Meta-heuristički algoritmi za određivanje kromatskog broja grafa

Bundalo, Lea

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:215985

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-16**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Lea Bundalo

META-HEURISTIČKI ALGORITMI ZA ODREĐIVANJE KROMATSKOG BROJA GRAFA

Diplomski rad

Voditelj rada:
doc. dr. sc. Goranka Nogo

Zagreb, 2021.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

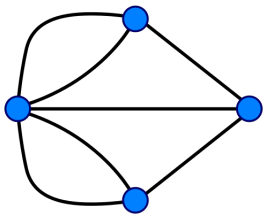
Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Grafovi su važni matematički objekti koji omogućuju jednostavni vizualni prikaz kompleksnih struktura. Početak teorije grafova seže do 1735. godine kada je Leonhard Euler objavio rješenje problema šetnje königsberškim mostovima. Naime, pitanje je mogu li građani Königsberga prošetati preko svih sedam mostova u gradu tako da preko svakog prijeđu točno jednom, a šetnju završe na početnom položaju. Graf koji prikazuje opisanu situaciju prikazan je na slici 0.1, a Euler je tada dokazao da navedena šetnja nije moguća.



Slika 0.1: Grafički prikaz königsberških mostova

Bojenje grafova započinje 1852. godine problemom četiri boje, tj. pitanjem može li se svaka karta obojiti najviše četirima bojama tako da su susjedne države obojene različitim bojama. Ako svaka država predstavlja točku u ravnini, a točke koje se odnose na susjedne države su spojene, dan je grafički prikaz ovog problema. 1976. godine Kenneth Appel i Wolfgang Haken objavljuju dokaz teorema četiri boje. Bitno je napomenuti da se teorem o četiri boje može primijeniti samo na planarne grafove, dok za ostale proizvoljne grafove ne postoji jednostavan način za odrediti broj potrebnih boja.

Problem bojenja grafa dijeli se na tri problema: bojenje vrhova, bojenje bridova i bojenje karata. U ovom radu promatrani su algoritmi za bojenje vrhova. Prvo poglavlje donosi uvod u teoriju grafova te opis problema i primjere bojenja grafova. U drugom poglavlju dana je klasifikacija i opis meta-heurističkih algoritama. U trećem poglavlju detaljnije su opisani konkretni algoritmi koji su implementirani nakon čega u četvrtom poglavlju slijedi analiza i usporedba rezultata.

Poglavlje 1

Definicija problema

1.1 Osnovni pojmovi teorije grafova

Najprije se navode osnovni pojmovi teorije grafova koji će biti potrebni za daljnje razumijevanje.

Definicija 1.1.1. Graf *je uređen par* (V, E)*, gdje je* V *proizvoljan neprazan konačan skup elemenata koji se nazivaju vrhovi, a* $E \subseteq \{\{A, B\} : A, B \in V\}$ *podskup svih dvočlanih podskupova skupa* V*. Elementi skupa* E *se nazivaju bridovi.*

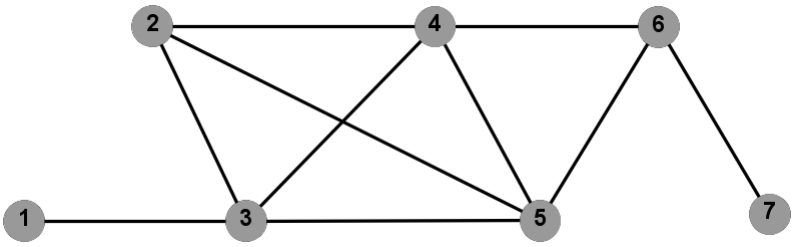
Kako se bridovi definiraju kao skupovi, nije važan redoslijed njihovih elemenata. Dakle, ne kažemo da brid povezuje prvi vrh s drugim, već su vrhovi međusobno povezani. Grafovi u kojima je ovo slučaj nazivaju se neusmjerenima i on njima će biti riječ u ovom radu.

Definicija 1.1.2. *Vrhovi* $A, B \in V$ *su* susjedni*, odnosno povezani bridom, ako je* $\{A, B\} \in E$*. Vrh* A *i brid* e *su* incidentni *ako je* $A \in e$*, tj. ako postoji* $B \in V$ *takav da* $e = \{A, B\}$*.*

Definicija 1.1.3. Stupanj *vrha* $A \in V$*, u oznaci* $d(A)$*, je broj bridova koji su incidentni s vrhom* A*.*

Definicija 1.1.4. Susjedstvo *vrha* $A \in V$*, u oznaci* $\Gamma_G(A)$*, je skup vrhova koji su susjedni s vrhom* A*, tj.* $\Gamma_G(A) = \{B \in V : \{A, B\} \in E\}$*.*

Primjer 1.1.5. *Graf* $G = (V, E)$*, gdje je* $V = \{1, 2, 3, 4, 5, 6, 7\}$*, a*
$E = \{\{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}, \{4, 6\}, \{5, 6\}, \{6, 7\}\}$*, prikazan je na slici 1.1. U tom grafu vrhovi* 5 *i* 6 *su susjedni, dok su vrh* 2 *i brid* $\{2, 5\}$ *incidentni. Stupanj vrha* 4 *je* $d(4) = 3$*.*



Slika 1.1: Primjer grafa

Definicija 1.1.6. Šetnja *u grafu* $G = (V, E)$ *je niz vrhova* $(v_1, v_2, \ldots, v_k)$*, pri čemu su* v_i *i* v_{i+1} *susjedni za* $i = 1, 2, \ldots, k-1$. Staza *je šetnja u kojoj su svi bridovi različiti, dok je* put *staza u kojoj su svi vrhovi različiti (osim eventualno prvog i zadnjeg - tada se takav put naziva ciklusom).*

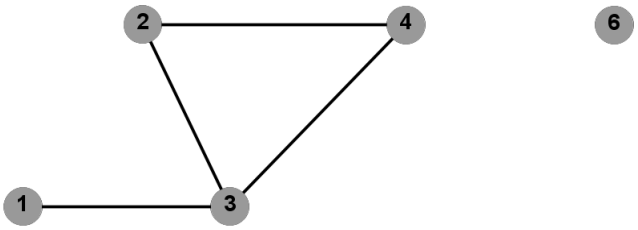
Definicija 1.1.7. *Graf* $G_1 = (V_1, E_1)$ *je* podgraf *grafa* $G = (V, E)$ *ako je* $V_1 \subseteq V$*, a* $E_1 \subseteq \{\{v_i, v_j\} \in E : v_i, v_j \in V_1\}$.

Definicija 1.1.8. *Graf* $G = (V, E)$ *je* povezan *ako između svaka dva vrha postoji put. U suprotnom je graf nepovezan.*

Definicija 1.1.9. Komponenta povezanosti *je maksimalan povezan neprazan podgraf.*

Primjer 1.1.10. *Slika 1.2 prikazuje podgraf* $G' = (V', E')$ *grafa* $G = (V, E)$ *sa slike 1.1. Ovdje je* $V' = \{1, 2, 3, 4, 6\}$*, a* $E' = \{\{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$*. Kako između vrhova* 3 *i* 5 *ne postoji put, graf* G' *je nepovezan. Komponente povezanosti od kojih se sastoji su:*

- $G_1 = (\{1, 2, 3, 4\}, \{\{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\})$
- $G_2 = (\{6\}, \emptyset)$.



Slika 1.2: Podgraf grafa sa slike 1.1

Definicija 1.1.11. Klika *je podskup vrhova grafa koji su svi međusobno povezani bridovima, tj. skup* $C \subseteq V$ *takav da vrijedi* $\{u, v\} \in E$ *za sve* $u, v \in C$.

1.2 Opis problema

Neka je $G = (V, E)$ graf s n vrhova i m bridova.

Definicija 1.2.1. *Neka je k prirodan broj.* k-bojenje *grafa G je funkcija $c : V \to \{1, 2, \ldots, k\}$ koja svakom vrhu grafa pridružuje jednu od k boja.*

Definicija 1.2.2. *Bojenje grafa je* pravilno *ako su susjedni vrhovi obojeni različitim bojama, tj. ako za sve $\{u, v\} \in E$ vrijedi $c(u) \neq c(v)$.*

Prilikom primjene algoritama ponekad se koriste i bojenja grafa koja ne pridružuju boju svim vrhovima grafa, već neke ostavljaju neobojenima.

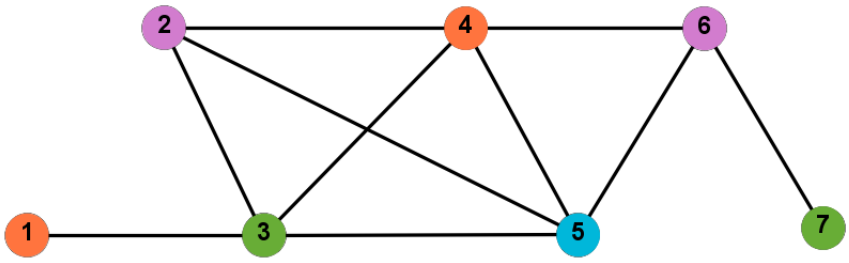
Definicija 1.2.3. *Bojenje grafa je* potpuno *ako svim vrhovima $v \in V$ pridružuje točno jednu boju $c(v) \in \{1, 2, \ldots, k\}$. U suprotnom je bojenje parcijalno.*

Definicija 1.2.4. *Graf je* k-obojiv *ako i samo ako je za njega moguće pronaći k-bojenje koje je istovremeno potpuno i pravilno.*

Kako je svaki graf očito n-obojiv s obzirom da je moguće svaki vrh grafa obojiti različitom bojom, ostaje pitanje koji je minimalan broj boja s kojima je graf pravilno obojiv.

Definicija 1.2.5. Kromatski broj *grafa G, u oznaci $\chi(G)$, je najmanji prirodan broj k takav da je G k-obojiv. Takvo k-bojenje naziva se* optimalnim.

Primjer 1.2.6. *Slika 1.3 prikazuje primjer pravilnog 4-bojenja grafa sa slike 1.1. Kako graf sadrži kliku $\{2, 3, 4, 5\}$ u kojoj su svi vrhovi povezani bridovima pa samim time moraju biti različite boje, ovo je i optimalno bojenje navedenog grafa.*



Slika 1.3: Primjer bojenja grafa sa slike 1.1

Problem bojenja grafa za proizvoljan graf $G = (V, E)$ nastoji svakom vrhu $v \in V$ pridružiti jednu od k boja tako da je bojenje pravilno te da koristi što manje boja, tj. da vrijedi:

- $c(u) \neq c(v)$, za sve $\{u, v\} \in E$
- k je minimalan.

Prostor rješenja problema bojenja grafa su sva k-bojenja za $k \leq n$. Algoritam koji bi sigurno pronašao optimalno rješenje je onaj koji bi prošao cijeli prostor rješenja i vratio ono najbolje. Međutim, prostor rješenja raste eksponencijalno u odnosu na n pa takav algoritam nije moguće provesti u realnom vremenu i s postojećom tehnologijom.

Problem minimizacije broja k može se lako svesti na problem odlučivanja. Za proizvoljan graf G umjesto pitanja „Koji je minimalan broj boja s kojima se graf G može obojiti?" postavlja se pitanje „Može li se graf G obojiti s k boja?". Primjenom problema odlučivanja iterativno za $k \leq n$ dolazi se i do odgovora na pitanje problema minimizacije. Međutim, čak i za takvu formulaciju problema, njegov znatno manji prostor rješenja također raste eksponencijalno u odnosu na n.

Definicija 1.2.7. *Problem odlučivanja pripada klasi problema P ako za njega postoji algoritam koji daje točan odgovor, a vrijeme izvršavanja ovisi polinomno o veličini ulaznih podataka.*

Definicija 1.2.8. *Problem odlučivanja pripada klasi problema NP ako za njega postoji polinoman algoritam koji ispituje točnost potencijalnog rješenja.*

Definicija 1.2.9. *Problem odlučivanja pripada klasi NP-potpunih problema ako pripada klasi NP i bilo koji drugi problem iz NP se može polinomno reducirati na njega.*

Definicija 1.2.10. *Problem pripada klasi NP-teških problema ako postoji NP-potpun problem koji se može polinomno reducirati na njega.*

Ako je zadano potencijalno rješenje problema odlučivanja može li se graf G obojiti s k boja, može se provjeriti u polinomnom vremenu da je to bojenje pravilno i da koristi točno k boja. Zbog toga problem odlučivanja za bojenje grafa pripada klasi NP.

Također, problem odlučivanja može li se graf G obojiti s k boja pripada klasi NP-potpunih problema. Dokaz za to oslanja se na činjenicu da je problem ispunjivosti logičke formule NP-potpun problem. Također, poznato je da ako je problem P_1 NP-potpun i problem P_2 se može polinomnim transformacijama svesti na problem P_1, tada je i problem P_2 NP-potpun. S obzirom da se problem odlučivanja može li se graf G obojiti s k boja može svesti na problem ispunjivosti logičke formule, on je NP-potpun.

Kako se problem pronalaska kromatskog broja grafa se može riješiti polinomno ograničenim brojem poziva problema odlučivanja može li se graf G obojiti s k boja, on pripada klasi NP-teških problema.

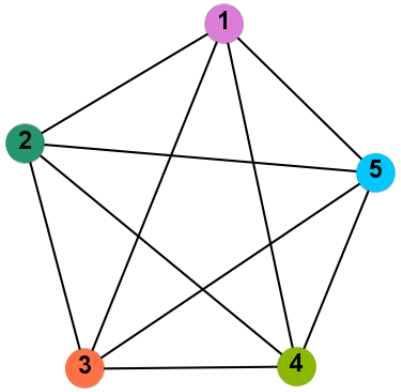
1.3 Jednostavni primjeri

Za neke vrste grafova jednostavno je odrediti pripadni kromatski broj.

Potpuni grafovi

Graf $G = (V, E)$ je *potpun* ako su u njemu svaka dva vrha povezana bridom, tj. za sve $u, v \in V$ vrijedi $\{u, v\} \in E$. Oznaka za potpuni graf s n vrhova je K_n.

Kako su u potpunom grafu svi vrhovi međusobno povezani, očito je da svi moraju biti različite boje pa je prema tome $\chi(K_n) = n$.

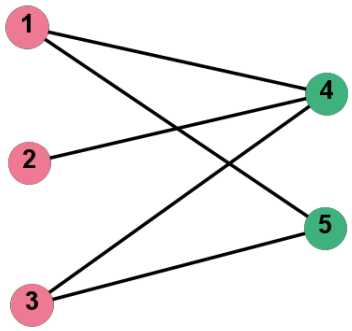


Slika 1.4: Primjer bojenja grafa K_5

Bipartitni grafovi

Graf $G = (V_1, V_2, E)$ je *bipartitan* ako mu vrhovi mogu biti particionirani u dva skupa, V_1 i V_2, tako da postoje bridovi samo između vrhova u V_1 i vrhova u V_2.

Kako ne postoje bridovi između vrhova koji se oba nalaze u skupu V_1, svi vrhovi u tom skupu mogu biti obojeni istom bojom. Isto vrijedi i za vrhove u skupu V_2. Dakle, svaki bipartitan graf se može obojiti u dvije boje, tj. $\chi(G) = 2$.



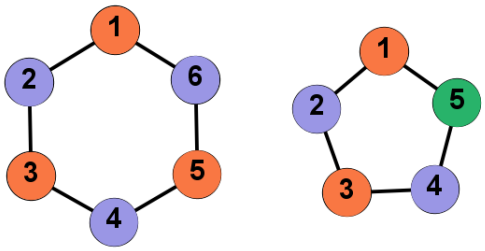
Slika 1.5: Primjer bojenja bipartitnog grafa u kojem je $V_1 = \{1, 2, 3\}$ i $V_2 = \{4, 5\}$

Ciklički grafovi

Graf $G = (V, E)$ je *ciklički* ako mu je za skup vrhova $V = \{v_1, \ldots, v_n\}$ skup bridova oblika $\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}, \{v_n, v_1\}\}$. Oznaka za ciklički graf s n vrhova je C_n.

Ako je n paran, ciklički graf C_n je ujedno i bipartitan pa se može obojiti s dvije boje. Bojenje se konstruira na način da se odabire početni vrh kojemu se pridruži prva boja nakon čega se kreće po grafu u smjeru kazaljke sata te se vrhovima dodjeljuju naizmjenično prva i druga boja. Drugim riječima, neparnim vrhovima se dodjeljuje prva boja, a parnim vrhovima druga. Zadnji vrh je tada moguće obojiti drugom bojom s obzirom da su prvi i $(n - 1)$. vrh neparni te stoga obojeni prvom bojom.

Ako je n neparan, potrebne su tri boje za obojiti ciklički graf C_n. Postupak bojenja započinje analogno kao u slučaju parnog broja n - neparnim vrhovima se dodjeljuje prva boja, a parnim vrhovima druga. Međutim, zadnji vrh nije moguće obojiti niti jednom od prve dvije boje. Naime, prvi vrh je neparan pa je obojen prvom bojom, a $(n - 1)$. je paran pa je obojen drugom bojom. Dakle, zadnji vrh je potrebno obojiti trećom bojom kako bi bojenje bilo pravilno.



Slika 1.6: Primjer bojenja cikličkih grafova C_6 i C_5

Poglavlje 2

Podjela meta-heurističkih algoritama

Meta-heuristike su algoritmi koji se koriste za pronalaženje ekstrema funkcije cilja za određeni problem, ali ne moraju nužno taj ekstrem i pronaći. Funkcija cilja u problemu bojenja grafova je funkcija koja za određeni graf određuje s koliko se boja on može pravilno obojiti. Ako je riječ o pronalasku kromatskog broja grafa, očito je da tu funkciju treba minimizirati. Kako je u prethodnom poglavlju opisano, svaki algoritam koji bi imao garanciju da će pronaći globalni minimum te funkcije za bilo koji graf je neefikasan. Točnije, vremenska složenost takvog algoritma ovisi eksponencijalno o ulaznim parametrima. Dakle, takvi algoritmi daju dobre rezultate za manje grafove, dok za veće grafove imaju preveliku vremensku složenost. Zato se koriste meta-heuristike - one pronalaze rješenje koje je dovoljno blizu optimalnom u znatno kraćem vremenskom roku, čak i za veće grafove.

Meta-heuristike se najjednostavnije dijele na one koje koriste jedno rješenje koje u svakom koraku pokušavaju poboljšati i one koje koriste populaciju rješenja. Neki od algoritama koji koriste jedno rješenje su tabu traženje i simulirano kaljenje, dok su primjeri populacijskih algoritama genetski algoritmi te algoritmi roja čestica. Neki od navedenih algoritama bit će detaljnije opisani u sljedećim poglavljima.

U ovom poglavlju promatra se druga podjela meta-heurističkih algoritama - ona s obzirom na prostor rješenja koji koriste. Kako bi neki graf bio k-obojiv za neki prirodan broj k, mora postojati k-bojenje koje je istovremeno potpuno i pravilno. Međutim, u postupku pronalaska takvog rješenja mogu se koristiti i parcijalna bojenja i bojenja koja nisu nužno pravilna. S obzirom na to, razlikuju se tri vrste prostora pretraživanja koje algoritmi mogu koristiti: prostor pravilnih i potpunih bojenja, prostor potpunih, ali ne nužno pravilnih bojenja te prostor parcijalnih, ali pravilnih bojenja. U nastavku je dan detaljniji opis svake od navedenih vrsta.

2.1 Prostor potpunih i pravilnih bojenja

Algoritam koji pronalazi potpuno i pravilno bojenje proizvoljnog grafa, ali ne nužno ono s najmanje boja je pohlepni algoritam (eng. *greedy*). On se koristi prilikom implementacije mnogih meta-heurističkih algoritama pa ga je na početku potrebno opisati.

Za pohlepni algoritam potrebno je imati unaprijed određenu permutaciju vrhova grafa te poredak boja koje će biti korištene. Pohlepni algoritam prolazi vrhovima grafa u zadanom poretku i svakom vrhu dodjeljuje prvu slobodnu boju, tj. prvu boju kojom nije obojen niti jedan susjed promatranog vrha. Jasno je da će po završetku pohlepnog algoritma biti pronađeno potpuno i pravilno bojenje grafa jer je u svakom koraku algoritma osigurano da bojenje ostaje pravilno. Algoritam 1 prikazuje pseudokod pohlepnog algoritma.

Vrlo je važna činjenica da za svaki graf postoji permutacija njegovih vrhova koja ako na nju primijenimo pohlepni algoritam, daje optimalno bojenje tog grafa. To slijedi iz svojstva da, ako je zadano pravilno bojenje grafa G i za njega se provodi pohlepni algoritam tako da prvo prolazi kroz sve vrhove koji su bili obojeni jednom bojom, zatim sve vrhove koji su bili obojeni drugom bojom itd., novo bojenje će također biti pravilno i imat će manje ili jednako boja kao prethodno. Dakle, ako je poznato optimalno bojenje grafa, upravo opisana permutacija vrhova bit će ona koja će ponovnom primjenom pohlepnog algoritma dati optimalno bojenje. Međutim, pronalazak takve permutacije također je problem eksponencijalne vremenske složenosti pa to ne može biti način na koji se efikasno rješava problem bojenja grafa.

Algorithm 1: Greedy algorithm

```
Data: graph G, permutation of the vertices π
for i = 1 to n do
    listOfColors = ∅;
    for j = 1 to n do
        if j ∈ Γ_G(i) & color(j) is not null then
            add color(j) to listOfColors;
        end
    end
    for c = 1 to n do
        if c ∉ listOfColors then
            color(i) = c;
            break;
        end
    end
end
```

Algoritmi koji za prostor pretraživanja koriste prostor potpunih i pravilnih bojenja uz ta ograničenja nastoje pronaći ono bojenje koje koristi najmanji broj boja. Najjednostavniji takav algoritam je onaj koji iterativno koristi pohlepni algoritam, a bazira se na prethodno opisanoj činjenici da se iz jednog pravilnog bojenja grafa dobrim odabirom permutacije vrhova i primjenom pohlepnog algoritma dolazi do bojenja koje koristi manji ili jednak broj boja kao prethodni. Također, takva permutacija vrhova odabire prvo vrhove koji su bili obojeni jednom bojom, zatim vrhove koji su bili obojeni drugom bojom itd., ali ostaje pitanje redoslijeda boja koji se koristi. Tri su tipična načina za odabir redoslijeda boja:

- s obzirom na broj vrhova koji su obojeni svakom od boja na način da se najprije uzimaju boje s kojima je obojeno najviše vrhova. Time se želi postići da što više vrhova ima istu boju.
- s obzirom na redoslijed boja u trenutnom bojenju na način da se uzima suprotan redoslijed. Ovime se želi postići da se vrhovi koji su prethodno bili iste boje sada rasporede između više boja.
- nasumični redoslijed kojim se želi izbjeći stalno ponavljanje istih rješenja.

Kako svaki od navedenih načina ima svoje prednosti, obično se kombinira njihovo korištenje. Najčešće se u svakoj iteraciji nasumično odabire jedan od njih u omjeru $5 : 5 : 3$.

2.2 Prostor potpunih, ali ne nužno pravilnih bojenja

Najčešće korištena vrsta prostora pretraživanja je ona koja sadrži potpuna, ali ne nužno pravilna bojenja zadanog grafa. Algoritmi koji ju koriste imaju unaprijed zadan broj boja k koje mogu koristiti pa zbog toga nije uvijek moguće postići pravilno bojenje. Prilikom inicijalizacije početnog bojenja, algoritam na neki način (npr. slučajnim odabirom ili pohlepnim algoritmom) mora svakom vrhu dodijeliti jednu od k boja. Ako za neki vrh ne postoji boja s kojom bi bojenje ostalo pravilno, algoritam mu svejedno dodjeljuje neku boju. Cilj algoritma je promjenama postojećeg bojenja postići pravilno bojenje zadanog grafa. U tom slučaju je graf k-obojiv pa se algoritam može provesti za $k - 1$ boja kako bi utvrdio minimalan broj boja za koji može pronaći pravilno bojenje.

S obzirom da su svim vrhovima dodijeljene boje, mora postojati način na koji se mogu razlikovati pravilna bojenja od onih koja to nisu, kako bi algoritam na kraju mogao odrediti koje je najbolje pronađeno bojenje. U tu svrhu se uvodi funkcija dobrote koja broji parove vrhova obojene istom bojom. Neka je G graf s n vrhova i m bridova te neka je c neko bojenje grafa G. Tada se definira:

$$f_1(c) = \sum_{\forall\{u,v\}\in E} g(u, v), \tag{2.1}$$

gdje je

$$g(u,v) = \begin{cases} 1, & \text{ako je } c(u) = c(v) \\ 0, & \text{inače.} \end{cases} \tag{2.2}$$

Primjer algoritma koji koristi ovaj prostor rješenja je simulirano kaljenje. To je algoritam koji se bazira na lokalnom pretraživanju prostora rješenja pri čemu u svakom koraku pokušava pronaći rješenje bolje od trenutnog, no ako to ne uspije, svejedno s nekom vjerojatnošću prelazi na lošije rješenje kako bi bolje istražio prostor rješenja. Kao i ostali algoritmi lokalnog pretraživanja, i ovaj ima operator susjedstva na temelju kojeg određuje među kojim bojenjima traži bolje rješenje od onog trenutnog. Treba primijetiti da je ovo drugačiji operator od operatora susjedstva za vrhove grafova koji jednostavno kaže s kojim je drugim vrhovima određeni vrh grafa povezan bridom. Operator susjedstva u algoritmu simuliranog kaljenja uvodi susjedstvo između na neki način sličnih bojenja grafa. Najjednostavniji primjer operatora susjedstva povezuje bojenja koja se razlikuju u boji točno jednog vrha. Dakle, ako se iz bojenja c grafa G promjenom boje jednog vrha može doći do bojenja d istog tog grafa, tada su bojenja c i d susjedna. Skup susjednih bojenja bojenju c označava se s $N(c)$.

Algoritam simuliranog kaljenja na početku kreira početno rješenje c i računa njegovu funkciju dobrote $f_1(c)$. Nakon toga algoritam u svakoj iteraciji nasumično odabire susjeda trenutnog bojenja d (nasumično odabire vrh grafa i boju u koju će ga promijeniti) te na temelju njegove funkcije dobrote $f_1(d)$ odlučuje hoće li to postati novo rješenje koje će se koristiti u sljedećoj iteraciji algoritma:

- ako je $f_1(d) \leq f_1(c)$, tada bojenje d postaje novo promatrano rješenje
- ako je $f_1(d) > f_1(c)$, tada također s određenom vjerojatnošću d postaje novo promatrano rješenje kako bi se bolje istražio prostor rješenja, tj. kako algoritam ne bi zaglavio u lokalnom optimumu. Vjerojatnost prelaska na lošije rješenje je $e^{-\delta/t}$, gdje je $\delta = |f_1(c) - f_1(d)|$, a t parametar koji predstavlja temperaturu. Temperatura je na početku izvođenja algoritma postavljena na neku veliku vrijednost kako bi algoritam istražio prostor rješenja. Tijekom izvođenja algoritma se vrijednost temperature smanjuje što rezultira sve rjeđim prihvaćanjem lošijeg rješenja kako bi algoritam došao što bliže nekom optimumu.

Algoritam 2 prikazuje pseudokod algoritma simuliranog kaljenja. U ovom algoritmu prikazana je i jedna metoda hlađenja, tj. smanjivanja temperature na način da se temperatura t svakih z iteracija algoritma množi s faktorom hlađenja $\alpha \in \langle 0, 1 \rangle$.

Algorithm 2: Simulated annealing algorithm

```
create initial coloring c;
choose an initial temperature t;
while stopping criterion is not met do
    randomly choose neighbour coloring d ∈ N(C);
    randomly generate number r ∈ ⟨0, 1⟩;
    i = 0;
    if f(d) ≤ f(c) then
        c = d;
    else if r ≤ e^{−δ/t} then
        c = d;
    i = i + 1;
    if i mod z == 0 then
        t = αt;
    end
end
```

2.3 Prostor parcijalnih, ali pravilnih bojenja

Posljednja i najmanje korištena vrsta prostora pretraživanja je prostor parcijalnih, ali pravilnih bojenja. Kao i kod prethodne vrste, algoritmi koji ju koriste imaju unaprijed zadan broj boja k pa nije uvijek moguće postići pravilno bojenje. Međutim, ovdje ne moraju svi vrhovi biti obojeni, već se oni koji se ne mogu obojiti tako da bojenje ostane pravilno čuvaju u skupu U neobojenih vrhova. Cilj algoritma je promjenama postojećeg bojenja postići pravilno bojenje zadanog grafa, tj. postići da na kraju bude $U = \emptyset$. U tom slučaju je graf k-obojiv pa se algoritam može provesti za $k - 1$ boja kako bi utvrdio minimalan broj boja za koji može pronaći pravilno bojenje.

Algoritam koji koristi ovaj prostor rješenja je tabu traženje. Ovaj algoritam se, kao i simulirano kaljenje, bazira na lokalnom pretraživanju prostora rješenja, ali za razliku od njega promatra sva susjedna bojenja i kao novo rješenje uzima najbolje od njih. Prednost opisane izmjene je što algoritam tabu traženja zna kada je došao do lokalnog optimuma jer u tom slučaju nema susjednog bojenja koje je bolje od trenutnog. U tom slučaju algoritam prelazi na lošije bojenje kako bi istražio prostor rješenja, ali sprječava cikličko ponašanje, tj. pretraživanje stalno istih rješenja, uvođenjem tabu liste. U svakoj iteraciji se trenutno posjećeno bojenje pohranjuje u tabu listu, što sprječava njegovo ponovno posjećivanje određeni vremenski period. Tabu traženje koristi i drugačiji operator susjedstva od onog koji koristi simulirano kaljenje. Naime, ako je poznato bojenje c grafa G, onda su njegovi susjedi sva bojenja d koja se mogu dobiti odabirom jednog vrha $v \in U$ kojem se

dodjeljuje boja j te se sve vrhove susjedne vrhu v čija je boja bila j proglašava neobojenima, tj. prebacuje u skup U. Kako ovdje nema parova vrhova obojenih istom bojom jer se radi o pravilnim bojenjima, uvodi se i nova funkcija dobrote koja broji koliko u bojenju ima neobojenih vrhova:

$$f_2(c) = |U|. \tag{2.3}$$

Algoritam tabu pretraživanja na početku kreira početno rješenje c i računa njegovu funkciju dobrote $f_2(c)$. Nakon toga u svakoj iteraciji algoritam promatra sve susjede trenutnog bojenja (sve izbore vrha $v \in U$ i sve boje koje mu mogu biti pridružene) te novo rješenje postaje ono s najboljom funkcijom dobrote, a koje se u toj iteraciji ne nalazi u tabu listi. Ovaj korak se provodi čak i ako niti jedno susjedno bojenje nije bolje od onog trenutno promatranog.

Algoritam 3 prikazuje pseudokod algoritma tabu pretraživanja. Duljina ostanka bojenja u tabu listi je postavljena na t iteracija, gdje je t unaprijed zadan broj.

Algorithm 3: Tabu search algorithm

```
create initial coloring c with the set of uncolored vertices U;
choose the number of iterations t;
tabuList = ∅;
while stopping criterion is not met do
    choose d ∈ N(c) such that (∀h ∈ N(c))(f2(d) ≤ f2(h));
    while d ∈ tabuList do
        N(c) = N(c)\{d};
        choose new d ∈ N(c) such that (∀h ∈ N(c))(f2(d) ≤ f2(h));
    end
    move coloring c into tabuList for t iterations;
    c = d;
end
```

Poglavlje 3

Implementacija

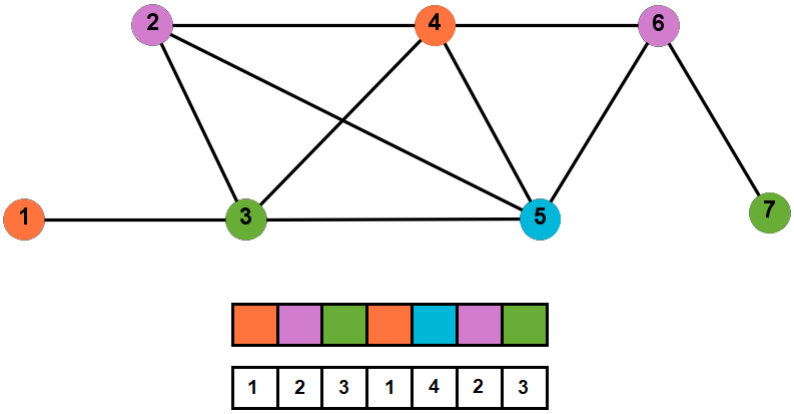
U ovom poglavlju bit će riječ o konkretnim implementacijama nekih meta-heurističkih algoritama za određivanje kromatskog broja grafa i njegovog optimalnog bojenja. Neka je $G = (V, E)$ proizvoljan graf s n vrhova i m bridova.

3.1 Genetski algoritam

Genetski algoritmi pripadaju skupini algoritama koja koristi populaciju rješenja te iz nje iterativno generira sve bolja i bolja rješenja. Prostor rješenja koji ovaj algoritam koristi je prostor potpunih, ali ne nužno pravilnih rješenja. Naziv genetski ili evolucijski algoritmi dobili su jer oponašaju principe prirodne evolucije poput križanja i mutacija. Pojedina bojenja unutar prostora pretraživanja se nazivaju *jedinke*, a skup jedinki se naziva *populacija*. Prikaz koji kodira neko bojenje traženog grafa naziva se *kromosomska reprezentacija*. U ovom slučaju kromosom je polje čija je duljina n, a vrijednosti, koje se nazivaju *geni*, su boje vrhova grafa. Na slici 3.1 prikazan je obojeni graf sa slike 1.3 te pripadna kromosomska reprezentacija. Prvi kromosom prikazan je pomoću boja kako bi se vidjela poveznica s grafom, dok su u drugom bojama dodijeljeni brojevi, što je praktičnije za implementaciju, pogotovo u slučaju većih grafova za koje je potreban velik broj boja.

Početna populacija

Prvi način kreiranja početne populacije je nasumični odabir boje vrha koja još nije korištena u jedinki koja se trenutno kreira. Dakle, svaki vrh pojedine jedinke obojen je različitom bojom, što rezultira n-bojenjem grafa G. Međutim, nakon pokretanja cjelokupnog genetskog algoritma na primjerima grafova, utvrđeno je da bi rezultati mogli biti bolji ako se početna populacija kreira na način da svaka jedinka koristi manje boja. Zato je implementiran drugi način koji svaku jedinku početne populacije kreira iz nasumične permutacije vrhova



Slika 3.1: Bojenje grafa sa slike 1.3, zajedno s pripadnom kromosomskom reprezentacijom

pokretanjem pohlepnog algoritma. Ovaj način je pokazao bolje rezultate kod konačnog pronalaska kromatskog broja grafa.

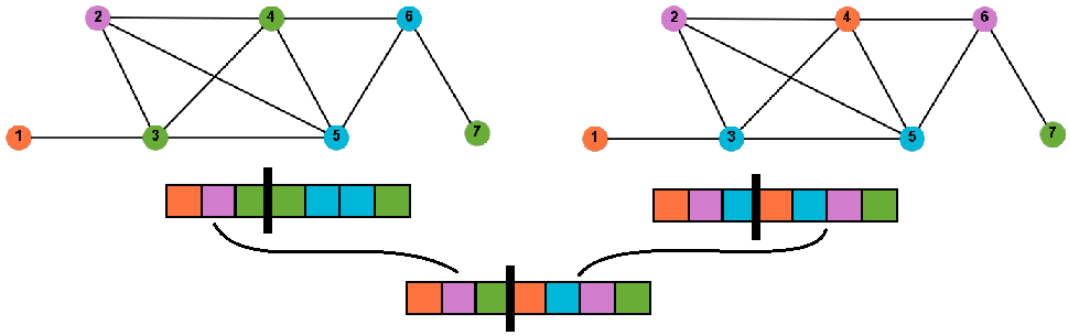
Nakon kreiranja početne populacije, u svakoj iteraciji algoritma jedinke se s određenom vjerojatnošću podvrgava genetskim operatorima: križanju i mutaciji te se na kraju svake iteracije odvija selekcija jedinki koje će biti korištene u sljedećoj iteraciji algoritma.

Križanje

Križanje je genetski operator koji uzima dvije jedinke populacije te na osnovu njihovog genetskog materijala kreira novu jedinku - dijete. Vrsta križanja koja je implementirana je križanje u jednoj točki. To znači da se izabire jedna točka unutar kromosoma te se dijete kreira tako da dio kromosoma prije izabrane točke naslijedi od jednog roditelja, a dio kromosoma nakon izabrane točke od drugog roditelja. Izbor te točke ovisi o tome je li bojenje koje predstavlja kromosom prvog roditelja pravilno ili nije. Ako je bojenje pravilno, točka se izabire nasumično. Međutim, ako bojenje nije pravilno, tada se točka izabire na način da se nalazi na mjestu prije prvog gena koji uzrokuje nepravilnost bojenja. Nakon što je točka izabrana, promatraju se oba bojenja koja mogu nastati iz istih roditelja - prvo dijete nasljeđuje početni dio kromosoma od prvog roditelja, a završni dio od drugog, dok drugo dijete nasljeđuje početni dio od drugog roditelja, a završni dio od prvog. Odabire se ono bojenje koje sadrži manje susjednih vrhova iste boje. Ako je taj broj jednak za oba bojenja, svejedno je koje će bojenje biti odabrano.

Primjer križanja dva nepravilna bojenja za graf sa slike 1.1 prikazan je na slici 3.2. Prvi gen koji uzrokuje nepravilnost bojenja prvog roditelja je vrh 4 jer je susjedan vrhu 3 koji je iste boje. Zato se točka u kojoj se provodi križanje nalazi prije vrha 4. Kromosom koji je dobiven ovim križanjem odgovara pravilnom bojenju grafa vidljivom na prethodnoj slici.

Drugo bojenje koje bi nastalo iz prikazanih roditelja naslijedilo bi završni dio kromosoma prvog roditelja pa bi u njemu susjedni vrhovi 5 i 6 bili iste boje. Dakle, bojenje prikazano na slici bi bilo odabrano kao bolje i dodano u populaciju.



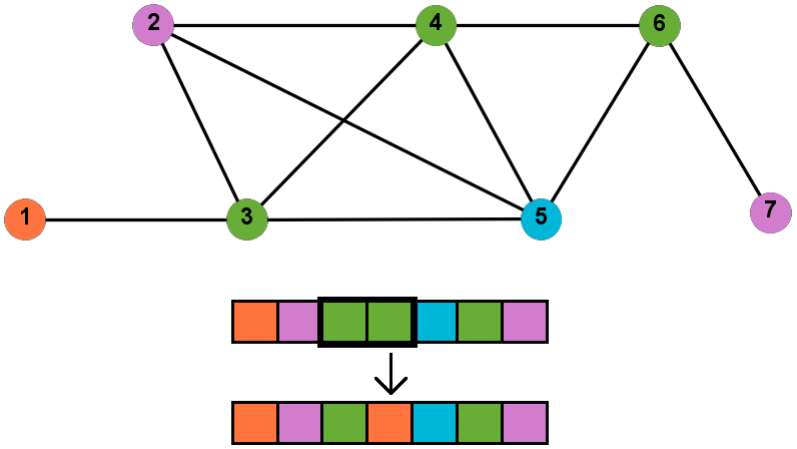
Slika 3.2: Primjer križanja dva nepravilna kromosoma čiji je rezultat pravilno bojenje kromosoma

Mutacija

Mutacija je genetski operator koji uzima samo jednu jedinku populacije te mijenjanjem njezinog genetskog materijala kreira novu jedinku. Mutacija se također provodi ovisno o tome je li bojenje koje predstavlja kromosom roditelja pravilno ili nije. Ako je bojenje pravilno, odabire se nasumično jedna boja te se svi geni koji su bili te boje bojaju nekom drugom bojom. Za svaki takav gen, promatraju se sve mogućnosti odabira nove boje i odabire se ona koja dovodi do najmanjeg broja susjednih vrhova iste boje. Ako bojenje nije pravilno, operator mutacije pronalazi prva dva susjedna vrha koji su obojeni istom bojom te mijenja boju jednog od njih kako bi lokalno popravio bojenje. Istražuju se sve mogućnosti promjene boje za oba pronađena vrha i odabire se, kao i u slučaju pravilnog bojenja, ona koja dovodi do najmanjeg broja susjednih vrhova iste boje.

Primjer mutacije nepravilnog bojenja za graf sa slike 1.1 prikazan je na slici 3.3. Prva dva vrha iste boje su vrhovi 3 i 4 pa oni dolaze u obzir kao vrhovi kojima će biti promijenjena boja. Kako vrh 3 ima po jednog susjeda svake boje, mijenjanjem njegove boje ne može se postići pravilno bojenje. S druge strane, vrh 4 ima dva susjeda čija je boja jednaka njegovoj, a nema susjeda narančaste boje. Mijenjanjem njegove boje u narančastu nastaje pravilno bojenje pa je to potez koji će operator mutacije napraviti.

U konkretnoj implementaciji populacija se dijeli na dva jednaka dijela. Iz prvog dijela biraju se jedinke koje će sudjelovati u križanju, a iz drugog dijela one koje će biti mutirane. Određuje se željeni broj potomaka te se postupci križanja i mutacije toliki broj puta iteriraju. U svakoj iteraciji se za križanje nasumično odabiru dva roditelja iz prve polovice



Slika 3.3: Primjer mutacije nepravilnog kromosoma čiji je rezultat pravilno bojenje kromosoma

populacije, a za mutaciju jedan roditelj iz druge polovice populacije. Generirana djeca se dodaju u populaciju kako bi se selekcijom moglo odrediti koje će jedinke biti sačuvane u populaciji, a koje odbačene.

Selekcija

Na kraju svake iteracije, uz populaciju koja je postojala na početku te iteracije dodane su još i jedinke nastale operatorima križanja i mutacije pa je potrebno napraviti selekciju koje će od tih jedinki biti dio populacije za sljedeću iteraciju. Za to se koristi funkcija dobrote koja je za pojedinu jedinku jednaka broju boja k koje se u njoj koriste, čemu se pribraja broj n za svaka dva susjedna vrha iste boje:

$$f(c) = k + \sum_{i=0}^{n} \sum_{j=0, j\neq i}^{n} p(i, j), \tag{3.1}$$

gdje je

$$p(i, j) = \begin{cases} n, & \text{ako je } \{i, j\} \in E \text{ i } c(i) = c(j) \\ 0, & \text{inače.} \end{cases} \tag{3.2}$$

Dakle, ako za bojenje c koje predstavlja neka jedinka vrijedi $f(c) < n$, tada je to bojenje sigurno pravilno, tj. u njemu ne postoje dva susjedna vrha iste boje.

Nakon izračunavanja funkcije dobrote, jedinke se sortiraju uzlazno po toj vrijednosti. Broj jedinki populacije na početku svake iteracije je unaprijed zadan. Za prvu polovicu jedinki koje ulaze u novu populaciju se biraju one najbolje iz sortiranog niza jedinki, a druga polovica se bira nasumično između preostalih jedinki.

Pseudokod algoritma i parametri

Algoritam 4 prikazuje pseudokod implementiranog genetskog algoritma. Parametri s kojima je algoritam pokretan su preuzeti iz članka [3], a vrijednosti su:

- M = veličina populacije = 50
- N_g = broj generacija = 200000
- P_c = vjerojatnost križanja = 30 %
- P_m = vjerojatnost mutacije = 80 %.

Dodatan parametar koji algoritam uvodi je broj iteracija nakon kojih se zaustavlja ako nije pronašao bolje rješenje:

- N_s = broj generacija prije zaustavljanja = 1000.

Algorithm 4: Genetic algorithm

choose the population size M;
choose the number of generations N_g;
choose the number of generations N_s for algorithm to stop if no better solution is found;
N_0 = number of offsprings = $\frac{M}{2}$;
generate initial population $currentPop$;
$numberOfColors = n$;
for $i = 0$ **to** M **do**
    calculate the objective function of each chromosome;
end
for $i = 0$ **to** N_g **do**
    if $bestFitness < numberOfColors$ **then**
        $numberOfColors = bestFitness$;
    end
    if *there is no better solution found after* N_s *generations* **then**
        break;
    end
    divide the population into two halves, P_1 and P_2;
    for $i = 1$ **to** N_0 **do**
        select two chromosomes from P_1;
        apply crossover with probability P_c;
        add offspring to the population;
    end
    for $i = 1$ **to** N_0 **do**
        select a chromosome from P_2;
        apply mutation with probability P_m;
        add offspring to the population;
    end
    calculate the objective function of each offspring;
    newPop = apply selection to currentPop and offspring;
    currentPop = newPop;
end

3.2 Algoritam roja čestica u kombinaciji s tabu traženjem

Algoritam roja čestica

Algoritam roja čestica je, kao i genetski algoritmi, algoritam koji rješenje traži na osnovi populacije. Također, to je algoritam inspiriran prirodom koji imitira ponašanje ptica unutar jata te se zbog toga populacija ovdje naziva rojem. Čestice u roju koriste se informacijama o svom do sada najboljem pronađenom rješenju te o najboljem rješenju svojih susjeda te na osnovu toga iterativno dolaze do sve boljih rješenja. Prostor rješenja koji ovaj algoritam koristi je prostor potpunih, ali ne nužno pravilnih rješenja koja se sastoje od točno unaprijed zadanog broja boja. Algoritam roja čestica je u originalnom obliku osmišljen za rješavanje problema u kojima je prostor rješenja kontinuiran. Zbog toga je u ovom slučaju potrebno redefinirati osnovne pojmove i operacije koje se koriste kako bi se algoritam mogao primijeniti na diskretan prostor rješenja. Najprije se navode definicije pojmova i operacija, a zatim konkretna primjena na prostor grafova.

Definicija 3.2.1. Diskretan prostor rješenja *je uređen par* (S, o)*, gdje je* S *skup rješenja, a* o *je operator pomoću kojeg se iz jednog rješenja* $s_1 \in S$ *dobiva drugo rješenje* $s_2 \in S$*. Operator mora biti reverzibilan i povezan, što znači da se iz proizvoljnog rješenja može konačnim brojem primjena operatora dobiti bilo koje drugo rješenje.*

Definicija 3.2.2. *U diskretnom prostoru rješenja* (S, o) *definira se* udaljenost *dvaju rješenja* $s_1 \in S$ *i* $s_2 \in S$ *kao najmanji broj primjena operatora s kojim se iz rješenja* s_1 *može dobiti rješenje* s_2*.*

Za ažuriranje položaja svake čestice u svakoj iteraciji algoritma potrebne su sljedeće operacije:

Definicija 3.2.3. Razlika dvaju rješenja $x, y \in S$ *je najkraći niz primjena operatora s kojim se iz rješenja* x *može dobiti rješenje* y*, tj.* $x - y = o(x), o(r), \ldots, o(t)$ *takav da je* $o(x) = r, o(r) = s, \ldots, o(t) = y$*. Ovaj niz primjena operatora naziva se brzina.*

Definicija 3.2.4. Produkt broja $\phi \in [0, 1]$ i brzine $v = o^1 o^2 \cdots o^d$ *je podskup brzine od prvih* k *primjena operatora* o*, tj.* $\phi v = o^1 o^2 \cdots o^k$*, gdje je* $k = [\phi d]$.

Definicija 3.2.5. Suma rješenja x i brzine $v = o^1 o^2 \cdots o^d$ *je novo rješenje koje nastaje primjenom* d *puta operatora* o *na rješenje* x*, tj.* $x + v = o(\ldots o(o(x)))$.

U konkretnom slučaju bojenja grafova, skup rješenja S jednak je skupu svih potpunih, ali ne nužno pravilnih rješenja koja se sastoje od točno k boja, gdje je k unaprijed zadan broj koji označava broj boja. Operator o djeluje tako da mijenja boju jednog vrha grafa.

Vrh grafa i boja u koju se on mijenja su ulazni parametri operatora. Operator se ne smije primijeniti na vrh koji je obojen drugačijom bojom od svih ostalih kako bi broj boja ostao fiksan.

Udaljenost dvaju rješenja je najmanji broj vrhova kojima je potrebno promijeniti boju kako bi se iz jednog rješenja dobilo drugo, a razlika dvaju rješenja je niz primjena operatora kojima se to postiže. Iako je intuitivno lako, programski ovo nije jednostavno izračunati jer se bojenja koja se mogu dobiti jedno iz drugog samo promjenom „imena" boja smatraju jednakima. Upravo zato se problem pronalaska razlike dvaju bojenja može svesti na problem dodjeljivanja kojega rješava mađarski ili Kuhn-Munkresov algoritam s vremenskom složenošću $O(|V|^3)$. Više o tom algoritmu može se pronaći u [4], a sam algoritam preuzet je s [2]. Algoritam 5 prikazuje pseudokod računanja razlike dvaju bojenja. Prvo se kreira matrica A koja se nakon toga šalje mađarskom algoritmu. Taj algoritam pronalazi parove boja $\{i, j\}$ takve da nakon promjene boje svih vrhova prvog bojenja boje i u boju j tražena dva bojenja imaju najmanji broj vrhova različite boje. Na kraju se kreira skup Γ vrhova kojima je, nakon opisane promjene „imena" boja u jednom bojenju, potrebno promijeniti boju kako bi se iz njega dobilo drugo bojenje.

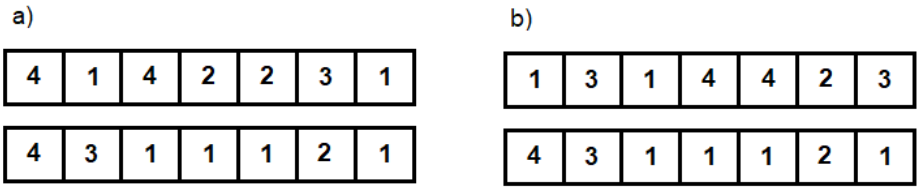
Algorithm 5: The difference of two colorings

Data: two colorings C^1 and C^2
create matrix $A_{k\times k}$;
for $i = 1$ **to** k **do**
  for $j = 1$ **to** k **do**
    $a_{ij} = |\{v \in V : C^1[v] = i \;\&\; C^2[v] = j\}|$;
  end
end
create maximum assignment $\Lambda = \{(i, j)\}$ using Hungarian algorithm;
$\Gamma = \emptyset$;
for $(i, j) \notin \Lambda$ **do**
  $\Gamma = \Gamma \cup \{v \in V : C^1[v] = i \;\&\; C^2[v] = j\}$;
end

Primjer primjene mađarskog algoritma i određivanja udaljenosti dvaju bojenja bit će opisan na bojenjima prikazanima na slici 3.4 a) za $k = 4$ i graf sa slike 1.1. Tada je matrica A sljedeća:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Mađarski algoritam vraća parove boja $(1,3),(2,4),(3,2)$ i $(4,1)$, što znači da je u prvom bojenju potrebno sve vrhove boje 1 zamijeniti bojom 3, vrhove boje 2 bojom 4 i tako dalje. Time je dobiveno bojenje prikazano na slici 3.4 b). Na kraju, skup Γ čine oni vrhovi grafa kojima se nakon ove promjene u promatranim bojenjima razlikuju boje, dakle $\Gamma = \{1, 4, 5 \text{ i } 7\}$.



Slika 3.4: a) Bojenja prije primjene mađarskog algoritma, b) Bojenja nakon primjene mađarskog algoritma i promjene boja prvog bojenja

Još jedan vrlo važan dio algoritma roja čestica je kreiranje početnog roja. Potrebno je paziti da početna bojenja budu što različitija kako bi čestice već na početku istražile što više prostora rješenja. To se postiže na način prikazan algoritmom 6. Algoritam se sastoji od generiranja k pozitivnih brojeva $n_1, \ldots, n_k$ koji označuju broj vrhova obojenih svakom od k boja, nakon čega se za svaku pojedinu boju i nasumično odabire n_i vrhova koji do tada još nisu obojeni.

Algorithm 6: Create initial coloring C

```
Data: graph G = (V, E) and number of colors k
Γ = V;
generate k postitive integers n_1, n_2, ..., n_k at
  random, such that Σ_{i=1}^{k} n_i = |V|;
for i = 1 to k do
    for j = 1 to n_i do
        choose a vertex v ∈ Γ at random;
        C[v] = i;
        Γ = Γ \ {v};
    end
end
```

Tabu traženje

Algoritam tabu traženja bio je opisan u poglavlju 2.3, ali u tom slučaju je koristio parcijalna i pravilna bojenja. Ovdje se koriste potpuna, ali ne nužno pravilna bojenja pa je i

algoritam tabu traženja izmijenjen. Funkcija dobrote koju treba minimizirati računa se kao broj susjednih vrhova koji imaju istu boju:

$$f(c) = \sum_{i=0}^{n} \sum_{j=0, j\neq i}^{n} p(i, j), \tag{3.3}$$

gdje je

$$p(i, j) = \begin{cases} 1, & \text{ako je } \{i, j\} \in E \text{ i } c(i) = c(j) \\ 0, & \text{inače.} \end{cases} \tag{3.4}$$

Za pravilna bojenja taj je broj jednak 0 pa se koristi kao indikator da je pravilno bojenje pronađeno i algoritam može stati. Također, MAX_{TABU} je parametar koji određuje broj iteracija nakon kojih algoritam staje čak i ako nije pronašao pravilno bojenje. Pseudokod je prikazan algoritmom 7. Oznaka $move(v, i)$ označava primjenu operatora o iz definicije prostora rješenja na vrh v, na način da mu promijeni boju u i. Oznaka $\delta(v, i)$ označava za koliko se smanjio broj konflikata, tj. susjednih vrhova koji imaju istu boju, nakon primjene operatora $move(v, i)$. Algoritam prima jedno bojenje C grafa G te u svakoj iteraciji pronalazi onaj vrh v i onu boju i koji imaju maksimalan $\delta(v, i)$, tj. najviše doprinose smanjenju funkcije dobrote promatranog bojenja. Na taj način algoritam iterativno pokušava, mijenjanjem boje jednog vrha u svakoj iteraciji, doći do pravilnog bojenja.

Algorithm 7: Tabu search algorithm

Data: graph $G = (V, E)$ and a k-coloring C of G
initialize new coloring $C^* = C$;
$tabuList = \emptyset$;
$iter = 0$;
while $iter < MAX_{TABU}$ **do**
  choose vertex v and color i with maximum $\delta(v, i)$, such that
   $move(v, i) \notin tabuList$;
  add $move(v, i)$ to tabuList for t iterations;
  C' = coloring generated from C by executing $move(v, i)$;
  if $f(C') < f(C)$ **then**
    $C^* = C'$;
  end
  if $f(C^*) = 0$ **then**
    break;
  end
  $iter = iter + 1$
end

Pseudokod algoritma i parametri

Algoritam 8 prikazuje pseudokod implementiranog algoritma roja čestica u kombinaciji s tabu traženjem. U svakoj iteraciji algoritma svaka čestica pokušava poboljšati svoje bojenje pomakom s određenom vjerojatnošću prema svom do tada pronađenom najboljem bojenju C^p, najboljem bojenju svojih susjeda C^g ili nasumično generiranom bojenju C^r. Na kraju se za svako novo generirano bojenje pokreće algoritam tabu traženja kako bi se to bojenje pokušalo dodatno poboljšati.

Kako algoritam roja čestica uobičajeno rješava problem k-obojivosti grafa G, a ne traženja kromatskog broja grafa, napravljena je dorada. Cijeli algoritam roja čestica se pokreće iterativno, pri čemu se broj boja smanjuje za 1 svaki put kada je prijašnje k-bojenje pronađeno, a zaustavlja se kada za neki k ne uspije pronaći k-bojenje u zadanom broju iteracija ili kada pronađe optimalno bojenje. S obzirom na navedenu doradu, potrebno je odrediti početni broj boja od kojeg algoritam kreće. Broj boja s kojima se graf sigurno može obojiti je n, broj njegovih vrhova, jer se svaki vrh može obojiti različitom bojom. Problem ovakvog algoritma je predugo trajanje jer je broj vrhova grafa često višestruko veći od optimalnog broja boja. Rješenje je postignuto jednim pokretanjem pohlepnog algoritma, opisanog u poglavlju 2.1, na početku algoritma roja čestica i postavljanjem početnog broja boja na onaj broj dobiven pohlepnim algoritmom.

Parametri s kojima je algoritam pokretan slični su onima iz članka [8]:

- $N = |S|$ = veličina roja = 10
- $prob_p$ = vjerojatnost da će čestica u pojedinoj iteraciji napraviti pomak prema svom do tada najboljem pronađenom rješenju = 40%
- $prob_n$ = vjerojatnost da će čestica u pojedinoj iteraciji napraviti pomak prema najboljem rješenju svojih susjeda = 50%
- $prob_r = 100\% - prob_p - prob_n$ = vjerojatnost da će čestica u pojedinoj iteraciji napraviti pomak prema nasumično generiranom rješenju = 10%
- MAX_{PSO} = maksimalan broj iteracija algoritma roja čestica = $5|V|$
- MAX_{TABU} = maksimalan broj iteracija algoritma tabu traženja = $5|V|$
- t = broj iteracija koje rješenje mora provesti u tabu listi = $U[0-9] + 0.6f(C)$, gdje je $U[0-9]$ nasumičan broj između 0 i 9, a C je trenutno promatrano bojenje.

Algorithm 8: Particle swarm algorithm combined with tabu search

Data: graph $G = (V, E)$
set k to the number of colors of the graph obtained by running the greedy algorithm;
while *the coloring of k colors is not found* **do**
  create the swarm S of N colorings;
  generate numbers $r_1, r_2, r_3 \in [0, 1]$ at random;
  $iter = 0$;
  while $iter < MAX_{PSO}$ **do**
    for *each* $C \in S$ **do**
      if $f(C) < f^p(C)$ **then**
        $C^p = C$;
      end
    end
    for *each* $C \in S$ **do**
      $C^g = \underset{C'}{\text{argmin}}\{f(C') : C' \in Neighbor(C) \subset S\}$;
    end
    for *each* $C \in S$ **do**
      generate a random number $\delta \in [0, 1$;
      if $\delta < prob_p$ **then**
        $C = C + r_1(C^p - C)$;
      end
      if $prob_p <= \delta < prob_p + prob_n$ **then**
        $C = C + r_2(C^g - C)$;
      end
      if $prob_p + prob_n <= \delta$ **then**
        $C = C + r_3(C^r - C)$;
      end
      run tabu search algorithm to improve coloring C;
    end
    if $f(C^g) = 0$ **then**
      $k = k - 1$;
      break;
    end
    $iter = iter + 1$;
  end
end

Poglavlje 4

Analiza i usporedba rezultata

Algoritmi iz prethodnog poglavlja implementirani su u programskom jeziku Java. Grafovi na kojima se algoritmi testiraju preuzeti su s [1], a radi se o bazi grafova DIMACS (Center for Discrete Mathematics and Theoretical Computer Science). Sve potrebne informacije o pojedinom grafu zadane su u datoteci koja sadrži tri vrste linija:

- linija koja počinje slovom c sadrži komentare koji opsuju graf
- linija koja počinje slovom p sadrži broj vrhova i bridova grafa
- linija koja počinje slovom e opisuje jedan brid grafa na način da sadrži dva vrha koja promatrani brid povezuje.

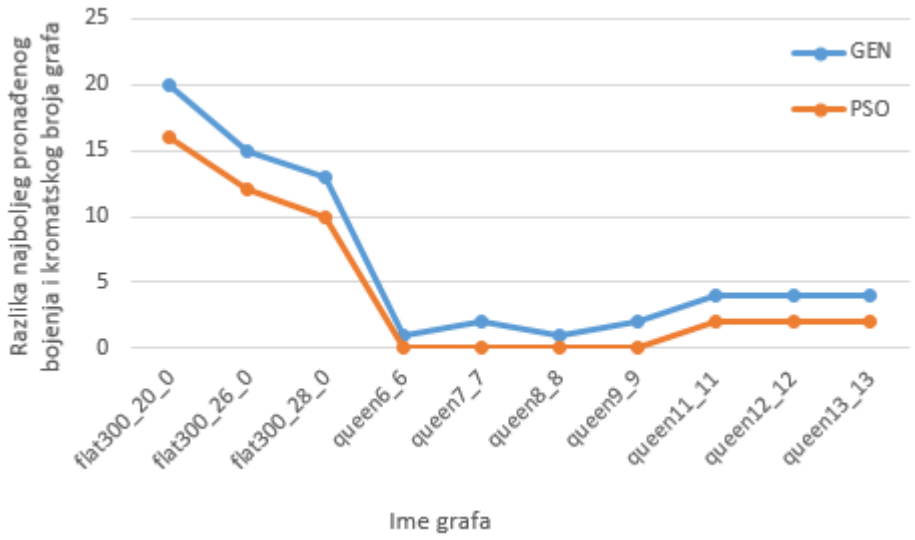
Dodatno, generirana je datoteka koja sadrži imena svih grafova i kromatski broj onih grafova za koje je on poznat. Ona se koristi kako bi se smanjilo trajanje algoritama - algoritam staje ako je pronašao pravilno bojenje koje sadrži točno onoliko boja koliki je kromatski broj grafa.

Nakon učitavanja grafa, pokreće se željeni meta-heuristički algoritam. Dva osnovna svojstva koja će biti promatrana su najmanji pronađeni broj boja te vrijeme potrebno za pronalazak bojenja s upravo tim brojem boja. Kako su za svaki graf oba implementirana algoritma pokretana više puta, navodi se prosječno vrijeme potrebno za pronalazak najboljeg bojenja. Rezultati pokretanja algoritama prikazani su u tablici 4.1. Kratica GEN označava stupce u kojima su prikazani rezultati pokretanja genetskog algoritma, a kratica PSO stupce u kojima su prikazani rezultati pokretanja algoritma roja čestica u kombinaciji s tabu traženjem. Znakom ? u stupcu $\chi(G)$ tablice označeno je da za pojedini graf nije poznat pripadni kromatski broj.

Ime grafa	Broj vrhova	Broj bridova	$\chi(G)$	Broj boja (GEN)	Vrijeme (GEN)	Broj boja (PSO)	Vrijeme (PSO)
david	87	812	11	11	1 s	11	1 s
huck	74	602	11	11	1 s	11	1 s
jean	80	508	10	10	1 s	10	1 s
DSJC125.1	125	736	?	7	1 s	6	1 s
DSJC250.1	250	3218	?	12	1 s	9	28 min
DSJR500.1	500	3555	?	14	1 s	12	27 min
DSJC1000.1	1000	49629	?	30	15 s	25	56 min
flat300_20_0	300	21375	20	40	11 min	36	47 min
flat300_26_0	300	21633	26	41	9 min	38	18 min
flat300_28_0	300	21695	28	41	3 min	38	3 min
fpsol2.i.1	496	11654	65	65	1 s	65	8 min
fpsol2.i.2	451	8691	30	30	1 s	30	8 min
inithx.i.1	864	18707	54	54	4 s	54	15 s
inithx.i.2	645	13979	31	31	3 s	31	5 min
mulsol.i.1	197	3925	49	49	1 s	49	25 s
mulsol.i.2	188	3885	31	31	1 s	31	1 min
mulsol.i.3	184	3916	31	31	1 s	31	30 s
myciel3	11	20	4	4	1 s	4	1 s
myciel4	23	71	5	5	1 s	5	1 s
myciel5	47	236	6	6	1 s	6	1 s
myciel6	95	755	7	7	1 s	7	1 s
myciel7	191	2360	8	8	1 s	8	2 s
queen5_5	25	320	5	5	1 s	5	1 s
queen6_6	36	580	7	8	1 s	7	2 s
queen7_7	49	952	7	9	2 s	7	10 s
queen8_8	64	1456	9	10	1 s	9	25 s
queen9_9	81	2112	10	12	1 s	10	40 s
queen11_11	121	3960	11	15	1 s	13	30 s
queen12_12	144	5192	12	16	8 s	14	50 s
queen13_13	169	6656	13	17	7 s	15	10 min
queen15_15	225	10360	?	20	10 s	18	25 s
queen16_16	256	12640	?	21	3 min	19	9 s
zeronin.i.1	211	4100	49	49	1 s	49	2 min
zeronin.i.2	211	3541	30	30	1 s	30	3 min
zeronin.i.3	206	3540	30	30	1 s	30	1 min

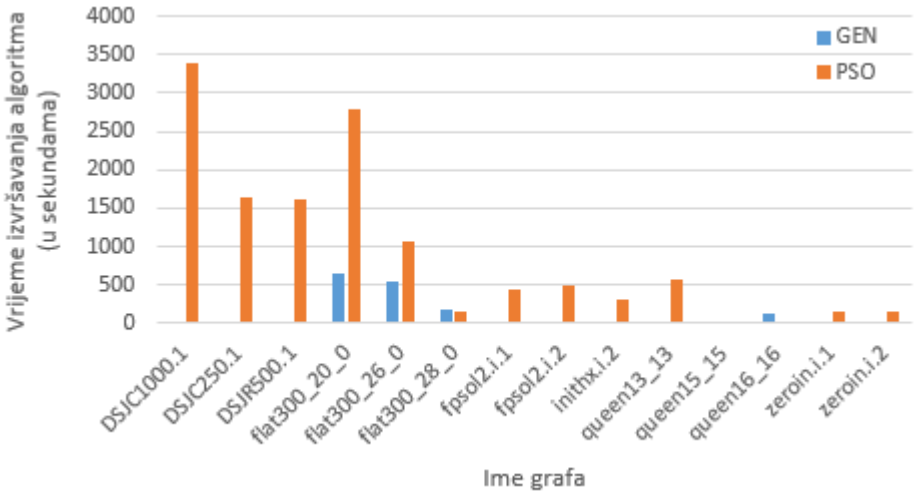
Tablica 4.1: Rezultati pokretanja genetskog algoritma i algoritma roja čestica na testnim grafovima

Za većinu grafova oba algoritma su pronašla optimalna rješenja. To nije bio slučaj samo za kompleksnije grafove, gdje niti jedan od algoritama nije pronašao optimalno rješenje. Za te grafove, razlike između broja boja najboljeg pronađenog pravilnog bojenja i kromatskog broja grafa prikazane su na slici 4.1. Algoritam roja čestica je za svaki takav graf pronašao rješenje koje koristi manji broj boja od rješenja kojeg je pronašao genetski algoritam.



Slika 4.1: Razlika između pronađenog broja boja i kromatskog broja za grafove za koje je poznat kromatski broj, a barem jedan algoritam nije pronašao optimalno bojenje

Međutim, cijena pronalaska boljeg rješenja je dulje vrijeme izvršavanja algoritma roja čestica. Na slici 4.2 prikazano je vrijeme izvršavanja oba algoritma za grafove za koje je utvrđena veća razlika u vremenima izvršavanja. Algoritam roja čestica ima dulje vrijeme izvršavanja za grafove za koje je pronašao bolja bojenja od genetskog algoritma, ali i za grafove za koje su algoritmi pronašli bojenja s jednakim brojem boja.



Slika 4.2: Vrijeme izvršavanja algoritama za grafove za koje je primijećena veća razlika u vremenu izvršavanja

S obzirom da svaki od dva implementirana algoritma ima jednu bitnu karakteristiku bolju od onog drugog, o konkretnom problemu kojeg algoritam treba riješiti ovisi koji je algoritam bolje odabrati. Ako je važno pronaći bojenje sa što manjim brojem boja, ako je moguće i optimalno, a dozvoljeno je za to potrošiti više vremena, algoritam roja čestica u kombinaciji s tabu traženjem je bolji odabir. S druge strane, ako je važno u što kraćem vremenu pronaći neko dovoljno dobro bojenje, tada je bolje koristiti genetski algoritam.

Bibliografija

[1] *Graph Coloring Instances*, https://mat.tepper.cmu.edu/COLOR/instances.html, posjećena 12.12.2020.

[2] *Hungarian Algorithm*, https://github.com/aalmi/HungarianAlgorithm, posjećena 15.1.2021.

[3] H. M. Harmanani, *A Method for the Minimum Coloring Problem Using Genetic Algorithms*, (2006), https://www.researchgate.net/publication/258452604_A_Method_for_the_Minimum_Coloring_Problem_Using_Genetic_Algorithms.

[4] H. W. Kuhn i B. Yaw, *The Hungarian method for the assignment problem*, Naval Res. Logist. Quart (1955), 83–97.

[5] R. M. R. Lewis, *A Guide to Graph Colouring*, Springer, 2016.

[6] Sean Luke, *Essentials of Metaheuristics*, Lulu, 2013.

[7] Robert Manger, *Teorija složenosti*, 2011, http://web.studenti.math.pmf.unizg.hr/~manger/tr/TR-V.pdf.

[8] J. Qin, Yin Y. i X. Ban, *Hybrid Discrete Particle Swarm Algorithm for Graph Coloring Problem*, Journal of computers (2011), br. 6, 1175–182.

Sažetak

Problem bojenja grafa sastoji se od pridruživanja boje svakom vrhu grafa tako da su susjedni vrhovi obojeni različitim bojama, pri čemu je broj boja minimalan. Takav minimalan mogući broj boja naziva se kromatski broj grafa. Kako je problem bojenja grafa NP-potpun, ne postoji algoritam koji bi za bilo koji graf u realnom vremenu odredio pripadni kromatski broj. Zato se za rješavanje tog problema koriste meta-heuristički algoritmi.

U ovom radu opisana su i implementirana dva meta-heuristička algoritma za određivanje kromatskog broja grafa - genetski algoritam i algoritam roja čestica u kombinaciji s tabu traženjem. Oba algoritma su populacijska i kao prostor rješenja koriste prostor potpunih, ali ne nužno pravilnih rješenja. Genetski algoritam iterativno, primjenom operatora križanja, mutacije i selekcije iz postojećih rješenja kreira nova, bolja rješenja. Algoritam roja čestica bazira se na kretanju čestice prema svom do sada pronađenom najboljem rješenju, najboljem pronađenom rješenju svojih susjeda ili nasumično generiranom rješenju kako bi istražila prostor rješenja i pronašla ono najbolje. Nakon pomaka pojedine čestice, pokreće se algoritam tabu traženja koji nastoji dodatno poboljšati trenutno rješenje.

Algoritmi su testirani na grafovima baze DIMACS. Rezultati pokazuju da oba algoritma za većinu grafova pronalaze optimalna rješenja. Za kompleksnije grafove za koje algoritmi ne pronalaze optimalna rješenja, algoritam roja čestica pronalazi pravilna bojenja s manjim brojem boja nego genetski algoritam, ali mu je za to potrebno dulje vrijeme. Zbog toga odabir algoritma za daljnje korištenje ovisi o konkretnom problemu koji je potrebno riješiti, tj. je li važno da algoritam pronađe rješenje sa što manjim brojem boja ili da u što kraćem vremenu pronađe dovoljno dobro rješenje.

Summary

The graph coloring problem consists of associating the color to each vertex of the graph so that the adjacent vertices are painted with different colors and the number of colors is minimal. Such a minimum number of colors is called the chromatic number of graph. Since the graph coloring problem is NP-complete, there is no algorithm that would determine the chromatic number of any graph in real time. That is why meta-heuristic algorithms are used to solve this problem.

Two meta-heuristic algorithms for determining the chromatic graph number are described and implemented in this thesis - genetic algorithm and hybrid particle swarm algorithm in combination with the taboo search. Both algorithms are population-based and use the solution space of complete improper colorings. The genetic algorithm iteratively, by applying the crossover, mutation and selection operators creates new, better solutions from existing solutions. The particle swarm algorithm is based on the movement of a particle towards its best solution so far, the best solution found by its neighbours or a randomly generated solution in order to explore the solution space and find the best coloring. After the movement of a particle, the tabu search algorithm attempts to further improve the current solution.

Algorithms were tested on the DIMACS graphs. The results show that both algorithms find optimal solutions for most graphs. For complex graphs the particle swarm algorithm finds proper coloring with fewer colors than the genetic algorithm, but it takes longer to find it. Therefore, the choice of the algorithm for further use depends on the specific problem that needs to be solved, i.e. whether it is important for the algorithm to find a solution with as few colors as possible or to find a good enough solution as soon as possible.

Životopis

Lea Bundalo rođena je 29. prosinca 1996. u Bjelovaru, gdje završava osnovnu školu i 2011. godine upisuje prirodoslovno-matematičku gimnaziju. Za vrijeme srednjoškolskog obrazovanja sudjeluje na natjecanjima iz matematike, fizike i logike. Također, pohađa teorijski smjer srednje glazbene škole Vatroslava Lisinskog u Bjelovaru. 2015. godine upisuje Preddiplomski sveučilišni studij Matematika na Prirodoslovno-matematičkom fakultetu Sveučilišta u Zagrebu, a 2018. godine diplomski sveučilišni studij Računarstvo i matematika. Za vrijeme druge godine preddiplomskog i prve godine diplomskog studija radi kao demonstrator kolegija Programiranje 1 i 2.