

MINIMAL COLOR SUM

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Marija Papović, Luka Miletić

Sadržaj

Uvod.....	2
Opšti rad I opis algoritma.....	3
Brute-force algoritam.....	3
Rešenja zasnovana na metaheuristikama.....	4
Lokalna pretraga.....	4
Simulirano kaljenje.....	5
Metoda promenljivih sredina (VNS).....	6
Genetski algoritam.....	7
Hibridni algoritam.....	9
Eksperimentalni rezultati nad nasim random generisanim instacama.....	9
Eksperimentalni rezultati nad primerima instanci iz prakse.....	11
Zaključak.....	16
Literatura:.....	17

Uvod

Bojenje grafova(GCP) je jedan od najpoznatijih i najviše rešavanih NP-teških problema kombinatorne optimizacije i ima veliki praktičan značaj.

Problem minimalne sume bojenja grafova (MSCP) je NP-težak problem izveden iz problema bojenja grafova, a uveli su ga Kubicka and Schwenk još 1989.godine .

Primenjuje se kod problema rasporeda, upravljanja skladištem, alokacije frekvencija u mobilnoj mreži, alokacije registara u optimizaciji kompajlera.

Zadatak ovog problema je pronaći pravilno bojenje čvorova grafa koristeći boje predstavljene prirodnim brojevima $\{1, 2, \dots\}$, tako da zbir boja dodeljenih čvorovima bude minimalan, a da nikoja dva susedna čvorova nisu obojena u istu boju.

Opšti rad I opis algoritma

U našem radu koristili smo dva tipa rešavanja problema.

Prvo rešenje jeste Brute-force algoritam, dok se drugi tip rešenja zasniva na metaheuristikama.

Kod oba pristupa generišemo permutacije koje predstavljaju redosled bojenja čvorova datog grafa, zatim bojimo čvorove tim redosledom najmanjom dopustivom bojom (prvim slobodnim brojem koju nisu iskoristili njegovi susedni čvorovi), računamo sumu tog bojenja, a najbolje rešenje čuvamo u globalnoj promenljivoj.

Brute-force algoritam

U ovom pristupu je potrebno za n čvorova generisati sve permutacije dužine n (ima ih $n!$).

Prvi od njih odgovara generisanju varijacija bez ponavljanja, gde se na svaku od n pozicija mogu postaviti elementi od 0 do $n-1$.

Navedena funkcija redom popunjava elemente niza dužine n . Promenljiva i čuva indeks naredne pozicije u nizu koju treba popuniti. Kada njena vrednost dostigne n , ceo niz je popunjen i tada se ispisuje generisana permutacija. Na poziciju i treba postavljati jedan po jedan element skupa od 0 do $n-1$ koji nije već postavljen na neku od prethodnih pozicija u nizu.

Pretraga tog dela niza zahteva linearno vreme i da se to ne bi radilo, tj. da bi se ubrzala procedura održava se i logički niz upotrebljen koji čuva informacije o tome koji elementi već postoje na prethodnim pozicijama: u nizu upotrebljen, na poziciji k je vrednost 1 ako i samo ako je element k već postavljen u deo niza pre pozicije i (tako se u konstantnom vremenu može proveriti da li je neki element već uključen u permutaciju). Prolazi se redom kroz sve kandidate (vrednosti od 0 do $n-1$), onaj koji nije već uključen u permutaciju postavlja se na mesto i , beleži se da je taj element upotrebljen i nastavlja se rekursivno generisanje permutacije. Nakon toga, element se povlači tako što se zabeleži da više nije upotrebljen i prelazi se na narednog kandidata.¹

Kada generišemo permutaciju, bojimo je najmanjom bojom (brojem) dostupnim u nizu *dopustive_boje* , a ako je zbir datog bojenja bolji (manji) od globalnog najboljeg, tada ažuriramo najbolje rešenje i prelazimo na narednu permutaciju, sve dok ne ispitamo sve permutacije.

Kako je za ovaj algoritam potrebno ispitati sve permutacije, testiranje na vecim grafovima (sa 12 i više čvorova) zahteva jako puno vremena, pa ovaj algoritam radi dobro samo na grafovima manjih dimenzija i garantuje nalaženje najboljeg rešenja jer ispituje sve moguće ispravne kombinacije bojenja grafa.

1 Iz knjige „Programiranje 2”, Mladen Nikolić i Predrag Janjičić

Rešenja zasnovana na metaheuristikama

Glavna stvar kod implementacija rešenja zasnovanih na metaheuristikama jeste funkcija cilja. Vrednost funkcije cilja za proizvoljno rešenje $x \in X$ izračunava se na sledeći način:

- Inicijalizuje se vektor x_boje dužine n , čiji su svi elementi nule (nula označava da čvoru još uvek nije dodeljena boja);
- Za čvor v_i koji je na redu za bojenje na osnovu zadatog vektora rešenja, određuje se vektor $zauzete_boje$, koji sadrži boje koje su već zauzete njegovim obojenim susedima;
- Nakon određivanja vektora $zauzete_boje$, koji sadrži već zauzete boje, čvoru v_i se dodeljuje boja jednaka $\min\{q : q \in Z^+, q \text{ ne pripada vektoru } zauzete_boje\}$;
- Ovaj postupak se ponavlja za sve čvorove v_i grafa G , *sve dok se ne ažurira ceo vektor x_boje* ;
- Vrednost funkcije cilja za rešenje x se dobija tako što se sabere svi elementi vektora x_boje , tj. $f(x) = \sum x_boje(i)$;

Ideja potiče iz pohlepnog algoritma za bojenje čvorova grafa. Složenost izračunavanja funkcije cilja će najviše zavisiti od broja grana grafa koji se testira. To znači da će vreme izvršavanja za grafove sa jako velikim brojem grana biti znatno duže od grafova sa manjim brojem grana.

Metode koje smo mi koristili:

1. Lokalna pretraga
2. Simulirano kaljenje
3. Metoda promenljivih okolina (VNS)
4. Genetski algoritam (turnirska selekcija)
5. Hibridni algoritam

Lokalna pretraga

Lokalna pretraga (eng. Local search) je jedna od heurističkih metoda za rešavanje NP teških problema optimizacije.

Algoritam počinje od jednog rešenja (koje mi inicijalizujemo na početku), a zatim iterativno prelazi na susedno rešenje (susedstvo je skup svih potencijalnih rešenja koja se razlikuju od trenutnog rešenja u minimalnoj mogućoj meri) . Ovo zahteva da se relacija susedstva definiše na prostoru za pretragu. Tipično, svako rešenje trenutnog kandidata ima više od jednog susednog rešenja. Izbor koji će se izabrati vrši se korišćenjem samo informacija o rešenjima u okolini trenutnog kandidata, pa otuda i naziv lokalna pretraga. Ukoliko je vrednost njegove funkcije cilja bolja od vrednosti funkcije cilja trenutnog rešenja, ažurira se trenutno rešenje.

Kada nema suseda za poboljšanje, lokalna pretraga se zaglavi u lokalno optimalnoj tački.

Implementacija lokalne pretrage:

- ➔ Generiši početno rešenje s
- ➔ Inicijalizuj najbolje rešenje $f^* = f(s)$
- ➔ Dok nije ispunjen kriterijum zaustavljanja, radi sledeće korake:
 - Generiši novo rešenje s' u okolini s
 - Ako je $f(s') < f(s)$ onda $s = s'$
 - Ako je $f(s') < f^*$ onda $f^* = f(s')$
- ➔ Ispisati vrednost rešenja f^*

U našem primeru početno rešenje jeste random generisana permutacija od n elemenata, a novo rešenje dobijamo tako što u datoj permutaciji zamenimo mesta dvema random izabranim pozicijama. Za kriterijum zaustavljanja uzeli smo broj iteracija koji korisnik unese.

Napomena : Lokalna pretraga ne daje garanciju da je bilo koje rešenje optimalno. Pretraga se može prekinuti kada se ispuni kriterijum zaustavljanja, a to može biti nakon određenog vremenskog ograničenja ili kada se najbolje rešenje do sada nije poboljšalo u određenom broju koraka.

Simulirano kaljenje

Algoritam simuliranog kaljenja (eng. *simulated annealing*, skraćeno SA) je zasnovan na procesu kaljenja čelika, čiji je cilj oplemenjivanje metala tako da on postane čvršći. Prvi korak u kaljenju čelika je zagrevanje do određene temperature, a zatim, nakon kratkog zadržavanja na toj temperaturi, počinje postepeno hlađenje. Pritom treba voditi o brzini hlađenja, jer brzo hlađenje može da uzrokuje pucanje metala.²

Simulirano kaljenje predstavlja optimizovani vid lokalne pretrage, a razlikuje se u tome što omogućava da se prihvati novo rešenje čak i ako njegova vrednost nije bolja od prethodnog. Na taj način se smanjuje verovatnoća konvergencije niza ka lokalnom optimumu koje nije i globalno. To se postiže tako što ukoliko vrednost funkcije cilja novog rešenja nije bolja od vrednosti funkcije cilja trenutnog, upoređuju se vrednosti unapred definisane funkcije p i proizvoljno izabrane vrednosti q iz intervala $(0,1)$. Ako je $p > q$, trenutno rešenje se ažurira novoizabranim. Takođe se, po potrebi, ažurira i vrednost najboljeg dostignutog rešenja.

Implementacija simuliranog kaljenja:

2 Preuzeto sa: <https://poincare.matf.bg.ac.rs/~denis.alicic/ri/5.html>

- ➔ Generiši početno rešenje s
- ➔ Inicijalizuj najbolje rešenje $f^* = f(s)$
- ➔ Dok nije ispunjen kriterijum zaustavljanja, radi sledeće korake:
 - Generiši novo rešenje s' u okolini s
 - Ako je $f(s') < f(s)$ onda $s = s'$
 - Ako je $f(s') \geq f(s)$ onda:
 - Ažurirati vrednost opadajuće funkcije p
 - Izabrati proizvoljno q iz intervala $[0,1]$
 - Ako je $p > q$ onda $s = s'$
 - Ako je $f(s') < f^*$ onda $f^* = f(s')$
- ➔ Ispisati vrednost rešenja f^*

Početno rešenje, funkcija cilja, funkcija gde menjamo trenutnog kandidata kao I kriterijum zaustavljanja se poklapaju sa funkcijama I kriterijumima u lokalnoj pretrazi. Opadajuću funkciju p definišemo na sledeći način:

$p = 1 / i^{1/2}$, gde je i trenutni broj iteracije (krećemo brojanje od 1 kako ne bi delili nulom).

Napomena : Simulirano kaljenje nam daje u većini slučajeva bolje rešenje od lokalne pretrage, ali i dalje nemamo garanciju da je rešenje koje dobijamo na kraju globalno najbolje.

Metoda promenljivih sredina (VNS)

Metoda promenljivih sredina (eng. *Variable neighborhood search*), koju su predložili Mladenović i Hansen 1997. godine, je metaheuristički metod za rešavanje skupa kombinatornih optimizacijskih i globalnih optimizacijskih problema. Istražuje udaljene sredine trenutno postojećeg rešenja i odatle prelazi na novo ako i samo ako je napravljeno poboljšanje. Metoda lokalne pretrage se više puta primenjuje da bi se od rešenja u okruženju došlo do lokalnog optimizma.

Glavna stvar po kojoj se ova metoda razlikuje od prethodnih jeste faza razmrđavanja koja nam omogućava da istražimo okoline koje su udaljenje od trenutne sredine koju istražujemo i na taj način smanjujemo verovatnoću da ćemo dobiti lokalni, a ne globalni optimum i utiče na stepen diverzifikacije.

Korak razmrđavanja se realizuje na sledeći način:

Okolina $N_k(x)$ rešenja x , za $1 \leq k \leq k_{\max}$, se definiše kao skup svih rešenja koja se od x dobijaju sledećim postupkom. Na slučajan način se izabere k različitih pozicija iz skupa $\{1, 2, \dots, n\}$. Zatim se vektor rešenja x prelomi na tih k mesta i svakom delu se obrne redosled elemenata. Primenom koraka razmrđavanja dobija se novi vektor rešenja $x' \in N_k(x)$.

Ovakvom fazom razmrđavanja će se očuvati dopustivost rešenja jer će svaki novi vektor x' biti jedna nova permutacija brojeva $1, 2, \dots, n$ na osnovu koje se određuje jedinstveni redosled dodeljivanja boja čvorovima. Ideja za ovakvu implementaciju razmrđavanja je da se na neki način očuva redosled

dodeljivanja boja, ali i da se „razdrma” toliko da se ispituju i neke dalje oblasti dopustivog prostora X . Zbog toga se prekidanjem veze na k mesta dobijeni lanci spajaju sa drugim lancima vektora rešenja, čime se obezbeđuje jači stepen diverzifikacije, dok se sa druge strane unutar jednog lanca rotiranjem redosleda obezbeđuje slabija diverzifikacija.

Implementacija simuliranog kaljenja:

- ➔ Generiši početno rešenje s
- ➔ Nađemo vrednost $f(s)$
- ➔ Dok nije ispunjen kriterijum zaustavljanja, radi sledeće korake:
 - Sve dok k ne dostigne vrednost k_{\max} radi sledeće korake:
 - Generiši novo rešenje s' koristeći opisanu funkciju razmrđavanja
 - Ako je $f(s') < f(s)$ ili $(f(s') = f(s) \text{ I } \text{random_br} < \text{move_prob})$ onda:
 - $s = s'$
 - $f(s) = f(s')$
 - prelazimo na drugu iteraciju
- ➔ Ispisati vrednost rešenja $f(s)$

K nam na početku svake iteracije kreće od vrednosti 1, a vrednost k_{\max} unosimo na ulazu, kao i konstantu move_prob .

random_br je broj koji nasumično generišemo na intervalu $[0,1]$.

Napomena : VNS nam daje bolje rešenje od lokalne pretrage i simuliranog kaljenja, a u zavisnosti od vrednosti k_{\max} i move_prob daje drugačija rešenja na istim test primerima. Kod VNS metode nam je najmanja verovatnoća da ćemo upasti u lokalni optimum iz razloga jer nam je stepen diverzifikacije najveći, ali i dalje nismo sigurni da je dato rešenje globalno najbolje.

Genetski algoritam

Genetski algoritam (eng. *genetic algorithm* - GA) jedan je od evolutivnih algoritama inspirisanih Darwinovom teorijom evolucije, koja kaže da unutar jedne populacije najčešće opstaju najbolje prilagođene jedinke (prilagođenost se određuje pomoću *fitness* funkcije). Pripada stohasticnim algoritmima pretrage sa heuristikom. Razvijen je 70-ih godina prošlog veka u Americi, a ključni autori su bili J. Holland, K. DeJong i D. Goldberg.

Ono šta karakteriše GA jeste:

- **Reprezentacija rešenja** (kodiranje i dekodiranje)
- **Mutacija**
- **Ukrštanje**
- **Selekcija**

Resenje evolutivnog algoritma kodira se pomocu hromozoma koji je sačinjen od gena, a geni su karakteristike rešenja. Predstavljanje našeg problema i rešenja kodiranjem jedan je od najtežih problema kod evolutivnih algoritama. Najčesce kod predstavlja niz fiksne dužine koji se sastoji iz biranih ili celih brojeva(kao u našem slučaju).

Selekcija predstavlja izbor jedinki koje će učestvovati u ukrštanju I ostaviti svoje potomstvo. Prateći Darwinovu teoriju, u selekciji najčesce (ali ne i uvek) treba da učestvuju najkvalitetnije jedinke čime će povećati prisutnost dobrih gena u narednim generacijama.

Potrebno je uvesti i pojam selekcionog pritiska. Selekcioni pritisak predstavlja vreme potrebno da se proizvede uniformna populacija jedinki - da se smanji genetski diverzitet toliko da preovladavaju karakteristike najboljih jedinki. Ukoliko se ovo dogodi previše, odnosno ukoliko je selekcioni pritisak previše visok, diverzitet gena će se toliko smanjiti da će se brzo konvergirati ka nekom maksimumu (najčesce lokalnom).

Mi smo u našem radu koristili turnirsku I ruletsku selekciju.

Turnirska - slučajno se bira skup jedinki i najbolja od njih (ona koja ima najbolju vrednost funkcije prilagodjenosti) se bira za selekciju.

Ruletska - jedinkama sa boljom funkcijom pogodnosti se daje veća verovatnoća da budu izabrane.

Ukrštanje predstavlja kombinovanje gena dva roditelja, pri čemu se dobijaju dva nova potomka. Postoje razni načini za ukrštanje, a mi smo implementirali ukrštanje prvog reda.

Opšta šema ukrštanja prvog reda:

1. Odaberi segment hromozoma prvog roditelja
2. Iskopirati ovaj segment u prvo dete
3. Iskopirati preostale vrednosti (brojeve) tako da:
 - Kopiranje počinje desno od kopiranog segmenta
 - Korištenjem redosleda datog drugim roditeljem
4. Slično se radi i za drugo dete

Mutacija predstavlja slučajnu promenu nekih gena određene jedinke. Ona se na svaku jedinku primenjuje sa nekom malom verovatnoćom. Ukoliko je verovatnoća mutacije premala, ili se uopšte ne primenjuje, može doći do preuranjene lokalne konvergencije. Ako je vrednost velika, genetski algoritam će prostor rešenja pretraživati bez jasnog fokusa.

Mi smo u našem radu koristili dva različita načina za mutiranje.

Mutacija zasnovana na zameni - izaberu se dve vrednosti na slučajan način i zamene se, na taj način održavamo većinu uređenja.

Mutacija zasnovana na mešanju - izabere se podskup pozicija na slučajan način I slučajno se reorganizuju vrednosti na tim pozicijama (pozicije ne moraju da budu uzastupne).

Napomena:

Mutacija zasnovana na mešanju nam se pokazala kao bolja pri testiranjima na istim instancama.

Elitizam- princip koji se zasniva na ideji da se najbolja ili više najboljih jedinki prenesu direktno u narednu generaciju.

Osnovna verzija genetskog algoritma se može prikazati sledećim pseudokodom:

- Učitavanje ulaznih podataka
- Generisanje početne populacije
- Određivanje funkcije prilagođenosti populacije
- Dok nije ispunjen kriterijum zaustavljanja:
 - Selekcija jedinki za primenu genetskih operatora
 - Ukrštanje za izabrane parove jedinki
 - Mutacija izabranih jedinki
 - Određivanje funkcije prilagođenosti populacije
- Ispis najboljeg rešenja

Funkcija prilagođenosti populacije (fitness funkcija) u našem slučaju predstavlja recipročnu vrednost sume obojenog grafa, što je fitness funkcija manja, zbir je veći pa je I jedinka bolja.

Hibridni algoritam

Hibridni algoritam predstavlja metodu koja kombinuje neke dve metaheuristike. Postoji mnogo hibridnih algoritama a mi smo ovde koristili jedan od poznatijih koji se zasniva na kombinaciji **genetskog algoritma** i **algoritma simuliranog kaljenja**. Posle svake iteracije u radu genetskog algoritma, pronalazi se najbolja jedinka, jedinka koja je najprilagodjenija i nad tom jedinkom se poziva algoritam simuliranog kaljenja u svrhu poboljšavanja tog najboljeg rešenja. Poziv simuliranog kaljenja nad tom jedinkom ne treba trajati predugo zbog fer upoređivanja sa drugim metodama čega smo bili svesni prilikom razvijanja odnosno pozivanja ove metode.

Eksperimentalni rezultati nad našim random generisanim instacama

Testiranja su vršena na računaru sa procesorom Intel Core i5-8250U(8) 3.4 GHz i 8GB RAM memorije.

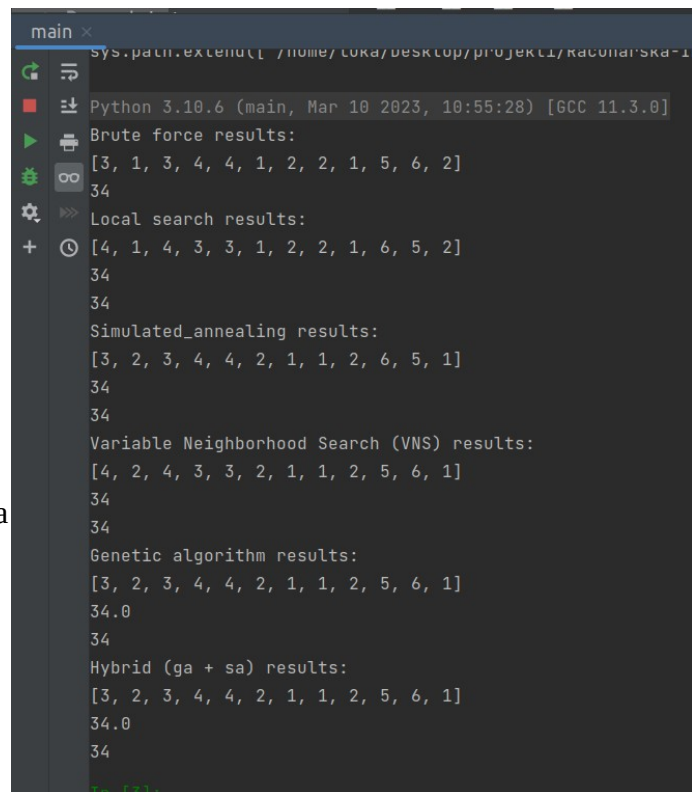
Testirane su razvijene metoda nad dve instace grafa koje su random generisani.

Prvi graf: small_random.txt ima 12 čvorova. Nad njim je pokrenut i brute force algoritam jer je graf manjih dimenzija i prihvatljivo je vreme izvršavanja. Sve metode su pronašle optimalno rešenje, isto kao i rešenje koje je dobijeno brute force algoritmom, sto nam ukazuje da metode ispravno rade.

Rezultati su sledeći:

1. Brute force(Optimal solution) : $Opt_{sol} = 34$
2. Lokalna pretraga: Sol = 34
3. Simulirano kaljenje: Sol = 34
4. Metoda promenljivih okolina: Sol = 34
5. Genetski algoritam: Sol = 34
6. Hibridni algoritam: Sol = 34

Pored rešenja, na slici je prikazan i način bojenja tog grafa predstavljen u vidu niza.(slika1)



```
main x
sys.path.extend(['../home/loka/desktop/projekti/racunar'ska-1

Python 3.10.6 (main, Mar 10 2023, 10:55:28) [GCC 11.3.0]

Brute force results:
[3, 1, 3, 4, 4, 1, 2, 2, 1, 5, 6, 2]
34

Local search results:
[4, 1, 4, 3, 3, 1, 2, 2, 1, 6, 5, 2]
34
34

Simulated_annealing results:
[3, 2, 3, 4, 4, 2, 1, 1, 2, 6, 5, 1]
34
34

Variable Neighborhood Search (VNS) results:
[4, 2, 4, 3, 3, 2, 1, 1, 2, 5, 6, 1]
34
34

Genetic algorithm results:
[3, 2, 3, 4, 4, 2, 1, 1, 2, 5, 6, 1]
34.0
34

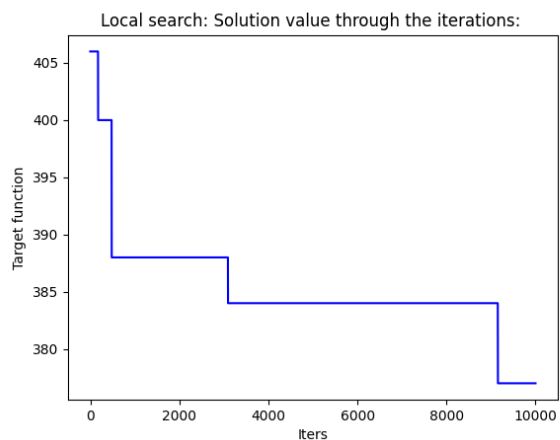
Hybrid (ga + sa) results:
[3, 2, 3, 4, 4, 2, 1, 1, 2, 5, 6, 1]
34.0
34
```

Slika 1

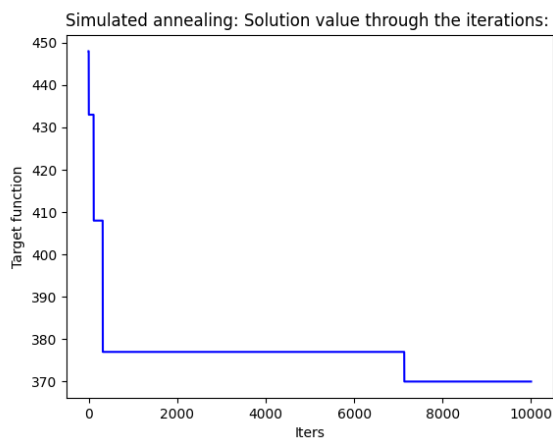
Drugi graf: random_graph.txt ima 50 čvorova. Metode su pokrenute samo po jednom i zabeleženi su sledeći rezultati. Imamo grafički prikaz rada metode, prikaz zavisnosti funkcije cilja koju je metoda nalazila tokom iteracija.

Lokalna pretraga: Sol = 377

Simulirano kaljenje: Sol = 370



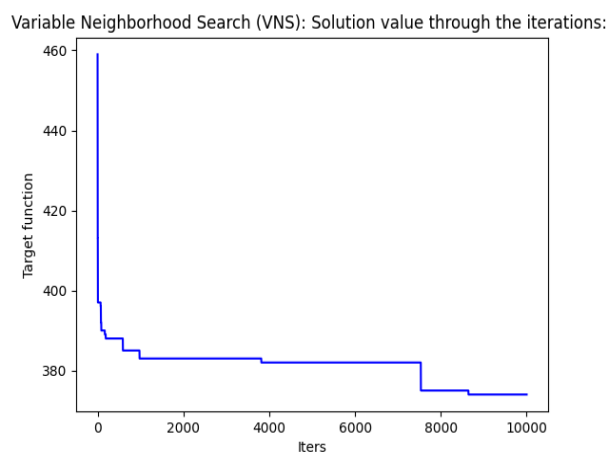
Slika 2



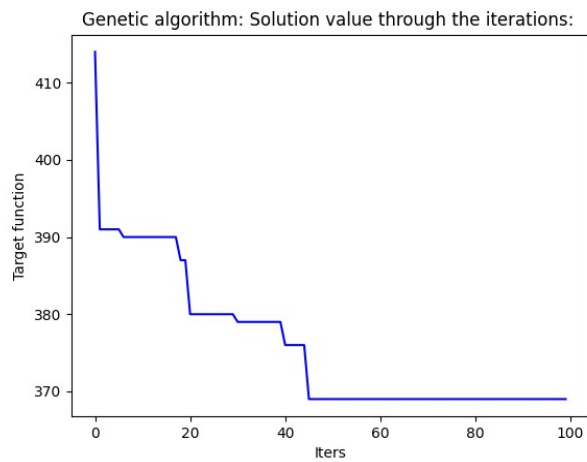
Slika 3

Metoda promenljivih okolina: Sol = 376

Genetski algoritam(ruletska selekcija): Sol = 369

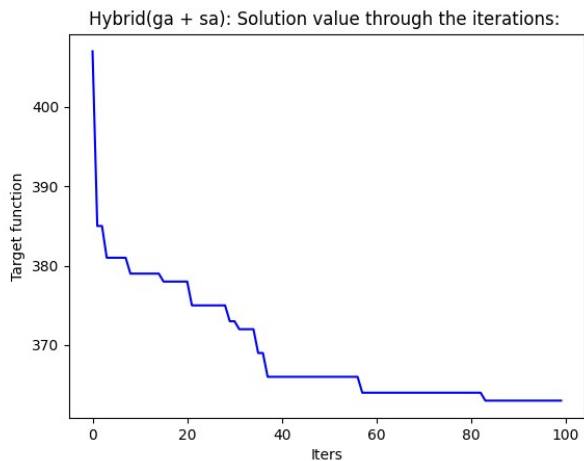


Slika 4



Slika 5

Hibridni algoritam(Genetski + Simulirano kaljenje): Sol = 363



Slika 6

Eksperimentalni rezultati nad primerima instanci iz prakse

Testiranja su vršena na računaru sa procesorom Intel Core i5-8250U(8) 3.4 GHz i 8GB RAM memorije.

Testirane su sve metode nad svim instancama preuzetim iz literature.

Instance su podeljene u dve grupe: Prva grupa predstavljaju instance koje imaju manje od 100 čvorova u grafu, a drugu grupu čine instance koje imaju više od 100 čvorova u grafu. Iznad svake tabele(tabela1-tabela6) stoji naziv instance, broj n koji predstavlja broj čvorova i broj m koji predstavlja broj grana u grafu.

Pomenute instance dostupne su na: <https://mat.tepper.cmu.edu/COLOR02/>.

Svaka metoda je pozvana 5 puta nad svakom instancom, pri čemu je zabeleženo ukupno vreme izvršavanja tih pet iteracija(**total time**), prosečno vreme izvršavanja jedne iteracije(**average time**), prosečna rešenja koja su dobijena kao rezultat izvršavanja metode(**average value**) i najbolje rešenje koje je metoda izračunala tokom tih 5 iteracija(**best value**).

Tabela1 – Tabela6 su organizovane po kolonama i to redom: naziv metode, ukupno vreme, prosečno vreme, prosečna vrednost i najbolja vrednost.

Pri testiranju instanci grupe 1, koji imaju manje od 100 čvorova u grafu, kod metoda lokalne pretrage i simuliranog kaljena, kao način traženja suseda korišćen je pristup zasnovan na zameni³. A kod testiranja instanci grupe 2, koji imaju više od 100 čvorova u grafu, kao način traženja suseda korišćen je pristup zasnovan na mešanju⁴ jer obezbeđuje intenzivniju promenu rešenja(što je pogodno za tako velike instance).

Prilikom upoređivanja rezultata testiranja, upoređivani su rezultati naših metoda sa IBM-ovim egzaktnim rešavačem koji se zove IBM ILOG CPLEX 12.8. On je za izabrane instance iz literature davao optimalno rešenje za manje od 2 sata. U tabelama ispod prvo je prikazano rešenje ovog egzaktnog rešavača, a ispod su rezultati naših metoda. Nijansama zelene boje obeležena su 4 najbolja rešenja koje su metode pronašle. Zatim sledi grafički prikaz rada te četiri najbolje metode predstavljen u vodu python plotova.s

3 Iz slajdova: Genetsko programiranje, prof.Aleksandar Kartelj

4 Iz slajdova: Genetsko programiranje, prof.Aleksandar Kartelj

Grupa 1: $n < 100$

Instance : 2-Insertions_3 ($n=37$, $m=72$)

IBM ILOG CPLEX 12.8 egzaktni rešavač: Optimalno rešenje za ovu instancu je $OPT_{sol} = 62$

Method name	Total time(sec)	Average time(sec)	Average value	Best value
Brute force	-	-	-	-
Local search	31.62	6.32	62	62
Simulated annealing	31.06	6.21	62	62
VNS($k_{max} = 3$)	73.18	14.64	62	62
VNS($k_{max} = 5$)	133.51	26.70	62	62
Genetic (roulette)	41.13	8.23	62	62
Genetic (tournament)	60.73	12.15	62	62
Hybrid (ga + sa)	75.93	15.19	62	62

Tabela 1

Instance: mug88_25 ($n=88$, $m=146$)

IBM ILOG CPLEX 12.8 egzaktni rešavač: Optimalno rešenje za ovu instancu je $OPT_{sol} = 178$

Method name	Total time(sec)	Average time(sec)	Average value	Best value
Brute force	-	-	-	-
Local search	169.04	33.81	180.2	179
Simulated annealing	164.05	32.81	180	179
VNS($k_{max} = 3$)	330.87	66.17	180	180
VNS($k_{max} = 5$)	688.486	137.68	178.6	178
Genetic (roulette)	207.79	41.56	179.4	178
Genetic (tournament)	369.13	73.83	178.8	178
Hybrid (ga + sa)	376.97	75.39	178.4	178

Tabela 2

Instance: myciel6 ($n=95$, $m=755$)

IBM ILOG CPLEX 12.8 egzaktni rešavač: Optimalno rešenje za ovu instancu je $OPT_{sol} = 189$

Method name	Total time(sec)	Average time(sec)	Average value	Best value
Brute force	-	-	-	-
Local search	210.79	42.16	191.8	190
Simulated annealing	211.89	42.38	193.6	192
VNS($k_{max} = 3$)	432.22	86.44	189.8	189
VNS($k_{max} = 5$)	855.382	171.07	189.4	189
Genetic (roulette)	256.08	51.22	190.4	189
Genetic (tournament)	441.36	88.27	194.4	194
Hybrid (ga + sa)	472.66	94.4	190.2	189

Tabela 3

Instance: mulsol.i.1 (n=197, m=3925)

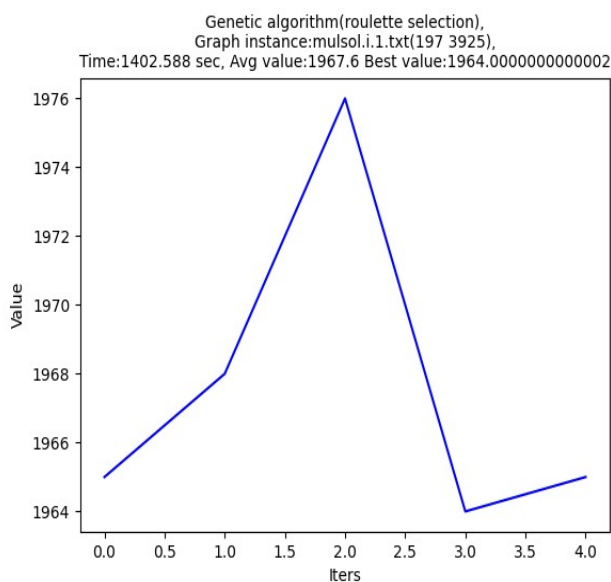
Grupa 2: $100 \leq n \leq 500$

IBM ILOG CPLEX 12.8 egzaktni rešavač: Optimalno rešenje za ovu instancu je $OPT_{sol} = 1957$

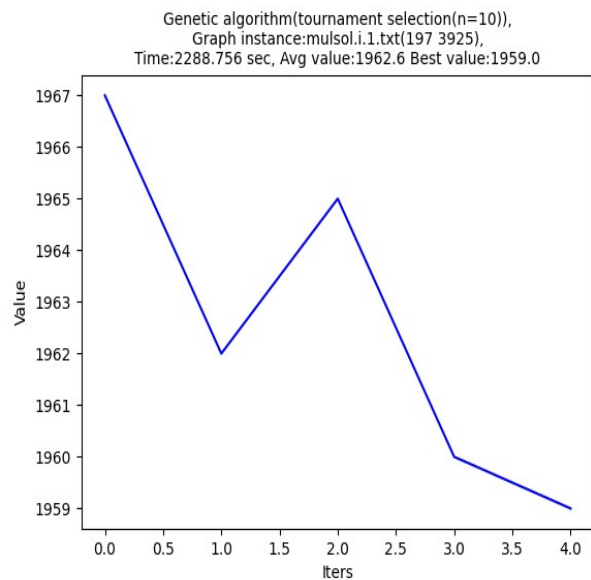
Method name	Total time(sec)	Average time(sec)	Average value	Best value
Brute force	-	-	-	-
Local search	1013.17	202.63	1984.4	1975
Simulated annealing	1017.02	203.4	1981.6	1967
VNS(k_max = 3)	2063.36	412.67	1979.8	1968
VNS(k_max = 5)	4109.84	821.9	1977.2	1973
Genetic (roulette)	1402.59	280.52	1967.6	1964
Genetic (tournament)	2288.76	457.75	1962.6	1959
Hybrid (ga + sa)	2428.7	485.74	1959.8	1958

Tabela 4

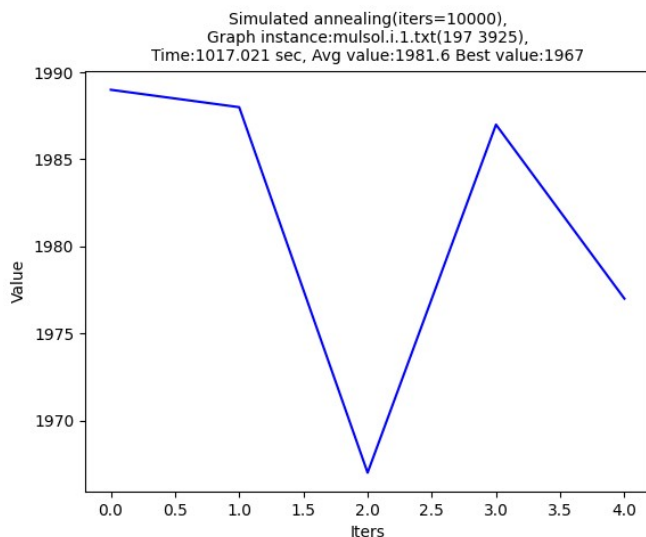
Grafički prikaz rada 4 najbolje metode za konkretnu instancu po iteracijama(i=5):



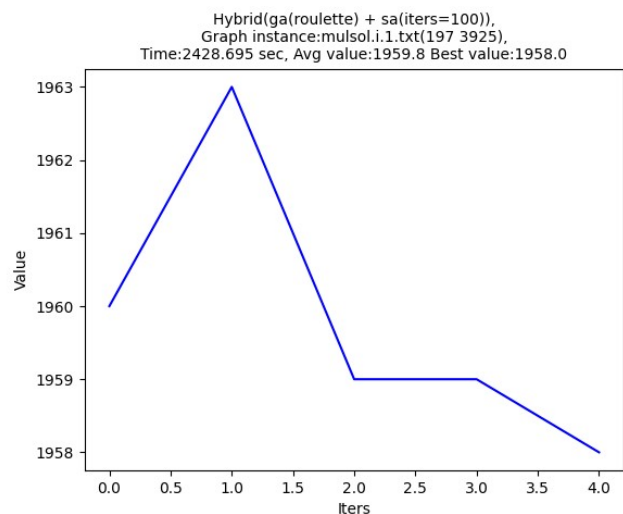
Slika 7



Slika 8



Slika 9



Slika 10

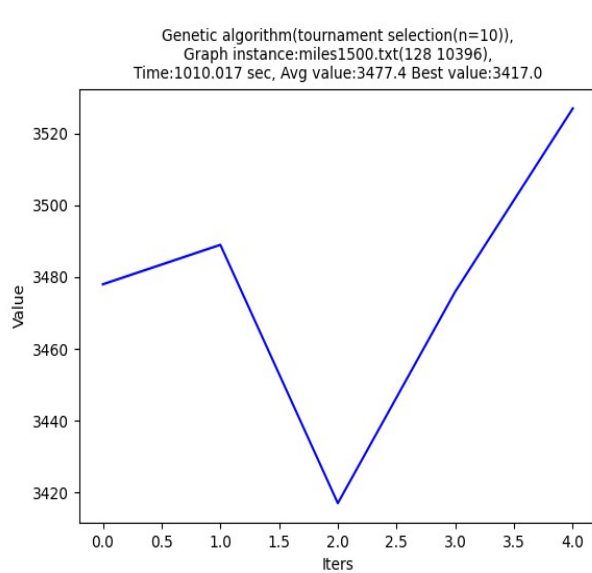
Instance: miles1500 (n=128, m=10396)

IBM ILOG CPLEX 12.8 egzaktни rešavač: Optimalno rešenje za ovu instancu je $OPT_{sol} = 3260$

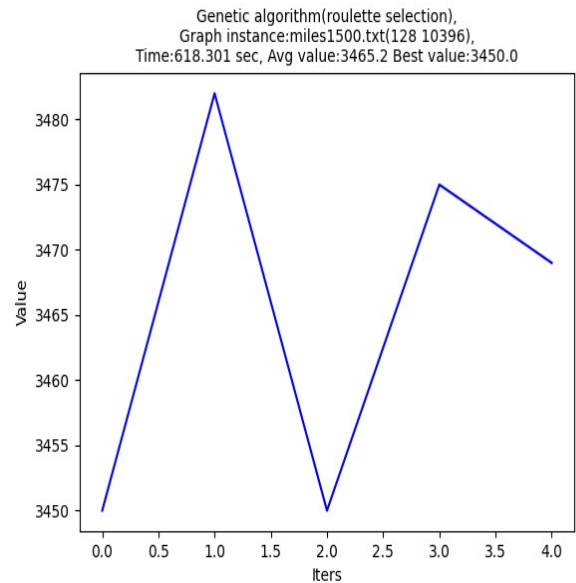
Method name	Total time(sec)	Average time(sec)	Average value	Best value
Brute force	-	-	-	-
Local search	538.43	107.69	3505.2	3489
Simulated annealing	540.86	108.17	3503.8	3492
VNS(k_max = 3)	1491.12	298.22	3523.6	3516
VNS(k_max = 5)	2193.52	438.79	3485.6	3465
Genetic (roulette)	618.3	123.66	3465.2	3450
Genetic (tournament)	1010.02	202	3477.4	3417
Hybrid (ga + sa)	1163.28	232.66	3431.8	3404

Tabela 5

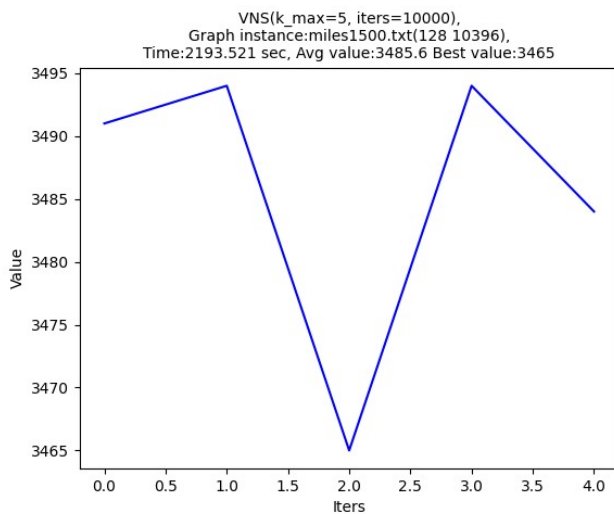
Grafički prikaz rada 4 najbolje metode za konkretnu instancu po iteracijama(i=5)



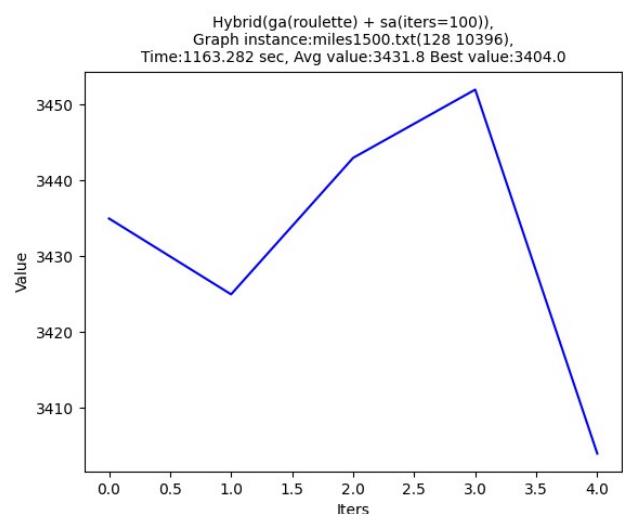
Slika 12



Slika 11



Slika 13



Slika 14

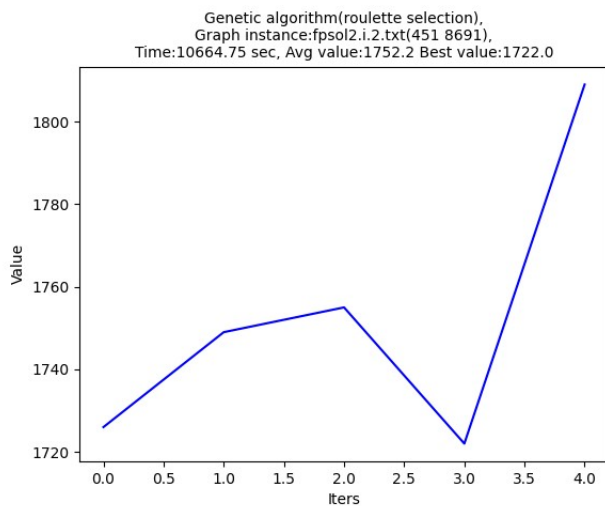
Instance: fpsol2.i.2 (n=451, m=8691)

IBM ILOG CPLEX 12.8 egzaktni rešavač: Optimalno rešenje za ovu instancu je $OPT_{sol} = 1668$

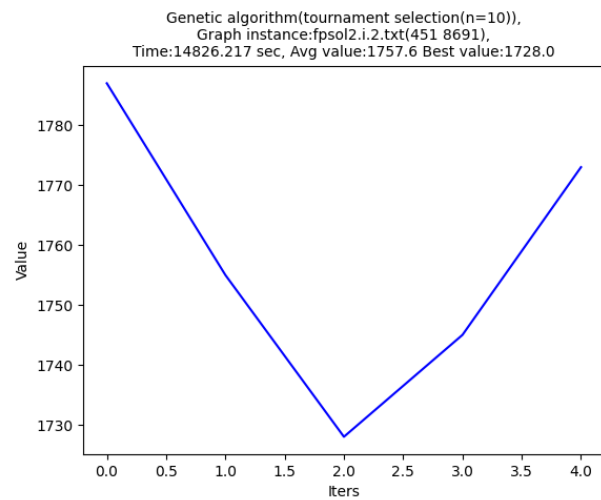
Method name	Total time(sec)	Average time(sec)	Average value	Best value
Brute force	-	-	-	-
Local search	5465.93	1093.19	1901.8	1879
Simulated annealing	5377.08	1075.42	1888.2	1860
VNS(k_max = 3)	10936.24	2187.25	1881.4	1860
VNS(k_max = 5)	21302.71	4260.54	1853.5	1800
Genetic (roulette)	10664.75	2132.95	1752.2	1722
Genetic (tournament)	14826.22	2965.24	1757.6	1728
Hybrid (ga + sa)	16057.83	3211.57	1701	1692

Tabela 6

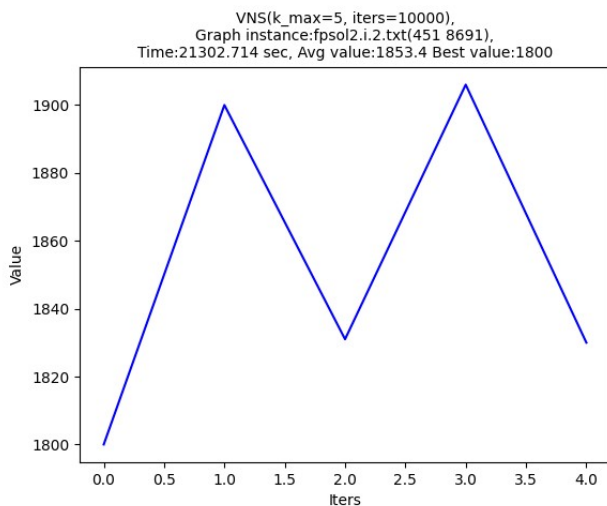
Grafički prikaz rada 4 najbolje metode za konkretnu instancu po iteracijama(i=5):



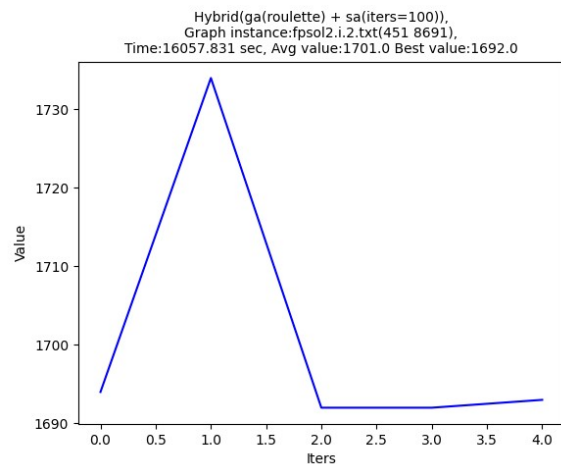
Slika 15



Slika 16



Slika 18



Slika 17

Zaključak

U ovom radu razmatrana je varijanta problema bojenja grafa, čiji je cilj pronalaženje pravilnog bojenja čvorova bojama predstavljenim prirodnim brojevima $\{1, 2, \dots\}$, tako da zbir boja dodeljenih čvorovima grafa bude minimalan. Opisani problem su predložili Kubicka , u oblasti teorije grafova, i Supovit, u oblasti VLSI (engl. Very Large Scale Integration) dizajna, i on spada u klasu NP-teških problema. MSCP ima veliki značaj u praksi jer se koristi za rešavanje problema raspoređivanja, rasporeda časova, alokacije registara, i mnogim drugim.

Za rešavanje ovog problem razvijene su metode: Lokalna pretraga, Simulirano kaljenje, Metoda promenljivih okolina poznatija kao VNS metoda, Genetski algoritam i Hibridni algoritam koji kombinuje Genetski algoritam sa algoritmom simuliranog kaljenja. Rezultati ovih metaheuristika su upoređeni sa IBM-ovim egzaktnim rešavačem i pokazuje se da su skoro sve metode našle optimalno rešenje za instance iz prve gupe, a to su grafovi sa manje od 100 čvorova. Što se tiče druge grupe, odnosno instance sa više od 100 čvorova u grafu, pomenute metode su dale približno bliska rešenja optimalnom rešenju od koji se najbolje pokazao Hibridni algoritam.

Literatura:

1. <http://poincare.matf.bg.ac.rs/~aleksandar.kartelj/?content=RI>
2. <https://poincare.matf.bg.ac.rs/~denis.alicic/ri/6.html>
3. <https://poincare.matf.bg.ac.rs/~denis.alicic/ri/5.html>
4. <http://poincare.matf.bg.ac.rs/~aleksandar.kartelj/nastava/RI2022/05.Evolutivna.izracunavanja.pdf>
5. <http://elibrary.matf.bg.ac.rs/bitstream/handle/123456789/5459/DjordjevicAleksandra.pdf?sequence=1>