
6. Način bojanja za lokalnu pretragu - CS_GLOBAL
7. Način bojanja za globalnu pretragu - CS_LOCAL

3.5.2. Detalji algoritma

Algoritam kreće od početnog rješenja (jedinke) koja se označava s X. Prikaz rješenja je klasičan, svaki čvor je element u nizu a vrijednost elementa je boja čvora. Susjedno rješenje od X u oznaci S je rješenje koje ima sve elemente niza jednake kao i X osim točno jednog elementa. U svakom koraku algoritma iz rješenja X generira se susjedno rješenje S. Ako rješenje S ima vrijednost funkcije cilja manju od X, tada

se X zamjenjuje s S. U slučaju da je vrijednost funkcije cilja veća, tada se rješenje prihvaća s vjerojatnošću (ovisi o trenutnoj temperaturi T):

$$P(X = S) = e^{-\frac{(f(S)-f(X))}{T}} \tag{3.1}$$

Kako temperatura T pada, vjerojatnost prihvaćanja pada u 0, uz pretpostavku $f(S) \geq f(X)$:

$$\lim_{T \to 0} e^{-\frac{f(S)-f(X)}{T}} = 0 \tag{3.2}$$

U ovoj implementaciji simuliranog kaljenja koriste se dvije petlje. Jedna vanjska koja služi za smanjivanje temperature, a druga unutarnja koja traži rješenja u susjedstvu s istom temperaturom T. Koriste se dvije vrste traženja susjeda: lokalna i globalna. Prvo služi kako bi se našlo bolje rješenje, a druga vrsta traženja služi kako bi se rješenje pogoršalo u svrhu prelaska u novo područje prostora stanja.

Algoritam 3.5: Pseudokôd simuliranog kaljenja

```
1   ALGORITHM(T_START, G, E, ALFA, PROP, CS_GLOBAL, CS_LOCAL):
2     init(X)
3     FOR i = 1 TO G:
4       FOR j = 1 TO E:
5         S = X
6         IF random() < PROP THEN
7           SETCOLOR(X[random()], CS_LOCAL)
8         ELSE
9           SETCOLOR(X[random()], CS_GLOBAL)
10        END IF
11        dE = f(S) - f(X)
12        IF ( dE < 0 OR random() < EXP(-dE/T) ) THEN
13          X = S
14        END IF
15      END FOR
16    T = T * ALFA
17    END FOR
18  END ALGORITHM
```

3.5.3. Karakteristike algoritma

Algoritam je relativno jednostavan za implementaciju. Dobra strana je što je za razliku od drugih promatranih algoritama ima manju memorijsku složenost, zbog toga što koristi samo dvije jedinke (X i S). Za parametar početne temperature T nije potrebno ispitivati najbolju vrijednost, već se može izračunati kao. Za prvi korak u algoritmu

potrebno je uzeti lošije rješenje s 50% vjerojatnosti. Prvi korak greške se procijeni prema problemu i ponašanju algoritma: $\tilde{dE} \sim f(S_1) - f(X_1)$.

$$\text{T\_start} = -\frac{dE}{ln(P)} = 1.443 \cdot dE \tag{3.3}$$

Parametar α potrebno je namjestiti kako bi temperatura ne bi padala prebrzo - zapinjanje u lokalnim optimumima, ili kako ne bi padala presporo - slaba konvergencija. Vremenska složenost algoritma je: $O(G \cdot E)$, što je veoma dobro svojstvo jer samo dva parametra utječu na vrijeme izvođenja i to polinomno.

3.6. Genetsko kaljenje

Genetsko kaljenje [4] algoritam je koji kombinira dvije ideje: genetski algoritam i simulirano kaljenje. Ideja je ista kao i kod simuliranog kaljenje, samo što se u sustavu nalazi više jedinki (rješenja). Rješenja međudjeluju preko parametra slobodne energije. Na početku zadana je početna energija, koja se ravnomjerno rasporedi po svim jedinkama. Za svaku jedinku u sustavu napravi se mutacija i dobiva se susjedno stanje. Nova jedinka ima neku vlastitu energiju (funkcija cilja) te se prihvaća kao novo rješenje ako je vlastita energija nove jedinke manja od zbroja stare jedinke i djela slobodne energije koju je stara čestica dobila. Tako slobodna energija utječe na odabir lošijih jedinki.

3.6.1. Parametri algoritma

1. Ukupan broj iteracija - G
2. Broj jedinki - N
3. Početna energija - E
4. Faktor hlađenja - α
5. Vjerojatnost lokalne vrste bojanja - $PROP$
6. Način bojanja za lokalnu pretragu - CS_GLOBAL
7. Način bojanja za globalnu pretragu - CS_LOCAL

3.6.2. Detalji algoritma

Na početku se inicijaliziraju jedinke. Slobodna energija se postavi na početnu energiju i algoritam kreće u rad. Svaka jedinka dobije dio slobodne energije u obliku praga T. To znači da jedinka može odstupati od svojeg roditelja za najviše prag T da bude prihvaćena kao nova jedinka. Ako je prihvaćena, energija koja nije iskorištena, a to je razlika energije stare jedinke i nove jedinke (gdje se prag T nadodaje staroj jedinki) dodaje se u slobodnu energiju za tu iteraciju. Na kraju, da se ne bi dogodila divergencija, slobodna energija se množi s nekim faktorom $\alpha \in [0, 1]$.

Algoritam 3.6: Pseudokôd genetskog kaljenja

```
1   ALGORITHM(G, N, E, ALFA, PROP, CS_GLOBAL, CS_LOCAL):
2     UNITS[N]
3     FOR i = 1 TO N:
4       init(UNIT[i])
5     END FOR
6     FREE_ENERGY = E
7     FOR j = 1 TO G:
8       T = FREE_ENERGY / N
9       FREE_ENERGY = 0
10      FOR i = 1 TO N:
11        new_unit = MUTATION(UNIT[i], PROP, CS_GLOBAL, CS_LOCAL)
12        IF f(new_unit) < f(UNIT[i]) + T THEN
13          D = f(UNIT[i]) + T - f(new_unit)
14          FREE_ENERGY = FREE_ENERGY + D
15          UNIT[i] = new_unit
16        END IF
17      END FOR
18    FREE_ENERGY = FREE_ENERGY * ALFA
19    END FOR
20  END ALGORITHM
```

Algoritam 3.7: Pseudokôd mutacije genetskog kaljenja

```
1   MUTATION(unit, PROP, CS_GLOBAL, CS_LOCAL):
2     IF random() < PROP THEN
3       SETCOLOR(unit[random()], CS_LOCAL)
4     ELSE
5       SETCOLOR(unit[random()], CS_GLOBAL)
6     END IF
7   END MUTATION
```

3.6.3. Karakteristike algoritma

Algoritam izgleda veoma slično kao i simulirano kaljenje, samo što se odluka za prihvaćanje novog rješenja donosi na temelju praga tj. odstupanja funkcije cilja od stare jedinke. Međudjelovanje jedinki temelji se na slobodnoj energiji, koja se u svakoj iteraciji skuplja kao dodatak koji nije potrošen. To znači da ako u nekom koraku ni jedna nova jedinka nije prihvaćena, slobodna energija će biti 0, i u sljedećoj iteraciji prihvaćat će se samo strogo bolje nove jedinke. S druge strane ako u nekoj iteraciji jedna jedinka postane puno bolja od roditelja, tada u sljedećoj iteraciji algoritam dopušta novim jedinkama da mogu biti puno gora od roditelja, što omogućuje izlaženje iz lokalnih optimuma i globalno pretraživanje prostora stanja. Za rješavanje optimizacijskog problema bojanja grafova koriste se dvije vrste dobivanja susjednog rješenja (mutacije). Kao i za neke Prethodne algoritme, koriste se lokalna i globalna vrsta bojanja. Vremenska složenost algoritma je: $O(G \cdot N)$. S time da je to nešto veća vremenska složenost od simuliranog kaljenja ako se u obzir uzme računanje funkcije cilja za N jedinki.

3.7. Pronalaženje najpogodnijeg pohlepnog algoritma

Pohlepni algoritma u ovom radu koristi se na dva načina. Prvo se koristi kao metoda koja dovodi rješenje u neko dovoljno dobro rješenje, nakon čega minimizaciju funkcije preuzima neka metaheuristika. Druga način korištenja je kao dio metaheuristike u obliku operatora (npr. mutacija kod genetskog algoritma), metoda za izbjegavanje lokalnog minimuma (tuneliranje) ili pak metoda koja osigurava da postotak promjene boja naspram inicijalnog grafa ne prođe zadani prag. Kod korištenja pohlepnih algoritama u praksi se često koriste *ad-hoc* metode konfiguriranja istog. Prednost ovakvih metoda je što u relativno kratkom vremenu mogu isprobati različite konfiguracije i zatim odabrati najbolju. Nedostatak metode je u tome što ta konfiguracija ne jamči dobar rad nad svim grafovima (*No free lunch* teorem). Kao uvod u hiper-heuristike i postavljanje temelja za budući rad, iz grafova se želi izvući znanje o utjecaju vrsta sortiranja čvorova. Ovaj dio rada ima dva cilja: pronaći optimalni operator usporedbe čvorova koji kombinira sve dostupne operatore tj. vrste sortiranja. Drugi cilj je dovesti u vezu vrstu sortiranja i načina bojanja tj. empirijski pokazati koja vrsta sortiranja je najbolja za koji vrstu bojanja. Rezultati bi mogli pokazati određenu vezu između svojstava grafova i vrste sortiranja što bi se moglo koristi kao uvod u istraživanje hiper-heuristika. Metoda pronalaženja pogodnog pohlepnog algoritma sastoji se od koraka:

1. Generiranje grafa
2. Odabira vrste bojanja za pohlepni algoritam
3. Pokretanje pohlepnog algoritma s vrstom sortiranja *RND*
4. Iz svih pokretanja pohlepnog algoritma uzeti određen postotak najboljih rezultata
5. Za svaki čvor izračunati relativnu poziciju kao $\frac{x+1}{N}$ gdje je x indeks u slučajno sortiranom nizu čvorova, a N ukupan broj čvorova u grafu.
6. Za svaku relativnu poziciju izračunati svojstva tog čvora (stupanj, zasićenje, koliziju, grešku)
7. Nekom od metoda regresije pronaći funkciju čiji su argumenti svojstva čvora, a rezultat je relativna pozicija

Tako dobivena funkcija može se koristiti kao vrsta sortiranja. Za neke vrste regresija veoma lako se interpretiraju rezultati pa je lako odrediti u kojoj mjeri svojstva čvora utječu na dobar rezultat. Za različite vrste bojanja moguće je uspoređivati funkcije i uočiti ovisnosti. Za različite grafove usporedbom funkcija može se izvući znanje o pogodnom algoritmu za pojedine vrste grafova.

3.8. Tuneliranje

Tuneliranje je tehnika koja se može ugraditi u bilo koji optimizacijski algoritam i služi za izlazak iz lokalnog minimuma. Tehnika je vrlo slična *mutacijama* kod evolucijskih algoritama. Razlika je u tome što ova tehnika ne odabire slučajno novu točku iz prostora pretraživanja stanja, već nastoji odabrati točku koja ima približnu vrijednost funkcije cilja kao i trenutna točka koja se promatra (npr. relativna razlika funkcija cilja mora biti manja od nekog praga). Kako nova točka u prostoru pretrage ima sličnu vrijednost funkcije cilja, algoritam nastoji pronaći minimum u toj okolini, teorijski to znači da funkcija cilja može samo padati. U praksi se može dogoditi da i poraste jer se za novu točku ne zahtijeva da vrijednosti budu identične već približno jednake. Nova točka traži se tako što se po redu pretražuje dimenzija po dimenzija linearno. U problemu optimizacije bojanja grafa to znači da se po redu za svaki čvor ispituju boje, i ako za trenutni čvor nova boja za čvor ne utječe na vrijednost greške tada se ta boja

ostavlja tome čvoru. Ako se u pohlepnom algoritmu kao vrsta bojanja koristi "*SWAP*" metoda dobiva se očekivani efekt.

U navedenom pseudokôdu nova točka postavlja se u trenutnu. Zatim se vanjska petlja vrti po dimenzijama i za svaku se ispituju se svi elementi (npr. boje). Element koji za apsolutno odstupanje daje vrijednost manju od praga T, postavlja se kao element za tu dimenziju i potraga kreće na sljedeću dimenziju.

Algoritam 3.8: Metoda tuneliranja - implementacija

```
1   TUNNELING(f, X)
2     NEW_X = X
3     FOR i = 1 TO X.length()
4       FOR EACH element in DIMENSION(i):
5         NEW_X.dimension(i) = element
6         IF |f(NEW_X)-f(X)| < f(X)*T:
7           BREAK
8         NEX_X.dimension(i) = X.dimension(i)
9       END FOR
10    END FOR
11    RETURN NEW_X
12  END TUNNELING
```

3.9. Upravljanje postotkom promijenjenih čvorova

Kako je sekundarni kriterij optimizacije da postotak promijenjenih čvorova naspram inicijalnog stanja ne prođe neku granicu (konkretno 66%) potrebno je ostvariti mehanizam koji to omogućuje. U ovom radu koristi se pristup težnje boje da postane početna. U vrste bojanja implementira se mehanizam biranje boja, koji ako je početna boja jedna od mogućih, tada je dodjela početne boje vjerojatnija od svih ostalih.

4. Infrastruktura

4.1. Uvod

Za rješavanje zadanog optimizacijskog problema bojanja grafova i testiranja navedenih algoritama odlučeno je da će se izgraditi infrastruktura koja omogućuje brzi razvoj i testiranje algoritama. Ideja je izgraditi radnu okolinu u kojoj korisnik može iskoristiti već napisane module za standardne operacije koje se koriste i jednostavno ih uvesti u svoj algoritam. Koristi se jezik *Java* zbog relativno visoke apstrakcije, mogućnosti objektnog oblikovanja aplikacije i prenosivosti. Veoma važan cilj koji se nastoji ostvariti je infrastruktura koja je lako nadopunjiva novim modulima za buduću upotrebu. Moduli aplikacije koji su razvijeni za ovaj rad:

1. *main* - glavna metoda za pokretanje algoritama
2. *common* - generičke strukture podataka
3. *input* - implementacije različitih ulaznih formata
4. *output* - implementacija različitih izlaznih formata
5. *algorithm* - apstrakcija algoritma bojanja grafova
6. *unit* - jedinka (prikaz rješenja) koja se koristi u populacijskim algoritmima
7. *algorithmset* - modul za spremanje implementacija algoritama
8. *greedy* - implementacija pohlepnog algoritma
9. *color_selector* - implementacije vrsta bojanja
10. *order* - implementacija različitih vrsta sortiranja čvorova
11. *agents* - implementacija agentskog algoritma
12. *genetic* - implementacija genetskog algoritma (strategije)

13. *simulated_annealing* - implementacija simuliranog kaljenja
14. *structure* - najviša apstrakcija grafa - jednostavni graf
15. *weight_graph* - apstrakcija težinskog grafa
16. *ericsson_graph* - graf koji posjeduje strukture pogodne za rješavanje optimizacijskog problema
17. *stat* - generičke (statičke) metode koje se koriste za računanje statistika ili svojstava koja ne spadaju u druge strukture
18. *machine_learning* - modul za izvlačenje značajki iz grafa i pripremu podataka za strojno učenje

U nastavku slijedi detaljni opis nekih modula koji su najvažniji za rad infrastrukture. Detaljne upute o korištenju infrastrukture priložene su u digitalnom obliku uz ovaj rad.

4.1.1. Modul *input*

Funkcija ovog modula je osigurati učitavanja raznih formata zapisa grafa u datotekama. Graf u datoteci može biti zapisan proizvoljnim formatom, pa je cilj modula da obuhvati što više formata. Jedan od formata je specifičan za optimizacijski problem. Implementirani su i neki jednostavni formati.

4.1.2. Modul "*output*"

Kao što je funkcija prethodnog modula učitavanje, tako je funkcija ovog modula zapis grafa u datoteku. Podržan je jednostavan format koji zapisuje identifikator čvora i njegovu boju. Podržan je zapis koji graf pretvara u format potpun za optimizacijski problem, ali je veličina datoteke koja nastaje duplo manja od originalnog ulaznog formata. Osim zapisa grafa, ovaj modul omogućuje i ispis statistika u drugim formatima koji se mogu koristi u drugim alatima, primjerice alatima za strojno učenje.

4.1.3. Modul "*algorithm*"

Modul implementira strategiju kao oblikovni obrazac. Korisnik pri pokretanju algoritma stvara kontekst u kojem navodi konkretan algoritam. Nakon toga sučelje prema korisniku je samo kontekst. Svi algoritmi moraju imati implementirano isto sučelje.

Kao što se može vidjeti u programskom odsječku 4.1 klasa *GraphAlgorithmContext* ima privatni atribut tj. referencu koja pokazuje na konkretan algoritam. Algoritam se zadaje preko konstruktora. Implementirane su dvije metode istog naziva: *startAlgorithm*, koje kao parametar primaju strukturu podataka graf (*Graph*). Metode se razlikuju u drugom parametru, gdje se kao argument može dati skup identifikatora čvorova koje algoritam ne smije bojati. To nije jednako kao i nepromjenjivi čvorovi (jer to algoritam rješava interno), već se tu zadaju čvorovi koji su možda prije obojeni i s pokretanjem novog algoritma korisnik ih ne želi mijenjati.

Algoritam 4.1: Definicija klase i metoda za oblikovni obrazac strategija

```
1
2  public class GraphAlgorithmContext {
3
4    private GraphColoringAlgorithm algorithm;
5
6    public GraphAlgorithmContext(GraphColoringAlgorithm
       algorithm){
7      this.algorithm = algorithm;
8    }
9
10   public void startAlgorithm(Graph graph){
11     algorithm.startAlgorithm(graph);
12   }
13
14   public void startAlgorithm(Graph graph, Set<Integer>
       touchableNodes){
15     algorithm.startAlgorithm(graph, touchableNodes);
16   }
17
18 }
```

U programskom odsječku 4.2 pokazana je implementacija klase *GraphColoringAlgorithm*. Klasa je apstrakcija algoritma za bojanje grafova. Pri stvaranju novog konkretnog algoritma korisnik je obavezan implementirati metodu *algorithm*, koja je središte implementacije svakog algoritma. Uz to korisnik ima mogućnost ispitivanja da li smije pobojati čvor ili ne - *checkNode*. Parametre algoritma korisnik zadaje pri pozivu konstruktora za konkretni algoritam.

Algoritam 4.2: Apstraktna klasa algoritma

```
1
2  public abstract class GraphColoringAlgorithm {
3
4    private Set<Integer> touchableNodes = null;
```

```
5     protected Graph graph;
6
7     protected Set<Integer> getTouchableNodes(){
8       return this.touchableNodes;
9     }
10
11    protected boolean checkNode(int index){
12      int id = graph.getNodeId(index);
13      if (touchableNodes == null)
14        return true;
15      return touchableNodes.contains(id);
16    }
17
18    protected abstract void algorithm();
19
20    public void startAlgorithm(Graph graph){
21      touchableNodes = null;
22      this.graph = graph;
23      this.algorithm();
24    }
25
26
27    public void startAlgorithm(Graph graph, Set<Integer>
          touchNodeId){
28      touchableNodes = touchNodeId;
29      this.graph = graph;
30      this.algorithm();
31    }
32  }
```

4.1.4. Modul "*unit*"

Modul "*unit*" je generička klasa koja se koristi kao prikaz rješenja u populacijskim algoritmima. Zbog prirode problema koja donosi grafove s velikim brojem čvorova i brojem bridova nemoguće je pamtiti više instanci grafa zbog nedostatka memorije. Zbog toga se pamte samo parovi čvor-boja, a struktura grafa zadana je referencom. Klasa *GeneralUnit* pruža operacije:

- Brzo dohvaćanje funkcije cilja - *getError()* - $O(1)$
- Dohvaćanje broja čvorova - *getSize()* - $O(1)$
- Postavljanje boje čvoru - *setColor(id, color)* - $O(H(v))$
- Postavljanje strukture graf u boje jedinke - *updateGraph()* - $O(V(G))$

5. Rezultati

5.1. Metode testiranja algoritama

5.1.1. Testiranje metaheuristika

Nekoliko rezultata za metaheuristike je bitno kako bi se mogle uspoređivati. Dva svojstva koja najviše utječu na dobrotu metaheuristike su najmanja postignuta vrijednost funkcije cilja i stvarno vrijeme izvođenja. Osim toga veoma bitan dio je pronalaženje optimalnih parametara metaheuristika. Zbog toga se na skupu ispitnih grafova jedna metaheuristika testira s drugačijim parametrima, te se najbolji dobiveni rezultat neke metaheuristike uspoređuje s ostalim najboljim rezultatima drugih metaheuristika. Rad metaheuristika proučava se kroz iteracije. Pronalazi se ovisnost broja korištenih iteracija i vrijednosti funkcije cilja, ovisnost broja korištenih iteracija i postotka promjene inicijalnih boja te ovisnost promjene inicijalnih boja o vrijednosti funkcije cilja. Takve ovisnosti mogu pokazati neke karakteristike algoritma, kao što su brzina konvergencije, vrijeme zadržavanja u lokalnim minimumima te dati smjer za korištenje vrsta odabira boja pri nekoj metaheuristici.

Metaheuristike se testiraju tako što se početno stanje grafa dovede pomoću pohlepnog algoritma u neku točku čija je funkcija cilja ima puno manju vrijednost od početnog grafa. Cilj pohlepnog algoritma je da osigura validno bojenje za svaki čvor, te da se greška dovede do manje vrijednosti. Koristi se pohlepni algoritam u pet koraka. Iz iteraciju u iteraciju pohlepni algoritam nastoji bojiti graf pomoću *SDO* sortiranja i *ABW* odabira boja. Ako algoritam zastane u lokalnom minimumu, pomoću *FIT* sortiranja i *MF* odabira boja nastoji se iskočiti iz tog minimuma. Pokazalo se da nakon nekog broja koraka pohlepni algoritam konvergira ili čak u nekim slučajevima divergira. Odabrano je pet koraka jer se eksperimentiranjem pokazalo kao najbolja opcija, a da bi algoritmi i grafovi bili usporedivi tj. da pohlepni algoritam ne bi utjecao na brzinu kovergencije, za sve slučajeve testiranja ostavljena je ta ista inačica.

5.2. Slučajno pretraživanje

Kako bi se opravdalo traženje optimizacijskog algoritma za ovaj problem, potrebno je pokazati kako uniformna raspodjela boja po grafu ne daje zadovoljavajuće rezultate. Za ovu svrhu korišten je pohlepni algoritam koji za sortiranje čvorova koristi *RND* metodu te također *RND* vrstu bojanja. Efekt je isti kao da se obilaze čvorovi i svakom se da slučajna boja iz njegove domene. Nad svakim grafom pokreće pohlepni algoritam pokreće se deset puta, nakon čeka se rezultat usrednjuje i uspoređuje s početnom pogreškom grafa.

Tablica 5.1: Popis grafova za testiranje

Naziv grafa	Broj čvorova	Broj bridova	Inicijalna pogreška	Slučajno bojanje
graph_38301107	700	489300	$2.26 \cdot 10^{09}$	$4.40 \cdot 10^{07}$
graph_75806963	700	489300	$2.11 \cdot 10^{09}$	$4.59 \cdot 10^{07}$
graph_2105155221	700	489300	$1.94 \cdot 10^{09}$	$4.52 \cdot 10^{07}$
graph_1259253950	700	489300	$2.04 \cdot 10^{09}$	$4.66 \cdot 10^{07}$
graph_1065301380	700	489300	$2.21 \cdot 10^{09}$	$4.57 \cdot 10^{07}$
graph_2144584312	500	249500	$1.58 \cdot 10^{09}$	$2.44 \cdot 10^{07}$
graph_596114851	500	249500	$1.57 \cdot 10^{09}$	$2.47 \cdot 10^{07}$
graph_1195753610	500	249500	$1.39 \cdot 10^{09}$	$2.39 \cdot 10^{07}$
graph_598677215	500	249500	$1.40 \cdot 10^{09}$	$2.28 \cdot 10^{07}$
graph_1367025351	500	249500	$1.70 \cdot 10^{09}$	$2.48 \cdot 10^{07}$
graph_1371311789	900	809100	$2.85 \cdot 10^{09}$	$7.63 \cdot 10^{07}$
graph_1707332235	900	809100	$2.87 \cdot 10^{09}$	$7.69 \cdot 10^{07}$
graph_1878030959	900	809100	$2.72 \cdot 10^{09}$	$7.30 \cdot 10^{07}$
graph_2056239330	900	809100	$2.67 \cdot 10^{09}$	$7.38 \cdot 10^{07}$
graph_1067592528	900	809100	$2.60 \cdot 10^{09}$	$7.43 \cdot 10^{07}$
Kansai-C	8937	2259249	$1.68 \cdot 10^{07}$	$7.09 \cdot 10^{08}$
Tokai-B	4244	915283	$1.29 \cdot 10^{09}$	$2.41 \cdot 10^{08}$
Tokai-A	17066	5298396	$1.83 \cdot 10^{10}$	$9.51 \cdot 10^{08}$
Kansai-B	4925	1283991	$7.32 \cdot 10^{07}$	$3.21 \cdot 10^{08}$
Tokai-C	10518	2557187	$5.75 \cdot 10^{09}$	$5.53 \cdot 10^{08}$
Kansai-A	18807	7359859	$4.69 \cdot 10^{09}$	$1.41 \cdot 10^{09}$

Kako je očekivana greška u svim grafovima reda veličine 10^3 ili manje, vidljivo je da slučajno bojanje ne daje dobre rezultate, već rezultati imaju jedan do dva redova veličine manje vrijednosti od inicijalnog bojanja. Razlika bi bila još manja kada bi inicjalna bojanja bila validna, ali kako se takvi čvorovi dodatno kažnjavaju (prema funkciji cilja), inicijalna pogreška je veća.

5.3. Agentski algoritam

5.4. Simulirano kaljenje

5.5. Genetski algoritam

5.6. Genetsko kaljenje

5.7. Metode strojnog učenja za odair pohlepnog algoritma

6. Zaključak

Zaključak.

LITERATURA

[1] Dr. Hussein Al-Omari i Khair Eddin Sabri. New graph coloring algorithms, 2006. ISSN 1549. URL thescipub.com/pdf/10.3844/jmssp.2006.439.441.

[2] D. Berend i T. Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30: 185–205, 2010. URL http://www.openu.ac.il/Personal_sites/tamirtassa/Download/Journals/lp_moments.pdf.

[3] Francesc Comellas i Javier Ozon. Graph coloring algorithms for assignment problems in radio networks. U *in Radio Networks", Applications of Neural Networks to Telecommunications*, stranice 49–56. Inc. Pub, 1995.

[4] Mostafa A. El-Hosseini, Aboul Ella Hassanien, Ajith Abraham, i Hameed Al-Qaheri. Genetic annealing optimization: Design and real world applications. U Jeng-Shyang Pan, Ajith Abraham, i Chin-Chen Chang, urednici, *ISDA (1)*, stranice 183–188. IEEE Computer Society, 2008. ISBN 978-0-7695-3382-7. URL http://dblp.uni-trier.de/db/conf/isda/isda2008-1.html#El-HosseiniHAA08.

[5] M. R. Garey, D. S. Johnson, i L. Stockmeyer. Some simplified np-complete problems. U *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, stranice 47–63, New York, NY, USA, 1974. ACM. doi: 10.1145/800119.803884. URL http://doi.acm.org/10.1145/800119.803884.

[6] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, i Catherine Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Oper. Res.*, 39(3):378–406, Svibanj 1991. ISSN 0030-364X. doi: 10.1287/opre.39.3.378. URL http://dx.doi.org/10.1287/opre.39.3.378.

[7] Zbigniew Kokosinski, Krzysztof Kwarciany, i Marcin Kolodziej. Efficient graph coloring with parallel genetic algorithms. *Computers and Artificial Intelligence*, 24(2):123–147, 2005. URL http://dblp.uni-trier.de/db/journals/cai/cai24.html#KokosinskiKK05.

[8] Francisco Luna, Enrique Alba, Antonio J. Nebro, i Salvador Pedraza. Evolutionary algorithms for real-world instances of the automatic frequency planning problem in gsm networks. U Carlos Cotta i Jano I. van Hemert, urednici, *EvoCOP*, svezak 4446 od *Lecture Notes in Computer Science*, stranice 108–120. Springer, 2007. ISBN 978-3-540-71614-3. URL http://dblp.uni-trier.de/db/conf/evoW/evocop2007.html#LunaANP07.

[9] Anna Marino i Robert I. Damper. Breaking the symmetry of the graph colouring problem with genetic algorithms. U Darrell Whitley, urednik, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, stranice 240–245, Las Vegas, Nevada, USA, 8 2000. ISBN 1-58133-109-7. URL http://citeseer.nj.nec.com/374201.html.

[10] Christine L. Mumford. New order-based crossovers for the graph coloring problem. U Thomas Philip Runarsson, Hans-Georg Beyer, Edmund K. Burke, Juan J. Merelo Guervós, L. Darrell Whitley, i Xin Yao, urednici, *PPSN*, svezak 4193 od *Lecture Notes in Computer Science*, stranice 880–889. Springer, 2006. ISBN 3-540-38990-3. URL http://dblp.uni-trier.de/db/conf/ppsn/ppsn2006.html#Mumford06.

[11] Riccardo Poli i Mario Graff. There is a free lunch for hyper-heuristics, genetic programming and computer scientists. U Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, i Marc Ebner, urednici, *EuroGP*, svezak 5481 od *Lecture Notes in Computer Science*, stranice 195–207. Springer, 2009. ISBN 978-3-642-01180-1. URL http://dblp.uni-trier.de/db/conf/eurogp/eurogp2009.html#PoliG09.

[12] Biman Ray, Anindya J Pal, Debnath Bhattacharyya, i Tai hoon Kim. An efficient ga with multipoint guided mutation for graph coloring problems. 2010.

[13] Ehsan Salari i Kourosh Eshghi. An aco algorithm for the graph coloring problem, 2008. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01662331.

[14] R.J. Wilson. *Introduction to graph theory*. Longman Scientific & technical. Longman, 1985. URL http://books.google.hr/books?id=1-nuAAAAMAAJ.

[15] D. H. Wolpert i W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, Travanj 1997. ISSN 1089-778X. doi: 10.1109/4235.585893. URL http://dx.doi.org/10.1109/4235.585893.

Bojanje grafova prilagodljivim metaheurističkim postupcima

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.