SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1754

RJEŠAVANJE PROBLEMA BOJANJA GRAFOVA PRIMJENOM HIBRIDNOG EVOLUCIJSKOG ALGORITMA

Hrvoje Kindl

Zagreb, rujan 2008.

Ovom prilikom zahvaljujem se svom bratu Filipu na nesebično ustupljenim računalnim resursima te pomoći u otkrivanju pogrešaka u praktičnom dijelu ovog rada. Također, zahvaljujem se svom mentoru doc. dr. sc. Domagoju Jakoboviću na susretljivosti i korisnim savjetima tijekom izrade ovog rada.

Naposlijetku, zahvaljujem se svima koji su mi pružili podršku i pomoć tijekom studija, posebno svojim roditeljima i djevojci Mariji.

Hrvoje Kindl
0036399493

Sažetak

U ovom diplomskom radu proučen je i opisan problem bojanja vrhova jednostavnih grafova te primjene tog problema na praktične probleme iz stvarnog života. Detaljno je istražena i programski ostvarena primjena hibridnog evolucijskog algoritma na rješavanje ovog problema. Ideja hibridnog evolucijskog algoritma sastoji se od ugradnje metode lokalne pretrage u evolucijski algoritam s populacijom rješenja. Također, predstavljena je nova klasa specijaliziranih operatora križanja za ovaj problem. Eksperimentalno je ispitana učinkovitost ostvarenog programskog rješenja na DIMACS ispitnim primjercima grafova. Naposlijetku, načinjena je usporedba s najboljim poznatim rezultatima iz literature i dane su smjernice za daljnje istraživanje.

Abstract

In this diploma thesis, the problem of coloring simple graphs and applications of this problem to real-life practical problems is studied and described. An application of a hybrid evolutionary algorithm to this problem is researched in detail and implemented. A very promising idea behind hybrid evolutionary algorithms is to embed local search into the framework of population based evolutionary algorithms. Also, a new class of highly specialized crossover operators for this problem is presented. The efficiency of our algorithmic implementation is tested on a set of DIMACS graph coloring instances. Finally, a comparison is made with the best known results from the literature and guidelines are given for further research.

Sadržaj

Popis korištenih kratica

NP	Non-deterministic Polynomial time
HEA	hibridni evolucijski algoritam
DIMACS	The Center for Discrete Mathematics and Theoretical Computer Science
PBG	problem bojanja grafova
LP	lokalna pretraga
GA	genetski algoritam
TP	tabu pretraga
TL	tabu lista
TV	tabu vrijeme
HAB	hibridni algoritam za bojanje
PKP	pohlepno križanje particija

1. Uvod

Problem bojanja grafova poznati je teški problem kombinatorne optimizacije. On se sastoji od bojanja vrhova grafa *G* s minimalnim brojem boja (kromatskim brojem X(*G*)) uz uvjet da su susjedni vrhovi različitih boja.

Postoji niz primjena bojanja grafova na važne praktične probleme. Neulazeći u detalje, samo neke od primjena uključuju problem raspoređivanja, problem alokacije registara, problem dodjele frekvencija te ispitivanje tiskanih pločica.

Nažalost, problem je *NP*-potpun. Čak i njegovo aproksimiranje pokazuje se vrlo teškim: nijedan algoritam polinomne složenosti nije u stanju obojati proizvoljan graf *G* koristeći broj boja manji od 2 * X(*G*). Empirijski rezultati bojanja slučajno generiranih grafova potvrdili su težinu problema. Egzaktni algoritmi eksponencijalne su složenosti i mogu rješavati problem u prihvatljivom vremenu za grafove do otprilike 100 vrhova.

Uzimajući u obzir da su šanse za pronalazak učinkovitog egzaktnog algoritma vrlo male, heuristički algoritmi potrebni su za veće instance grafova. Nova, uspješna heuristika za bojanje grafova je hibridni evolucijski algoritam (HEA). Ideja HEA sastoji se od ugradnje lokalne pretrage u evolucijski algoritam s populacijom rješenja. Osim učinkovite metode lokalne pretrage, ključ uspjeha ovakvog algoritma leži u specijaliziranom operatoru križanja. Operator križanja mora biti pažljivo osmišljen i prilagođen problemu koji se rješava. Dizajn prikladnog operatora križanja zahtijeva prvo identifikaciju svojstava koja su značajna za rješavani problem i razvoj rekombinacijskog mehanizma koji prenosi ta svojstva s roditelja na potomke. Osnovna je ideja koristiti operator križanja kako bi se dobila nova zanimljiva rješenja koja se onda dalje unapređuju metodom lokalne pretrage. U ovome radu detaljno je proučen i implementiran jedan konkretan HEA. Kao metoda lokalne pretrage u njemu korištena je tabu pretraga, točnije poznati algoritam Tabucol.

Rad je organiziran po poglavljma kako slijedi. U poglavlju 2 prvo su dane definicije potrebne za pristup problemu bojanja grafova. Zatim je dan kratki pregled heuristika koje se koriste za njegovo rješavanje. Na kraju poglavlja opisane su primjene ovoga problema na praktične probleme. U poglavlju 3 opisana je metoda tabu pretrage te njeni osnovni koncepti i napredne strategije. Poglavlje 4 sadrži detaljnu analizu i opis implementacije svih komponenti HEA pridajući posebnu pozornost operatoru križanja i metodi lokalne pretrage. U poglavlju 5 prvo je opisan DIMACS format zapisa grafa, a zatim su predstavljeni rezultati eksperimenta izvršenih na DIMACS ispitnim primjercima grafova. Na kraju poglavlja rezultati dobiveni s HEA uspoređeni su s rezultatima dobivenim Tabucol algoritmom te s najboljim poznatim rezultatima iz literature. U poglavlju 6 dana su neka korisna zapažanja i smjernice za daljnje istraživanje. Zadnja dva poglavlja 7 i 8 sadrže zaključak i popis korištene literature.

2. Problem bojanja grafova

2.1 Definicije

2.1.1 Definicije teorije grafova

Definicija 1. **Jednostavni graf** G sastoji se od uređenog para $G = (V, E)$ sa sljedećim svojstvima:

- $V = \{v_0, ..., v_n\}$ je neprazan, konačan skup čije elemente zovemo **vrhovi** grafa G,
- $E = \{(v_i, v_j)\}$ je konačan skup različitih dvočnanih (neuređenih) podskupova skupa V koje zovemo **bridovi**.

Uočimo da smo prethodnom definicijom *jednostavnog* grafa isključili mogućnost da su dva vrha spojena s više bridova (budući smo E definirali kao skup), te da postoji brid koji spaja vrh sa samim sobom (jer smo svaki brid definirali kao dvočlani podskup). Primjer jednostavnog grafa prikazuje slika 2.1.

U daljnjem tekstu pod pojmom graf podrazumijevat ćemo jednostavni graf. Također umjesto da pišemo "skup V grafa G", pisat ćemo kraće "skup $V(G)$", analogno za bridove "skup $E(G)$".

Definicija 2. Za brid $e = \{v, u\}$ kažemo da **spaja** vrhove v i u i bez mogućnosti zabune kraće ga pišemo vu. U toj situaciji kažemo da su vrhovi v i u grafa G **susjedni**. Također, kažemo da su vrhovi v i u **incidentni** s bridom e.

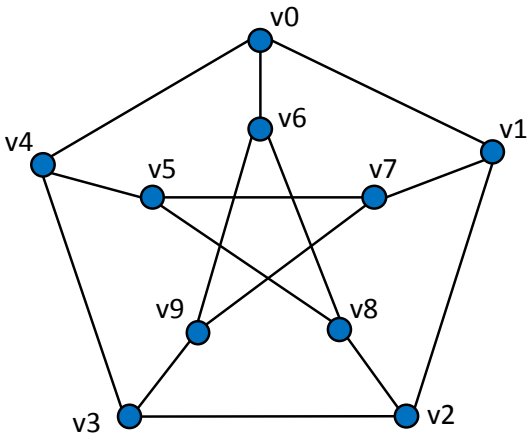
Definicija 3. Skup vrhova koji su susjedni vrhu v zovemo **susjedstvo** vrha v i označavamo s oznakom $H(v)$.

Definicija 4. **Stupanj vrha** v grafa G jednak je broju bridova koji su incidentni s v. Označavamo ga s $d(v)$. Najveći stupanj vrha u grafu označavamo s $\Delta(G)$.

Definicija 5. Neka je definirana funkcija $w : E \rightarrow \mathbf{R}$ koja svakom bridu e grafa G pridružuje realan broj $w(e)$ kojeg ćemo zvati **težina brida**. Takav graf zovemo **težinski graf**.

Definicija 6. Za zadane disjunktne grafove $G_1 = (V(G_1), E(G_1))$ i $G_2 = (V(G_2), E(G_2))$ definiramo njihovu **uniju** $G_1 \cup G_2$ kao graf $G_1 \cup G_2 = (V(G_1) \cup V(G_2), E(G_1) \cup E(G_2))$.

Definicija 7. Graf je **povezan** ako se ne može prikazati kao unija neka dva grafa. U suprotnom kažemo da je graf **nepovezan**. Svaki se nepovezani graf dakle može prikazati kao unija povezanih grafova.



Slika 2.1 Primjer jednostavnog grafa:
Petersenov graf

Definicija 8. Za graf *G* kažemo da je **regularan**, ako su svi njegovi vrhovi istog stupnja. Kažemo da je *G r*-regularan ako je $d(v) = r, \forall v \in V(G)$. Cijeli broj *r* tada ćemo zvati **stupanj regularnosti** grafa *G*.

Definicija 9. Jednostavni graf kod kojeg su svaka dva vrha susjedna zovemo **potpuni graf**. Potpuni graf s *n* vrhova označavamo s K_n.

Definicija 10. Povezani 2-regularni graf zovemo **ciklički graf** ili kratko **ciklus**. Ciklus s *n* vrhova označavamo s C_n te je riječ o **neparnom ciklusu** kada je *n* neparan odnosno **parnom ciklusu** kada je *n* paran.

Većina prethodnih definicija, kao i dobar uvod u teoriju grafova općenito, može se pronaći u literaturi [20].

2.1.2 Definicije računske teorije složenosti

Definicija 11. U računskoj teoriji složenosti, **problem odluke** je problem koji uvijek ima odgovor da ili ne.

Definicija 12. Problemi odluke za koje postoje algoritmi koji daju odgovor, a čije vrijeme izvršavanja ovisi *polinomno* o veličini ulaznih podataka spadaju u **klasu *P*** i zovemo ih ***P*-problemima**.

Definicija 13. Kažemo da je neki problem odluke sadržan u **klasi *NP***, odnosno naziva se ***NP*-problem**, ako za njega postoji algoritam koji u *polinomnom* vremenu može ispitati je li neko predloženo rješenje stvarno rješenje danog problema.

Definicija 14. ***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.

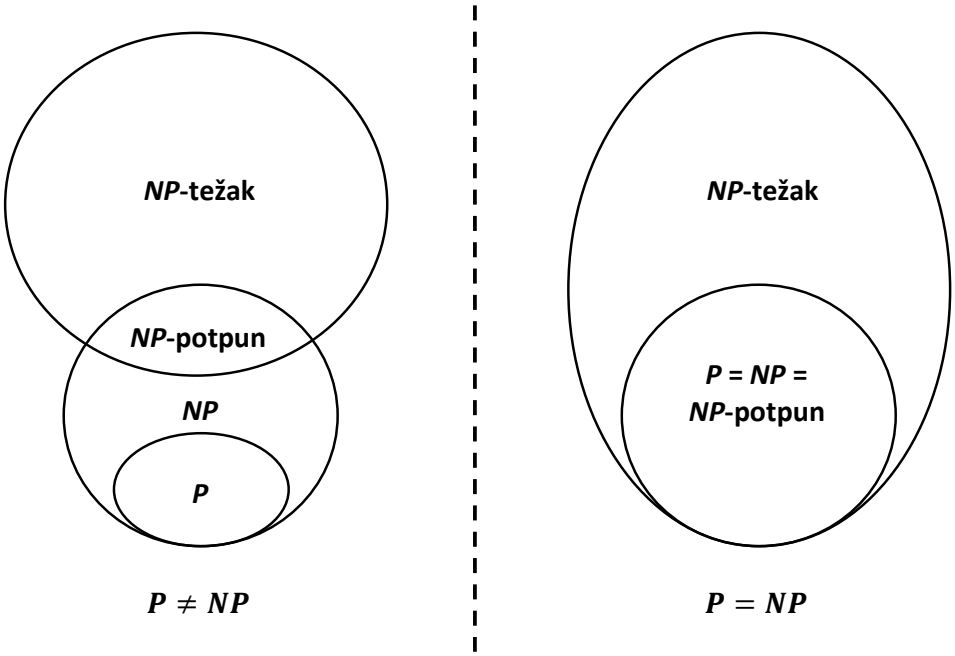
Iz prethodnih definicija lako se izvodi da je svaki *P*-problem ujedno i *NP*-problem, odnosno $\boldsymbol{P \subseteq NP}$. Važno je istaknuti da do danas nije poznato jesu li *NP*-problemi ujedno i *P*-problemi, odnosno je li $\boldsymbol{P = NP}$?

Kada bi se pokazalo da samo jedan od *NP*-potpunih problema leži u *P*, slijedilo bi $\boldsymbol{P = NP}$, jer tada bi se svaki problem iz *NP* mogao polinomno reducirati na njega.

Pretpostavlja se da je $\boldsymbol{P \neq NP}$, međutim zasada to još nitko nije dokazao. Ovo je jedno od najvažnijih otvorenih pitanja u teoretskom računarstvu, s obzirom na široke implikacije koje bi rješenje tog pitanja imalo. Clay Mathematics Institute je 2000. godine objavio da će isplatiti nagradu od milijun USD prvoj osobi koja dokaže rješenje [28].

Definicija 15. Kažemo da je problem ***NP*-težak** akko postoji *NP*-potpun problem koji ima svojstvo da se može *polinomno* reducirati na njega.

Uočimo da smo prethodnom definicijom dopustili da *NP*-težak problem može biti bilo kakav problem, a ne samo problem odluke. Odnosi klasa složenosti za slučaj $\boldsymbol{P \neq NP}$ te slučaj $\boldsymbol{P = NP}$ prikazani su Vennovim dijagramom na slici 2.2.



Slika 2.2 Vennov dijagram: Odnosi klasa složenosti

2.1.3 Definicija problema

Definicija 16. Neka je definirana funkcija $c : V \rightarrow \mathbf{N}$ koja pridružuje svakom vrhu grafa *G* prirodan broj *c*(*v*) kojeg ćemo zvati **boja vrha**. Pridruživanje boja vrhovima grafa, odnosno funkciju *c*, zovemo **bojanje vrhova grafa** ili kraće **bojanje grafa**.

Definicija 17. Bojanje vrhova grafa *G* s najviše *k* boja zovemo ***k*-bojanje** grafa *G*.

Definicija 18. **Legalno *k*-bojanje** grafa *G* je bojanje grafa *G* s najviše *k* boja tako da su susjednim vrhovima pridružene različite boje.

U daljnjem tekstu ćemo radi jednostavnosti možda neki put pisati "problem bojanja" grafa *G*, dok ćemo zapravo misliti na problem legalnog *k*-bojanja grafa *G*.

Definicija 19. Graf *G* je ***k*-obojiv** akko postoji legalno *k*-bojanje od *G*.

Definicija 20. Ako je graf G *k*-obojiv ali nije (*k-1*)-obojiv, kažemo da je *G* ***k*-kromatski**, odnosno kažemo da je **kromatski broj** od *G* jednak *k* i pišemo *X*(*G*) = *k*.

U sljedećim teoremima dati ćemo rezultate u vezi gornje ograde za kromatski broj proizvoljnog grafa.

Teorem 1. **(Brooks)** Za proizvoljan graf *G* vrijedi:

$$\mathrm{X}(G) \leq \Delta(G) + 1 \tag{2.1}$$

Nadalje, ako je *G* povezan, tada u gornjem izrazu vrijedi jednakost akko je *G* potpun graf ili neparan ciklus.

Teorem 2. **(Stacho)** Kromatski broj proizvoljnog grafa *G* zadovoljava nejednakost:

$$\mathrm{X}(G) \leq \Delta_2(G) + 1 \tag{2.2}$$

gdje je:

$$\Delta_2(G) = \max_{u \in V(G)} \max_{\substack{v \in H(u) \\ d(v) \leq d(u)}} d(v)$$

Dakle, $\Delta_2(G)$ je najveći stupanj vrha *v* u grafu G uz uvjet da je *v* susjedan barem jednom vrhu *u* čiji je stupanj veći ili jednak stupnju od *v*.

Primjetimo da izraz (2.2) daje strožu gornju ogradu od izraza (2.1) ako nijedna dva vrha najvećeg stupnja nisu susjedna.

Korisno je primjetiti da je problem *k*-bojanja grafa *G* zapravo ekvivalentan problemu *k*-diobe grafa *G*. Pojašnjenje slijedi u nastavku.

Definicija 21. Podskup skupa *V*(*G*) zovemo **nezavisni skup** ako niti jedna dva vrha u njemu nisu susjedna.

Definicija 22. Brid koji spaja dva vrha iste boje zovemo **konfliktni brid**.

Definicija 23. **Konfliktni vrhovi** definiraju se kao vrhovi koji se nalaze na krajevima konfliktnog brida.

Definicija 24. Particiju skupa *V*(*G*) na *k* disjunktnih nepraznih podskupova V_1 ,..., V_k tako da vrijedi $V(G) = \bigcup_{i=1}^{k} V_i$ zovemo ***k*-dioba** grafa *G*. Ako su podskupovi V_1 ,..., V_k ujedno i *nezavisni* skupovi onda se radi o **legalnoj *k*-diobi** grafa *G*.

Podskupove V_1 ,..., V_k ponekad zovemo i **klase boja**.

Teorem 3. Graf *G* je *k*-obojiv akko postoji legalna *k*-dioba grafa *G*.

Dokaz. ($\Longrightarrow$) Neka je *G* *k*-obojiv. Tada postoji legalno *k*-bojanje od *G*. Sada *V*(*G*) podijelimo u *k* podskupova V_1 ,..., V_k tako da svaki podskup sadrži samo vrhove iste boje. Ovime smo dobili legalnu *k*-diobu od *G*.
($\Longleftarrow$) Neka postoji legalna *k*-dioba od *G* na V_1 ,..., V_k. Tada vrhove u svakom podskupu V_i obojamo drugom bojom. Tako smo dobili legalno *k*-bojanje, pa je *G* *k*-obojiv. ■

Problem pronalaženja kromatskog broja X(*G*) zadanog grafa *G* je ***NP*-težak** problem. Odgovarajući problem odluke, odnosno problem pronalaženja odgovora na pitanje postoji li legalno *k*-bojanje zadanog grafa *G* je ***NP*-potpun** problem [24].

U daljnjem tekstu ćemo ponekad umjesto "problem bojanja grafa" pisati skraćeno **PBG**, a umjesto "problem *k*-bojanja grafa" pisati ***k*-PBG**.

2.1.4 Definicije pojmova

U sljedećim definicijama pod pojmom "rješenje" misli se na bilo koje rješenje iz skupa dozvoljenih rješenja za neki problem, ne nužno optimalno.

Definicija 25. U matematici, pojam **optimizacija** odnosi se na proučavanje problema u kojima se želi minimizirati ili maksimizirati realnu funkciju na način da se sistematski biraju vrijednosti njenih varijabli iz dozvoljenog skupa vrijednosti.

Definicija 26. **Kombinatorna optimizacija** je grana optimizacije. Njena domena je optimizacija problema gdje je skup dozvoljenih rješenja diskretan ili se može reducirati na diskretni skup, a cilj je pronaći optimalno rješenje.

Definicija 27. Funkciju nad kojom provodimo optimizaciju zovemo **funkcija cilja**.

Definicija 28. U računarskoj znanosti, **heuristički algoritam** ili kraće **heuristika** je algoritam koji žrtvuje pronalaženje optimalnog rješenja u svrhu skraćivanja vremena izvođenja.

Definicija 29. **Metaheuristika** je općenita strategija koja pomaže u vođenju neke heuristike prema boljim rješenjima, a primjenjuje se na općenitim klasama problema.

Definicija 30. U algoritmima optimizacije, skup svih dozvoljenih rješenja nazivamo **prostor pretrage**.

Definicija 31. Podskup prostora pretrage ***N*(*x*)** za koji vrijedi: $N(x)$ = {skup rješenja koja su u *relaciji susjedstva* s rješenjem x} zovemo **susjedstvo** rješenja x.

Relacija susjedstva je binarna, simetrična i netranzitivna relacija. Kažemo da su rješenja iz skupa $N(x)$ **susjedna** rješenja od x.

Objasnimo relaciju susjedstva na sljedećem jednostavnom primjeru. Neka je prostor pretrage skup točaka iz $\mathbf{Z_n}$ oblika $x = \{x_1, ..., x_n\}$, $x_i \in \mathbf{Z}$, onda možemo definirati jednu relaciju susjedstva ovako: dva rješenja (točke) x, y su susjedna ako im udaljenost iznosi $d = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2} = 1$. To zapravo znači da im se točno jedna koordinata razlikuje za ±1.

Definicija 32. **Lokalna pretraga (*LP*)** je metaheuristika za rješavanje računski teških optimizacijskih problema.

Algoritam lokalne pretrage započinje pretragu od nekog početnog rješenja i onda uzastopnim iteracijama prelazi na susjedna rješenja [26]. To je moguće samo ako je definirana relacija susjedstva na prostoru pretrage. Obično svako rješenje ima više susjednih rješenja, a izbor slijedećeg rješenja obavlja se samo na temelju informacija o susjednim rješenjima. Algoritam pretražuje prostor rješenja sve dok optimalno rješenje nije pronađeno ili dok neki drugi uvjet zaustavljanja nije zadovoljen (npr. maksimalni broj iteracija algoritma).

2.1.5 Strategije za rješavanje problema bojanja grafova

Općenito, u osmišljavanju algoritma za rješavanje specifičnog optimizacijskog problema potrebno je definirati prostor pretrage koji će se pretraživati te funkciju cilja koju će se minimizirati. Kod algoritama lokalne pretrage, primjerice, potrebno je definirati još i relaciju susjedstva. Ovi elementi čine ono što nazivamo *strategija pretrage*. Ovdje ćemo klasificirati strategije pretrage u četiri kategorije. Dvije od njih rješavaju problem bojanja grafova, dok druge dvije razmatraju fiksan broj boja te rješavaju problem *k*-bojanja grafova. Riječ je o sljedećim strategijama:

- ***legalna strategija*:** prostor pretrage *S* sadrži sva legalna bojanja i cilj je pronaći rješenje $x \in S$ koje koristi najmanji mogući broj boja. Minimizacija funkcije cilja $f(x) = -\sum_{i=1}^{k} |V_i|^2$ potiče smanjivanje najmanjih klasa boja te njihovu eventualnu eliminaciju.
- ***kaznena strategija*:** prostor pretrage *S* sadrži sva, ne nužno legalna bojanja i cilj je pronaći legalno bojanje $x \in S$ koje koristi najmanji mogući broj boja. Funkcija cilja u ovome slučaju mora u isto vrijeme poticati smanjivanje broja korištenih boja i smanjivanje broja konfliktnih bridova. To se može postići minimizacijom $f(x) = \sum_{i=1}^{k} 2|V_i||E_i| - \sum_{i=1}^{k} |V_i|^2$ gdje $|E_i|$ predstavlja skup bridova s oba kraja u skupu V_i. Dokazano je u [15] da svi lokalni minimumi ove funkcije odgovaraju legalnim bojanjima.
- ***k-kaznena strategija*:** broj boja *k* je fiksan, prostor pretrage *S* sadrži sva, ne nužno legalna *k*-bojanja i cilj je pronaći legalno *k*-bojanje $x \in S$. Minimizacijom funkcije cilja $f(x) = \sum_{i=1}^{k} |E_i|$ smanjujemo broj konfliktnih bridova.
- ***k-legalna parcijalna strategija*:** broj boja *k* je fiksan, prostor pretrage *S* sadrži sva legalna parcijalna *k*-bojanja i cilj je pronaći rješenje $x \in S$ u kojem su svi vrhovi obojeni. Rješenje $x \in S$ može se predstaviti particijom (V_1 ,..., V_k, V_{k+1}) skupa vrhova gdje skupovi V_1 ,..., V_k predstavljaju legalnu *k*-diobu dok skup V_{k+1} sadrži neobojene, odnosno nerazvrstane vrhove. Funkcija cilja može biti $f(x) = \sum_{v \in V_{k+1}} d(v)$ gdje *d*(*v*) predstavlja stupanj vrha *v*.

Ova podjela prikazana je pregledno u tablici 2.1.

Tablica 2.1 Strategije za rješavanje PBG

Prostor pretrage	*k* nije fiksan		*k* fiksan	
	legalna bojanja	sva bojanja	sva *k*-bojanja	legalna parcijalna *k*-bojanja
Funkcija cilja	$-\sum_{i=1}^{k} \lvert V_i \rvert^2$	$\sum_{i=1}^{k} \lvert V_i \rvert (2\lvert E_i \rvert - \lvert V_i \rvert)$	$\sum_{i=1}^{k} \lvert E_i \rvert$	$\sum_{v \in V_{k+1}} d(v)$
Strategija pretrage	legalna	kaznena	*k*-kaznena	*k*-legalna parcijalna

2.2 Heuristike za bojanje grafova

2.2.1 Lokalna pretraga

Nekoliko algoritama lokalne pretrage se pokazalo vrlo uspješnim u rješavanju problema bojanja grafova. To su obično algoritmi temeljeni na simuliranom kaljenju ili tabu pretrazi. Uglavnom se razlikuju u načinu definiranja prostora pretrage, funkcije cilja te relacije susjedstva. Više o primjeni ovih heuristika na problem bojanja grafova može se pronaći u radovima iz literature [1], [8], [14], [16] i [19].

2.2.2 Genetski algoritam

U ranim 90tim, Davis [3] je implementirao genetski algoritam za rješavanje problema bojanja grafova. Njegov GA kodira rješenja kao poredak vrhova. Preciznije, rješenje se kodira kao permutacija vrhova. Kako bi se izračunala njegova dobrota, pohlepnim algoritmom svakom vrhu u slijedu dodijeljuje se prva dozvoljena boja. Ovakav algoritam daje slabe rezultate. Od tada niti jedan rad u kojem se koristi čisti GA za rješavanje problema bojanja grafova nije objavljen. Raširena je pretpostavka da čisti GA nije kompetitivan za rješavanje ovoga problema.

2.2.3 Hibridni algoritmi

Hibridni algoritmi kombiniraju metode lokalne pretrage s evolucijskim algoritmima. Prvi radovi u kojima se primjenjuje ovaj koncept na problem bojanja grafova su od autora Costa, Hertz i Dubuis [2] 1995. godine te Fleurent i Ferland [7] 1996. godine. U tim radovima koristi se populacija rješenja te operator križanja kao i u običnom GA, samo se umjesto nasumičnog operatora mutacije koristi metoda lokalne pretrage. Zato ovaj tip algoritma zovemo još i *genetska lokalna pretraga* (GLP). Ovakav pristup dao je rezultate neznatno bolje od čiste lokalne pretrage. Algoritmi koji koriste populaciju rješenja i lokalnu pretragu, ali bez križanja predloženi su u radu Morgensterna [19] 1996. godine i daju izvrsne rezultate.

Novi tip GLP algoritama predožen je u radovima Dorne i Hao [6] te Galinier i Hao [9]. U oba rada evolucija populacije odvija se na prilično jednostavan i standardan način, dok je kao metoda lokalne pretrage korištena tabu pretraga, točnije algoritam Tabucol. Rezultati koje su dobili na velikim primjercima grafova bolji su od bilo kojih prethodno postignutih. Razlog ovome uspjehu leži u snazi njihovih specijaliziranih operatora križanja. Glass i Prügel-Bennett u svom radu [11] analizirali su utjecaj zamijene Tabucol algoritma s metodom nabržeg spusta u algoritmu Galiniera i Hao-a, pritom koristeći puno veću populaciju. Njihovi eksperimenti pokazuju da se ovim načinom mogu dobiti rješenja iste kvalitete ali uz znatno veću količinu računanja.

2.2.4 Gradnja nezavisnih skupova

Ovaj pristup sastoji se od uzastopne gradnje klasa boja tako se svaki puta pronađe maksimalni nezavisni skup u grafu te se onda odstrane pripadajući mu vrhovi iz grafa. Na taj način je problem bojanja grafova sveden na problem uzastopnog pronalaženja maksimalnog nezavisnog skupa u grafu. Različite metode za pronalaženje maksimalnog nezavisnog skupa predložene su u radovima [7], [14], [15] i [19]. Ovakav pristup pokazuje se jednim od općenito najuspješnijih za rješavanje problema bojanja velikih, slučajno generiranih grafova.

U radu Galiniera i Hertza [8] može se pronaći detaljniji povijesni pregled heuristika korištenih za rješavanje problema bojanja grafova.

2.3 Primjena na praktične probleme

2.3.1 Problemi raspoređivanja

Mnogi problemi iz domene raspoređivanja mogu se opisati nizom ograničenja kojima definiramo koji se parovi poslova, uz zadane resurse, (ne)mogu izvoditi istovremeno [5]. Primjerice, u izradi rasporeda nastave na fakultetu, dva predmeta koja predaje isti predavač ne smiju biti zakazana u istom vremenskom intervalu. Slično tome, dva predmeta koje sluša ista grupa studenata ne smiju biti zakazana u istom vremenskom intervalu. Problem određivanja minimalnog broja vremenskih intervala (sati) potrebnih da bi se održala nastava iz svih predmeta je ustvari problem bojanja grafova. Problemi izrade rasporeda obrađeni su detaljno u literaturi [4].

2.3.2 Dodjela frekvencija

Razmotrimo mrežu radio odašiljača, gdje svaki odašiljač mora imati svoju radnu frekvenciju. Ako dva odašiljača koja su prostorno blizu imaju istu frekvenciju, tada postoji mogućnost interferencije. U najjednostavnijem modelu takvi odašiljači trebaju imati različite radne frekvencije, a cilj je koristiti što manji ukupan broj frekvencija [22]. Bojanje grafova je prirodan pristup ovome problemu. Vrhovi grafa predstavljaju odašiljače i brid povezuje dva vrha u grafu akko među njima postoji mogućnost interferencije. Frekvencije tada odgovaraju bojama koje treba dodijeliti vrhovima uz uvjet da susjedni vrhovi moraju biti različite boje. Pretpostavlja se da su frekvencije diskretne te su boje predstavljene cijelim brojevima. U složenijim modelima zahtijeva se veća razlika u radnoj frekvenciji za parove odašiljača koji su bliže jedan drugome i cilj je minimizirati frekvencijski raspon raspodijele, odnosno razliku između najviše i najniže korištene frekvencije. Ovako postavljen problem može se modelirati tako da svakom bridu dodijelimo težinu - pozitivan cijeli broj koji predstavlja minimalnu razliku između boja dvaju vrhova koje taj brid spaja. Ovaj problem pojavljuje se ne samo u radio mrežama, nego u svim mrežama koje koriste elektomagnetske valove za komunikaciju, primjerice u onima koje koriste svjetlost i svjetlovode.

2.3.3 Alokacija registara

Jedna vrlo aktivna primjena bojanja grafova jest u rješavanju problema alokacije registara. Taj problem je u stvari problem dodjele programskih varijabli ograničenom broju fizičkih registara za vrijeme izvođenja programa. Varijablama pohranjenim u registre moguće je znatno brže pristupiti nego onima koje nisu u pohranjene u registre. Obično imamo puno više varijabli nego raspoloživih registara, tako da je nužno dodijeliti više varijabli pojedinim registrima. Za varijable kažemo da su u konfliktu ako se obje koriste u istom kraćem vremenskom intervalu tokom izvođenja programa, primjerice u istoj metodi. Cilj je dodijeliti registrima varijable koje nisu u konfliktu i na taj način minimizirati uporabu ne-registarske memorije. Jednostavan pristup ovom problemu jest načiniti graf gdje vrhovi predstavljaju varijable dok bridovi predstavljaju konflikt između varijabli, odnosno vrhova. Ako postoji legalno bojanje toga grafa s brojem boja manjim ili jednakim broju raspoloživih registara, tada je alokacija registara bez konflikta moguća.

2.3.4 Ispitivanje tiskanih pločica

Razmotrimo problem ispitivanja tiskanih pločica na neželjene kratke spojeve. Da bi se ubrzao proces isptivanja imamo sljedeće razmišljanje. Načinimo graf gdje vrhovi odgovaraju mrežama na pločici a brid postoji između dva vrha ako postoji mogućnost kratkog spoja između odgovarajućih mreža. Bojanjem ovoga grafa zapravo radimo particiju skupa mreža na takozvane "super-mreže", gdje se mreže u svakoj super-mreži mogu paralelno ispitivati od kratkog spoja prema svim ostalim mrežama. Na taj način ubrzavamo proces ispitivanja.

2.3.5 Sudoku slagalica

Ovdje ćemo pokazati jednu možda manje važnu, ali interesantnu primjenu bojanja grafova. Riječ je naime o danas vrlo popularnoj slagalici *sudoku*. Ova slagalica u svom originalnom obliku sastoji se od kvadratnog polja dimenzija 9 x 9 koje je pak podijeljeno na 9 podpolja dimenzija 3 x 3. Primjer nerješene sudoku slagalice prikazan je na slici 2.3 na sljedećoj stranici. Vidimo da su samo neki od ukupno 81 kvadratića u polju popunjeni znamenkama (od 1 do 9) i upravo ovi popunjeni kvadratići zadaju slagalicu.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Slika 2.3 Primjer zadane sudoku slagalice

Cilj slagalice je popuniti sve prazne kvadratiće u polju znamenkama od 1 do 9 uz uvjet da se u svakom retku polja i svakom stupcu polja te u svakom od 9 označenih podpolja znamenke od 1 do 9 pojavljuju točno jedanput.

Pokažimo sada kako se problem rješavanja sudoku slagalica može se formulirati kao problem *k*-bojanja grafova. Prvo označimo retke i stupce polja znamenkama od 1 do 9 te podpolja slovima od A do I kako je prikazano na slici 2.4.

	1	2	3	4	5	6	7	8	9
1									
2		A			B			C	
3									
4									
5		D			E			F	
6									
7									
8		G			H			I	
9									

Slika 2.4 Oznake redaka, stupaca i podpolja
u sudoku slagalici

Sada zadanu sudoku slagalicu možemo predstaviti kao parcijalno 9-bojanje grafa s 81 vrhom (po jedan vrh za svaki kvadratić u polju). Vrhovima ćemo pridijeliti oznake prema položaju odgovarajućih kvadratića u polju na sljedeći način. Dakle, svaki vrh označen je uređenom trojkom (x, y, z), gdje su x, y i z redom oznake retka, stupca i podpolja kojem odgovarajući kvadratić pripada. Brid postoji između dva različita vrha (x, y, z) i (x', y', z') ako i samo ako je ispunjen barem jedan od sljedeća 3 uvjeta:

(1) $x = x'$ (nalaze se u istom retku),

(2) $y = y'$ (nalaze se u istom stupcu),

(3) $z = z'$ (nalaze se u istom podpolju).

Rješenje zadane slagalice je legalno 9-bojanje grafa (znamenkama od 1 do 9) koji smo upravo konstruirali. Ovaj postupak može se poopćiti na sudoku slagalice dimenzija n x n. Detaljnija matematička analiza ove slagalice može se pronaći u literaturi [27].

3. Tabu pretraga

3.1 Osnovni koncepti

Tabu pretraga (*TP*) je matematička optimizacijska metoda koja pripada klasi lokalnih pretraga. Tabu pretraga koristi memorijske strukture i kada pronađe potencijalno rješenje, označi ga kao **tabu** (zabranjeno) kako se više ne bi vraćala na njega. Sva rješenja označena kao tabu pamte se u takozvanoj **tabu listi**. Tabu pretraga pripisuje se *Fredu Gloveru* [30].

Tabu lista (*TL*) je memorijska struktura pomoću koje se određuje koja su rješenja iz prostora rješenja trenutno zabranjena (imaju **tabu** status). Broj iteracija za vrijeme kojih je stavka sadržana u tabu listi zabranjena zove se **tabu vrijeme.** Označavat ćemo ga oznakom *TV*.

Definicija 33. Zamjena trenutnog rješenja x s nekim rješnjem x' iz njegova susjedstva $N(x)$ zovemo **potez**.

Definicija 34. Vraćanje na već posjećeno rješenje u procesu pretrage prostora rješenja nazivamo **kruženje**, a niz poteza koji smo pri tome učinili zovemo **ciklus**.

Osnovni princip TP-a je korištenje algoritma lokalne pretrage koji kada naiđe na lokalni optimum dopušta poteze koji vode rješenjima lošijim od trenutnog. Pronađeni lokalni optimum (rješenje) pamti se u tabu listi, i vraćanje na njega spriječeno je korištenjem tabu liste. Ova ideja izbjegavanja kruženja je ključna i dovodi se u vezu s konceptima umjetne inteligencije.

Umjesto da u tabu listi pamtimo sama rješenja, jedna varijacija principa je da tabu lista sadrži atribute, odnosno brani rješenja koja posjeduju određene atribute (npr. kod bojanja grafova, sva rješenja kod kojih je vrh v_i boje c_j su zabranjena). Duga varijacija je da TL sadrži poteze, odnosno brani određene poteze (npr. kod bojanja grafova, promjena boje vrha v_t zabranjena je slijedećih n iteracija).

Tabu liste koje sadrže atribute ili poteze mogu biti učinkovite u nekim domenama, ali povlače novi problem. Mogu braniti povoljna rješenja čak i kada nema opasnosti od kruženja [30]. Zato je nužno definirati **kriterij prihvaćanja**, koji privremeno ukida tabu status poteza. Najjednostavniji i najčešće korišteni kriterij prihvaćanja (korišten praktički u svim implemetacijama TP-a) je da se uvijek dopušta potez ako vodi rješenju koje je bolje od dosad najboljeg pronađenog rješenja. Očito, ovo novo rješenje nije bilo posjećeno. Ključno pravilo kod svih kriterija prihvaćanja je da se uvijek mogu dopustiti potezi koji neće uzrokovati kruženje, bez obzira na tabu listu.

Najvažniji koraci u implemetaciji bilo koje lokalne pretrage pa tako i TP-a su definiranje prostora pretrage, te definiranje relacije susjedstva. O njima najviše ovisi učinkovitost pretrage i mora ih se pažljivo definirati. Za to je potrebno dobro poznavanje prirode problema.

3.1.1 Skica algoritma u pseudo-kodu

Sada smo u mogućnosti dati općenitu skicu algoritma TP-a uvažavajući sve dosada rečeno. Bez smanjenja općenitosti pretpostavimo da želimo minimizirati funkciju $f(x)$ na nekoj domeni. Koritistiti ćemo verziju TP-a koja pri svakoj iteraciji bira najbolji dozvoljeni potez (ovo je najčešći slučaj).

***Oznake*:**

- ***x***, trenutno rješenje
- ***x****, nabolje pronađeno rješenje
- ***f****, vrijednost od $f(x^*)$
- ***N(x)***, susjedstvo od x
- ***TL,*** tabu lista
- ***TV,*** tabu vrijeme

***Algoritam TS*:**

Inicijalizacija:

odaberi (konstruiraj) početno rješenje x_0;

postavi: $x := x_0$, $x^* := x_0$, $f^* := f(x_0)$, $TL := \emptyset$;

Pretraga:

```
dok (uvjet_zaustavljanja nije zadovoljen) radi
{
    odaberi najbolje dozvoljeno rješenje x' iz N(x);
    postavi x := x';
    ako je f(x) < f*, onda postavi f* := f(x),  x* := x;
    stavi x u TL na TV iteracija;
}
vrati rješenje x*;
```

3.1.2 Uvjet zaustavljanja

Ako nije poznata optimalna vrijednost funkcije koja se minimizira, pretraga bi teoretski mogla trajati beskonačno. U praksi, očito pretraga se mora zaustaviti u neko vrijeme. Najčešći uvjeti zaustavljanja su:

- nakon fiksnog broja iteracija (ili fiksnog vremena procesora),
- nakon određenog broja iteracija bez napretka u minimiziranju funkcije,
- nakon što funkcija dosegne neku unaprijed definiranu vrijednost.

Kao uvjet zaustavljanja mogu se koristiti i sve kombinacije gore navedenih uvjeta.

3.2 Naprednije strategije

3.2.1 Intenzifikacija

Osnovna ideja intenzifikacije jest da, kako bi inteligentno biće vjerojatno postupilo, treba detaljnije istražiti dijelove prostora pretrage koji se čine "plodnijim" kako bi bili sigurni da su najbolja rješenja u tim područjima pronađena. Isto tako, ona se bazira na forsiranju poteza i atributa rješenja koji su se "povijesno" pokazali dobrima [12]. S vremena na vrijeme, može se zaustaviti običnu pretragu i provesti fazu intenzifikacije.

Generalno gledajući, intenzifikacija je temeljena na nekoj memorijskoj strukturi, poput tabu liste. Tako primjerice u toj strukturi možemo pratiti koliko dugo su se neki atributi rješenja održali u trenutnom rješenju bez prekida. Kod bojanja grafova, možemo promatrati koliko je iteracija boja pojedinog vrha u trenutnom rješenju ostala nepromijenjena. Tipičan pristup intenzifikacije bi onda bio da započnemo pretragu ispočetka od najboljeg pronađenog rješenja fiksirajući neke parove vrhova i boja koji se čine "najpovoljnijima".

Jedna drugačija strategija intenzifikacije bila bi da tijekom pretrage pamtimo "najbolja" rješenja i da se onda u fazi intenzifikacije vratimo na njih, ovaj puta s izmijenjenom relacijom susjedstva, omogućavajući "moćnije" ili raznovrsnije poteze. Kod problema bojanja grafova mogli bi tako dopustiti složenije poteze od mijenjanja boje jednog jedinog vrha, primjerice korištenjem takozvane "*ejection chain*" metode.

Intenzifikacija se koristi u mnogim TP implementacijama, ali nije nužna. Razlog tome je što je u većini slučajeva obična pretraga jednostavno dovoljno detaljna. Zato nema potrebe trošiti vrijeme istražujući detaljnije dijelove prostora koji su već bili posjećeni.

3.2.2 Diverzifikacija

Jedan od glavnih problema svih metoda baziranih na lokalnoj pretrazi, a to uključuje i TP (usprkos povoljnom utjecaju tabu liste), jest da teže biti previše "lokalne" kako samo ime kaže. To znači da provode većinu, ako ne i čitavo vrijeme pretrage na ograničenom dijelu prostora pretrage [10]. Negativna posljedica ove činjenice jest da, iako dobra rješenja mogu biti pronađena, može se dogoditi da "najzanimljivija" područja prostora pretrage ostanu neistražena i da tako ostanemo na rješenjima koja su još uvijek prilično daleko od najboljih.

Diverzifikacija je algoritamski mehanizam koji pokušava umanjiti ovaj problem na način da "tjera" pretragu u do tada neistražena područja prostora pretrage. Obično je bazirana na nekoj vrsti dugoročne memorije o tijeku pretrage. Tako se primjerice pamti ukupan broj iteracija (od početka pretrage) u kojima su neki atributi rješenja bili zastupljeni. Kod bojanja grafova možemo tako pamtiti koliko je ukupno iteracija neki vrh bio određene boje.

Postoje dvije glavne strategije diverzifikacije. Prva fiksira nekoliko od početka pretrage rijetko korištenih atributa rješenja i započinje novu pretragu. Druga metoda integrira diverzifikaciju direktno u običnu pretragu tako da pri ocjeni mogućih poteza uzme u obzir frekvenciju (učestalost) pojedinih atributa rješenja.

Za kraj treba naglasiti da je osiguravanje "dobre" diverzifikacije možda jedan od najkritičnijih koraka u dizajnu algoritma TP-a. Treba mu pristupiti s pažnjom odmah rano u razvoju i, ako rezultati nisu u skladu s očekivanim, revidirati ono što je napravljeno.

4. Hibridni evolucijski algoritam

4.1 Evolucija genetskog algoritma

Genetski algoritam (GA) je heuristička globalna optimizacijska metoda. Ona pripada klasi evolucijskih algoritama koji imitiraju evoluciju u prirodi koristeći pritom razne operatore poput selekcije, križanja i mutacije nad populacijom rješenja [23].

Standardni genetski algoritam u početku je definiran kao općenita metoda temeljena na "slijepim" genetskim operatorima (križanje, mutacija, ...) koja rješava bilo koji problem čiji je prostor pretrage kodiran u skup nizova bitova. Ovakvim pristupom dobivamo robustnu metodu koja radi razmjerno dobro na širokom skupu optimizacijskih problema ali gotovo nikad ne postiže najbolje rezultate za neki konkretan problem [18]. Tu se ukazuje praktična potreba da se GA pažljivo prilagodi svakom pojedinom rješavanom problemu. Ovo je takozvani *ortogonalni* pristup, gdje je cilj proizvesti najbolji optimizacijski algoritam za dani problem, a ne robustan algoritam za općenitu upotrebu. Ovakav pristup nadalje implicira što veću integraciju specifičnog znanja o domeni problema u sam genetski algoritam, odnosno u njegove genetske operatore i zapis rješenja.

Iskustveno se došlo do još jednog bitnog zaključka u vezi učinkovitosti GA. Naime, GA je prilično učinkovit u pronalaženju globalno dobrih rješenja, ali je neučinkovit u pronalaženju onih nekoliko koraka - mutacija do samog globalnog optimuma. Imajući ovo na umu pokazuje se dobrom praksom kombinirati GA s drugim optimizacijskim metodama koje su bolje u tom segmentu, naime metodama lokalne pretrage. Metode lokalne pretrage općenito su vrlo učinkovite u pronalaženju dobrih rješenja u nekom ograničenom području prostora pretrage. Ovakavim pristupom dolazimo do koncepta takozvanog hibridnog evolucijskog algoritma (HEA).

Primjetimo da smo do istog koncepta mogli stići i drugim putem, promatrajući nedostatke lokalne pretrage. Naime vrlo često je prostor pretrage u rješavanom problemu ogroman, a lokalna pretraga je u pravilu ograničena na neki dio tog prostora. Dakle moramo osigurati neku strategiju diverzifikacije, odnosno mogućnost većih skokova u prostoru pretrage na neistražena područja. Kako bismo to ostvarili prirodno nam se nameće korištenje neke vrste centralne memorije. U našem slučaju populacija rješenja ima ulogu centralne memorije i zajedno s operatorima križanja i selekcije implementira strategiju diverzifikacije u proizvoljnoj metodi lokalne pretrage. Sada proces pretrage možemo zamisliti tako da svako križanje u populaciji predstavimo kao skok na novo rješenje u prostoru pretrage, a lokalnu pretragu kao traženje lokalnog optimuma u okolini tog rješenja.

4.2 Specijalizirani operator križanja

Prethodno smo ustanovili da dizajn operatora križanja zahtijeva identifikaciju svojstava koja su značajna za rješavani problem i razvoj rekombinacijskog mehanizma koji prenosi ta svojstva sa roditelja na potomke. Što se tiče problema bojanja grafova, u osnovi postoje dva različita pristupa ovisno o tome da li se pod bojanjem grafa podrazumijeva dodjela boja vrhovima grafa ili particija skupa vrhova na klase boja. Razlike između ova dva pristupa pregledno su prikazane u tablici 4.1.

Tablica 4.1 Razlike pristupa dodjele i pristupa particije

	Pristup dodjele	*Pristup particije*
Rješenje	dodjela boja vrhovima grafa $c : V \rightarrow \{1, \ldots, k\}$	particija skupa vrhova grafa $x = \{V_1, \ldots, V_k\}$
Elementarna karakteristika	par vrh-boja (v, i): boja i dodijeljena vrhu v	neprazan skup vrhova $\{v_1, \ldots, v_q\}$ koji pripadaju istom V_i
Operator križanja	križanje parova $c(v) := c_1(v)$ ili $c_2(v)$	križanje particija

Pristup dodjele gleda na k-bojanje kao na dodjelu boja vrhovima grafa, odnosno preslikavanje $c : V \rightarrow \{1,\ldots,k\}$ iz skupa vrhova V u skup od k boja. Sada je prirodno prikazati rješenje x vektorom $(c(v_1),\ldots,c(v_n))$ duljine $n = |V|$, gdje je svaki $c(v_i)$ predstavlja boju pridruženu vrhu v_i. Tada je moguće definirati uniformno križanje parova na sljedeći način: za dana dva roditelja $x_1 = (c_1(v_1),\ldots,c_1(v_n))$ te $x_2 = (c_2(v_1),\ldots,c_2(v_n))$ izgradi se potomak $x = (c(v_1),\ldots,c(v_n))$ tako da postavi $c(v_i) := c_1(v_i)$ ili $c(v_i) := c_2(v_i)$ za svaki vrh v_i s jednakom vjerojatnošću od 0.5. Ovakav operator križanja može se poboljšati na sljedeći način: ako je vrh v konfliktan vrh u točno jednom od roditelja onda on automatski nasljeđuje boju od roditelja u kojem nije konfliktan. Kod pristupa dodjele možemo opaziti da je svojstvo koje se prenosi križanjem zapravo par (vrh, boja): odnosno pojedini vrh poprima pojedinu boju. Takav par, ako se uzme sam za sebe, nema nikakvo značenje u kontekstu bojanja grafova jer sve boje imaju istu ravnopravnu ulogu. To proizlazi iz činjenice da permutiranjem skupa boja u nekom k-bojanju ne dobivamo novo rješenje već samo jedno od $k!$ različito zapisanih a u osnovi istih rješenja.

Za problem bojanja grafova biti će prikladnije i značajnije razmatrati neprazan skup vrhova koji sačinjavanju neku klasu boje. Ovo vodi drugom pristupu kojeg nazivamo "pristup particije". Ovim pristupom rješenjem se smatra particija skupa vrhova na klase boja te operator križanja ima zadaću prenijeti klase boja ili podskupove klasa boja na potomke. Sada je na temelju ovog općenitog principa moguće zamisliti više različitih operatora križanja. Jedna od mogućnosti je da se od podskupova klasa boja dvaju roditelja izgradi parcijalno rješenje maksimalne veličine i da se zatim nadopuni do potpunog rješenja. Preciznije, za dana dva roditelja $x_1 = \{V_1^1,\ldots,V_k^1\}$ i $x_2 = \{V_1^2,\ldots,V_k^2\}$ parcijalno rješenje će biti skup $\{V_1,\ldots,V_k\}$ disjunktnih skupova vrhova sa sljedećim svojstvima:

- svaki podskup V_i sadržan je u klasi boje jednog od dva roditelja, odnosno vrijedi: $\forall i\ (1 \leq i \leq k)\ \exists j : V_i \subseteq V_j^1\ ili\ V_i \subseteq V_j^2$, što opet povlači da su svi V_i *nezavisni* skupovi;
- unija svih V_i je maksimalne veličine, odnosno kardinalnost od $|\bigcup_{1 \leq i \leq k} V_i|$ je maksimalna;
- utjecaj oba roditelja je podjednak jer je otprilike polovica skupova V_i sadržana u klasi prvog roditelja dok je druga polovica sadržana u klasi drugog roditelja.

Jedan od načina da se konstruira parcijalno rješenje jest da se svaki V_i gradi sukcesivno na pohlepan način: roditelji se razmatraju naizmjence jedan za drugim te u razmatranom roditelju biramo klasu boje s najvećim brojem (nedodjeljenih) vrhova kao sljedeći skup V_i u djetetu. Primjetimo da ovakav operator križanja, kojeg ćemo zvati Pohlepno Križanje Particija (PKP), svaki puta generira jedno dijete.

Galinier i Hao proveli su eksperimente koji su potvrdili relevantnost informacija koje prenose "particijski" operatori križanja na svoje potomke. Oni su se u svome eksperimentu ograničili na parove vrhova i pokazali sljedeće. Ako promatramo skup različitih *k*-bojanja nekog grafa i pogledamo učestalost pojave da su određena dva nesusjedna vrha svrstana u istu klasu boje, možemo zaključiti da se neki nesusjedni parovi vrhova puno češće svrstavaju u istu klasu boje nego drugi [9]. Ova činjenica upućuje na pomisao da se baš klase boja nalaze u samoj srži problema bojanja grafova.

4.3 Prostor pretrage i funkcija cilja

Kako bismo rješili problem *k*-bojanja grafa, razmatramo skup svih mogućih particija skupa *V*(*G*) na *k* nepraznih podskupova. Primjetimo da ovdje ubrajamo i particije koje nisu legalne *k*-diobe. Ovime smo definirali naš prostor pretrage kojeg označavamo sa *S*. Kako bismo odredili njegovu veličinu u ovisnosti o broju vrhova *n* te broju boja *k*, primjetimo da se to zapravo može izračunati kao broj načina na koji možemo podijeliti skup od *n* elemenata u *k* nepraznih disjunktnih podskupova. Ako dopustimo da neki od *k* podskupova mogu biti prazni, lako dobijemo da je $|S| = k^n / k!$. Međutim ako zahtijevamo da je svih *k* skupova neprazno, tada veličinu prostora pretrage karakteriziraju *Stirlingovi brojevi druge vrste* [29]. Oznaka za Stirlingov broj druge vrste je *S*(*n*, *k*) gdje je *n* broj elemenata, a *k* broj podskupova. Eksplicitna formula (4.1) nešto je složenijeg oblika i glasi:

$$S(n,k) = \left\{ {n \atop k} \right\} = \sum_{j=1}^{k} (-1)^{k-j} \frac{j^{n-1}}{(j-1)!\,(k-j)!} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n \qquad (4.1)$$

Kako bismo imali predodžbu kako se Strilingovi brojevi druge vrste ponašaju te koliko brzo rastu, dane su njihove vrijednosti u tablici 4.2 za sasvim malene *n* i *k*. Primjetimo da se najveće vrijednosti postižu kada je *k* blizu *n*/2.

Tablica 4.2 Vrijednosti Stirlingovih brojeva druge vrste

n**k	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	B_n
0	1										1
1	0	1									1
2	0	1	1								2
3	0	1	3	1							5
4	0	1	7	6	1						15
5	0	1	15	25	10	1					52
6	0	1	31	90	65	15	1				203
7	0	1	63	301	350	140	21	1			877
8	0	1	127	966	1701	150	266	28	1		4140
9	0	1	255	3025	7770	6951	2646	462	36	1	21147

Bellov broj B_n predstavlja ukupan broj particija skupa od *n* elemenata i računa se pomoću izraza (4.2):

$$B_n = \sum_{k=0}^{n} S(n,k) \tag{4.2}$$

Pomoću Bellovog broja računamo veličinu prostora pretrage za problem legalnog bojanja grafa od *n* vrhova s *minimalnim* brojem boja, odnosno za problem pronalaženja kromatskog broja grafa *X*(*G*). Prvih nekoliko Bellovih brojeva dano je u zadnjem stupcu tablice 4.2.

Funkcija cilja f vrlo je jednostavna. Ona mjeri ukupan broj konfliktnih bridova. Dakle za rješenje $x = (V_1, ..., V_k)$ iz S, $f(x)$ jednak je $\sum_{i=1}^{k} |E_i|$ gdje $|E_i|$ predstavlja skup bridova kojima oba kraja (vrha) leže u istom podskupu V_i. Cilj algoritma je pronaći *k*-bojanje *x* tako da je $f(x) = 0$ (legalno *k*-bojanje). Sada možemo pregledno zapisati:

- rješenje $x \in S$ je bilo koja particija $x = (V_1, ..., V_k)$ skupa V na k nepraznih podskupova;
- $\forall x \in S$, definiramo $f(x) = |e \in E$: oba kraja od e leže u istom podskupu $V_i \in x|$.

4.4 Opći postupak

Kako bismo rješili problem bojanja grafa, odnosno legalnog bojanja grafa s najmanjim mogućim brojem boja, konkretan pristup sastoji se od sljedećih koraka. Prvo pronađemo legalno k-bojanje za neki (dovoljno velik) broj boja $k = k_0$. Zatim, kada je legalno k-bojanje pronađeno isti se algoritam uzastopno koristi za pronalazak k-bojanja sa sve manjim brojem boja ($k = k_0 - 1$, $k_0 - 2$, ...). Na ovaj način problem bojanja grafa sveden je na uzastopno rješavanje sve težih problema k-bojanja.

U nastavku dan je detaljan opis takozvanog *Hibridnog Algoritma za Bojanje* (HAB) koji rješava problem k-bojanja grafa. Slijedi algoritam u pseudokodu:

***Hibridni Algoritam za Bojanje (HAB)*:**

Ulaz: graf $G = (V, E)$, cijeli broj $k > 0$

Izlaz: najbolje pronađeno rješenje x^*

Početak:

P = *InicijalizirajPopulaciju*($|P|$);

dok (!*UvjetZaustavljanja*()) radi
{
    (x_1, x_2) = *OdaberiRoditelje*(P);

    x = *OperatorKrižanja*(x_1, x_2);

    x = *LokalnaPretraga*(x, L);

    P = *AžurirajPopulaciju*(P, x)

}

Kraj.

U HAB-u, populacija P je skup rješenja fiksne veličine $|P|$. Algoritam prvo gradi inicijalnu populaciju rješenja te zatim izvodi niz iteracija koje zovemo *generacije*. U svakoj generaciji, dva rješenja - roditelja x_1 i x_2 su odabrana iz populacije i operator križanja se primjenjuje kako bi stvorili novo rješenje - dijete x. Nakon toga, primjenjuje se metoda lokalne pretrage s fiksnim brojem iteracija L kako bi se smanjio broj konfliktih bridova u djetetu. Naposlijetku, "poboljšano" dijete x ubacuje se u populaciju gdje zamjenjuje jedno od postojećih rješenja. Cijeli ovaj postupak se ponavlja dok uvjet zaustavljanja nije ispunjen, obično je to kada se dosegne unaprijed određen broj iteracija.

Primjetimo da se ovaj hibridni algoritam dosta razlikuje od običnog genetskog algoritma. Osnovna razlika je dakako korištenje metode lokalne pretrage na mjestu nasumičnog operatora mutacije. Još jedna bitna razlika je u operatoru selekcije koji u običnom GA najčešće potiče preživljavanje i reprodukciju najboljih jedinki dok je u HAB-u selekcija više nasumična, što ćemo poslije pokazati.

4.5 Značajke Hibridnog Algoritma za Bojanje

4.5.1 Operator inicijalizacije

Operator inicijalizacije *InicijalizirajPopulaciju*(|*P*|) inicijalizira populaciju *P* od |*P*| rješenja. Kako bismo stvorili svako pojedino rješenje u populaciji koristimo pohlepni algoritam koji radi na sljedeći način. Započinjemo s *k* praznih klasa boja $V_1 = \ldots = V_k = \emptyset$ i u svakom koraku biramo vrh $v \in V$ takav da *v* ima minimalan broj dopuštenih klasa (klasa koje ne sadrže vrh susjedan vrhu *v*). Kako bismo stavili *v* u neku klasu boje, biramo između dopuštenih onu klasu V_i s najmanjim indeksom *i*. Ovaj postupak općenito ne može uvijek dodijeliti sve vrhove klasama. Svaki od nedodijeljenih vrhova se onda stavlja u nasumce odabranu klasu boje. Kada su svi vrhovi dodijeljeni, rješenje se još "popravlja" metodom lokalne pretrage kroz *L* iteracija.

Zahvaljujući nasumičnosti pohlepnog algoritma i poboljšanja lokalnom pretragom, rješenja u inicijalnoj populaciji su prilično različita jedna od drugog. Ovo je vrlo važno za populacijske algoritme jer suviše homogena populacija ne može učinkovito evoluirati.

4.5.2 Operator selekcije

Operator selekcije ostvaren je zapravo pomoću dvije metode, *OdaberiRoditelje*(*P*) i *AžurirajPopulaciju*(*P*, *x*). Metoda *OdaberiRoditelje*(*P*) iz populacije *P* nasumce bira dva rješenja na koje će se primjeniti operator križanja. Nasumičnim odabirom, umjesto biranjem najboljih jedinki sprječavamo pretjerano brzu konvergenciju populacije, odnosno održavamo raznolikost.

Metoda *AžurirajPopulaciju*(*P*, *x*) ubacuje dijete *x* na mjesto lošijeg od dva roditelja. U slučaju izjednačenih roditelja jedan od njih eliminira se nasumce. Uočimo da je ovakvom jednostavnom logikom najbolja jedinka u populaciji uvijek sačuvana od eliminacije[1]. Ovo svojstvo GA naziva se e*litizam.* Može se pokazati da GA s ugrađenim elitizmom, iz generacije u generaciju, asimptotski teži ka globalnom optimumu [13].

[1] Također, sačuvana je od bilo kakve izmjene, odnosno mutacije.

4.5.3 Operator križanja

OperatorKrižanja(x_1, x_2) koji se koristi zapravo je prethodno spomenuti operator zvan Pohlepno Križanje Particija (PKP). Algoritam koji implementira PKP radi tako da za dva roditelja $x_1 = \{V_1^1, \ldots, V_k^1\}$ i $x_2 = \{V_1^2, \ldots, V_k^2\}$ odabrana operatorom *OdaberiRoditelje*(P), gradi dijete $x = \{V_1, \ldots, V_k\}$ na sljedeći način.

***Operator križanja (PKP)*:**

Ulaz: rješenja $x_1 = \{V_1^1, \ldots, V_k^1\}$ i $x_2 = \{V_1^2, \ldots, V_k^2\}$

Izlaz: rješenje $x = \{V_1, \ldots, V_k\}$

Početak:

za j ($1 \leq j \leq k$)
{

ako (j neparan), onda $A := 1$, inače $A := 2$;

odaberi i takav da je klasa V_i^A maksimalnog kardinaliteta;

$V_j := V_i^A$;

odstrani sve vrhove iz klase V_j iz rješenja x_1 i x_2;

}
Nasumce dodijeli klasama preostale vrhove iz $V - (V_1 \cup \ldots \cup V_k)$;

Kraj.

Algoritam gradi korak po korak k klasa $V_1, \ldots, V_k$ djeteta kako slijedi. Razmatramo roditelja x_1 ($A = 1$) ili x_2 ($A = 2$) ovisno o tome je li j neparan ili paran. U razmatranom roditelju odabiremo klasu koja ima najveći broj vrhova i ta klasa postaje klasa V_j. Zatim odstranimo sve te vrhove iz roditelja x_1 i x_2. Nakon k koraka neki će vrhovi moguće ostati nedodijeljeni klasama u djetetu. Te vrhove nasumično podijelimo po klasama djeteta.

4.5.4 Operator mutacije

Kao operator mutacije koristi se metoda lokalne pretrage - algoritam Tabucol čiji je detaljan opis dan u poglavlju 4.6.

4.5.5 Uvjet zaustavljanja

Uvjet zaustavljanja vrlo je jednostavan. Algoritam se zaustavlja kada je dostignut unaprijed zadani maksimalni (ukupan) broj iteracija *MaxIter* ili ako je pronađeno legalno *k*-bojanje. Treba naglasiti da se broj iteracija u HAB-u odnosi na ukupan broj iteracija lokalne pretrage, konkretno, ukupan broj iteracija Tabucol algoritma. Galinier i Hao u svom radu [9] navode još jedan uvjet zaustavljanja. Taj uvjet zaustavlja algoritam ako je raznolikost populacije pala ispod unaprijed zadanog minimuma. Ovaj uvjet nije implementiran u praktičnom dijelu ovog rada.

4.5.6 Raznolikost populacije

Kod genetskih algoritama poznata je činjenica da raznolikost populacije ima važan utjecaj na učinkovitost algoritma. Brz pad raznolikosti populacije vodi ka preuranjenoj konvergenciji algoritma, vjerojatno nekom lokalnom optimumu. S druge strane, konvergencija je potrebna da bi algoritam imao svoju svrhu. Zato je ključno na dobar način kontrolirati raznolikost populacije tokom pretrage.

Kako bismo mogli pratiti raznolikost, nužno je definirati smislenu mjeru udaljenosti između neke dvije jedinke - rješenja. Kada je riječ o jednostavnim GA i binarnom zapisu rješenja koristi se *Hammingova udaljenost*. Međutim za naš problem ona nema značenja i nije primjenjiva. Zato uvodimo mjeru udaljenosti između dva rješenja na sljedeći način.

Definicija 35. Transformaciju nad rješenjem *x* koja se sastoji od promjene klase boje točno jednog vrha zovemo **elementarna transformacija** ili kraće **1-potez**.

Definicija 36. **Udaljenost** između proizvoljna dva rješenja $d(x_1, x_2)$ definiramo kao minimalan broj potrebnih elementarnih transformacija kojima rješenje x_1 možemo pretvoriti u rješenje x_2.

Očigledno mora vrijediti $d(x_2, x_1) = d(x_1, x_2)$.

Definicija 37. **Raznolikost populacije** definiramo kao prosječnu udaljenost između svih rješenja u populaciji i označavamo velikim slovom *D*.

Možemo je izračunati pomoću izraza (4.3):

$$D = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d(x_i, x_j) \tag{4.3}$$

S oznakom $D(g)$ označavamo raznolikost populacije nakon *g* generacija algoritma.

Primjetimo sada da udaljenost između neka dva rješenja nije baš trivijalno izračunati. Naime, kako bismo pobrojali minimalan broj potrebnih elementarnih transformacija moramo ih točno odrediti. Pogledajmo slučaj kada u dva različita rješenja x_1 i x_2 ista klase boje (identični podskup vrhova od V) ima u prvom rješenju jednu oznaku V_i a u drugom drugačiju oznaku V_j, $i \neq j$. Kako oznake klasa boje ne smiju praviti razliku između dva rješenja, bilo bi pogrešno brojati elementarne transformacije koje bi dotičnu klasu boje iz prvog rješenja pretvorile u odgovarajuću klasu boje iz drugog rješenja. Ako permutiramo oznake klasa boja u nekom rješenju ne dobivamo novo rješenje!

Dakle moramo prvo preimenovati klase boja iz prvog rješenja kako bi što više "odgovarale" klasama boja iz drugog rješenja. Preciznije, moramo pronaći bijekciju $\sigma : \{1 \dots k\} \to \{1 \dots k\}$ tako da je "presjek" $p(x_1, x_2) = \sum_{i=1}^{k} \left| V_i^1 \cap V_{\sigma(i)}^2 \right|$ između dva rješenja maksimalan. Očito, moramo najprije poznavati vrijednost od $\left| V_i^1 \cap V_j^2 \right|$ za sve kombinacije i i j. Ove vrijednosti pogodno je pohraniti u matricu k x k elemenata. Primjetimo odmah da je broj različitih bijekcija jednak broju permutacija k-članog skupa i iznosi $k!$. Naivan pristup sada bio bi izračunati presjek za svaku od $k!$ različitih bijekcija i odabrati onu za koju je on maksimalan. Kako je ovaj pristup faktorijelne složenosti, lako se može pokazati da je već za malo veće vrijednosti od k nemoguće dobiti rezultat u razumnom vremenu[2]. Srećom, ovaj je problem zapravo poznati problem kombinatorne optimizacije i postoji algoritam koji ga rješava u polinomnom vremenu. Naime, radi se o *problemu pridruživanja*, a najpoznatiji algoritam koji ga rješava sa složenošću od $O(k^3)$ je takozvani *Munkresov algoritam*[3]. Ovaj algoritam (zbog svoje kompleksnosti) nije implementiran u praktičnom dijelu ovog rada. Detaljan opis algoritma nešto je duži i može se pronaći u literaturi [21].

[2] Kada bi račun presjeka za svaku od $k!$ bijekcija trajao samo 1 milisekundu, već uz k = 12 imali bismo ukupno 479,001,600 bijekcija i bilo bi potrebno više od 133 sata računanja da ih sve izračunamo.

[3] Poznat je još pod imenima: *Kuhn-Munkresov algoritam* te *mađarski algoritam* [25].

4.6 Algoritam Tabucol

4.6.1 Opis algoritma

Tabucol je implementacija TP-a za problem *k*-bojanja grafova. Tabucol su osmislili i objavili *Hertz* i *de Werra* 1987. godine. Od tada su mnogi autori koristili Tabucol kao potprogram u svojim programima. Svi ti autori zadržali su sve glavne karakteristike izvornog algoritma, ali isto tako uvodili su neke modifikacije kako bi postigli veću učinkovitost. Ovdje ćemo predstaviti jednu modificiranu, poboljšanu verziju Tabucola.

Algoritam iterativno mijenja klasu boje jednog (ne uvijek istog) vrha, a cilj je progresivno smanjivati ukupan broj konfliktnih bridova dok legalno bojanje nije uspostavljeno. Tabu lista se koristi u svrhu "bježanja" iz lokalnih optimuma te kako bi se izbjeglo kratkoročno kruženje (u kratkim ciklusima). Slijedi detaljniji opis algoritma.

Prostor pretrage S za algoritam je skup svih k-bojanja danog grafa G. Rješenje $x \in S$ je particija skupa vrhova $V(G)$ na k klasa boja V_1 ,..., V_k, tako da pritom vrijedi da je boja vrha v, $c(v) = i$ akko je $v \in V_i$. Funkcija cilja f mjeri ukupan broj konfliktnih bridova. Dakle za rješenje $x = (V_1, ..., V_k)$ iz S, $f(x)$ jednak je $\sum_{i=1}^{k} |E_i|$ gdje $|E_i|$ predstavlja skup bridova kojima su oba kraja (vrha) u skupu V_i. Cilj Tabucola je pronaći k-bojanje x tako da je $f(x) = 0$ (legalno k-bojanje).

U daljnjem tekstu ćemo poradi jednostavnosti mjesto klasa boje V_i, pisati naprosto boja i. Elementarnu transformaciju nad rješenjem x definiranu u poglavlju 4.5.6 ovdje ćemo nazivati **1-potez**.

Za vrh v te boju $i \neq c(v)$ s oznakom **(v, i)** označavamo 1-potez koji pridružuje boju i vrhu v, te rješenje koje nastaje ovim potezom označavamo s $x + (v, i)$.

Tada k-bojanje $x' = x + (v, i)$ možemo opisati na sljedeći način:

- $c'(v) = i$
- $c'(w) = c(w)$ za sve $w \in V(G) - \{v\}$

Susjedstvo $N(x)$ rješenja $x \in S$ je definirano kao skup k-bojanja koja se mogu dobiti iz x primjenom jednog 1-poteza. Dakle, $N(x)$ sadrži $|V|(k - 1)$ rješenja. "Dobrota" 1-poteza (v, i) primjenjenog na rješenje x može se izmjeriti kao $\delta(v, i) = f(x) - f(x + (v, i))$. Strogo pozitivna (odnosno negativna) vrijednost od $\delta(v, i)$ upućuje na smanjenje (odnosno povećanje) vrijednosti funkcije cilja. Važnu ulogu u Tabucolu ima broj *konfliktnih vrhova* (vrhova koji su krajevi konfliktnog brida) u trenutnom rješenju. S **$F(x)$** označavamo broj konfliktnih vrhova u rješenju x.

Definicija 38. 1-potez (v, i) koji uključuje konfliktni vrh v zovemo **kritični 1-potez**.

Radi učinkovitosti, Tabucol izvodi samo kritične 1-poteze. Tabu lista u Tabucolu pamti poteze, a ne rješenja. Kada je 1-potez (v, i) primjenjen na rješenje x, u tabu listu zapisuje se par (v, $c(v)$) što znači da je zabranjeno ponovno dodijeliti boju $c(v)$ vrhu v na određen broj iteracija. Trajanje tabu statusa 1-poteza (v, $c(v)$) ovisi o broju konfliktnih vrhova u rješenju x te o dva parametra A i b. Preciznije, kada je 1-potez (v, i) primjenjen na rješenje x tada 1-potez (v, $c(v)$) postaje tabu na točno $\boldsymbol{A + b*F(x)}$ iteracija.

Definicija 39. Za 1-potez (v, i) kažemo da je **potencijalni 1-potez** ako je kritičan i nije tabu, ili ako je $f(x + (v, i)) = 0$ (odnosno $\delta(v, i) = f(x)$).

U prethodnoj definiciji, zadnji uvjet je elementarni uvjet prihvaćanja koji osigurava ne propuštanje legalnog k-bojanja. U svakoj iteraciji, Tabucol izvodi "najbolji" (s najvećom vrijednosti $\delta(v, i)$) potencijalni 1-potez. Ako postoji više "najboljih" 1-poteza (s istom vrijednosti $\delta(v, i)$), tada se radi slučajni odabir. Algoritam se zaustavlja kada je $f(x) = 0$ i vraća rješenje x koje predstavlja legalno k-bojanje. Postoji i drugi uvjet zaustavljanja koji zaustavlja algoritam na temelju ukupnog broja izvedenih iteracija. Mogu se koristiti i još neki uvjeti zaustavljanja, primjerice, ograničenje u ukupnom vremenu izvođenja algoritma ili pak maksimalnom broju iteracija bez poboljšanja najboljeg pronađenog rješenja, itd. Slijedi algoritam Tabucol u pseudo-kodu.

***Algoritam Tabucol*:**

Ulaz: graf $G = (V, E)$, cijeli broj $k > 0$

Parametri: *MaxIter*, A i b

Izlaz: rješenje x^*

Inicijalizacija:

postavi: $x^* := x$, $iter = 0$, $TL := \emptyset$;

Pretraga:

```
dok (f(x) > 0 AND iter <= MaxIter) radi
{
    postavi  iter := iter + 1;
    odaberi najbolji potencijalni 1-potez (v, i);
    stavi 1-potez (v, c(v)) u TL na A + b*F(x) iteracija;
    postavi  x := x + (v, i);
    ako je f(x) < f(x*), onda postavi  x* := x;
}
vrati rješenje x*;
```

4.6.2 Značajke Tabucol algoritma

U sljedećim cjelinama detaljnije ćemo opisati razne značajke (engl. *features*) Tabucola. Dat ćemo logičku podlogu tim značajkama i opisati njihovu važnost.

(1) Samo kritični 1-potezi se razmatraju.

(2) Svi potencijalni 1-potezi su uzeti u obzir (umjesto samo slučajnog uzorka).

(3) Koristi se elementarni uvjet prihvaćanja.

(4) Duljina tabu liste se povećava s brojem konfliktnih vrhova i ovisi o dva parametra *A* i *b.*

(5) Prikladne strukture podataka mogu bitno smanjiti trošak računanja.

4.6.2.1 Kritični 1-potezi

Tabucol razmatra samo kritične 1-poteze. Ova činjenica olakšava vođenje pretrage prema "dobrim" područjima u prostoru pretrage. Može se uočiti da dopuštanjem primjene 1-poteza koji nisu kritični postaje puno teže "pobjeći" iz lokalnog optimuma. Razlog tome jest činjenica da obično postoje vrhovi koji nisu konfliktni i koji isto tako mogu poprimiti novu boju *i* bez stvaranja novog konfliktnog brida (u tom slučaju $\delta(v, i) = 0$). Inzistirajući da u 1-potezu (v, i) vrh v mora biti konfliktni vrh prisiljavamo algoritam da se bavi s onim vrhovma koji su zaslužni za postojanje konflikata. Primjetimo također da vrijednost funkcije cilja opada drastično na samom početku pretrage i da algoritam općenito troši većinu svoga vremena pokušavajući eliminirati posljednje konfliktne bridove. Dakle, ako izuzmemo sam početak pretrage, uvijek postoji ograničen manji broj konfliktnih vrhova i moguća prednost ovoga pristupa je u skraćivanju vremena potrebnog za jednu iteraciju algoritma.

4.6.2.2 Potpuno susjedstvo

Potez koji se izvodi u svakoj iteraciji je "najbolji" potencijalni 1-potez (izjednačeni potezi podvrgavaju se slučajnom odabiru). Ovo nije bio slučaj u originalnoj verziji Tabucola gdje se "najbolji" potez birao u slučajnom uzorku potencijalnih 1-poteza. Veličina uzorka obično je bila manji postotak ukupnog broja potencijalnih 1-poteza. Eksperimentalno je pokazano da korištenje premalog uzorka kompromitira učinkovitost heuristike [8].

4.6.2.3 Uvjet prihvaćanja

Uvjet prihvaćanja korišten u Tabucolu vrlo je jednostavan: tabu status 1-poteza (v, i) je poništen ako taj potez vodi do legalnog bojanja nakon kojeg pretraga može stati.

4.6.2.4 Tabu vrijeme

Ukupan broj kritičnih 1-poteza je proporcionalan ukupnom broju konfliktnih vrhova te stoga može znatno varirati tijekom čitave pretrage. Iz istog razloga tabu vrijeme isto tako dinamično varira. Eksperimentalno je jasno utvrđeno da je konstantno tabu vrijeme općenito neprikladno za upotrebu - samo mali dijelovi prostora pretrage su istraženi ako koristimo premalu vrijednost dok korištenje prevelike vrijednosti djeluje previše restriktivno. Jedan važan aspekt ovdje jest kako odrediti prikladne vrijednosti parametara A i b. Galinier i Hao u svom radu [9] predlažu postaviti $b = 0.6$ te u svakoj iteraciji algoritma slučajno odabrati A iz intevala [0, 9]. Ove vrijednosti čine se dobrima, barem za instance grafova korištene u njihovim eksperimentima. Ipak ove vrijednosti mogle bi biti neprikladne za druge grafove i ova dva parametra bi u idealnom slučaju trebalo prilagoditi svakoj instanci grafa posebno.

4.6.2.5 Strukture podataka

Najveći dio vremena u svakoj iteraciji (a onda i u cijelom algoritmu) troši se na odabir "najboljeg" kritičnog (potencijalnog) 1-poteza. Naivan pristup vršenju ovog odabira bio bi izračunati vrijednost (dobrotu) $\delta(v, i)$ svakog pojedinog kritičnog 1-poteza, te potom izabrati najboljega. Primjetimo da je za k-bojanje $x = (V_1, ..., V_k)$, vrijednost $\delta(v, i)$ jednaka broju vrhova susjednih vrhu v u skupu $V_{c(v)}$ umanjena za broj vrhova susjednih vrhu v u skupu V_i. Ova vrijednost može se izračunati sa složenošću $O(|\Gamma(v)|)$, gdje $\Gamma(v)$ označava skup vrhova susjednih vrhu v. Zaista, može se inicijalno postaviti $\delta(v, i) = 0$ i onda za svaki susjedni vrh $w \in \Gamma(v)$ učiniti sljedeće: uvećati $\delta(v, i)$ za jedan ako je $c(w) = c(v)$ te smanjiti $\delta(v, i)$ za jedan ako je $c(w) = i$. Kako postoji $O(kF(x))$ kritičnih 1-poteza, računska složenost jedne iteracije iznosi otprilike $O(kF(x)*|V(G)|)$.

Postoji i učinkovitiji način računanja koji daje manju računsku složenost jedne iteracije. Neka $\gamma(v, i)$ označava broj vrhova koji su susjedni vrhu v te im je pridružena boja i u trenutnom rješenju x. Pretpostavimo da su ove vrijednosti $\gamma(v, i)$ pohranjene u matrici veličine $|V(G)| \times k$ koju ćemo zvati γ-matrica. Ovu γ-matricu možemo popuniti odgovarajućim vrijednostima na početku pretrage i onda ju ažurirati u svakoj iteraciji pri primjeni 1-poteza na trenutno rješenje x. Dobrota 1-poteza (v, i) sada se može izračunati u konstantnoj složenosti $O(1)$ jer je $\delta(v, i) = \gamma(v, c(v)) - \gamma(v, i)$. Primjetimo također da sada možemo u konstantnoj složenosti odrediti je li vrh v konfliktan vrh jer vrijedi sljedeća tvrdnja: vrh v je konfliktan akko je $\gamma(v, c(v)) > 0$. Pronalaženje najboljeg kritičnog 1-poteza može se sada izvesti u vremenu $O(k*F(x))$ pretraživanjem γ-matrice. Kada radimo prijelaz s rješenja x na rješenje $x + (v, i)$ možemo ažurirati γ-matricu u vremenu $O(|\Gamma(v)|)$. Zaista, za svaki vrh $w \in \Gamma(v)$ dovoljno je uvećati vrijednost $\Gamma(w, i)$ za jedan i umanjiti vrijednost $\Gamma(w, c(v))$ za jedan. Dakle jedna iteracija ima složenost $O(\max\{k*F(x), |\Gamma(v)|\})$. Primjetimo da se složenost nalaženja "najboljeg" kritičnog 1-poteza može još dalje popraviti pohranjivanjem γ-matrice u podatkovnu strukturu hrpe (engl. *heap*). Tabu lista može se pohraniti u matricu veličine $|V| \times k$ koju ćemo označavati s TL, gdje onda $TL(v, i)$ sadrži index iteracije u kojoj će 1-potez (v, i) prestati imati tabu status (vrijedi: 1-potez (v, i) je tabu akko $TL(v, i) \geq iter$). Dakle s ovakvom strukturom podataka moguće je u konstantnom vremenu odrediti je li 1-potez tabu ili ne. Čak štoviše, kada se 1-potez (v, i) primjeni na x, ažuriranje tabu liste TL može se obaviti u konstantnom vremenu jer je dovoljno postaviti $TL(v, i)$ na vrijednost $iter + A + b*F(x)$.

4.6.3 Implementacija Tabucola

Algoritam je implementiran sa svim gore navedenim značajkama. Treba spomenuti da se ne koristi podatkovna struktura hrpe za pohranjivanje *γ*-matrice te da jedna iteracija algoritma ima složenost $O(\max\{kF(x), |\Gamma(v)|\})$. Uvjet zaustavljanja definira se na sljedeći način: algoritam se zaustavlja kada je pronađeno legalno *k*-bojanje ili je broj izvedenih iteracija dostigao unaprijed određeni broj (*MaxIter*). Neki glavni elementi algoritma dani su pregledno u tablici 4.3.

Tablica 4.3 Elementi algoritma

Naziv:	*Oznaka:*	*Vrijednost:*
Prostor pretrage	S	skup svih *k*-bojanja od G
Elementarna transformacija: 1-potez	(v, i)	$c'(v) = i$, $c'(w) = c(w)$ za sve $w \in V(G) - \{v\}$
Susjedstvo rješenja	$N(x)$	skup svih *k*-bojanja oblika $x' = x + (v, i)$
Funkcija cilja	$f(x)$	$\sum_{i=1}^{k} \lvert E_i \rvert$
Dobrota poteza	$\delta(v, i)$	$f(x) - f(x + (v, i))$
Parametar	A	slučajna vrijednost iz intervala [0, 9]
Parametar	b	0,6
Tabu vrijeme	TV	$A + b*F(x)$

5. Rezultati

5.1 DIMACS format

The Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) je znanstvena ustanova nastala suradnjom *Rugters* i *Princeton* sveučilišta te istraživačkih tvrtki *AT&T*, *Bell Labs*, *Telcordia*, i *NEC*. Od 1990. godine DIMACS sponzorira implementacijska natjecanja (engl. *implementation challenges*) kako bi se odredila učinkovitost algoritama za rješavanje problema od interesa. Dosada se održalo devet DIMACS natjecanja od kojih je drugo po redu (1992. godine)[4] obrađivalo NP-teške probleme te među njima i problem bojanja grafova. Jedan od ciljeva DIMACS natjecanja je olakšati ispitivanje i usporedbu algoritama i heuristika [17]. U tu svrhu, za problem bojanja grafova, objavljeni su ispitni primjerci grafova i osmišljen je DIMACS format zapisa grafa u datoteku. Zapravo postoje dvije vrste datoteka, ulazna - u kojoj se definira problem, te izlazna - u kojoj se zapisuje rješenje. Datoteke sadrže ASCII znakove i sastoje se od nekoliko različitih vrsta linija. Svaka linija započinje jednim znakom – oznakom vrste linije, a zaključena je s *end-of-line* znakom. Različita polja u liniji odvojena su barem jednim znakom razmaka.

5.1.1 Format ulazne datoteke

Ulazna datoteka sadrži sve informacije o grafu potrebne za definiranje problema bojanja. U ovom formatu datoteke vrhovi grafa numerirani su prirodnim brojevima od 1 do *n*. Za datoteke se pretpostavlja da su sintaksno ispravne i dosljedne: oznake vrhova su valjane, vrhovi su jedinstveno označeni, točno *m* bridova je definirano itd. Ona se sastoji od sljedećih vrsta linija.

- **Linija komentara**

Linije komentara daju informacije o datoteci razumljive ljudima i zanemarene su pri čitanju od strane programa. One se mogu pojaviti bilo gdje u datoteci. Svaka linija komentara započinje sa znakom malog slova **c**.

```
c Ovo je primjer komentara.
```

- **Linija vrste problema**

Postoji točno jedna linija ove vrste u svakoj ulaznoj datoteci. Ona se mora pojaviti prije bilo kakve linije za opis vrhova ili bridova. Ona je sljedećeg formata.

```
p FORMAT VERTICES EDGES
```

Znak malog slova **p** na početku linije označava da se radi o liniji vrste problema. Polje `FORMAT` u skladu s vrstom problema (bojanje grafova) treba sadržavati riječ "`edge`". Polje `VERTICES` sadrži broj vrhova *n*, dok polje `EDGES` sadrži broj bridova grafa *m*.

[4] Više o tome na: http://dimacs.rutgers.edu/Challenges/

- **Linija brida**

Postoji točno jedna linija brida za svaki brid u grafu. Svaki brid (*v*, *w*) pojavljuje se točno jedanput u ulaznoj datoteci i ne smije se ponavljati u obliku (*w*, *v*).

```
e V W
```

Znak malog slova **e** označava da se radi o liniji brida. Za brid (*v*, *w*) polja `V` i `W` označavaju njegove krajeve (vrhove).

5.1.2 Format izlazne datoteke

Svaki algoritam za rješavanje problema bojanja grafova trebao bi stvoriti izlaznu datoteku. Ovdje ćemo malo odstupiti od DIMACS norme za izlazne datoteke. Razlog tome je što je DIMACS format za izlazne datoteke namijenjen za više različitih vrsta problema (bojanje grafova je jedan od njih), te ne dopušta zapisivanje nekih specifičnih informacija koje su nam od interesa. Dakle, izlazna datoteka modificiranog formata sastoji se od sljedećih vrsta linija.

- **Linija komentara**

Za nju vrijede ista pravila kao i u ulaznoj datoteci. Unutar komentara možemo zapisati neke korisne informacije kao što su: ukupan broj izvedenih iteracija algoritma, ukupno vrijeme izvođenja, parametre algoritma (kod HAB-a $|P|$ i L, kod Tabucola A i b) te broj konfliktnih bridova po završetku algoritma.

- **Linija rješenja**

Postoji točno jedna linija rješenja u svakoj izlaznoj datoteci. Ona se mora pojaviti prije bilo kakve linije za opis vrhova ili bridova. Ona je sljedećeg formata.

```
s TYPE VERTICES SOLUTION
```

Znak malog slova **s** označava da se radi o liniji rješenja. Polje `TYPE` označava tip rješenja sadržanog u datoteci. U ovom slučaju treba sadržavati riječ "`col`" što označava da se radi o problemu bojanja grafova. Polje `VERTICES` sadrži broj vrhova grafa *n*, dok polje `SOLUTION` sadrži broj boja *k* korišten za bojanje grafa.

- **Linija boje vrha**

Postoji točno jedna linija ove vrste za svaki vrh grafa.

```
v V N
```

Znak malog slova **v** označava da se radi o liniji boje vrha. Polje `V` sadrži oznaku vrha, dok polje `N` sadrži odgovarajuću boju.

5.2 Rezultati eksperimenta

U ovom poglavlju prezentirani su rezultati eksperimenta dobiveni našom implementacijom HAB-a na nekim poznatim primjercima grafova. Istovremeno, napravljene su usporedbe tih rezultata s rezultatima naše implementacije Tabucol algoritma. Naposlijetku, navedeni su najbolji[5] poznati rezultati za dane grafove, i napravljena je usporedba s našim rezultatima.

5.2.1 Ispitni primjeri grafova

Korišteni su sljedeći grafovi iz dobro znanog drugog DIMACS implementacijskog natjecanja (*David S. Johnson* & *Michael A. Trick*).[6]

- tri slučajna grafa: *DSJC250.5*, *DSJC500.5* i *DSJC1000.5*.
  Ovi grafovi imaju redom 250, 500 i 1000 vrhova te nepoznat kromatski broj. Dobiveni su spajanjem para vrhova (*v*, *w*) bridom s vjerojatnošću *p* = 0.5 za svaki par. Parovi se ne ponavljaju u obliku (*w*, *v*).
- dva Leightonova grafa: *le450_15c* i *le450_25c*.
  Ovo su strukturirani grafovi od 450 vrhova, konstruirani s poznatim kromatskim brojem (redom 15 i 25).
- dva "ravna" grafa: *flat300_28* i *flat1000_76*.
  Ovo su također strukturirani grafovi od redom 300 i 1000 vrhova, konstruirani s poznatim kromatskim brojem (redom 28 i 76). Konstruirani su tako da imaju *n* vrhova te *k* particija skupa vrhova, a svaka particija ima $\lceil n/k \rceil$ ili $\lfloor n/k \rfloor$ vrhova. Treba napomenuti da se ovdje pod atributom "ravni" grafovi ne misli na *planarne* grafove.

Ovi grafovi su nam interesantni jer su dobro istraženi te predstavljaju dobru referencu za usporedbu rezultata. Štoviše ovi grafovi su teški i predstavljaju pravi izazov algoritmima za bojanje grafova.

5.2.2 Postavke eksperimenta

Za programsko ostvarenje algoritma korišten je razvojni alat *Microsoft Visual Studio 2005* te *Microsoft .NET Framework 2.0*. Kao jezik ostvarenja korišten je *Microsoft Visual C#.*

Ispitivanje za različite primjere grafova odvijalo se na dva različita računala podjednako. Dio ispitivanja na 3 slučajna grafa izvršavao se na stolnom računalu s *2.21 GHz AMD Athlon 64* procesorom te *1.0 GB RAM*. Preostali dio ispitivanja na 4 strukturirana grafa izvršavao se na prijenosnom računalu *HP nx8220* s *2.0 GHz Intel Pentium M* procesorom te *1.0 GB RAM*. Pri tome je na oba računala korišten *MS Windows XP SP2* operativni sustav. Ispitivanje se izvršavalo u pozadini, u dretvi s niskim prioritetom izvršavanja.

[5] Ovdje se misli na rezultate navedene u literaturi [8] i [9].
[6] Dostupno putem: ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/

5.2.2.1 Kriterij ocjene učinkovitosti

Kako bismo ocjenili učinkovitost HAB-a i usporedili ga s drugim algoritmima potrebno je razmotriti nekoliko različitih kriterija. Za proizvoljan graf prvi kriterij korišten je *kvaliteta* najboljeg pronađenog rješenja, odnosno, najmanja vrijednost od *k* za koju algoritam uspijeva pronaći legalno *k*-bojanje. Također zanima nas ocijeniti algoritam za različite vrijednosti od *k* za neki graf. Za dani graf i danu vrijednost od *k* (zadani problem *k*-bojanja) te konkretni uvjet zaustavljanja (u našem slučaju maksimalni dopušteni broj iteracija - *MaxIter*), razmatramo dva kriterija: *robustnost* i *količinu računanja*. Robustnost mjerimo kao omjer broja pokušaja: "(uspješno pokušaja)/(ukupno pokušaja)". Uspješan pokušaj je onaj koji završava pronalaskom legalnog *k*-bojanja dok je neuspješan pokušaj onaj u kojem legalno k-bojanje nije pronađeno već je algoritam zaustavljen na temelju broja iteracija. Naravno, ovaj omjer ovisiti će uvelike o parametrima *k* i *MaxIter* za dani graf. Kako bi se izmjerila količina računanja koristimo (prosječan) broj iteracija potreban za uspješan pokušaj. Također zanimljiv podatak predstavlja nam i (prosječan) broj generacija potreban za uspješan pokušaj. Nije nam toliko bitno samo vrijeme izvođenja jer ono ovisi o konfiguraciji računala na kojem se algoritam izvodi, štoviše ono ovisi i o opterećenju procesora drugim procesima. Objektivna mjera količine računanja bila bi zapravo: "(broj izvedenih iteracija) x (složenost jedne iteracije)", no kako složenost jedne iteracije ne znamo eksplicitno izraziti nećemo ju koristiti. Identični kriteriji mogu se koristiti i za ocjenu Tabucol algoritma tako da je moguće napraviti direktnu usporedbu ova dva algoritma.

5.2.2.2 Parametri

Dva glavna parametra u HAB-u su veličina populacije $|P|$ i duljina (broj iteracija) lokalne pretrage *L* nakon križanja. Parametar $|P|$ iznosi 10 za većinu eksperimenata (samo za lakše instance problema iznosi 5). Ova veličina u razmjerima tipičnih GA predstavlja malenu populaciju. Korištenje veće populacije pogoduje sprječavanju preuranjene konvergencije algoritma. S druge strane, za pohranu veće populacije potrebno je proporcionalno više memorije i što je važnije, veća količina računanja za inicijalizaciju populacije. Prisjetimo se da se inicijalizacija populacije odvija za svakog člana pohlepnim algoritmom nakon kojeg slijedi *L* iteracija lokalne pretrage. Detaljniji opis ovog postupka dan je u poglavlju 4.5.1. Ova dodatna količina računanja za inicijalizaciju veće populacije nije neznatna, pogotovo kada se radi o većim primjercima grafova. Treba još spomenuti da uz način evolucije koji je implementiran u HAB-u veća populacija znatno sporije konvergira. Ako bi se htjelo koristiti znatno veću populaciju, potrebno bi bilo u svakoj generaciji izmjeniti većinu ili sve jedinke u trenutnoj populaciji, a ne samo jednu kako je to trenutno slučaj.

Parametar *L* ima kritičniji utjecaj na učinkovitost algoritma. Pokazati ćemo njegov utjecaj sljedećim jednostavnim primjerom. Uzmemo graf *DSCJ500.5* i fiksiramo $k = 49$. Zatim iskušamo algoritam za različite vrijedosti od *L*: $L = 250; 500; 1000; 2000; 4000; 8000$ uz najveći dopušteni broj iteracija $MaxIter = 3*10^6$. Za svaku vrijednost od *L* pokrenemo algoritam 20 puta, izbrojimo uspješne pokušaje te izračunamo prosječan broj iteracija potreban za uspješan pokušaj. U tablici 5.1 prikazani su dobiveni rezultati. Primjetimo odmah da parametar *L* određuje relativan omjer između broja križanja i broja iteracija lokalne pretrage u nekom ukupnom broju iteracija (*MaxIter*).

Primjećujemo da za suviše malene vrijednosti od *L* (*L* = 250) algoritam uopće ne uspijeva pronaći legalno *k*-bojanje. Kako *L* raste, tako raste i broj uspješnh pokušaja, sve do *L* = 2000, a onda opet pada. Ovo upućuje na zaključak da postoji neka optimalna vrijednost od *L* obzirom na uspješnost, odnosno robustnost algoritma. Pokazuje se da ova vrijednost nije ista za sve instance problema *k*-bojanja, već je valja za svaku instancu posebno odrediti. Naši rezultati ovdje odudaraju od onih koje su dobili Galinier i Hao. Naime, oni su u svom radu proveli sličan eksperiment (samo do *L* = 4000) i dobili da je uspješnost algoritma za *L* = 4000 veća nego za *L* = 2000. Na temelju ovoga zaključili su da uspješnost (samo) raste kako *L* raste. Ovaj zaključak sigurno nije ispravan uz fiksan broj iteracija *MaxIter* za svaki pokušaj. To možemo jednostavno vidjeti ako zamislimo ekstreman slučaj kada je *L* = *MaxIter*, tada se HAB praktički svodi na Tabucol. Naime, Tabucol upoće ne uspijeva pronaći legalno *k*-bojanje ovog grafa za *k* = 49, što ćemo pokazati poslije kod usporedbe Tabucola i HAB-a. Također, treba primjetiti da je uz veći *L* potreban i veći prosječan broj iteracija do pronalaska legalnog *k*-bojanja. To pak znači da povećavanjem parametra *L* činimo algoritam sporijim. Ovo zadnje implicira da ne postoji idealna vrijednost ovog parametra već moramo pronaći kompromis između brzine i uspješnosti algoritma za danu instancu problema. Naime, pokazuje se kasnije da je za rješavanje težih instanci problema (manji *k*) povoljan veći *L*.

Tablica 5.1 Utjecaj duljine lokalne pretrage na učinkovitost HAB-a

Graf	*k*	*(|P|, L)*	*Broj uspješnih pokušaja*	*Prosječan broj iteracija*	*Prosječan broj generacija*
DSCJ500.5	49	(10, 250)	0/20	-	-
		(10, 500)	2/20	950,000	1891
		(10, 1000)	15/20	1,160,000	1148
		(10, 2000)	18/20	1,350,000	667
		(10, 4000)	15/20	1,890,000	462
		(10, 8000)	5/20	2,650,000	321

Osim ova dva parametra, potrebno je podesiti i dva parametra koja se odnose na algoritam Tabucol unutar HAB-a. To su parametri *A* i *b* potrebni za računanje tabu vremena u svakoj iteraciji algoritma. Oni su postavljeni kako Galinier i Hao u svom radu predlažu. Parametar *A* se u svakoj iteraciji algoritma bira slučajno iz intervala [0, 9] dok je vrijednost od *b* konstantno postavljena na 0.6. Ovi parametri ne podešavaju se za svaki graf posebno (najbolji način) nego se biraju na isti način za sve grafove.

5.2.3 Rezultati

5.2.3.1 Usporedba s Tabucol algoritmom

HAB koristi Tabucol algoritam kao jednu od glavnih komponenata i zato je interesantno pogledati kakvi su rezultati HAB-a u usporedbi s Tabucolom. U eksperimentu za svaki od izabranih grafova fiksiramo različite vrijednosti broja boja *k* od većih pa do manjih. Tako dobivamo različite instance problema *k*-bojanja. Na primjer, "najbolje" poznato *k*-bojanje za graf *DSJC500.5* zahtijeva 48 boja i u tom slučaju uzmemo *k* = 52 do 48 (veće vrijednosti daju "prelake" instance problema). Na ovaj način dobivamo redom sve teže instance problema za svaki pojedini graf. Sada, kako bi "riješili" svaku pojedinu instancu problema pružimo Tabucolu i HAB-u više pokušaja, odnosno pokrenemo ih više puta (10, 5 ili 3). Ukupni maksimalni broj iteracija postavljamo većinom na *MaxIter* = $1*10^7$, samo za najteže instance problema dopuštamo više (čak do $8*10^7$). Postoje dvije bitne razlike između Tabucola i HAB-a koje valja istaknuti. Prvo, Tabucol ne koristi populaciju rješenja. Drugo, Tabucol za inicijalizaciju početnog rješenja ne koristi pohlepni algoritam već ga slučajno generira. Tablica 5.2 prikazuje rezultate algoritama na 3 slučajna grafa, dok tablica 5.3 prikazuje rezultate algoritama na 4 strukturirana grafa. U ovim tablicama svaki redak odgovara jednoj instanci problema i sadrži broj boja *k*, robustnost[7], prosječan broj iteracija[8], prosječan broj generacija, parametre (*L*, |*P*|) te prosječno vrijeme izvođenja[9].

Iz tablice 5.2 analiziramo rezultate eksperimenta za 18 različitih instanci problema *k*-bojanja za 3 slučajna grafa. Prvo možemo opaziti da Tabucol i HAB uspijevaju pronaći bojanja s istim najmanjim brojem boja *k* za graf od 250 vrhova. Također, primjećujemo da je za lakše instance problema kod ovog grafa Tabucolu u prosjeku potrebno manje iteracija da pronađe legalno rješenje (*k*-bojanje). Ovo se može objasniti na sljedeći način. HAB potroši jedan dio svojih iteracija na jedinke u populaciji koje neće uroditi legalnim rješenjem, dok Tabucol sve svoje iteracije "korisno" utroši. Ova "prednost" Tabucola vidljiva je uglavnom na instancama problema kod kojih se legalno rješenje prosječno nalazi za manje od 100,000 iteracija. Rijetko je moguća i na težim instancama, vjerojatno onda kada početno slučajno rješenje Tabucola "padne" blizu traženog legalnog rješenja. Primjetimo da HAB-u treba u prosjeku više nego upola manje iteracija za pronalazak legalnog rješenja kada je *k* = 28. Kod težih instanci problema HAB-ova premoć biti će još vidljivija.

Za grafove od 500 i 1000 vrhova, HAB pronalazi bolja rješenja od Tabucola. Točnije, kod grafa *DSCJ500.5* bolji je za 2 boje, dok je kod grafa *DSCJ1000.5* bolji za čak 6 boja. Ovo je velika razlika i izvrstan rezultat, uzevši u obzir da je Tabucol jedan od najuspješnijih algoritama lokalne pretrage za PBG. Sada promotrimo u tablici vrijednosti od *k* za koje oba algoritma uspijevaju pronaći legalno *k*-bojanje. Ako usporedimo prosječan broj iteracija potreban za pronalazak legalnog rješenja, vidimo da je HAB-u potrebno manje iteracija za sve razmatrane instance problema. Također, možemo primjetiti da za dani graf, rješavanje težih instanci problema zahtijeva veći *L*.

---

[7] Izraženu kao omjer "(uspješno pokušaja)/(ukupno pokušaja)".
[8] Prosječan broj iteracija odnosi se samo na uspješne pokušaje.
[9] Prosječan broj generacija i prosječno vrijeme izvođenja odnose se samo na uspješne pokušaje HAB-a.

Tablica 5.2 Rezultati algoritma na slučajnim grafovima

		Tabucol		*HAB*				
Graf	*k*	*Robustnost*	*Prosječan broj iteracija*	*Robustnost*	*Prosječan broj iteracija*	*Prosječan broj generacija*	*Parametri (|P|, L)*	*Prosječno vrijeme izvođenja [s]*
DSJC250.5	28	8/10	4,410,000	9/10	2,030,000	1004	(10, 2000)	37
	29	10/10	64,000	10/10	170,000	159	(10, 1000)	5.6
	30	10/10	11,000	10/10	31,000	112	(10, 250)	2.8
DSJC500.5	48	-	-	8/10	8,170,000	940	(10, 8000)	370
	49	0/10	-	10/10	1,320,000	650	(10, 2000)	95
	50	10/10	1,720,000	10/10	495,000	698	(10, 700)	58
	51	10/10	290,000	10/10	128,000	250	(5, 500)	19
	52	10/10	64,000	10/10	64,000	122	(5, 500)	14
DSJC1000.5	83	-	-	1/3	70,100,000	2911	(10, 24000)	10673
	84	-	-	1/5	32,900,000	1362	(10, 16000)	3865
	85	-	-	3/5	20,900,000	1296	(10, 16000)	4624
	86	-	-	4/5	6,360,000	1125	(10, 5600)	1550
	87	-	-	5/5	2,690,000	951	(10, 2800)	1107
	88	0/5	-	5/5	1,310,000	927	(10, 1400)	579
	89	1/5	8,200,000	5/5	655,000	926	(10, 700)	501
	90	4/5	4,760,000	5/5	433,000	855	(10, 500)	494
	91	5/5	830,000	5/5	287,000	405	(5, 700)	243
	92	5/5	660,000	5/5	256,000	507	(5, 500)	230

Pogledajmo još jedan detalj. Naime, može se opaziti da je kod grafa *DSCJ1000.5* prosječno vrijeme izvođenja duže za $k = 85$ nego za $k = 84$ unatoč činjenici da je za legalno 85-bojanje potreban manji prosječan broj iteracija. Ovu pojavu možemo lako objasniti. Prvo, primjetimo da složenost izvođenja jedne iteracije HAB-a odgovara složenosti izvođenja jedne iteracije Tabucola. Dakle, ona ovisi o broju konfliktnih vrhova $F(x)$ u trenutnom promatranom rješenju.

Taj broj u nekim slučajevima može biti razmjerno visok tijekom većeg dijela pretrage, da bi na kraju brže opadao do nule, odnosno legalnog rješenja. Tada, veći dio pretrage imamo veću složenost jedne iteracije algoritma, a time i duže vrijeme izvođenja. Suprotno tome, broj konfliktnih vrhova može brže opadati u početnom dijelu pretrage (najčešći slučaj) da bi pri kraju polako došao do nule. Tada pak, veći dio pretrage imamo manju složenost jedne iteracije algoritma, a time i kraće vrijeme izvođenja. Osim ovoga, moramo uzeti u obzir i različito opterećenje procesora drugim procesima za vrijeme izvođenja algoritma.

Tablica 5.3 Rezultati algoritma na strukturiranim grafovima

		Tabucol		*HAB*				
Graf	*k*	*Robustnost*	*Prosječan broj iteracija*	*Robustnost*	*Prosječan broj iteracija*	*Prosječan broj generacija*	*Parametri (|P|, L)*	*Prosječno vrijeme izvođenja [s]*
le450_15c	15	0/10	-	10/10	526,000	84	(10, 5600)	16
	16	9/10	1,300,000	10/10	68,000	87	(10, 700)	5.8
	17	10/10	32,000	10/10	66,000	178	(10, 350)	6.4
le450_25c	26	10/10	160,000	10/10	3,400,000	1688	(10, 2000)	73
	27	10/10	13,000	10/10	28,000	61	(10, 400)	3.5
flat300_28	31	8/10	4,200,000	10/10	3,020,000	1499	(10, 2000)	68
	32	10/10	200,000	10/10	127,000	352	(10, 350)	7.4
flat1000_76	83	-	-	4/5	40,600,000	1680	(10, 24000)	5313
	84	-	-	3/5	14,200,000	879	(10, 16000)	1939
	85	-	-	5/5	5,980,000	737	(10, 8000)	951
	86	-	-	5/5	3,060,000	754	(10, 4000)	631
	87	0/5	-	5/5	1,150,000	567	(10, 2000)	354
	88	1/5	8,180,000	5/5	965,000	473	(10, 2000)	372

Iz tablice 5.3 analiziramo rezultate eksperimenta za 14 različitih instanci problema *k*-bojanja za 4 strukturirana grafa. Možemo opaziti slične odnose u rezultatima kao kod bojanja 3 slučajna grafa. Primjetimo međutim malo veće odstupanje kod grafa *le450_25c*, gdje Tabucol uspijeva pronaći rješenja iste kvalitete uz znatno manji prosječan broj iteracija. U većini ostalih instanci problema HAB pronalazi bolja rješenja i brži je od Tabucola.

Sve u svemu, možemo zaključiti da je HAB znatno uspješniji od Tabucola u bojanju svih predstavljenih grafova, posebice većih primjeraka. Ovo pak upućuje na zaključak da HAB križanjem uspijeva dobiti nove, potencijalno dobre konfiguracije u prostoru rješenja do kojih sam Tabucol ne uspijeva doći.

5.2.3.2 Usporedba s najboljim rezultatima

Sada ćemo usporediti rezultate dobivene HAB-om s najboljim znanim rezultatima iz literature. Zanima nas prvenstveno kriterij kvalitete najboljeg pronađenog rješenja, odnosno najmanja vrijednost od *k* za koju je algoritam uspio pronaći legalno *k*-bojanje. Tablica 5.4 prikazuje komparativne rezultate za sedam odabranih grafova. Svaki redak u tablici odgovara jednom grafu. U drugom stupcu tablice, ako je poznat, naveden je kromatski broj grafa *X(*G*)*. U trećem stupcu navedena je najmanja vrijednost od *k* za koju je pronađeno legalno *k*-bojanje *bilo kojim* algoritmom. U četvrtom stupcu navedeni su rezultati Tabucol algoritma, dok su u petom stupcu navedeni rezultati HAB-a. Naposljetku, u šestom i sedmom stupcu navedene su razlike između HAB-a i najboljih rezultata općenito, te rezultata Tabucol algoritma. Iz tablice možemo vidjeti da HAB uspijeva izjednačiti najbolji postignuti rezultat za sve dane grafove, osim za graf *le450_25c* gdje je slabiji za jednu boju.

Većina od ponajboljih poznatih rezultata koji nisu postignuti algoritmom poput HAB-a, postignuti su populacijskim algoritmom s lokalnom pretragom (koja koristi dvije različite relacije susjedstva) i strategijom uzastopne gradnje nezavisnih skupova [19]. Strategija uzastopne gradnje nezavisnih skupova, jedno vrijeme, bila je jedina učinkovita strategija za slučajne grafove s većim brojem vrhova (više od 300). Danas ona to više nije, jer su najbolji rezultati većinom postignuti hibridnim evolucijskim algoritmima poput HAB-a.

Tablica 5.4 Usporedba s najboljim znanim rezultatima

Graf	*X(G)*	*Najbolji*	*Tabucol*	*HAB*	*Razlika*[10]	*Razlika*[11]
DSJC250.5	-	28	28	28	0	0
DSJC500.5	-	48	50	48	0	-2
DSJC1000.5	-	83	89	83	0	-6
le450_15c	15	15	16	15	0	-1
le450_25c	25	25	26	26	1	0
flat300_28	28	31	31	31	0	0
flat1000_76	76	83	88	83	0	-5

[10] Razlika između naših rezultata HAB-a i najboljih rezultata postignutih *bilo kojim* algoritmom.
[11] Razlika između naših rezultata HAB-a i naših rezultata Tabucol algoritma.

6. Diskusija

6.1 Kontrola raznolikosti populacije

Već smo prije spomenuli da je jedna od najkritičnijih stavki za učinkovitost svakog evolucijskog algoritma raznolikost populacije. U praktičnom dijelu ovog rada (zbog svoje kompleksnosti) nije implementirana metoda za praćenje raznolikosti, ali je u poglavlju 4.5.6 predstavljena logika te način njenog računanja. Dakle, kako bi smo mogli bolje procijeniti učinkovitost našeg algoritma potrebno bi bilo prvo implementirati dotičnu metodu. Ovdje ćemo sada iznijeti jedan zaključak na temelju rezultata eksperimenta poznatih nam iz literature [9]. Radi se naime o utjecaju parametra *L*, odnosno duljine lokalne pretrage neposredno nakon križanja, na raznolikost populacije i samim time na učinkovitost algoritma. Već smo spomenuli da parametar *L* određuje relativan omjer između broja križanja i broja iteracija lokalne pretrage u algoritmu. Naime, pokazuje se da se s većom vrijednosti od *L* bolje čuva raznolikost populacije tijekom pretrage. Intuitivno, dva su razloga ovome. Prvi i manje važan razlog je očito manji ukupan broj križanja koji se uspije izvesti unutar maksimalnog broja iteracija algoritma. Drugi razlog leži u pretpostavci da se dužom lokalnom pretragom nakon križanja dobiva dijete koje je više udaljeno od svojih roditelja u prostoru pretrage. Ovu pretpostavku valjalo bi eksperimentalno ispitati.

Dakako, postoje drugi načini održavanja raznolikosti populacije i ovdje ćemo ih navesti nekoliko. Prvi je upravo taj da se unutar lokalne pretrage ugradi mehanizam koji bi sprječavao da dijete nakon njenog učinka postane previše slično jednom od roditelja. Drugi, očit, način bio bi u dizajnu operatora križanja koji bi svaki puta stvarao dijete koje je dovoljno udaljeno od svojih roditelja. Treći način bio bi da se jednostavno upotrijebi veća populacija rješenja. Bitno je ovdje opet istaknuti da nije cilj održavati konstantnu raznolikost populacije jer time bi se zaustavila potrebna konvergencija algortima, cilj je samo spriječiti prebrzu konvergenciju.

6.2 Novi operatori križanja

Ključ uspjeha HAB-a je u njegovom specijaliziranom operatoru križanja zvanom PKP. No ovaj operator tek je jedan iz šire klase operatora križanja. Ta novu klasu operatora križanja možemo nazvati klasom *particijskih* operatora križanja. Predstaviti ćemo ovdje tek nekoliko ideja za nove operatore križanja. Sve ideje počivaju na početnom preimenovanju klasa boja jednog roditelja kako bi dva roditelja bila što sličnija, upravo na način kako je to opisano u poglavlju 4.5.6 o raznolikosti populacije. Dakle imamo funkciju preimenovanja klasa boja jednog roditelja $\sigma : \{1 \dots k\} \to \{1 \dots k\}$ koja postiže da je suma $\sum_{i=1}^{k} \left| V_i^1 \cap V_{\sigma(i)}^2 \right|$ maksimalna. Kada je preimenovanje obavljeno možemo zamisliti više načina rekombinacije klasa boja i stvaranja novoga potomka. Prvi je jednostavno uniformno križanje. Postiže se tako da se svakom pojedinom vrhu u potomku dodijeli klasa boje iz jednog od dva roditelja s istom vjerojatnošću od 0.5. Primjetimo da, kako smo prethodno preimenovali klase roditelja da budu što sličniji, većina vrhova zapravo može pripasti samo jednoj te istoj klasi boje. Drugi način stvaranja potomka je da svaka klasa boje V_i u potomku poprimi presjek odgovarajućih klasa iz oba roditelja $V_i^1 \cap V_{\sigma(i)}^2$ te da se preostali vrhovi onda pridjele klasama nekim pohlepnim algoritmom ili možda nasumično. Treći način bio bi da se sukcesivno izvršavaju sljedeća tri koraka za izgradnju svake klase boje V_i u potomku. U prvom koraku pronađemo j takav da je presjek $\left| V_j^1 \cap V_{\sigma(j)}^2 \right|$ maksimalan, zatim u drugom koraku izgradimo V_i od unije $V_j^1 \cup V_{\sigma(j)}^2$, te naposlijetku, u trećem koraku odstranimo vrhove klase V_i iz odgovarajućih klasa boja u oba roditelja.

Ovo je tek nekoliko ideja za nove operatore i sigurno ih se može osmisliti još, samo bi pri osmišljavanju trebalo imati na umu što se njima želi postići.

6.3 Evolucija populacije

Evolucija populacije u HAB-u je prilično je jednostavna i možda bi bilo zanimljivo iskušati neke drugačije pristupe. Za početak moglo bi se pokušati eksperimentirati s većom populacijom rješenja. Nadalje, što se tiče selekcije u populaciji, u svakoj generaciji bira se samo dva roditelja koji daju jedno dijete. Nakon toga, to dijete zamijeni slabijeg od dva roditelja. Na ovaj način imamo samo jednog novog člana populacije u svakoj generaciji. Bilo bi zanimljivo to promijeniti. Naime, moglo bi se pokušati s reprodukcijom više parova jedinki koji bi opet proizveli više novih jedinki - djece. Ovdje se valja pobrinuti samo da je najbolja jedinka u populaciji uvijek zaštićena od eliminacije (jer želimo svojstvo elitizma). Sada bi se moralo učiniti nešto kraćom duljinu lokalne pretrage L, jer u protivnom bi pretraga mogla postati previše spora.

Još jedna ideja bila bi varirati neke parametre algoritma dinamički za vrijeme pretrage, ovisno o toku pretrage. Ovdje se misli prvenstveno na duljinu lokalne pretrage L, te parametre A i b za određivanje tabu vremena. Primjerice, mogli bismo tako parametar L postaviti na manju vrijednost u samom početku pretrage te ga postepeno povećavati kako broj konfliktnih vrhova u najboljem pronađenom rješenju opada. Na taj način bi nakon početnog grubog pretraživanja prostora pretrage za plodnim područijma sve finije pretraživali upravo te dijelove. Ovime bi zapravo postigli jedan oblik intenzifikacije pretrage.

7. Zaključak

U ovom radu predstavljen je hibridni evolucijski algoritam (HEA) kao nova uspješna heuristika za rješavanje problema bojanja grafova. On se temelji na evolucijskom algoritmu s populacijom rješenja i dvije ključne komponente. Prva komponenta je nova klasa operatora križanja, specijalno prilagođena problemu bojanja grafova. Operatori križanja iz te klase imaju svojstvo da promatraju bojanje grafa kao particiju grafa na nezavisne skupove odnosno klase boja. Oni prenose podskupove tih klasa s roditelja na potomke. Druga ključna komponenta je zamjena nasumičnog operatora mutacije s moćnom metodom lokalne pretrage, u ovom slučaju tabu pretragom.

Ovakav algoritam implementiran je i ispitan na skupu DIMACS ispitnih primjeraka grafova. Rezultati eksperimenta pokazuju da je HEA znatno uspješniji u rješavanju problema bojanja grafova od algoritma tabu pretrage koji je njegova ključna komponenta. Ovo upućuje na važnost operacije križanja u procesu pretrage koja očigledno služi kao sredstvo diverzifikacije. Nadalje, naš HEA uspijeva se izjednačiti s najboljim poznatim rezultatima iz literature za ove ispitne primjerke grafova. Ako se uzme u obzir činjenica da su najbolji poznati rezultati također postignuti hibridnim evolucijskim algoritmima, dolazimo do zaključka da su upravo oni danas najmoćnija heuristika za rješavanje ovog problema. Ovo pak potiče interes za primjenu HEA na druge teške probleme kombinatorne optimizacije.

Ideje za daljnje istraživanje i razvoj na ovom području leže u pronalasku učinkovite metode upravljanja s raznolikosti populacije, dizajnu novih specijaliziranih operatora križanja te novim načinima vođenja populacije rješenja.

8. Literatura

[1] Chams, M., de Werra, D., & Hertz, A. (1987). Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research, vol. 32* , 260–266.

[2] Costa, D., Hertz, A., & Dubuis, O. (1995). Embedding of a sequential procedure within an evolutionary algorithm for coloring problem in graphs. *Journal of Heuristics, vol. 1, no. 1* , 105–128.

[3] Davis, L. (1991). Order-based genetic algorithms and the graph coloring problem. *Handbook of genetic algorithms* , 72-90.

[4] de Werra, D. (1985). An introduction to timetabling. *European Journal of Operations Research* , 19:151-162.

[5] DIMACS. (1992). *Clique and Coloring Problems, A Brief Introduction, with Project Ideas.*

[6] Dorne, R., & Hao, J. K. (1998). A new genetic local search algorithm for graph coloring. *Lecture notes in computer science, vol. 1498* , 745–754.

[7] Fleurent, C., & Ferland, J. A. (1996). Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research* , 63:437–61.

[8] Galinier, P., & Hertz, A. (2006). A survey of local search methods for graph coloring. *Computers & Operations Research 33* , 2547–2562.

[9] Galinier, P., & Kao-Hao, J. (1999). Hybrid Evolutionary Algorithms for Graph Coloring. *Journal of Combinatorial Optimization 3* , 379–397.

[10] Gendreau, M. (2002). *An Introduction To Tabu Search.* Dohvaćeno iz University of Oslo - Faculty of Mathematics and Natural Sciences - Department of Informatics: http://www.ifi.uio.no/infheur/Bakgrunn/Intro_to_TS_Gendreau.htm

[11] Glass, C., & Prügel-Bennett, A. (2003). Genetic algorithm for graph colouring: exploration of Galinier and Hao's algorithm. *Journal of Combinatorial Optimization* , 7:229–236.

[12] Glover, F., & Laguna, M. (1997). *Tabu Search.* Boston: Kluwer Academic Publishers.

[13] Golub, M. (2004). Genetski algoritam, Prvi dio. Zagreb.

[14] Hertz, A., & de Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing, vol. 39* , 39:345–51.

[15] Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (1991). Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations Research, vol. 39, no. 3* , 378–406.

[16] Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Reasearch of the National Bureau of Standards* , 84:489-506.

[17] Massey, B. (2001). *DIMACS Graph Format.* Dohvaćeno iz Rolland Balzon Philippe - Works & Research: http://prolland.free.fr/works/research/dsat/dimacs.html

[18] Mitchell, M. (1991). *Review of L. D. Davis, Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold.

[19] Morgenstern, C. (1996). Distributed Coloration Neighborhood Search. *Proceedings of the 2nd DIMACS Implementation Challenge* , 335–358.

[20] Pavčević, M.-O. (2006). *Uvod u teoriju grafova.* Zagreb: Element.

[21] Pilgrim, R. A. (2008). *Munkres' Assignment Algorithm.* Dohvaćeno iz Murray State University: http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html

[22] Waters, R. J. (2005). Doctor's thesis: "Graph Colouring and Frequency Assignment". University of Nottingham.

[23] Wikipedia. (2008). *Genetic algorithm*. Dohvaćeno iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Genetic_algorithm

[24] Wikipedia. (2008). *Graph coloring*. Dohvaćeno iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Graph_coloring

[25] Wikipedia. (2008). *Hungarian algorithm*. Dohvaćeno iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Hungarian_algorithm

[26] Wikipedia. (2008). *Local search (optimization)*. Dohvaćeno iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Local_search_(optimization)

[27] Wikipedia. (2008). *Mathematics of Sudoku*. Dohvaćeno iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Mathematics_of_Sudoku

[28] Wikipedia. (2008). *P = NP problem.* Dohvaćeno iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/P_vs_NP

[29] Wikipedia. (2008). *Stirling numbers of the second kind*. Dohvaćeno iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Stirling_numbers_of_the_second_kind

[30] Wikipedia. (2008). *Tabu search*. Dohvaćeno iz Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/Tabu_search