SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 659

Bojanje grafa prilagodljivim postupcima lokalne pretrage

Luka Marasović

Zagreb, lipanj 2014.

SADRŽAJ

1. Uvod

Tema diplomskog rada je problem bojanja grafova i njegova primjena na varijantu problema, problem dodjele radio frekvencija koji se pojavljuje u različitim tipovima bežičnih komunikacijskih mreža. Kako je problem bojanja grafova NP-težak postoji niz predloženih metoda za rješavanje od kojih su se neke pokazale manje ili više uspješne. Svrha ovog rada je implementirati metode koje će biti dovoljno robusne i brze kako bi mogle ponuditi kvalitetna rješenja za grafove čija veličina uvelike odskače od grafova na kojima se testiraju najmoderniji heuristički algoritmi. (9; 6)
Rad je strukturiran tako da drugo poglavlje sadrži definicije koje se koriste kad razmatramo problem bojanja grafova, njegovu računsku složenost i pojmove koje vezujemo uz algoritme s kojima radimo. U ovom poglavlju je dan pregled dostupnih varijanti bojanja grafova vezanih uz dodjelu frekvencija i na kraju poglavlja je opisan problem s kojim radimo. Treće poglavlje sadrži pregled algoritama koji su korišteni u radu, a četvrto poglavlje sadrži programsko ostvarenje. U petom poglavlju je napravljena analiza ulaznih grafova i pregled dobivenih rezultata. Zadnje poglavlje sadrži popis korištene literature, a u predzadnjem poglavlju se ocjenjuje uspješnost primijenjenih algoritama s obzirom na zadani problem rada.

2. Problem bojanja grafova

Problem bojanja grafova je vezan uz teoriju grafova, granu matematike koja proučava grafove. Osim definicija vezanih uz teoriju grafova, dodatno ćemo razmatrati definicije računske teorije složenosti te opis varijanti problema dodjele radio frekvencija. Na samom kraju poglavlja opisan je problem koji rad obrađuje.

2.1. Definicije

2.1.1. Definicije teorije grafova

Definicija 1. ***Graf*** G *sastoji se od nepraznog konačnog skupa* $V(G)$*, čije elemente zovemo vrhovi grafa* G *i konačnog skupa* $E(G)$ *različitih dvočlanih podskupova skupa* $V(G)$ *koje zovemo bridovi. Skup* $V(G)$ *zovemo skup vrhova i ako je jasno o kojem je grafu* G *riječ označava se kraće samo s* V*, a skup* $E(G)$ *zovemo skup bridova i označava se kraće samo s* E*.*

Definicija 2. *Za brid* $e = \upsilon, \omega$ *kažemo da spaja vrhove* υ *i* ω *i bez mogućnosti zabune kraće ga pišemo* $\upsilon\omega$*. U toj situaciji kažemo da su vrhovi* υ *i* ω *grafa* G ***susjedni****. Također, kažemo da je vrh* υ ***incidentan*** *s bridom* e *te da je vrh* ω ***incidentan*** *s* e*. Brid koji započinje i završava u istom vrhu naziva se* ***petlja****. Brid je višestruk ako postoji drugi brid kojemu odgovaraju isti vrhovi.*

Definicija 3. *Graf se naziva* ***jednostavnim*** *ako je neusmjeren (bridovi nemaju smjer), nema petlji i između bilo koja dva vrha nema više od jednog brida.*

Definicija 4. ***Stupanj*** *vrha* υ *grafa G je broj bridova koji su incidentni s* υ*. Označavamo ga s* $deg(\upsilon)$*. Dogovorno, ako je vrh* υ *petlja, onda ona broju* $deg(\upsilon)$ *doprinosi s 2. Vrh stupnja 0 zovemo izolirani vrh, a vrh stupnja 1 zovemo krajnji vrh.*

Definicija 5. *Za zadane disjunktne grafove* $G_1 = (V(G_1),\ E(G_1))$ *i* $G_2 = (V(G_2),\ E(G_2))$ *definiramo njihovu uniju* $G_1 \bigcup G_2$ *kao graf*
$G_1 \bigcup G_2 = (V(G_1) \bigcup V(G_2), E(G_1) \bigcup E(G_2))$.

Definicija 6. *Graf je **povezan** ako se ne može prikazati kao unija neka dva grafa. U suprotnom kažemo da je graf **nepovezan**.*

Definicija 7. ***Podgraf** grafa G je graf čiji vrhovi pripadaju skupu V(G), a bridovi skupu E(G).*

Definicija 8. ***Bojanje** zadanog grafa G je pridruživanje nekog skupa boja skupu vrhova od G, tako da svakom vrhu pridružimo jednu boju, tako da su susjednim vrhovima pridružene različite boje. Za bojanje ćemo reći da je **k-bojanje** ako smo upotrijebili k boja u ovom pridruživanju. Graf G je **k-obojiv**, ako postoji s-bojanje grafa G, za neki s <= k.*

Definicija 9. *Ako je graf G k-obojiv, ali nije (k-1)-obojiv, onda kažemo da je G **k-kromatski**. Dotični najmanji broj boja **k** s kojim se graf može pobojati nazivamo **kromatski broj** te ga označavamo s χ**(G)**.*

Definicija 10. *Brid koji spaja dva vrha iste boje zovemo **konfliktni brid**.*

Definicija 11. ***Konfliktni vrhovi** definiraju se kao vrhovi koji se nalaze na krajevima konfliktnog brida.*

Definicija 12. *Težinski graf je uređeni par (G,ω) gdje je G graf, a ω:I$\rightarrow R_0^+$ funkcija koju nazivamo **težinska funkcija**.*

Definicija 13. ***Prostor rješenja** Ω - gledajući problem bojanja grafa $(G(V,E),k)$ gdje je k broj dostupnih boja, prostor rješenja Ω predstavlja sva moguća bojanja od G; pa je $\mid \Omega \mid = \mid V \mid^k$.*

Definicija 14. ***Susjedstvo** S' nekog bojanja $s \in \Omega$ je skup svih mogućih susjeda s' koje je moguće dobiti promjenom boje jednom vrhu $s \rightarrow s'$.*

Definicija 15. *Bojanje s za graf $G(V,E)$ nazivamo **kompletnim** ako svi vrhovi $v \in V$ imaju definiranu boju $c(v) \in \{1,...,k\}$, inače govorimo o **parcijalnom** bojanju.*

Definicija 16. *Za bojanje s grafa $G(V,E)$ u kojem ne postoji par vrhova $u,v \in V$ takvih da $\{u,v\} \in E$ i $c(v) = c(u)$ govorimo o **legalnom** bojanju. Ako u bojanju pak postoji par vrhova koji zadovoljavaju $c(v) = c(u)$ tj. imamo konfliktnih vrhova govorimo o **nelegalnom** bojanju.*

2.1.2. Definicije računske teorije složenosti

Definicija 17. *Problem odluke je problem koji uvijek ima odgovor da ili ne.*

Definicija 18. *Problem odluke nazivamo P-problemom ako imamo algoritam koji daje odgovor na njega u vremenu polinomno ovisnom o ulaznim podacima.*

Definicija 19. *Kažemo da je neki problem odluke sadržan u klasi NP, odnosno naziva se N-problem, ako za njega postoji algoritam koji u polinomnom vremenu može ispitati je li neko predloženo rješenje stvarno rješenje danog problema.*

Definicija 20. *NP-potpun problem je problem odluke koji je sadržan u klasi NP i koji ima svojstvo da se svaki drugi problem iz klase NP može polinomno reducirati na njega.*

Definicija 21. *Kažemo da je problem NP-težak ako postoji NP-potpun problem koji ima svojstvo da se može polinomno reducirati na njega.*

2.1.3. Definicije algoritama

Definicija 22. *Heuristika je tehnika rješavanja koja brže od klasičnih pristupa dolazi do približnog rješenja. Nauštrb brzine mijenjamo optimalnost, točnost ili preciznost.*

Definicija 23. *Metaheuristika je skup algoritamskih koncepata koje koristimo za definiranje i usmjeravanje heurističkih metoda primjenjivih na širok skup problema.*

2.2. Problem dodjele frekvencije

Kako se problem dodjele frekvencija primjenjuje na različite probleme kao što su GSM mreže, satelitska komunikacija, prijenos televizijskih kanala koji svaki imaju svoj set specifičnih zahtjeva u industriji postoji više modela problema (1). Osnovni problem dodjele frekvencija sastoji se od tri dijela:

Ograničenja Definiramo ograničenja koja moramo ispoštovati kako bi rješenje bilo ispravno.

Interferencija Definiramo interferenciju, tj. kako i kada smatramo da dva vrha imaju interakciju.

Funkcija cilja Definiramo funkciju koju minimiziramo s obzirom na ograničenja i interferenciju.

Interferencije možemo prikazati uz pomoć interferencijskog grafa G = (V,E) gdje je svaka antena vrh v ∈ V. Svi vrhovi υ i ω za koje su moguće interferencije signala su povezani bridom $\{\upsilon,\omega\}$. Ako varijanta problema zahtjeva dodjelu više frekvencija po jednoj anteni, u interferencijskom grafu je moguće dodati više vrhova. Za svaki par frekvencija $f \in F(\upsilon)$ i $g \in G(\omega)$ kažnjavamo s obzirom na definiranu interferenciju što označavamo s $p_{\upsilon\omega}(f, g)$. Kada primijeniti kaznu ovisi od modela, česta je uporaba sljedećih pravila udaljenosti frekvencija:

1. $|f - g|$ ako su frekvencije udaljene manje od $d_{\upsilon\omega}$.
2. $|f - g| = 0$ Kazniti ako se radi o istim frekvencijama.
3. $|f - g| = 1$ Kazniti ako se radi o susjednim frekvencijama.

Ova pravila se mogu koristiti i u kombinacijama, te s različitim utjecajem na kaznu. Ako za određeni $p_{\upsilon\omega}$ odaberemo iznimno velike vrijednosti, udaljenost frekvencija može postati čvrsto ograničenje. Ako gledamo podjelu s obzirom na rješenje zadanog modela dijelimo ih na kompletna i parcijalna rješenja. U kompletna spada *feasible frequency assignent problem* (F-FAP) a u parcijalnim imamo sljedeće varijante:

Max-FAP *Maximum Service FAP* kao funkciju cilja postavlja dodjelu što više frekvencija - moguće je da vrhu bude dodijeljeno manje frekvencija od broja zadanih.

MB-FAP *Minumum Blocking FAP* za funkciju cilja postavlja težinski prosjek vjerojatnosti blokiranja za sve vrhove $v \in V$ s $n(v)$ u okruženju gdje su pojedini vrhovi aktivni u nekom trenutku kao prilikom poziva.

MO-FAP *Minimum Order FAP* kao funkciju cilja postavlja broj upotrijebljenih frekvencija.

MS-FAP *Minimum Span FAP* kao funkciju cilja želi minimizirati raspon između najmanje i najveće dodijeljene frekvencije.

MI-FAP *Minimum Interference FAP* kao funkciju cilja koristi sumu kazni za odabir frekvencija te ne gleda na kazne kao čvrsta ograničenja koja je potrebno zadovoljiti, iako je dodatno moguće primijeniti i takva ograničenja.

2.3. Opis problema

Cilj rada je optimizirati parametre *Radio Access Network-a* (RAN) koji je dio telekomunikacijske tehnologije koja omogućava spajanje raznih uređaja na mrežu. U okviru

zadatka potrebno je za svaku antenu dodijeliti *Scrambling Code* (SC) koji služi kako bi uređaj spojen na mrežu prilikom kretanja kroz prostor mogao odrediti antenu na koju će se spojiti. Ako dvije bliske antene dijele identični kod, dolazi do problema. Rješenje ovog problema je dodjela različitih SC-a svakoj anteni, no to nije uvijek moguće s obzirom na ograničeni broj kodova i velik broj antena.

2.3.1. Model problema

Problem je zapisan u obliku težinskog grafa G u kojem svaki vrh $v \in V(G)$ predstavlja antenu. Ako postoji mogućnost da uređaj koji se spaja na mrežu nije siguran koju antenu koristiti, između vrhova v i ω koji predstavljaju te antene dodajemo brid $e = v, \omega$. Težina brida e ovisi o udaljenosti antena i jačini signala. Boje u grafu predstavljaju SC te su inicijalno postavljene na pripadne vrijednosti.
Cilj optimizacije je minimizirati sumu (MI-FAP) svih bridova koji imaju vrhove jednake boje uz sljedeća pravila:

1. Svaki vrh može pripadati trima grupama: A, B ili C. Bridovi između vrhova različitih grupa ne doprinose sumi.
2. Ako je zabranjena promjena boje vrhu, vrh se ne smije bojati.
3. Bridovi ne doprinose sumi ako nije dozvoljena promjena boje za oba vrha.
4. Svaki vrh kojem je dozvoljena promjena boje ima domenu - listu boja iz kojih je moguć odabir.

Ispravnim bojanjem se smatra bojanje koje zadovoljava sljedeće uvjete:

1. Vrhovi kojima nije dozvoljena promjena imaju inicijalnu boju.
2. Svaki vrh kojem je dozvoljena promjena ima boju iz pripadne mu domene.
3. Ukupan postotak promijenjenih boja vrhova od inicijalnog stanja grafa ne prelazi 66%.
4. Vrijeme izvođenja je ograničeno na 4 sata.

3. Algoritmi

Problem bojanja grafova je NP-težak optimizacijski problem, a provjera je li graf $k - obojiv$ za $k \geq 3$ je NP-kompletan problem. Uz pretpostavku $P \neq NP$ nema algoritma koji može riješiti problem u polinomijalnom vremenu. Iako nemamo algoritme koji brzo mogu pronaći optimalno rješenje, razvijene su i upotrijebljene razne metode kojima se postižu relativno dobri rezultati (9; 6). Dostupne metode možemo podijeliti u dvije skupine: konstruktivne i optimizacijske metode(9).
Konstruktivne metode uzimaju neobojen graf G i odabirom boje za pojedini vrh grafa stvaraju k-obojen graf gdje je $k \geq \chi(G)$.
Optimizacijske metode istražuju prostor rješenja često u kombinaciji s jednostavnim konstruktivnim metodama. Razlikujemo ih po bojanjima s kojima rade:

Kompletna legalna bojanja Metoda radi s legalnim k-bojanjem i permutacijama rješenja se pokušava smanjiti k.

Kompletna nelegalna bojanja U bojanju su dozvoljeni konfliktni vrhovi, te se često za funkciju cilja koristi broj konfliktnih vrhova koja se minimizira.

Parcijalna legalna bojanja Prilikom bojanja imamo parcijalno legalno bojanje i skup neobojenih vrhova V' te je cilj pronaći rješenje za koje je V' = $\emptyset$.

Algoritmi korišteni u radu prilikom rada optimiziraju kompletno nelegalno bojanje. Koriste se pohlepni algoritmi za brzo dobivanje relativno dobrih bojanja i perturbaciju rješenja te algoritmi lokalne pretrage za pretragu prostora rješenja. U određenoj problematici korišten je genetski algoritam u kombinaciji s neuronskom mrežom kako bi se ispitala mogućnost učenja nekih karakteristika dobrog redoslijeda bojanja. Određeni dijelovi algoritama koji su izmjenjivi za više pristupa su izdvojeni u operatore.

3.1. Operatori

U radu (2) se razmatraju pohlepni algoritmi i vidljivo je da korištene heuristike imaju sljedeće ponašanje - prvo se vrši odabir vrha koji ćemo bojati, i zatim se pomoću nekih

lokalnih svojstava vrha donosi odluka o boji koji su u ovom radu podijeljeni u operatore odabira boje i odabira vrha. Za genetski algoritam i algoritme lokalne pretrage dodatno definiramo i funkcije cilja koje dotični žele minimizirati.

3.1.1. Odabir boje

MF *Minimise Fitness*
: Odabire se boja u kojoj je najmanja funkcija greške.

MC *Minimise Collision*
: Odabire se boja za koju imamo najmanje konfliktnih susjeda.

MD *Minimise Domain*
: Odabire se boja koja se najmanje pojavljuje u domenama susjednih vrhova.

ABW *Awoid Bigger Weights*
: Odabir boje se vrši eliminacijom potencijalnih boja kandidata iz skupa dostupnih boja gledajući boje koje imaju susjedni vrhovi s kojima dijelimo najteže bridove. Ako postupkom eliminacije u listi dostupnih boja ostane samo jedna, odabiremo nju, a ako ih je više, odabiremo početnu boju vrha ako je to dozvoljeno stanje, a ako ne, proizvoljno vršimo odabir.

ABWMD *Awoid Bigger Weights Minimise Domain*
: Slučaj kad imamo više preostalih boja prilikom izvođenja operatora *ABW* proširujemo dodatnom logikom - od preostalih boja odabiremo onu boju koja se nalazi u najmanje domena susjednih vrhova.

SA *Stochastic Acceptance*
: Nasumično vršimo odabir boje iz skupa dostupnih, a odluku o prihvaćanju boje prihvaćamo s vjerojatnošću ovisnoj o funkciji greške u toj boji. Postupak se ponavlja dok neka boja nije prihvaćena.

RND *Random*
: Nova boja se nasumično odabire iz skupa dostupnih boja.

3.1.2. Odabir vrha

Odabir vrha se vrši na temelju redoslijeda koji formiramo na temelju neke od značajki vrhova.

DEGREE Vrhovi su sortirani po stupnju, od višeg prema manjem stupnju. (LDO) (2)

FITNESS Vrhovi su sortirani po sumi težina konfliktnih bridova. Vrhovi s većim vrijednostima imaju prednost.

MAX FITNESS DIFFERENCE Vrhovi su sortirani po maksimalnoj razlici sume težina konfliktnih bridova koju mogu poprimiti u nekoj drugoj boji tj. vrh koji za sljedeći korak nudi najveće smanjenje ima prednost.

COLLISION Vrhovi su sortirani po broju konfliktnih bridova, prvo odabiremo vrhove s više konfliktnih susjeda.

MAX COLLISION DIFFERENCE Vrhovi su sortirani po razlici u broju konfliktnih bridova koju mogu potencijalno postići, tj. vrhovi koji najviše mogu smanjiti broj susjeda s kojima dijele boju imaju prednost.

SATURATION Vrhovi se sortiraju po zasićenosti, odnosno broju boja koje se nalaze u njegovom susjedstvu.(SDO) (2) vrhovi s više boja u susjedstvu imaju prednost.

SATURATION DEGREE Vrhovi su sortirani po zasićenosti, a za slučajeve kad imamo jednake vrijednosti, primjenjujemo *Degree*.

COLOR Vrhovi su sortirani po boji.

FREE COLORS Vrhovi su sortirani po broju dostupnih boja gdje su dostupne boje one boje u njihovoj domeni koje se ne pojavljuju u susjedstvu. Prednost imaju vrhovi s manje dostupnih boja.

HYBRID Kombinacija *Free Colors* i *Max Collision Difference*. Prednost pri odabiru imaju vrhovi koji imaju mali broj dostupnih boja, a ako imamo jednake vrijednosti, uspoređujemo koji vrh će najviše smanjiti broj konfliktnih bridova.

3.1.3. Funkcija cilja

Algoritmi lokalne pretrage i genetskog algoritma prilikom donošenja odluka razmatraju više potencijalnih rješenja koje mogu vrednovati po različitim funkcijama cilja:

– **Suma težina** Suma težina je zbroj težina svih konfliktnih bridova.

– **Suma konfliktnih bridova** Suma konfliktnih bridova je broj konfliktnih bridova u grafu.

– **Postotak promijenjenih vrhova** S obzirom na inicijalno bojanje grafa, gledamo postotak vrhova koji su promijenili boju.

3.2. Pohlepni algoritmi

Pohlepni algoritmi u svakom koraku biraju lokalno najbolje rješenje kako bi potencijalno pronašli globalni optimum. U većini slučajeva daju samo relativno brzu aproksimaciju najboljeg rješenja. Kao takvi, često se koriste u problemu bojanja grafova kako bi dali polaznu točku ostalim algoritmima, bio to populacijski algoritam ili algoritam lokalne pretrage. Najranija rješenja predlažu *first-fit* pristup koji uzima svaki vrh i pridjeljuje mu najmanju dostupnu boju za koju nema konflikta i pri tome kvaliteta bojanja ovisi o poretku bojanja. Kao predloženi poredak spominje se sortiranje po stupnju nekog vrha te dinamičko kretanje po grafu tako da idemo u vrh koji ima najveću zasićenost (2). Korišteni algoritam radi s kompletnim nelegalnim bojanjem te u svakom koraku odabire jedan vrh i donosi odluku koju boju pridjeliti tom vrhu. Jednom donesena odluka se ne mijenja, već se radi prolaz kroz ostale vrhove koji nisu posjećeni. U nastavku je prikazan pseudokod pohlepnog algoritma koji koristi operator odabira boje *ColorSelect* te koristi operator odabira vrha *Comparator* koji se koristi za sortiranje vrhova. *NodeList* predstavlja listu vrhova s kojom radimo.

Algoritam 3.1: Pseudokôd pohlepnog algoritma

```
1  INPUT
2    NodeList
3    Comparator
4    ColorSelect
5  WHILE NodeList not empty
6    Comparator.sort(NodeList);
7    node := NodeList.First;
8    IF (node.allowedchange == TRUE)
9      node.color := ColorSelect.select(node);
10   END IF
11   NodeList.RemoveFirst;
12 END WHILE
```

3.3. Genetski algoritam

Genetski algoritmi (u nastavku GA) su heuristička metoda optimiranja koja imitira prirodu i prirodni evolucijski proces te predstavlja robustan proces pretraživanja prostora rješenja i kao takav se često nameće kao logičan izbor za problem bojanja grafova. Najbolji rezultati (9) koji se postižu na većinama instanci problema bojanja grafova koriste genetske algoritme uz kombinaciju lokalne pretrage. GA su specifični po tome što rade s populacijom rješenja tj. jedinkama, i pri tome koriste operatore križanja i

mutacije. Križanje se koristi kako bi od dvije ili više jedinki koje nazivamo roditelji dobili nove jedinke. Mutacija je unarni operator koji vrši neku promjenu na jedinki. Kako će se prikazati pojedino rješenje te implementacija operatora križanja i mutacije varira s obzirom na problematiku koja se rješava. Većina GA koriste binarni zapis pojedinog rješenja, no nekad to nije praktično. Cilj GA u ovom radu nije konkretno bojanje te shodno tome pojedino rješenje neće biti zapis boje za pojedini vrh, već će se tražiti redoslijed vrhova za koji pohlepni algoritam uz neki operator odabira boje daje dobre rezultate. Ovaj redoslijed će se iskoristiti za učenje neuronske mreže koja bi potencijalno mogla otkriti sortiranje vrhova na temelju nekih njihovih obilježja.

3.3.1. Zapis jedinke

Jedinka u GA predstavlja jedan zapis rješenja, te više jedinki čini populaciju. Kako bi rješenje prikazali kao redoslijed iskorištena je lista vrhova u kojem se vrhovi ne ponavljaju, dakle više jedinki možemo gledati kao permutirane nizove istog skupa vrhova. Cijeli zapis rješenja često nazivamo kromosom, a pojedini vrh gen.

3.3.2. Selekcija

Kako bi GA konvergirao, bitna komponenta je selekcija. Selekcijom iz populacije biramo roditelje koji će stvoriti novu jedinku ili kompletnu novu populaciju. U radu se za jednog roditelja odabire najbolja jedinka u trenutnoj populaciji po nekoj funkciji cilja kao što je suma konfliktnih bridova, broj konfliktnih bridova, ili pak postotak promijenjenih vrhova, a drugi roditelj je nasumično generirani permutirani niz vrhova. Eksperimentalno se pokazalo da zbog veličine kromosoma (3000+ vrhova) GA relativno brzo prestanu konvergirati i operator mutacije ovdje nije adekvatan da ponudi izlaz iz lokalnog optimuma.

3.3.3. Križanje

Križanje u GA omogućava robusnu pretragu prostora rješenja i često dolazi do izražaja na početku rada algoritma. Za križanje se koristi *Partially Matched Crossover* (8) zbog svojstva da za dva ulazna permutirana niza vraća permutirani niz. Nova jedinka se dobije tako da dio niza uzmemo iz jednog roditelja, a ostale dijelove iz drugog roditelja pazeći pri tome da geni nisu već prisutni. Ako je gen već prisutan u novoj jedinci, vršimo zamjenu tako da pronađemo lokaciju gena u prvom roditelju, i zamijenimo ga

s genom koji se na toj lokaciji nalazi u drugom roditelju. Postupak se ponavlja sve dok više ne možemo napraviti zamjenu.

3.3.4. Mutacija

Mutacija u GA ima ulogu čuvanja raznolikosti jer bolja rješenja obično relativno brzo potisnu lošija. Mutacija obično utječe na mali broj gena unutar kromosoma jedinke, i to s malom vjerojatnošću. Za mutaciju koristimo zamjenu dva gena uz određenu stopu mutacije. Nakon što se obavi operator križanja, za svaki gen u zapisu određujemo je li nasumičan broj iz intervala $0,1$ manji od zadane stope mutacije. Svaki put kad pronađemo dva gena koji zadovoljavaju stopu, zamijenimo ih.

3.3.5. Detalji algoritma

Korišten je generacijski GA, što znači da algoritam iz populacije odabire roditelje i iz njih stvara novu populaciju nad kojom se ponovno vrši odabir i stvara nova populacija koja se u ovom kontekstu naziva generacija. Eksperimentalno je utvrđeno da ABW operator odabira boje pruža zadovoljavajuće rezultate s obzirom na sumu konfliktnih bridova i postotka promijenjenih vrhova te se koristi kao operator odabira boje kojem tražimo najbolji redoslijed bojanja. Parametri koji utječu na rad algoritma su:

– **startSeed** Moguće je dodati jedinku kao polazišnu točku algoritmu, to može biti raspored vrhova koji je dobiven primjenom nekog od operatora odabira vrha.

– **populationSize** Veličina populacije po generaciji.

– **numberOfGenerations** Broj generacija prije zaustavljanja algoritma.

Algoritam 3.2: Pseudokôd genetskog algoritma

```
1  INPUT
2   NodeList
3   startSeed
4   populationSize
5   numberOfGenerations
6  INITIALIZATION
7   chromosomeBest := startSeed;
8  FOR (i = 0 to numberOfGenerations)
9   FOR(j = 0 to populationSize)
10    chromosomeRandom := generateRandomChromosome;
11    population[j] := mutate(pmx(chromosomeBest,
          chromosomeRandom));
12  END FOR
13  chromosomeBest := selectBest(population);
```

3.4. Umjetna neuronska mreža

Umjetne neuronske mreže isto kao genetski algoritmi inspiraciju pronalaze u prirodi, tj. u funkcioniranju neuronskih mreža, te često izostavljamo pridjev umjetne koji se pretpostavlja iz konteksta. Neuronske mreže se sastoje od neurona i sinapsi koje omogućuju interakcije između neurona uz relativno jednostavna pravila i dovoljno velik broj postiže kompleksne operacije uz paralelizam. Umjetna neuronska mreža je masivno paralelni distribuirani procesor koji je dobar za pamćenje iskustvenog znanja te s neuronskim mrežama dijeli dva aspekta:

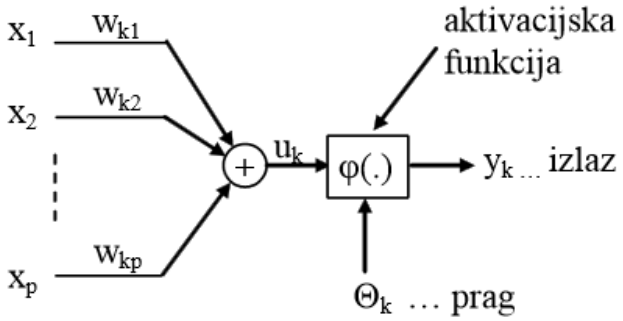
– Proces učenja kojim se stječe znanje

– Veze između neurona se koriste za spremanje znanja

Neuronske mreže su dobre za razne probleme kao što je prepoznavanje znakova, predviđanje kretanja cijena dionica, tržišta, klasifikaciju i razne druge probleme.

3.4.1. Karakteristike neuronske mreže

Neuronske mreže karakterizira način na koji predstavljaju neuron, aktivacijska funkcija koja određuje hoće li i kakav izlaz taj model neurona dati, te arhitektura same mreže. Često korišteni model predstavlja skup sinapsi koji svaki ima svoju težinu ($w_{k1}..w_{kp}$), sumator za zbrajanje težinskih ulaza ($x_1..x_p$) na sinapsama, te nelinearnu aktivacijsku funkciju φ koja ograničava izlaz. Prag Θ_k se često reprezentira kao dodatni ulaz fiksne težine sinapse -1 (12).

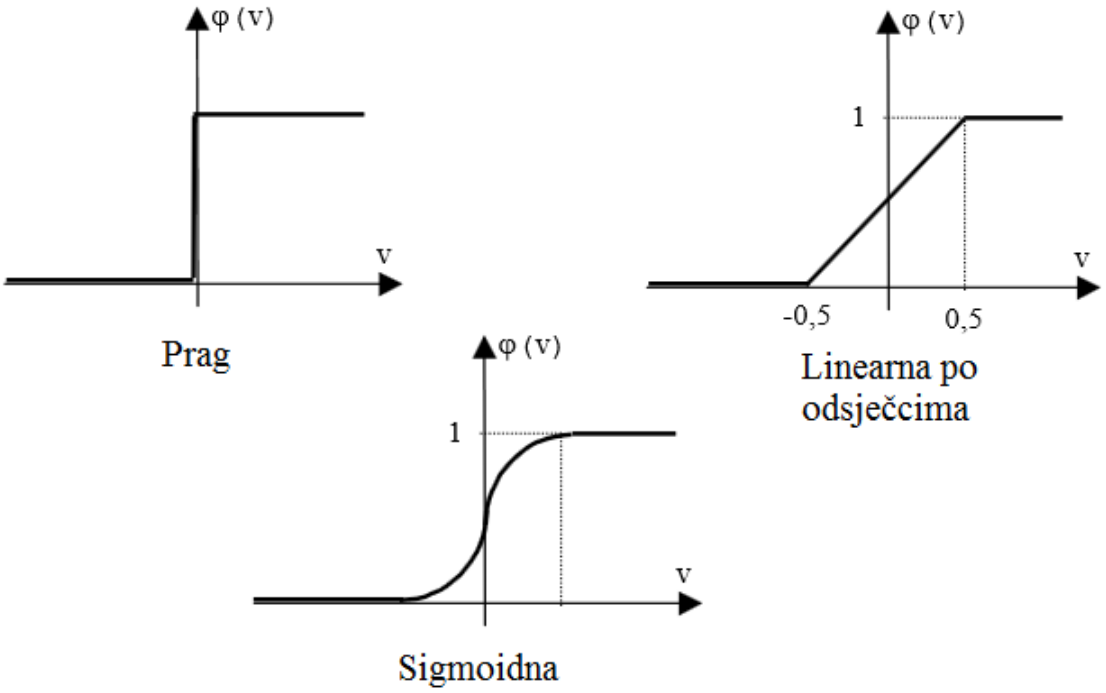


$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

$$y_k = \varphi\left(u_k - \Theta_k\right)$$

Slika 3.1: Model neurona

Za aktivacijske funkcije tri najčešće korištene su prikazane na slici 3.2.



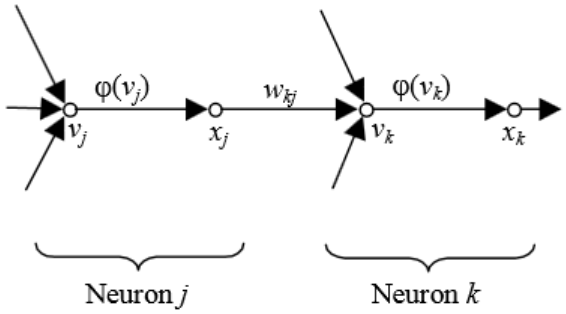
Slika 3.2: Aktivacijske funkcije

Postoji više načina kako povezati neurone u mrežu kod umjetnih neuronskih mreža te arhitekture dijelimo na:

- Jednoslojne mreže bez povratnih veza
- Višeslojne mreže bez povratnih veza
- Mreže s povratnim vezama
- Ljestvičaste mreže

3.4.2. Učenje neuronske mreže

Kako bi neuronska mreža mogla analizirati podatke i na ulazne podatke pružiti zadovoljavajuće izlaze potrebno ju je prethodno naučiti, tj. na neki način izmjeniti težine na sinapsama. Učenje neuronske mreže je definirano načinom na koji se odvija promjena slobodnih parametara (11).



Slika 3.3: Interne aktivnosti neurona i sinaptičke veze

Pomoću v_j i v_k označavamo interne aktivnosti, a s x_j i x_k označavamo izlaze neurona j i k. Neuronska mreža se mijenja postepeno, pa ako u koraku n promijenimo iznos sinaptičke težine $w_{kj}(n)$ za $\Delta w_{kj}(n)$ dobijemo novu vrijednost u koraku $n+1$:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \tag{3.1}$$

Algoritme učenja razlikujemo po načinu na koji izračunavaju $\Delta w_{kj}(n)$:

- Učenje korekcijom pogreške
- Hebbovo učenje
- Kompetetivno učenje
- Boltzmanovo učenje
- Thorndikeovo učenje

Paradigme učenja određuju način na koji se odnosi neuronska mreža prema okolini:

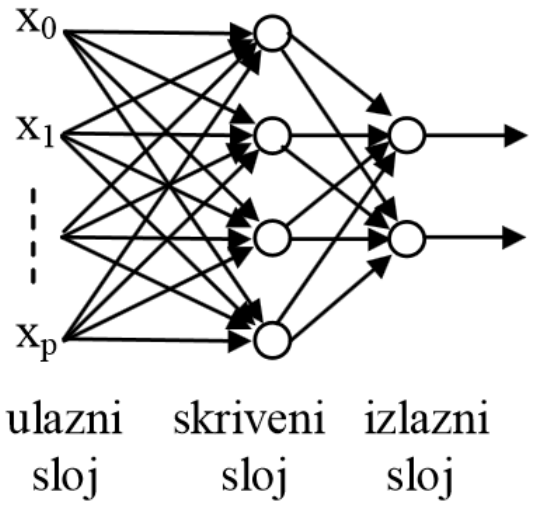
Učenje pod nadzorom Učitelj ima znanje o okolini u obliku parova ulaz-izlaz te se parametri mreže mijenjaju pod utjecajem ulaznih vektora i signala pogreške. Proces je iterativan dok se ne dosegne zadovoljavajuće ponašanje nakon čega učitelj više nije potreban. Učenje može biti u *on-line* ili *off-line* načinu rada. U *off-line* načinu rada sustav prvo uči, i nakon toga se mreža više ne mijenja. U *on-line* načinu rada sustav prilikom rada uči i mijenja konfiguraciju.

Učenje podrškom Uči se ulazno-izlazno preslikavanje na način da se maksimizira indeks, skalarna vrijednost kojim se mjeri kvaliteta učenja, nemamo znanje kolika je greška, već je li korak učenja dobar.

Učenje bez nadzora Nema učitelja koji upravlja procesom učenja već se koriste mjere kvalitete znanja koje mreža mora naučiti.

3.4.3. Detalji algoritma

Cilj neuronske mreže u radu jest naučiti sortirati vrhove na temelju značajki iz vrhova grafa. Pretpostavka koju koristimo prilikom učenja neuronske mreže jest da mreža naučena na jednom grafu može biti korištena na drugim instancama. Za arhitekturu neuronske mreže je odabrana višeslojna mreža koja na ulaze prima značajke vrha, a na izlazu daje prioritet bojanja kao broj na intervalu [0,1]. Neuroni u mreži imaju sigmoidnu aktivacijsku funkciju.



Slika 3.4: Arhitektura neuronske mreže

Korištena paradigma učenja je učenje pod nadzorom u off-line načinu rada. Za algoritam učenja se koristi *Resilient Propagation* (RPROP) algoritam, koji je inačica algoritma s povratnom propagacijom. Algoritam povratne propagacije (10) računa kvadratnu pogrešku $E(n)$ razlike željenog $d_j(n)$ i dobivenog odziva $y_j(n)$ te koristi metodu najbržeg spusta za modifikaciju $\Delta w_{kj}(n)$.

$$e_j(n) = d_j(n) - y_j(n) \tag{3.2}$$

$$E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{3.3}$$

$$\Delta w_{kj}(n) = -\eta \frac{\partial E}{\partial w_{kj}}(n) \tag{3.4}$$

Ovakav način učenja ovisi o parametru η. Male vrijednosti uzrokuju sporo učenje, a prevelike vrijednosti mogu uzrokovati nestabilnost pa se često koristi dodatni izraz za

inerciju koji uspori brzinu promjene korekcije težine.

$$\Delta w_{kj}(n) = -\eta \frac{\partial E}{\partial w_{kj}}(n) + \mu \Delta w_{kj}(n-1). \tag{3.5}$$

RPROP (14) za učenje koristi individualne vrijednosti za svaku težinu, te primjenjuje zbrajanje ili oduzimanje te vrijednosti ovisno o predznaku gradijenta.

$$\Delta w_{ij}(n) = \begin{cases} +\Delta_{ij}(n), & \text{ako } \frac{\partial E}{\partial w_{kj}}(n) > 0. \\ -\Delta_{ij}(n), & \text{ako } \frac{\partial E}{\partial w_{kj}}(n) < 0. \\ 0, & \text{inače.} \end{cases} \tag{3.6}$$

Nakon svakog koraka u kojem mijenjamo težine pomoću formule 3.1, mijenjamo i vrijednost $\Delta_{ij}(n)$ na sljedeći način:

$$\Delta_{ij}(n) = \begin{cases} \eta^+ \cdot \Delta_{ij}(n-1), & \text{ako } \frac{\partial E}{\partial w_{kj}}(n-1) \cdot \frac{\partial E}{\partial w_{kj}}(n) > 0. \\ \eta^- \Delta_{ij}(n-1), & \text{ako } \frac{\partial E}{\partial w_{kj}}(n-1) \cdot \frac{\partial E}{\partial w_{kj}}(n)) < 0. \\ \Delta_{ij}(n-1), & \text{inače.} \end{cases} \tag{3.7}$$

Za η^+ i η^- se tipično uzmu vrijednosti 1.2 i 0.5. RPROP brzo uči, ne zahtjeva optimiranje parametara η i μ te u većini slučajeva daje puno bolje rezultate nego obični algoritam povratne propagacije.

3.5. Lokalna pretraga

Lokalna pretraga je metaheuristička metoda koja služi za rješavanje teških optimizacijskih problema. Rad algoritma se temelji na kretanju kroz prostor rješenja koristeći lokalne promjene dok se ne pronađe optimalno rješenje ili bude zadovoljen kriterij zaustavljanja. Problem koji optimizirano mora biti formuliran tako da možemo vrjednovati pojedina rješenja iz susjedstva trenutnog rješenja te primijeniti lokalnu promjenu.

Tabu pretraga (7) je metaheuristička metoda koja koristi metode lokalne pretrage uz dodatnu modifikaciju tabu liste - liste učinjenih poteza kako bi se otežao povratak u prethodno ispitano rješenje $s \in \Omega$. Postoji više načina na koje je moguće definirati tabu listu (3):

Statična tabu lista Koristi se u originalnoj verziji tabu pretrage, jednom odabrana veličina tabu liste se ne mijenja.

Dinamična tabu lista Veličina ovisi o trenutnom bojanju i potezu koji je izvršen da

se dođe u njega. Ne koriste se informacije o prethodno posjećenim bojanjima, već se koristi formula $\alpha \cdot n_c$ gdje je α konstantan faktor, a n_c broj konfliktnih vrhova.

Reaktivna tabu lista Veličina liste se temelji na potencijalno cijeloj povijesti pretrage prostora rješenja. Veličina tabu liste se mijenja prilikom posjete već posjećenog rješenja.

Često se bolji rezultati postižu ako se izabere nasumična vrijednost izračunu veličine tabu liste t iz intervala $[a(t), b(t)]$, tj. $t_r := UNIFORM(a(t), b(t))$. Za TABUCOL algoritam, dobri rezultati su dobiveni za vrijednosti $a(t) = t$ i $b(t) = t + 10$ za $t = \alpha \cdot n_c$(3).

3.5.1. TABUCOL

Algoritam TABUCOL (7) se često koristi u hibridnim algoritmima za bojanje grafova koji postižu relativno dobre rezultate (9).

Algoritam 3.3: Pseudokôd TABUCOL algoritma

```
1  INPUT
2     G = (V,E)
3     k = number 0f colors
4     |T| = size of tabu list
5     rep = numbr of neighbours in sample
6     nbmax = maximum number of iterations
7  INITIALIZATION
8    Generate random solution s
9    nbiter := 0
10   choose arbitrary tabu list T
11 WHILE f(s) > 0 and nbiter<nbmax
12   generate rep neighbours
13     IF f(s_new) < f(s) stop generating
14   s_new := best neighbour generated
15   update tabu list T
16   s := s_new
17   nbiter:=nbiter+1
18 END WHILE
```

Funkcija $f(s)$ označava broj bridova u grafu koji imaju oba vrha jednake boje. Susjedi s_i generiraju se tako da iz skupa vrhova incidentnih konfliktnim bridovima odaberemo nasumično jedan vrh. Za $x \in V_i$ gdje je i boja vrha x nasumično odaberemo boju j koja se ne nalazi u tabu listi. Ako susjed ima bolju $f(s)$ prekidamo s generiranjem. Tabu lista se dobije na sljedeći način: svakom učinjenom promjenom

bojanja iz V_i u V_j u tabu listu stavimo par (x, i) tj. povratak vrha x u boju i je tabu (7).

3.5.2. ILS

Algoritam *Iterative Local Search* za probleme bojanja grafa (4)(5) se temelji na iterativnom mijenjanju lokalno optimalnih rješenja koja dobivamo lokalnom pretragom. Perturbacija mora biti dovoljno jaka da pomakne algoritam iz lokalnog optimuma, ali ne previše agresivna kako ne bi izgubili kvalitetu rješenja do kojeg smo došli.

Algoritam 3.4: Pseudokôd ILS algoritma

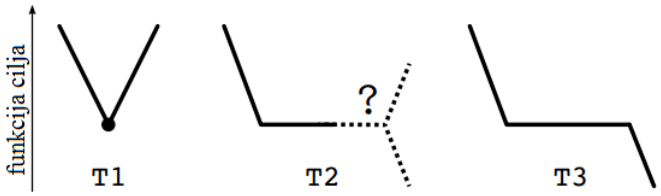
```
1  INPUT
2    G = (V,E)
3  INITIALIZATION
4    Generate initial solution s
5    s := LocalSearch(s)
6  REPEAT
7    s1 := Perturbation(s, history)
8    s2 := LocalSearch(s1)
9    s = AcceptanceCriterion(s,s2, history)
10 UNTIL termination condition
```

3.5.3. Neutralnost

Neutralnim susjedom smatramo susjedno rješenje s' za neki s ako imaju jednake vrijednosti funkcije cilja (13). Skup neutralnih susjeda nekom rješenju $s \in \Omega$ je tada skup svih susjeda $s' \in S'$ za koje vrijedi $f(s') = f(s)$.
Stupanj neutralnosti je broj neutralnih susjeda nekom bojanju s. Ako ima mnogo rješenja s visokim stupnjem neutralnosti imamo neutralan reljef funkcije cilja koji ima mnogo platoa. Plato je ovdje svako rješenje koji ima jednaku funkciju cilja. Prosjek stupnjeva neutralnosti uzimamo kao mjeru da okarakteriziramo neutralnost nekog grafa.
Važno svojstvo neutralne funkcije cilja za susjedstvo je mogućnost izvođenja neutralne nasumične šetnje, kojom iz trenutnog rješenja s prelazimo u novo neutralno rješenje s'. Ako rješenje s koje pripada platou ima susjeda s' s boljom funkcijom cilja, nazivamo ga portal. U ovisnosti o ovom svojstvu moguća su tri slučaja neutralne nasumične šetnje:

- Plato kojem pripada samo jedno rješenje i nema neutralnih susjeda (Slika 3.5 T1).
- Plato za kojeg nismo naišli na portal (Slika 3.5 T2).

– Plato kojem smo pronašli portal (Slika 3.5 T3).



Slika 3.5: Tipovi platoa

3.5.4. Detalji algoritma

Algoritam predložen u radu je modularna verzija tabu pretrage koja koristi operatore odabira boje, vrha i operator funkcije cilja. Algoritam koristi dinamičnu tabu listu koja pamti svako prošlo stanje u obliku para <vrh, boja> i ne dozvoljava promjenu boje vrhu određeni broj bojanja koji dinamički odredimo $\alpha \cdot n_c + t_r$ gdje je α neki broj, n_c broj konfliktnih vrhova i t_r nasumičan broj. Važno je napomenuti da ovdje, kao u *TABUCOL* algoritmu, ako vrh nudi bolju funkciju cilja tabu se zanemaruje. Broj susjeda koje generiramo prilikom svake iteracije također se dinamički odredi s $\beta \cdot n_c$ gdje je β neka pozitivna vrijednost. Susjede dobivamo primjenom operatora bojanja nad nekim vrhom i od trenutnog bojanja s se razlikuju u bojanju jednog vrha. Generiranje susjeda se prekida ako novi susjed ima bolju funkciju cilja od trenutnog bojanja grafa. Redoslijed stvaranja susjeda je utvrđen operatorom odabira vrha. Ako u skupu generiranih susjeda nema niti jednog susjeda s' za kojeg funkcija cilja $f(s') < f(s)$ kreće se u proceduru neutralne šetnje kako bi potencijalno naišli na portal koji ima susjeda s manjom funkcijom cilja. Ako određeni broj ponavljanja zadan s ulaznim parametrom algoritam ne pronađe bolje bojanje, algoritam se zaustavlja. Parametri koje algoritam koristi su:

1. Lista vrhova nad kojima izvodimo algoritam - nodeList
2. Broj pokušaja neutralne šetnje - neutralityCount
3. α - alpha
4. β - beta
5. Maksimalni broj iteracija bez promjene - maxIter
6. Operator odabira boje - colorSelector

7. Operator odabira vrha - comparator

8. Funkcija cilja - objectiveF

Algoritam 3.5: Pseudokôd tabu pretrage

```
1  INPUT
2    nodeList, neutralityCount, alpha, beta, maxIterations,
        colorSelector, comparator, objectiveF
3  INITIALIZATION
4    s := currentSolution(nodeList)
5    minObjectiveF = objectiveF(s)
6  REPEAT
7    iteration := iteration + 1
8    n_c := numberOfConflictNodes(s)
9    comparator.sort(nodeList)
10   FOR node(i) in nodeList
11     s_new :=colorSelector(node(i))
12     IF node(i).tabuCounter < 0
13       neighbours.add(s_new)
14     ELSE IF objectiveF(s_new) < minObjectiveF
15       neighbours.add(s_new)
16       minObjectiveF = objectiveF(s_new)
17       iterations := 0
18       BREAK
19     END IF
20   END FOR
21   s_best := selectBest(neighbours)
22   decreaseTabuCounter(nodeList)
23   s_best.node.tabuCounter = alpha * n_c + random(1,10)
24   IF objectiveF(s_best) < objectiveF(s)
25     s: = s_best
26   ELSE
27     s: =neutralityWalk(nodeList, neutralityCount, objectiveF)
28   END IF
29 UNTIL iteration > maxIteration
```

Procedura neutralne šetnje se pokušava kretati po platou funkcije cilja tako da uzima nasumično vrhove iz skupa konfliktnih i vrhova koji nisu u inicijalnoj boji i nasumično im dodjeljuje neutralnu boju s obzirom na funkciju cilja. Neutralnom bojom se također smatra i trenutno dodijeljena boja. Odabirom vrhova koji nisu u inicijalnoj boji želimo potencijalno kontrolirati postotak promijenjenih vrhova.

Algoritam 3.6: Pseudokôd neutralne šetnje

```
1  INPUT
2    nodeList, neutralityCount, objectiveF
3  INITIALIZATION
```

```
4    s := currentSolution(nodeList)
5    minObjectiveF = objectiveF(s)
6    conflictNodes = getConflictNodes(nodeList)
7    otherNodes = getNonConflictNodes(nodeList)
8    FOR i = 1 TO neutralityCount
9      node := selectRandom(conflictNodes)
10     colorList = objectiveF.getNeutralList(node)
11     node.setColor(random(colorList))
12   END FOR
13   FOR i = 1 TO neutralityCount
14     node := selectRandom(otherNodes)
15     colorList = objectiveF.getNeutralList(node)
16     node.setColor(random(colorList))
17   END FOR
```

4. Programsko rješenje

Prilikom izrade i oblikovanja programskog dijela diplomskog rada u obzir su uzeti različiti zahtjevi i ograničenja te su korišteni principi objektnog oblikovanja kako bi se ostvarila modularnost i jednostavno testiranje zasebnih dijelova. Za implementaciju je korišten Java programski jezik, te otvorene biblioteke *Encog* za neuronske mreže .

4.1. Pomoćne strukture

Prilikom izvođenja algoritama, vidljivo je da u većini situacija koristimo veliki broj informacija o nekom vrhu kako bi donijeli odluku o bojanju ili sortiranju. Jedna mogućnost je prilikom svakog poziva metode koja implementira neko svojstvo nanovo računati vrijednosti. Ovakav način se isplati u svim situacijama gdje bi nam inicijalizacija pomoćnih struktura predstavljala većinu izvođenja. S obzirom na prirodu algoritama, poglavito lokalne pretrage koja za svaku odluku ispituje mnogo mogućih bojanja, i izvođenje u više iteracija, te ponukano sličnom implementacijama (5), odlučeno je korištenje pomoćnih struktura koje se osvježavaju prilikom poziva *setColor* nad bilo kojim objektom *Node*. Cijena inicijalizacije pojedine strukture je u najgorim slučajevima $O(n^2 \cdot k)$ za inicijalizaciju i $O(n \cdot k)$ za osvježavanje pomoćne strukture pri svakom bojanju gdje je n broj vrhova a k broj boja. Ova cijena je opravdana kroz uzastopno izvođenje pohlepnih algoritama i lokalne pretrage. Pomoćne strukture pamte i računaju vrijednosti nekog svojstva za sve boje domene vrha kojem pripadaju. U osnovi implementiraju sljedeće metode:

put(color, value) Metoda koju pozivamo kad želimo dodati neku vrijednost za određenu boju.

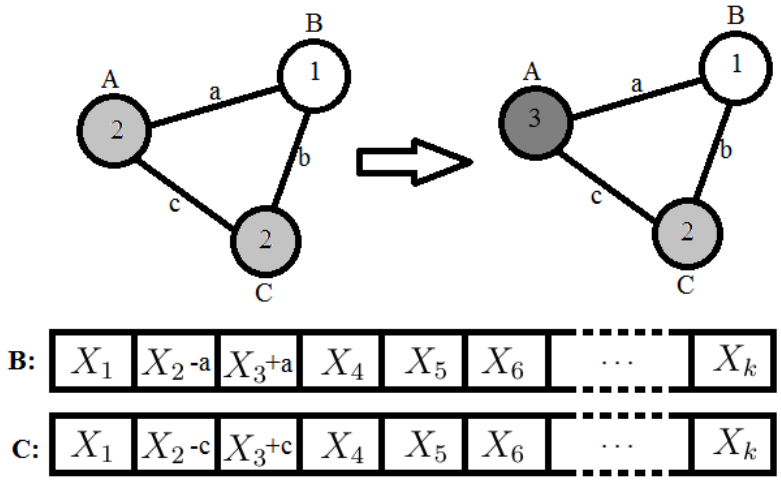
remove(color, value) Metoda koju pozivamo kad želimo ukloniti neku vrijednost za određenu boju.

get(color) Metoda za čitanje vrijednosti boje *color*.

entrySet() Metoda koja vraća uređene parove <boja, vrijednost>. Iako postoje me-

tode kojima možemo dohvatiti individualne vrijednosti, ova opcija je ostavljena ako neki algoritam zahtjeva potpuni skup vrijednosti a ne želimo obavljati mnogo poziva.

Svaki vrh sadrži svoje vlastite pomoćne strukture koje stvara prilikom poziva metode *initialize*. Računanje većine metoda koje objekt *Node* pruža na korištenje ostalim objektima je delegirano internim pomoćnim strukturama. U konstruktor pomoćne strukture primaju boje koje je potrebno pratiti - ovo je potrebno jer ako se traži vrijednost za boju van domene, vraća se vrijednost nula. Prilikom promjene boje, svi susjedi vrhu koji je promijenio boju ažuriraju vrijednosti za pojedine boje vidljivo na slici 4.1.



Slika 4.1: Osvježavanje pomoćnih struktura

4.1.1. FitnessMap

Pomoćna struktura za izračun sume težina je implementirana razredom *FitnessMap* koja prati vrijednost sume težina konfliktnih bridova za boje domene vrha kojem pripada. Implementira sljedeće metode:

get(Integer color) Dohvat vrijednosti zapisane za boju *color*.

put(Integer color, double value) Za boju *color* dodajemo vrijednost *value*.

remove(Integer color, double value) Vrijednost boje *color* smanjujemo za *value*.

getMinColor Prilikom svakog poziva *put* i *remove* pamti se boja za koju je postignuta minimalna vrijednost te metodom *getMinColor* možemo dobiti tu boju.

maxDifference(Integer color) Pozivom *maxDifference* vraća se razlika vrijednosti

za boju *color* i minimalne vrijednosti koju je moguće doseći u nekoj od dostupnih boja.

4.1.2. CollisionMap

U pomoćnoj strukturi *CollisionMap* se pamti broj konfliktnih bridova ako bi se vrh koji sadrži ovu pomoćnu strukturu imao ako bi se nalazio u određenoj boji iz svoje domene. Specifičnost ove strukture naspram *FitnessMap* je da znamo da prilikom dodavanja i uklanjanja vrijednosti radimo s jediničnim vrijednostima zbog načina i trenutka kad osvježavamo vrijednosti tj. kad neki vrh promjeni boju, njegovi susjedi će imati jednu koliziju manje u njegovoj prijašnjoj boji, i jednu koliziju više u njegovoj novoj boji. Implementira sljedeće metode:

get(Integer color) Dohvat vrijednosti za određenu boju.

put(Integer color) Dodavanje vrijednosti za određenu boju.

remove(Integer color) Smanjivanje vrijednosti za određenu boju.

getMinColor Dohvat boje s najmanjim brojem kolizija.

maxDifference(Integer color) Dohvat razlike broja kolizija za boju *color* i minimalnog dostiživog broja kolizija.

size() Metoda vraća broj boja gdje je upisana vrijednost veća od nula i u kontekstu vrha predstavlja saturaciju tj. broj različitih boja koje se pojavljuju u susjeda.

4.1.3. DomainMap

Ova struktura se nakon inicijalizacije ne osvježava, jer se radi vrijednostima koje ostaju iste prilikom izvođenja algoritama. U prilog ovoj činjenici ide i izostavljanje *remove* metode, koja u ovom kontekstu nema smisla osim ako postoji scenario u kojem bi uklanjali vrhove. Struktura za svaki vrh pamti koliko puta se svaka boja iz domene vrha pojavljuje u susjednim vrhovima. Motivacija za ovu strukturu je kako bi prilikom odabira boje ostavili što više slobodnih boja susjedima. Učinjena je optimizacija kako bi proces inicijalizacije bio što jeftinija operacija zato što upis atomarnih vrijednosti za svaku boju svake domene svakog susjeda predstavlja popriličan trošak. Inicijalizacija pomoćnih struktura prolazi kroz sve bridove i vrši upis što daje složenost O(E) gdje je E broj bridova. Upis svake zasebne boje nameće trošak od d - broja boja u domeni, što bi povećalo složenost na O(E*d). Iz ovog razloga, DomainMap ima znanje o dostupnim domenama, i prilikom upisa vrijednosti u DomainMap definiramo koliko se određenih

domena nalazi u susjedstvu vrha, te se interno vrijednosti povećaju za svaku zasebnu boju, ako je boja prisutna u domeni vrha s kojim radimo. Sljedeće metode koristimo pri radu s *DomainMap*:

get(Integer color) Dohvaćamo vrijednost upisanu za određenu boju. Ako je *DomainMap* ispravno inicijaliziran, ova činjenica nam govori koliko susjeda ima mogućnost odabrati upravo ovu boju.

put(Integer domain, Integer count) Metoda za dodavanje vrijednosti u *DomainMap*. Vidimo da su ulazni parametri domena, i broj istih - što je upravo zbog navedenih optimizacija koje su potrebne zbog veličine ulaznih grafova.

getMinColor Dohvat boje koja se najmanje pojavljuje u susjeda.

maxDifference(Integer color) Metoda vraća razliku pojavljivanja boje *color* u domenama susjeda i boje koja se minimalno pojavljuje u domenama susjeda.

4.2. Struktura grafa

4.2.1. Graph

Razred koji implementira graf. Jednom učitan i instanciran objekt razreda *Graph* služi nam kao polazna točka za dohvat vrhova koji mu pripadaju. Za pohranu vrhova moguće je implementirati jednu od reprezentacija:

Lista susjedstva - svi vrhovi su susjedni nekom vrhu. Može biti implementirano kao tablica ili kao dvostruko povezana lista.

Matrični zapis - koristi se matrica [vrhovi x vrhovi] ili [vrhovi x bridovi].

Kako je količina memorije potrebna za matrični zapis prevelika, te nam konkretno ne treba O(1) pristup određenom bridu, odabrana je lista susjedstva tako da svaki vrh ima listu bridova koji sadrže pokazivače na susjedni vrh. Metode koje su dostupne prilikom rada s *Graph* objektom:

getFitness Računa se funkcija pogreške za trenutno stanje grafa.

getFitnessByGroups Vraća parove <grupa, greška> za trenutno stanje grafa i za zasebne grupe unutar grafa.

getChanged Vraća postotak vrhova koji nemaju boju koju su imali prilikom učitavanja grafa.

getChangedByGroups Vraća parove <grupa, postotak promjene> za trenutno stanje grafa i za zasebne grupe unutar grafa.

checkConstraints Izvodi se provjera ograničenja i ako ima nekih nepravilnosti ispisuju se na konzolu.

getNode(int id) Dohvaća se referenca vrha koji ima id kao ulazni argument.

getGroupedColoring Vraća listu objekata *Coloring* koji služe za spremanje trenutnog bojanja grafa zasebno po grupama.

getComponents Vraća listu vrhova razdvojenih u komponente, tj. disjunktne skupove vrhova koji u uniji čine kompletno učitani graf.

groupNodes Vraća više listi vrhova koji su razdvojeni po grupama. Grupe unutar okvira problema nemaju međusobnih interakcija i problem je moguće podijeliti na više neovisnih dijelova.rekcijom pogreške

4.2.2. Node

Razred koji implementira vrh u grafu. Svaki instancirani objekt razreda *Node* sadrži listu objekata *Bridge* pomoću kojih možemo dohvatiti susjedne mu vrhove. Također imamo veliki broj metoda koji enkapsuliraju funkcionalnost pomoćnih struktura:

initialise Metoda koja inicijalizira pomoćne strukture te sortira bridove po težini kako određeni operatori (ABW, ABWMD) nebi ponavljali akciju sortiranja.

getColor Metoda koja vraća trenutnu boju vrha.

setColor Metoda koja postavlja boju vrha i ujedno osvježava pomoćne strukture.

getSaturation Služi za dohvat zasićenosti vrha.

getFitness Metoda koja vraća sumu težina konfliktnih bridova za vrh s trenutnom bojom.

getFitness(int color) Metoda koja vraća sumu težina konfliktnih bridova za vrh ako bi bio u boji zadanoj kao ulazni argument *color*.

getMinFitnessColor Metoda koja vraća boju za koju je suma težina konfliktnih bridova minimalna. Boja je iz domene dostupnih boja za zadani vrh.

getMaxFitnessDifference Metoda koja vraća najveću razliku fitnesa koju vrh može postići gledajući trenutni fitnes.

getFitnessMap Metoda koja vraća uređene parove <boja, fitnes> za sve boje u koje je vrhu dozvoljeno ići.

getCollisions Metoda koja vraća broj konfliktnih bridova za vrh u trenutnoj boji.

getCollisions(int color) Metoda koja vraća broj konfliktnih bridova za vrh u boji zadanoj kao ulazni parametar *color*.

getMinCollisionColor Metoda koja vraća boju u kojoj je minimalan broj konfliktnih bridova. Boja je iz domene dostupnih boja za zadani vrh.

getMaxCollisionDifference Metoda koja vraća najveću razliku konfliktnih bridova za trenutnu boju i neku dostupnu boju iz domene.

getCollisionMap Metoda koja vraća uređene parove boja, $< brojkonfliktnihbridova >$ za sve dostupne boje iz domene vrha.

getFreeColors Metoda vraća broj boja iz domene koje se ne pojavljuju u trenutnom bojanju susjeda.

setNeuralNetwork Metoda kojom postavljamo referencu vrha na neku neuronsku mrežu generiranu bibliotekom *Encog*.

getNetworkOutput(boolean calculate) Metoda vraća izlaz neuronske mreže koja je postavljena metodom *setNeuralNetwork*. Ulazni argument *calculate* utječe hoće li se jednom izračunata vrijednost ponovno računati. Ovakav način računanja je zbog pitanja performansi koji predstavlja veći trošak.

getDomainColorCount(int color) Metoda vraća koliko puta se određena boja pojavljuje u domenama susjeda.

getFeatures Metoda vraća polje vrijednosti koje koristimo za učenje neuronske mreže.

4.2.3. Bridge

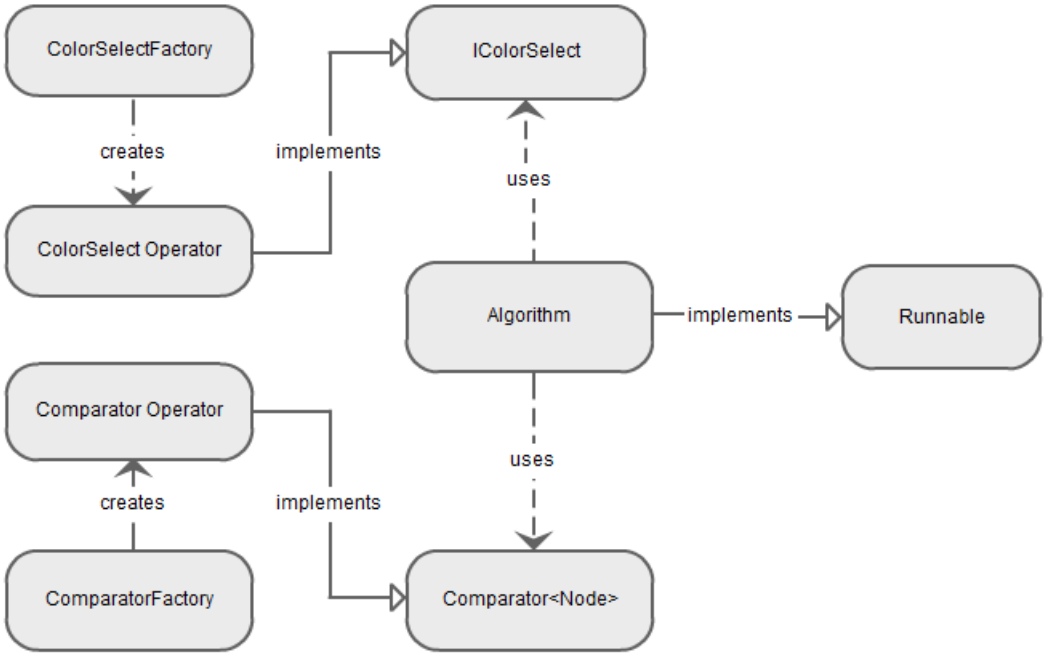
Razred koji implementira brid u grafu. U konstruktor razreda *Bridge* šaljemo argumente težine koju će brid imati, i vrh na koji pokaziva. Prvi vrh je implicitno zadan s pripadnošću određenoj listi objekta *Node*. Kako svaki objekt *Node* prilikom inicijalizacije sortirane pripadne mu vrhove, ovaj razred implementira sučelje *Comparable<Bridge>* kako bi mogli koristiti ugrađene metode kolekcija u Javi.

4.2.4. Domain

Razred *Domain* implementira specifičnost zadatka s kojim radimo a to je ograničeni set boja koje pripadaju nekoj domeni. Svaki objekt *Node* ima referencu na određeni objekt *Domain* ovisno ako je zadana pripadnost nekoj domeni za vrh. Prilikom postavljanja nove boje, *Node* provjerava nalazi li se u listi boja objekta *Domain* boja koju želimo dodijeliti vrhu.

4.3. Strukture algoritama

Kako bi izdvojili zajedničke dijelove skupine algoritama i omogućili jednostavno naknadno dodavanje novih operatora koji bi isti koristili, korišteni su principi u literaturi poznati kao *Dependancy Injection* i objektni oblikovni obrazac *Strategy*. Algoritmi prilikom stvaranja u konstruktoru primaju objekte o kojima njihovo izvođenje ovisi, a za njihovo stvaranje su dostupne 'tvornice' tj. razredi koji implementiraju oblikovni obrazac *Factory* (slika 4.2). Dodatno je implementirana mogućnost izvođenja u dretvama po grupama, što proizlazi iz definicije zadatka gdje zasebne grupe možemo promatrati kao zasebne grafove čiji su vrhovi međusobno disjunktni a u uniji daju kompletan ulazni graf.



Slika 4.2: Dijagram klasa

4.3.1. Strukture za višedretvenost

Algoritmi implementiraju sučelje *Runnable* (slika 4.2) koje je dio *Java* biblioteka. Svi razredi koji implementiraju ovo sučelje obvezuju se implementirati metodu *run()* koju ne zovemo direktno nego proslijedimo taj objekt razredu Thread i nad njim pozivamo *start()*. Poziv metode *thread.join()* blokira izvođenje glavne dretve sve dok *thread* ne završi.

4.3.2. Strukture odabira boje

Razredi implementiraju zajedničko sučelje *IColorSelect* koje definira jednu metodu *select(Node node)* čime je ostvaren *Strategy* oblikovni obrazac, koji nam omogućava da na svim mjestima gdje radimo s *IColorSelect* sučeljem, možemo ponuditi neki od razreda koji poštuju njegov ugovor.

IColorSelect Sučelje koje definira jednu javnu metodu *select(Node node)* koja treba vratiti boju tj. Integer.

ColorSelectFactory Razred koji stvara objekte koji implementiraju *IColorSelect*. Statične metoda *make* zapravo vraća*IColorSelect* pozivatelju.

4.3.3. Strukture odabira vrha

Kako koristimo kolekciju *List* za spremanje vrhova, iskorišten je razred *Collections* koji implementira statične metode za rad s kolekcijama. Metoda koje je korištena je *sort(List l, Comparator c)*. Dio koji je potreban je razred koji implementira sučelje *Comparator*, u ovom slučaju *Comparator<Node>* pošto je lista parametrizirana za rad s *Node* objektima. *Comparator* definira metodu *compareTo* koja prima dva objekta za usporedbu, u ovom slučaju dva Node objekta.

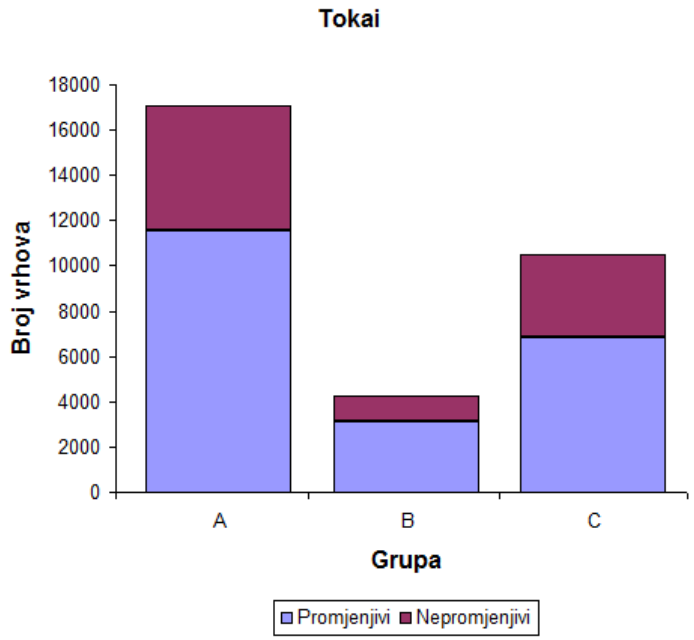
5. Rezultati

5.1. Grafovi za ispitivanje

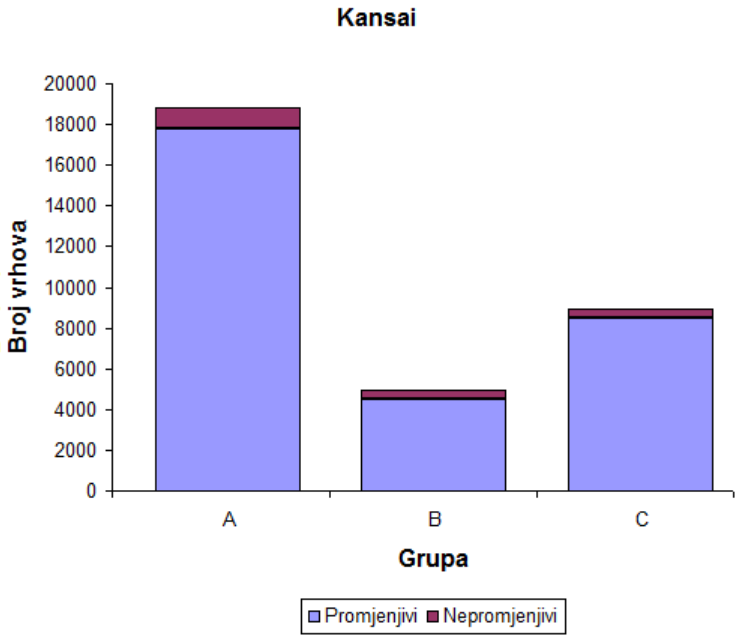
Grafovi za ispitivanje podijeljeni su u dvije skupine, grafovi problema dodjele frekvencija te grafovi klasičnog problema bojanja grafa.

5.1.1. Problem dodjele frekvencija

Za ispitivanje problema dodjele frekvencija opisanog u poglavlju 2.3.1 dostupna su dva grafa: Tokai i Kansai graf. Oba grafa sadrže vrhove koji pripadaju jednoj od 3 grupe: A, B ili C. Ako se prilikom testiranja obrađuje samo podskup vrhova, dodana je oznaka grupe $-X$ na kraj naziva grafa gdje X označava grupu. Grupe nisu podjednako velike, te sadrže određeni broj vrhova kojima nije dozvoljena promjena boje. Ovi grafovi se koriste kako bi se odredili parametri i operatori koji nude najbolje rezultate.

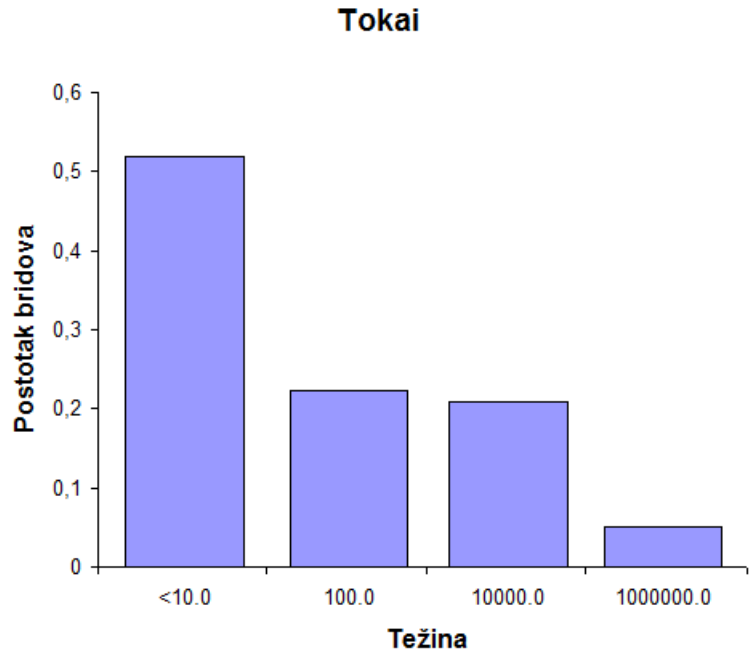


Slika 5.1: Podjela vrhova po grupama za Tokai

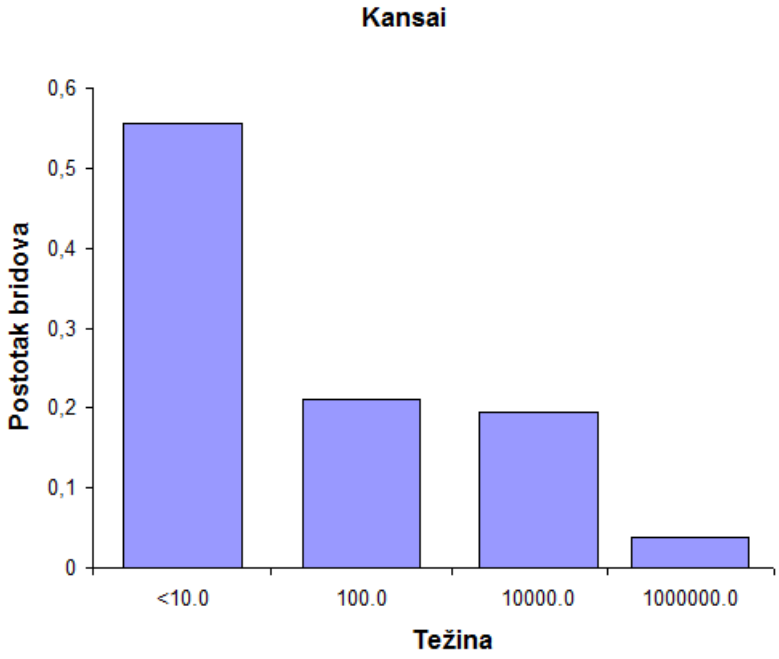


Slika 5.2: Podjela vrhova po grupama za Kansai

Vidimo da iako su sličnih struktura (slika 5.1 i 5.2), Tokai graf ima veći postotak vrhova kojima ne smijemo mijenjati boju. Također, prosječan broj bridova po vrhu je 667.48 za Kansai te 551.14 za Tokai. U slikama 5.3 i 5.4 vidimo da zadanim velikim težinama bridova se nameću čvrsta ograničenja, te da je potencijalan cilj dobrih rješenja imati samo konfliktne bridove koji imaju težine manje od 10.0.



Slika 5.3: Raspodjela bridova za Tokai



Slika 5.4: Raspodjela bridova za Kansai

5.1.2. Klasični problem bojanja grafa

Za ispitivanje primjene na klasičan problem bojanja koristimo bazu znanstvene ustanove *The Center for Discrete Mathematics and Theoretical Computer Science* (DI-

MACS) koja od 1990. godine sponzorira implementacijska natjecanja u kojima se obrađuju NP-teški problemi. Za grafove iz baze u većini slučajeva znamo kromatski broj te najbolje legalno k-bojanje. Korišteni grafovi s obzirom na karakteristike imaju sljedeću podjelu:

dsjcX.Y Slučani grafovi s X vrhova i vjerojatnošću $p = 0.Y$ za spajanjem para vrhova.

flatX_K Grafovi konstruirani s poznatim kromatskim brojem. X je broj vrhova, K je kromatski broj.

le450_K Grafovi s 450 vrhova i kromatskim brojem K.

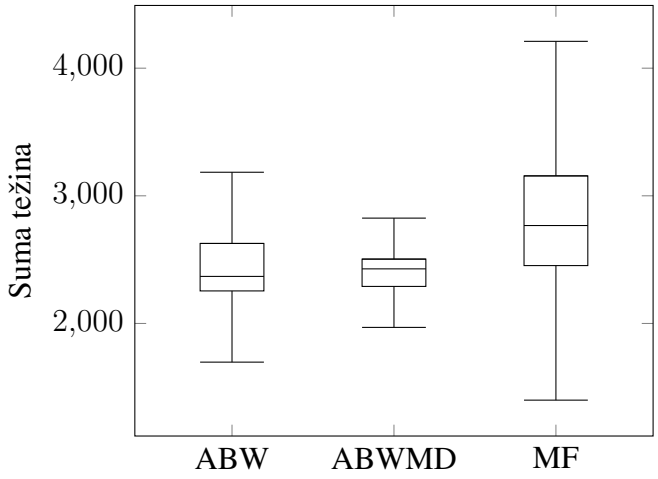
dsjrX.Y Slučajni grafovi generirani uniformnim odabirom točaka u kvadratu i povezivanjem parova vrhova unutar neke udaljenosti.

Kako bi rad s klasičnim problemom bio moguć, prilikom učitavanja zadaje se broj boja koje su dostupne. Svi vrhovi imaju identičnu domenu kojoj pripada niz od $1, 2, ..., k$ boja gdje je k zadani broj boja. Inicijalna boja svim vrhovima je 1 te svi pripadaju grupi A. Težina svih bridova je postavljena na 1.0.

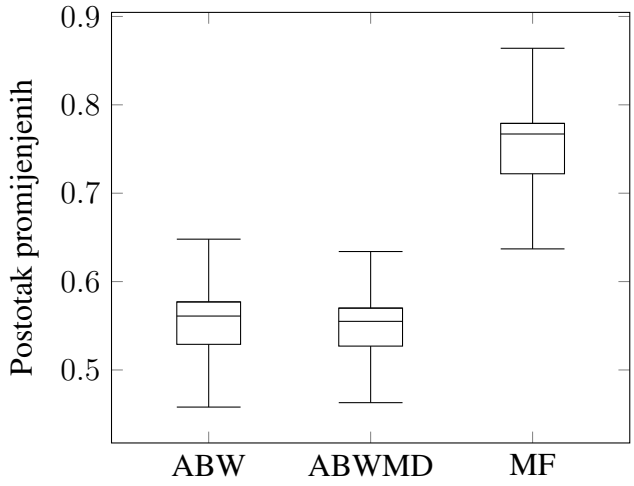
5.2. Pohlepni algoritam

U okviru ispitivanja pohlepnih algoritama napravljena je usporedba operatora odabira boje i operatora odabira vrha te kakve karakteristike ima inicijalno bojanje ovisno o operatoru na grafu *Tokai-A*. Kvaliteta bojanja ovisi o sumi težina i postotku promijenjenih vrhova. Operatori odabira vrha *MC*, *MD*, *RND* i *SA* izdvojeni su iz pregleda rezultata jer se nisu pokazali kompetitivni za inicijalno bojanje.

5.2.1. Operator odabira boje



Slika 5.5: Suma težina - Tokai-A

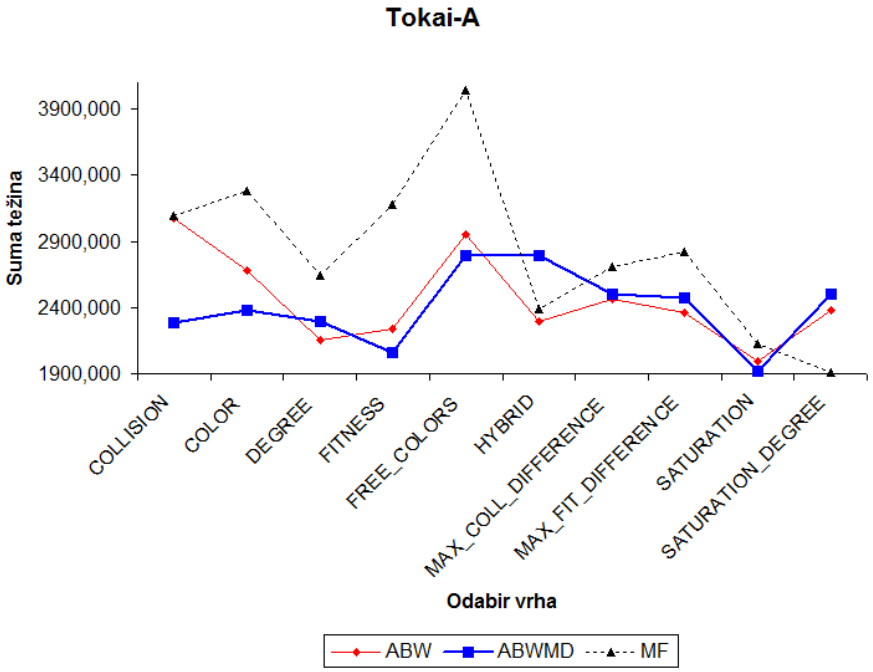


Slika 5.6: Postotak promjene - Tokai-A

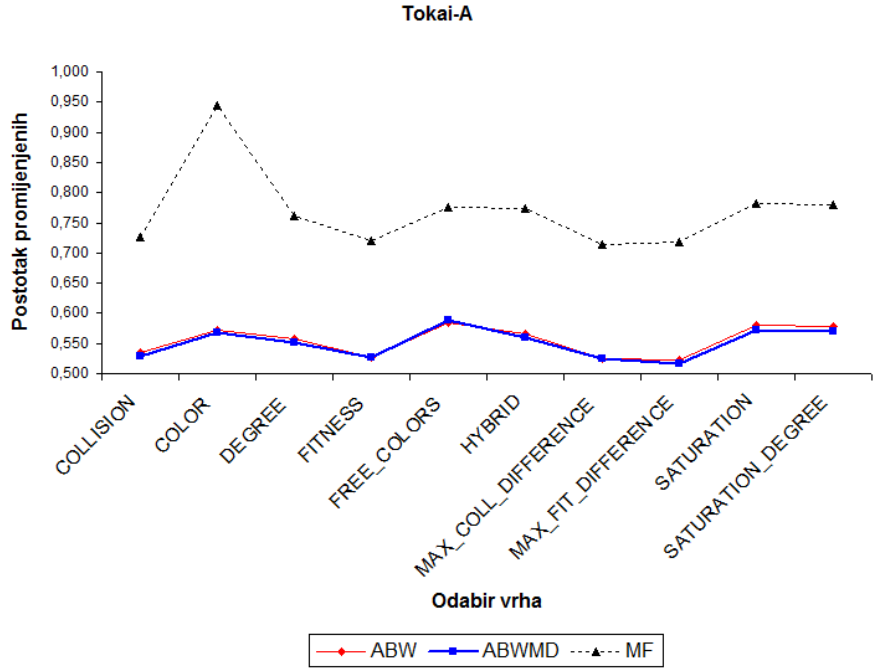
Na slikama 5.5 i 5.6 dan je pregled vrijednosti funkcija cilja nakon 5 iteracija pohlepnog algoritma za svaki operator odabira boje uz različite operatore odabira vrha.

5.2.2. Operator odabira vrha

Prilikom ispitivanja operatora odabira vrha koristimo najbolje operatore odabira boje iz prošlih ispitivanja: *ABW*,*ABWMD* i *MF*. Uspoređujemo rezultate dobivene nakon 5 iteracija pohlepnog algoritma. Na slici 5.7 i 5.8 su rezultati izvođenja za različite operatore.



Slika 5.7: Suma težina za različite operatore odabira vrha



Slika 5.8: Postotak promijenjenih za različite operatore odabira vrha

5.3. Genetski algoritam

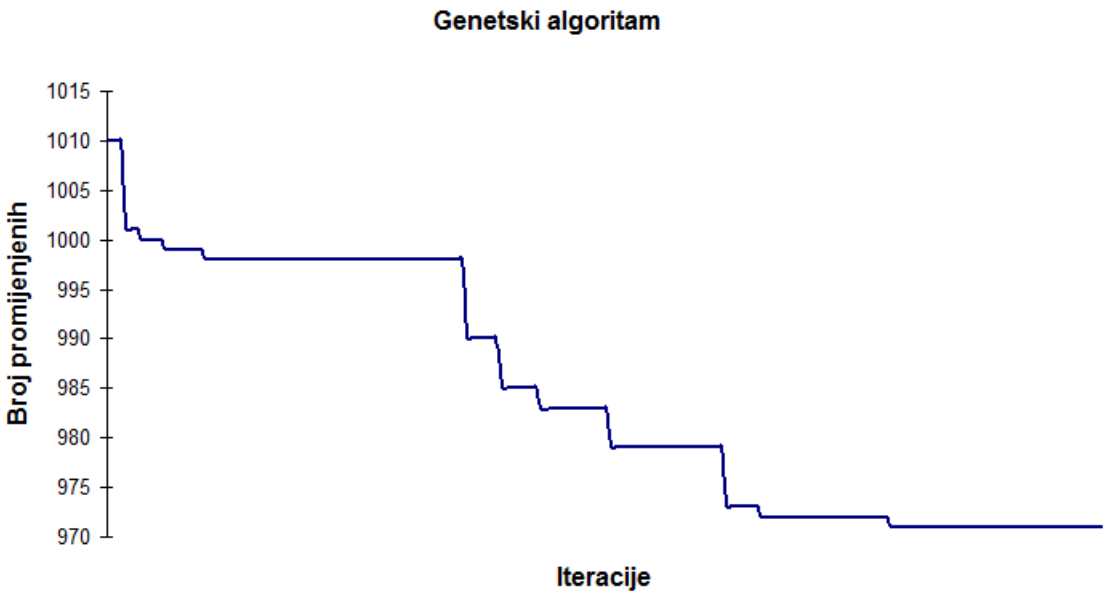
Ispitivanjem pohlepnih algoritama s raznim kombinacijama operatora pokazalo se kako operator odabira vrha dosta utječe na kvalitetu bojanja. Ako ne mijenjamo operator odabira boje, genetskim algoritmom bi za klasičnu inačicu problema bojanja grafa koristeći konstruktivne metode poput *first-fit* operatora odabira boje trebali pronaći optimalno rješenje. Primjenom genetskog algoritma pronađen je redoslijed *Tokai-B* grafa u kojem se minimizira postotak promijenjenih (slika 5.9) uz primjenu sljedećih parametara:

–startSeed Raspored vrhova za izvođenje pohlepnog algoritma uz operatore *ABW* i *COLLISION*.

–populationSize **200**

–numberOfGenerations **200**

–funkcija cilja **Broj promijenjenih vrhova**



Slika 5.9: Kretanje funkcije cilja za genetski algoritam

5.4. Neuronska mreža

Za neuronsku mrežu je korišten *Encog Machine Learning Framework* koji je dostupan kao biblioteka za Javu. Za učenje se koristi *Tokai-B* graf, a testiramo naučenu mrežu na *Tokai-A* i *Tokai-B* grafu. Značajke koje neuronska mreža gleda kod vrha su:

–Stupanj vrha

–Broj konfliktnih bridova

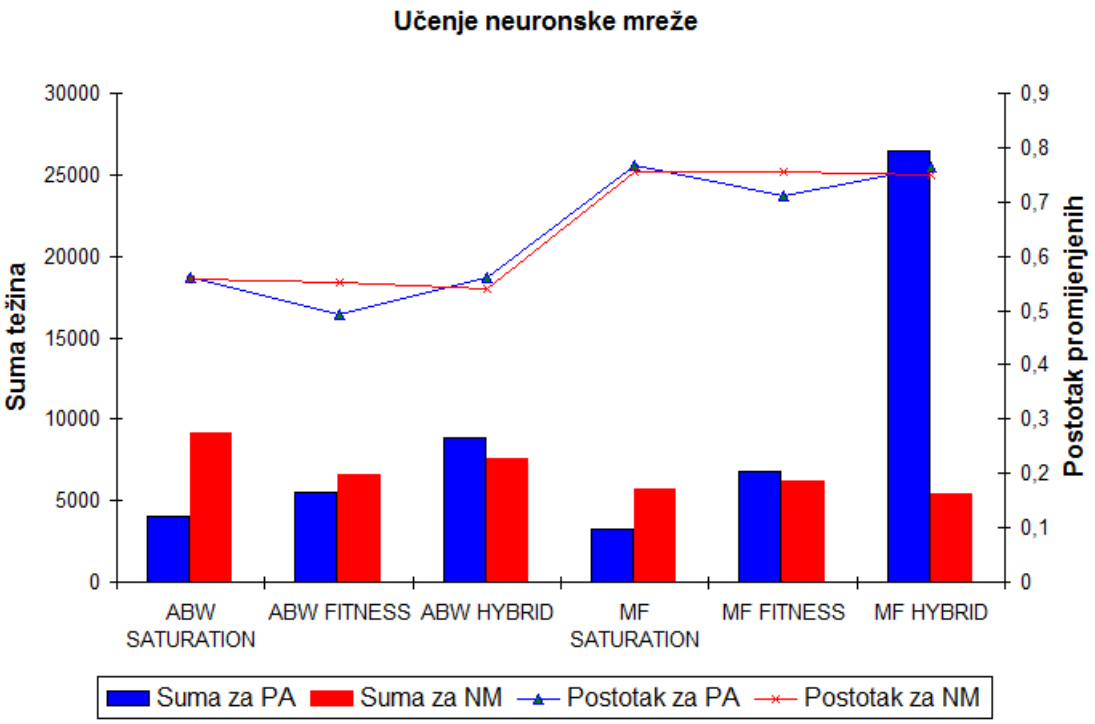
–Broj boja u susjedstvu

–Suma težina konfliktnih bridova

–Broj slobodnih boja

–Maksimalna razlika konfliktnih bridova u trenutnoj boji i nekoj boji domene

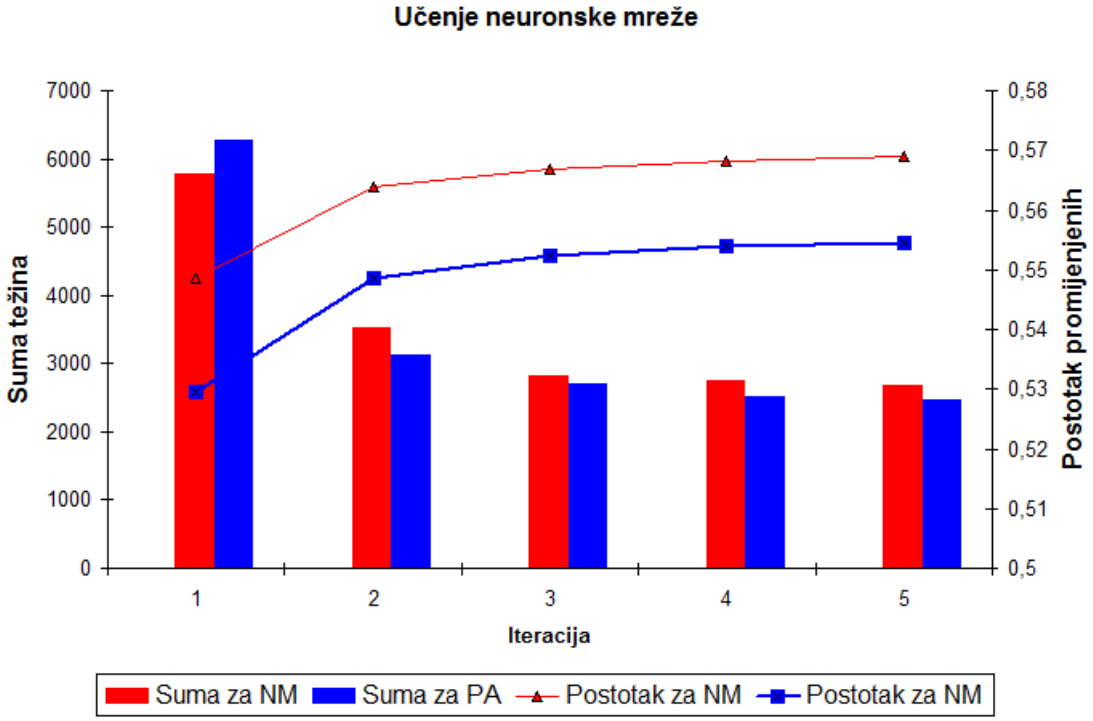
–Maksimalna razlika sume težina bridova u trenutnoj boji i nekoj boji domene



Slika 5.10: Neuronska mreža za operatore

Na slici 5.10 vidimo kakve rezultate postiže pohlepni algoritam (oznaka PA) i neuronska mreža (oznaka NM) koja uči sortiranje od rezultata pohlepnog. Neuronska mreža uči novi operator sortiranja koji se ne treba osvježavati dinamički prilikom svakog odabira vrha već na temelju konačnog poretka i inicijalnog stanja grafa pokušava pronaći poziciju vrha na intervalu [0,1] što predstavlja relativnu poziciju vrha u redoslijedu

bojanja. Sortiranje se obavlja gledajući izlaz neuronske mreže za značajke pojedinog vrha uzimajući prije manje vrijednosti.



Slika 5.11: Naučeni operator odabira vrha

Naučen je operator odabira vrha koji koristi ulazni niz generiran genetskim algoritmom . Genetski algoritam je prilikom generiranja rješenja koristio operator *ABW* te se isti koristi prilikom ispitivanja. U rezultatima na slici 5.11 su uspoređeni prosječni rezultati neuronske mreže (oznaka NM) naspram prosječnih rezultata pohlepnog algoritma (oznaka PA) za operator odabira boje *ABW*.

5.5. Lokalna pretraga

Ispitivanje se provodi na *Tokai-A* grafu. Graf se inicijalno optimizira s 5 prolaza pohlepnog algoritma kombinacije *ABW* i *SDO* operatora nakon čega suma težina grafa *Tokai-A* iznosi 1920.745. Prilikom ispitivanja koristi se operator odabira boje *MF*, operator odabira vrha *COLLISION* i suma težina za funkciju cilja. Prilikom ispitivanja ostali parametri su postavljeni na sljedeće vrijednosti:

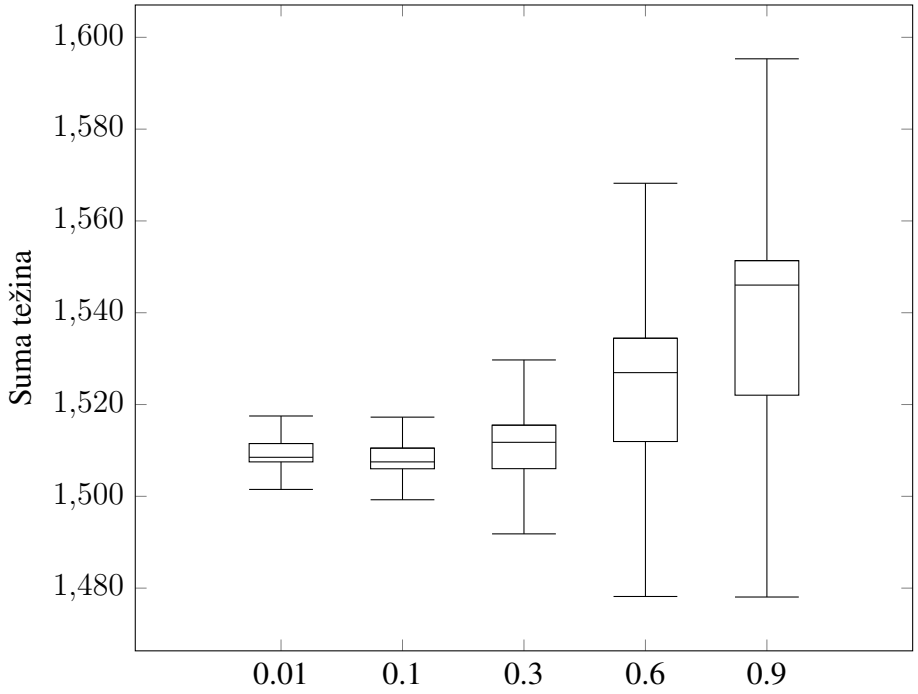
–α **0.01**

–β **1.0**

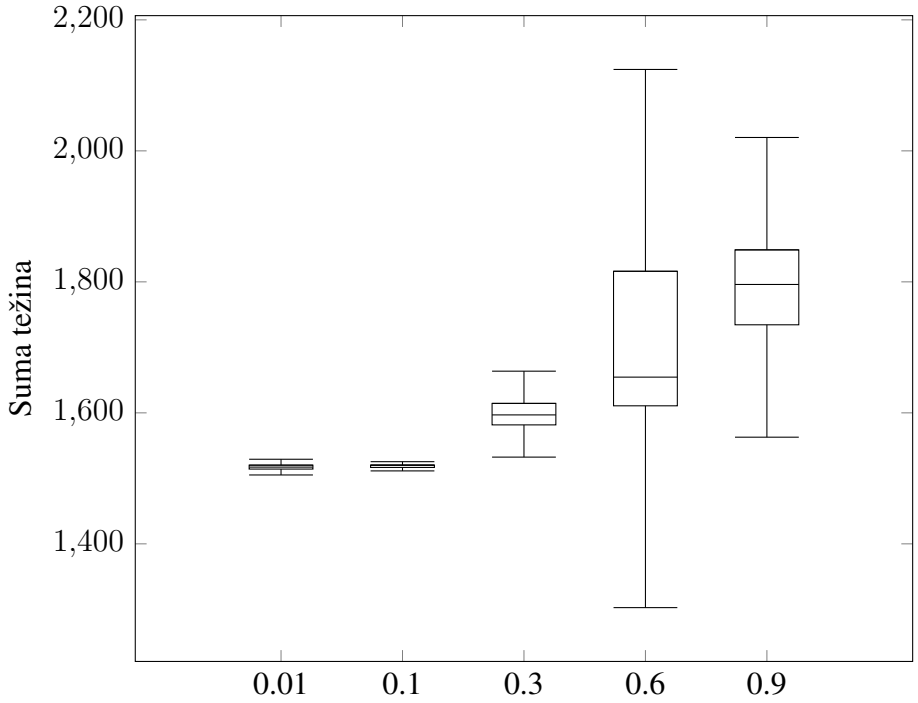
–neutrality **0**

–iterations **500**

Na slikama 5.12,5.13, 5.14, 5.15, 5.16, 5.17 prikazani su rezultati u ovisnosti o pojedinom parametru.

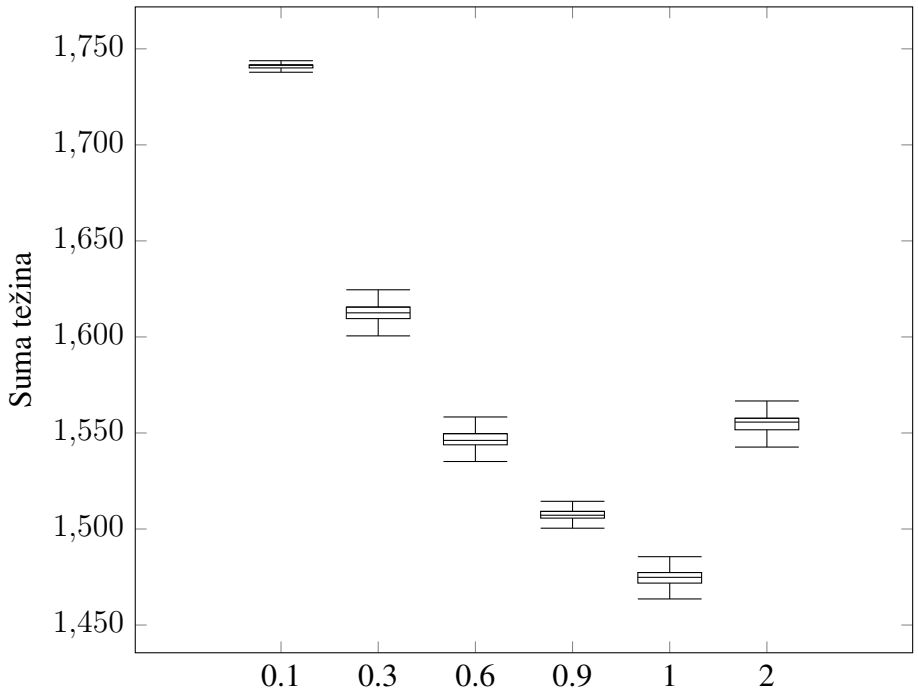


Slika 5.12: Ispitivanje α



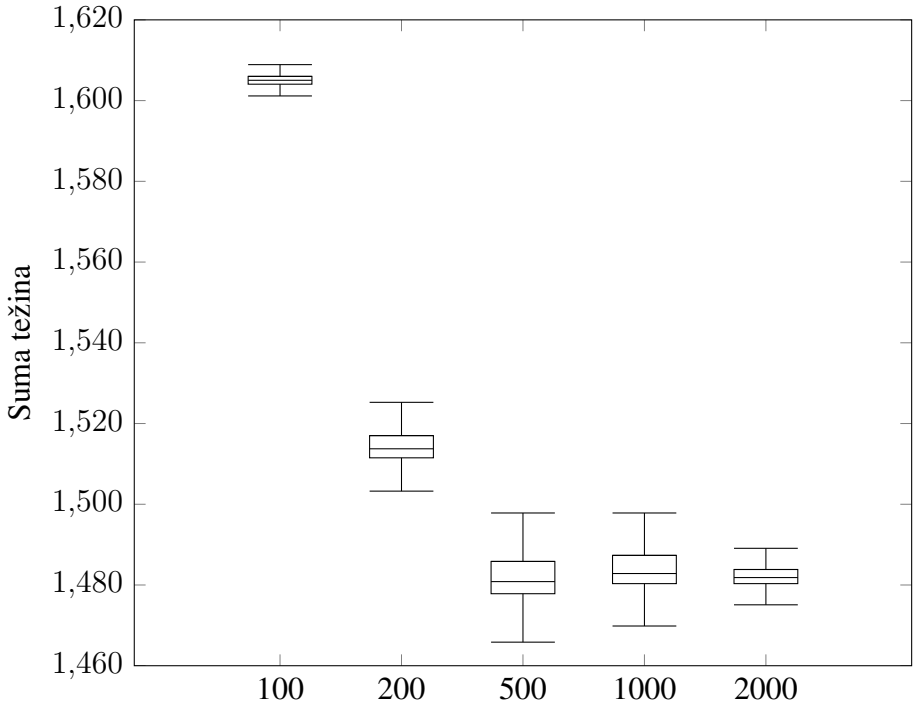
Slika 5.13: Ispitivanje α uz striktno poštivanje tabu list

Vidimo na slikama 5.12 i 5.13 utjecaj ignoriranja tabu liste.

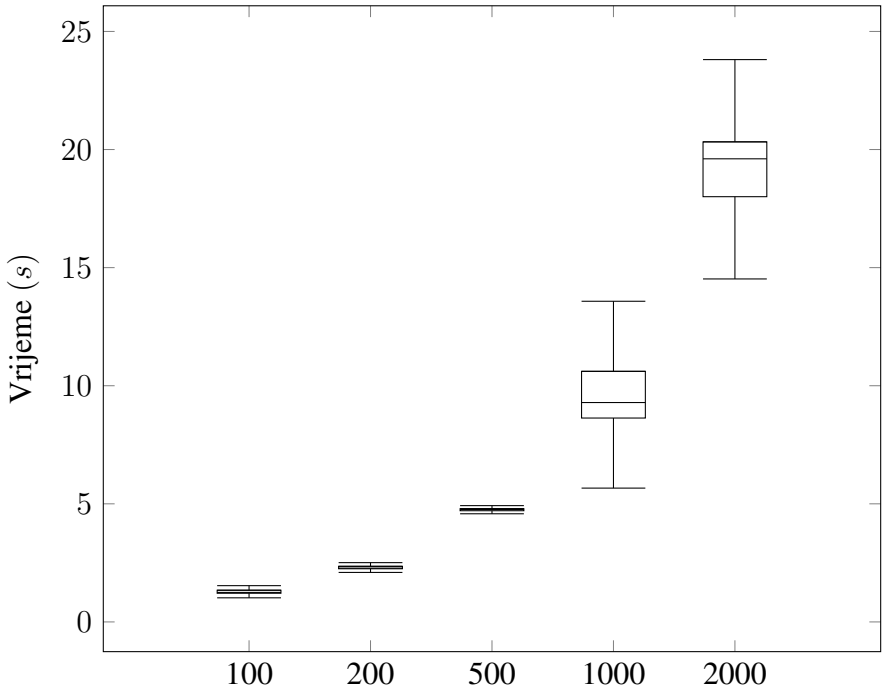


Slika 5.14: Ispitivanje β

Na slici 5.14 najbolje rezultate postižemo za $\beta = 1$.

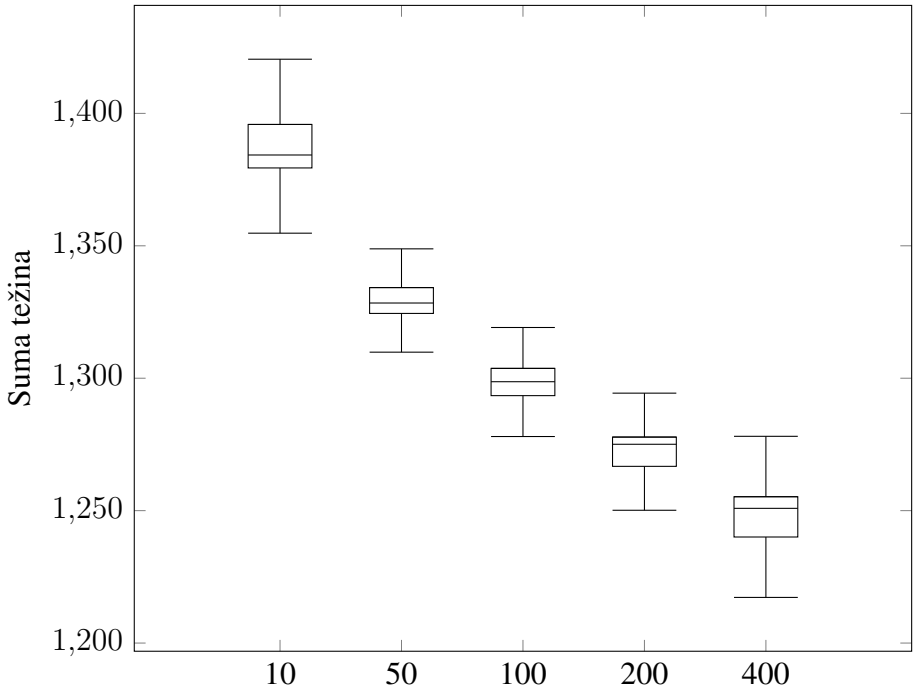


Slika 5.15: Funkcija cilja za broj iteracija



Slika 5.16: Vrijeme izvođenja za broj iteracija

Na slikama 5.15 i 5.16 vidimo da je 500 iteracija bez pomaka dobar indikator kako lokalna pretraga ne može pronaći bolje rješenje.



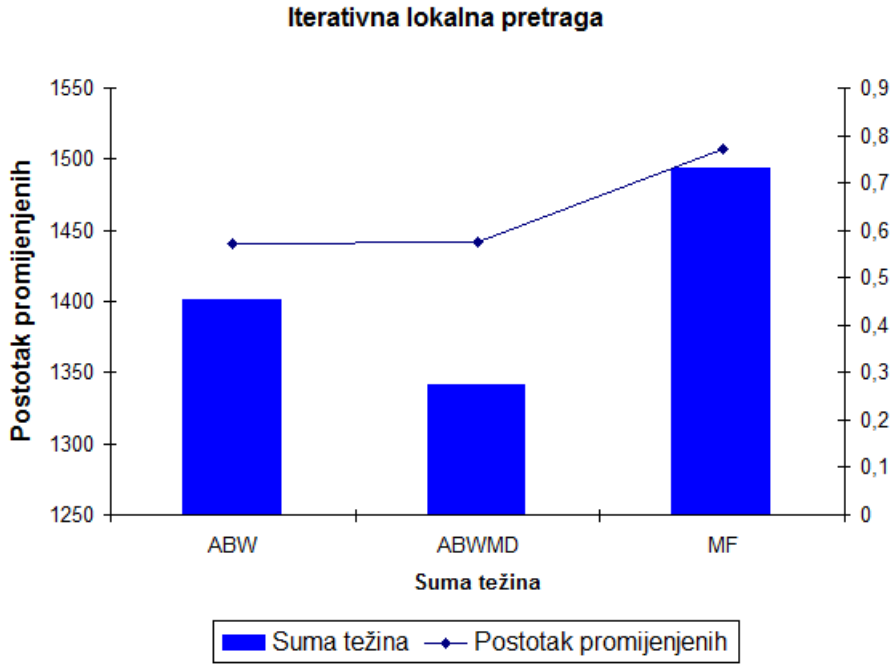
Slika 5.17: Ispitivanje parametra neutralne šetnje

Agresivnije postavke neutralne šetnje pružaju bolje rezultate kao što vidimo na slici 5.17.

5.6. Iterativna lokalna pretraga

5.6.1. Perturbacija

Nakon što lokalna pretraga dosegne kriterij zaustavljanja potrebno je perturbirati postojeće rješenje kako bi potencijalno proučili druga područja prostora rješenja. Za mijenjanje rješenja se koristi pohlepni algoritam a testiranjem ćemo utvrditi koji operatori pohlepnog omogućavaju kvalitetnu pretragu prostora rješenja. Slika 5.18 prikazuje rezultate 3 najbolja operatora odabira boje za graf *Tokai-A* s obzirom na funkcije cilja.



Slika 5.18: Iterativno izvođenje lokalne pretrage s pohlepnim algoritmom

U tablici 5.1 navedeni su rezultati ostalih operatore odabira boje za *Tokai-A*.

Tablica 5.1: Funkcije cilja za zadane operatore

	Suma težina	Postotak promijenjenih
MC	8,89E+07	0,76979
MD	3,13E+11	0,78001
RND	9,58E+08	0,99437
SA	1,70E+07	0,996882

U tablici 5.2 dan je pregled rezultata za različite operatore za *dsjc500.5* graf s ograničenjem dostupnih boja na kromatski broj.

Tablica 5.2: Funkcije cilja za zadane operatore

	Suma težina	Postotak promijenjenih
ABW	88	0,976
ABWMD	80	0,978
MF	90	0,98
MC	80	0,978
MD	41764	0,4
RND	886	0,976
SA	866	0,98

5.6.2. Usporedba s poznatim rezultatima

Za Tokai graf najbolje poznato rješenje ima sumu težina 1153 uz nepoznat postotak promijenjenih vrhova, dok za Kansai nema dostupnih rezultata za usporedbu. U tablici 5.3 dan je prikaz rezultata dobivenih uz korištenje optimalnih parametara gdje je f_1 suma težina konfliktnih bridova a f_2 postotak promijenjenih vrhova.

Tablica 5.3: Rezultati za Tokai i Kansai

		A	B	C	Ukupno
Tokai	f1	1058,516	0	2	1060,515733
	f2	0,558028755	0,349583066	0,464563463	0,498114437
Kansai	f1	5069,525922	0	51,21750043	5120,743422
	f2	0,602809778	0,30933981	0,341391509	0,487696403

U tablici 5.4 prikazana su najbolja legalna k-bojanja postignuta iterativnom lokalnom pretragom na DIMACS grafovima.

Tablica 5.4: Usporedba za klasičan problem bojanja grafa

	najbolji k	**postignut k**	**broj uspjeha**
dsjc500.1	12	13	1/10
dsjc500.5	48	54	7/10
dsjc1000.1	20	22	4/10
le450_25c	25	27	7/10
le450_25d	25	27	4/10
dsjr500.1c	84	86	5/10
dsjr500.5	122	130	4/10
flat300_28_0	28	34	2/10
flat1000_50_0	50	94	1/10
r250.5	65	68	5/10

5.7. Analiza rezultata

Operatori odabira boje i odabira vrha su iznimno bitni kako bi se graf dobio kvalitetno inicijalno bojanje. Rezultatima u tablici 5.1 se također pokazalo kako tabu algoritam ne može dovoljno brzo popraviti funkciju cilja ako izlaz iz pohlepnog nije zadovoljavajući. Operator bojanja *ABW* i njegova varijanta *ABWMD* daju optimalne rezultate. Najbolji operator odabira vrha je *SATURATION* koji je predložen u radu (2).

Genetski algoritam treba relativno puno vremena kako bi konvergirao, a rezultat je marginalno bolji od bojanja koji se postiže pohlepnim algoritmom. Neuronska mreža također ne uspijeva ponuditi rezultate koji bi motivirali njenu upotrebu u pohlepnom algoritmu.

Analizom parametara lokalne pretrage, možemo uočiti par stvari. Ignoriranje tabu liste ako se može doseći bolja funkcija cilja smanjiva važnost parametra α. Osim toga, lokalna pretraga nudi bolje rezultate za male vrijednosti α parametra. U slučaju β parametra, očito je bolje da koristimo sve konfliktne vrhove u pretrazi prostora rješenja, ali zanimljivi su loši rezultati za $\beta = 2$. Čini se da algoritam tada preferira poteze koji se nalaze izvan konfliktnih vrhova, nešto poput neutralne šetnje. Parametar neutralne šetnje pokazuju bolje rezultate za agresivnije postavke što je očekivano ponašanje.

Iterativna lokalna pretraga nad problemom u radu postiže najbolje poznate rezultate za sumu težina. Usporedbom postignutih rezultata i najboljih rezultata na klasičnom problemu bojanja grafa vidimo veliko odstupanje za grafove *flatX_K* dok su za ostale grafove rezultati relativno dobri. Ovo može biti posljedica preferiranja inicijalne boje prilikom optimiziranja te nemogućnosti promatranja vrhova kao da pripadaju određenoj klasi boja a ne konktretnoj boji.

6. Zaključak

U ovom radu istražena je mogućnost primjene pohlepnih algoritama i tabu pretrage na rješavanje optimizacijskog problema bojanja velikih težinskih grafova. Kako tabu pretraga sporo konvergira, za inicijalno bojanje grafa iskorišten je pohlepni algoritam. Dodatno se pokazalo kako iterativno izvođenje pohlepnog algoritma i lokalne pretrage nudi najbolje rezultate i uspijeva poboljšati trenutno poznate rezultate za zadane probleme.

Kvaliteta dobivenih rješenja ovisi o pohlepnom algoritmu, čije izvođenje je podijeljeno na operator odabira boje i odabira vrha. Kako bi potencijalno naučili dobar operator, iskorištena je neuronska mreža koja koristi rezultate koje dobijemo primjenom genetskog algoritma no naučeni operatori ne mogu postići kvalitetu bojanja na drugim grafovima. Ovo može biti zbog dinamične prirode operatora koji uvijek koristi aktualno stanje grafa za donošenje odluke, što u slučaju neuronske mreže nije moguće zbog ograničenja na vremensko izvođenje, ili pak zbog nedostatka informacija koje neuronska mreža koristi prilikom računanja izlaza.

Rezultati su pokazali kako je svojstvo neutralnosti i neutralna šetnja (13) iznimno bitna i većim fokusom na neutralnu šetnju u slučaju lokalnog optimuma postižu se bolji rezultati.

Postupak iterativnog izvođenja tabu pretrage je primijenjen i na klasični problem bojanja grafova gdje su postignuti rezultati relativno blizu najboljim poznatim rezultatima za slične metaheuristike.

LITERATURA

[1] KarenI. Aardal, StanP.M. van Hoesel, ArieM.C.A. Koster, Carlo Mannino, i Antonio Sassano. Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129, 2007. ISSN 0254-5330. doi: 10.1007/s10479-007-0178-0. URL http://dx.doi.org/10.1007/s10479-007-0178-0.

[2] Dr. Hussein Al-Omari i Khair Eddin Sabri. New graph coloring algorithms, 2006. ISSN 1549-3636. URL thescipub.com/pdf/10.3844/jmssp.2006.439.441.

[3] Ivo Blöchliger i Nicolas Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers Operations Research*, 35(3):960 – 975, 2008. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2006.05.014. URL http://www.sciencedirect.com/science/article/pii/S0305054806001420. Part Special Issue: New Trends in Locational Analysis.

[4] Massimiliano Caramia i Paolo Dell'Olmo. Coloring graphs by iterated local search traversing feasible and infeasible solutions. *Discrete Appl. Math.*, 156(2):201–217, Siječanj 2008. ISSN 0166-218X. doi: 10.1016/j.dam.2006.07.013. URL http://dx.doi.org/10.1016/j.dam.2006.07.013.

[5] Marco Chiarandini i Thomas Stützle. An application of iterated local search to graph coloring problem. U *PROCEEDINGS OF THE COMPUTATIONAL SYMPOSIUM ON GRAPH COLORING AND ITS GENERALIZATIONS*, stranice 112–125, 2002.

[6] Philippe Galinier, Jean-Philippe Hamiez, Jin-Kao Hao, i Daniel Porumbel. Recent advances in graph vertex coloring. U Ivan Zelinka, Václav Snášel, i Ajith Abraham, urednici, *Handbook of Optimization*, svezak 38 od *Intelligent Systems Reference Library*, stranice 505–528. Springer Berlin Heidelberg, 2013.

ISBN 978-3-642-30503-0. doi: 10.1007/978-3-642-30504-7_20. URL http://dx.doi.org/10.1007/978-3-642-30504-7_20.

[7] A. Hertz i D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, Prosinac 1987. ISSN 0010-485X. doi: 10.1007/BF02239976. URL http://dx.doi.org/10.1007/BF02239976.

[8] Rubicite Interactive. Pmx crossover, Lipanj 2014. URL http://www.rubicite.com/Tutorials/GeneticAlgorithms/CrossoverOperators/PMXCrossoverOperator.aspx/.

[9] R. Lewis, J. Thompson, C. Mumford, i J. Gillard. A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers Operations Research*, 39(9):1933 – 1950, 2012. ISSN 0305-0548. doi: http://dx.doi.org/10.1016/j.cor.2011.08.010. URL http://www.sciencedirect.com/science/article/pii/S0305054811002425.

[10] S. Lončarić. Višeslojni perceptron, Lipanj 2014. URL http://www.fer.unizg.hr/_download/repository/06-ViseslojniPerceptron-1s.pdf/.

[11] S. Lončarić. Proces učenja, Lipanj 2014. URL http://www.fer.unizg.hr/_download/repository/02-ProcesUcenja-1s.pdf/.

[12] S. Lončarić. Uvod u neuronske mreže, Lipanj 2014. URL http://www.fer.unizg.hr/_download/repository/01-Uvod-1s.pdf/.

[13] Marie-Eléonore Marmion, Aymeric Blot, Laetitia Jourdan, i Clarisse Dhaenens. Neutrality in the graph coloring problem. U Giuseppe Nicosia i Panos Pardalos, urednici, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, stranice 125–130. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-44972-7. doi: 10.1007/978-3-642-44973-4_14. URL http://dx.doi.org/10.1007/978-3-642-44973-4_14.

[14] Martin Riedmiller i I. Rprop. Rprop - description and implementation details, 1994.

Bojanje grafa prilagodljivim postupcima lokalne pretrage

Sažetak

U diplomskom radu je proučena primjena metaheuristika na optimizaciju parametara *Radio Access Network-a* koji je modeliran kao težinski graf. Cilj optimizacije je minimizacija sume konfliktnih bridova uz zadana ograničenja. Za optimizaciju se koriste pohlepni algoritmi te tabu pretraga. Dodatno je istražena mogućnost učenja pohlepnog algoritma uz pomoć neuronske mreže i genetskog algoritma. Eksperimentalno su određeni parametri koji nude najbolje rezultate te je napravljena usporedba s najboljim rezultatima.

Ključne riječi: bojanje grafova, metaheuristika, pohlepni algoritam, tabu pretraga, genetski algoritam, umjetna neuronska mreža

Title

Abstract

This thesis deals with the application of metaheuristics to optimize parameters for *Radio Access Network* which is modeled as a weighted graph. The goal of optimization is to minimize the sum of conflicting edges given the constraints. Greedy algorithms and tabu search are used for optimisation. Possibility of learning new greedy algorithm was explored with the help of neural networks and genetic algorithms. Parameters that provide best results are experimentally determined and comparison is made with the best known results.

Keywords: graph coloring, metaheuristics, greedy algorithm, tabu search, genetic algorithm, artificial neural network