Rješavanje problema rasporeda pomoću meta-heuristika

Deduš, Vedran

Rješavanje problema rasporeda pomoću meta-heuristika

Deduš, Vedran

Rješavanje problema rasporeda pomoću meta-heuristika

Deduš, Vedran

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Vedran Deduš

RJEŠAVANJE PROBLEMA RASPOREDA POMOĆU META-HEURISTIKA

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

U dosta faza našeg života javlja se potreba za organizacijom nekih resursa. Problemi mogu biti razni, primjerice organiziranje rada u smjenama, redoslijeda događanja na sportskim natjecanjima ili rasporeda sati za fakultete, koji ćemo proučavati u ovom radu.

Prije početka semestra potrebno je složiti raspored sati za fakultet. Satničar treba uskladiti različite smjerove na fakultetu, kako bi studentima bilo omogućeno pohađati sva predavanja. Za svako predavanje treba naći slobodnu predavaonicu i nastavnika koji će održati predavanje. Ako se ovaj problem rješava ručno, to može potrajati tjednima ili čak mjesecima. Težina izrade rasporeda ovisi od fakulteta do fakulteta, jer nemaju svi fakulteti istu strukturu niti jednake uvjete za održavanje nastave. Problem rasporeda sati na nekim fakultetima može biti NP-težak i u ovom radu ćemo promatrati takvu instancu problema.

Kao i za mnoge druge NP-teške probleme, za rješavanje koristimo meta-heuristike, odnosno algoritme koji daju dovoljno dobra (ne nužno najbolja) rješenja za traženi problem. Genetski algoritam, koji opisujemo u ovom radu, inspiriran je prirodom, točnije procesom evolucije, pa tako kroz iteracije (ekvivalent generacijama neke populacije) pokušava poboljšati skup rješenja.

U poglavlju 1 definiramo instancu problema koji ćemo rješavati. U poglavlju 2 detaljnije opisujemo genetski algoritam, a implementacija rješenja je opisana u poglavlju 3. Usporedba rezultata s rezultatima drugih radova prikazana je u poglavlju 4.

Poglavlje 1

Definicija problema

1.1 Motivacija

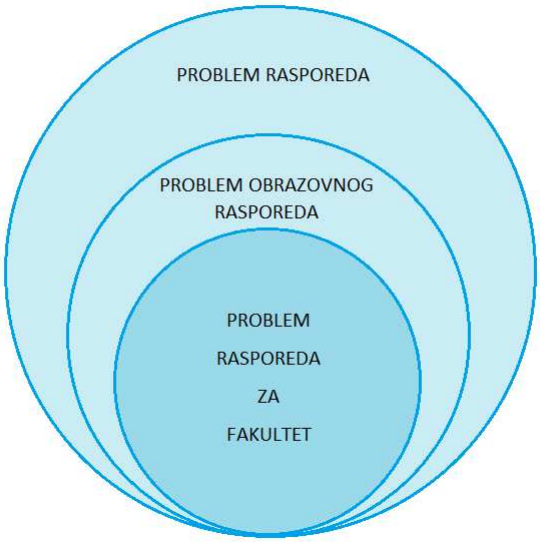
Problemi rasporeda često se javljaju u stvarnom životu i već za male slučajeve znaju biti teško rješivi. Jedan od prvih primjera koji se javlja je problem rasporeda radnika na poslovima. Poznat takav problem je problem medicinskih sestara (engl. *Nurse Scheduling Problem*), gdje treba složiti raspored za rad medicinskih sestara u smjenama. U slaganju rasporeda smjena treba paziti da:

- u svakom trenutku u bolnici mora biti dovoljan broj medicinskih sestara
- medicinske sestre ne smiju raditi uzastopne smjene
- svaka medicinska sestra mora imati slobodne dane u svakom tjednu
- svaka medicinska sestra mora odraditi dovoljan broj sati u zadanom periodu.

Potproblem problema rasporeda je problem obrazovnog rasporeda (Slika 1.1). Primjer problem obrazovnog rasporeda bi bio problem rasporeda sati za škole - u razdoblju od osam sati treba organizirati sva predviđena predavanja tako da vrijedi:

- dva predavanja istom razredu ne smiju biti u istom terminu
- jedan nastavnik ne smije biti na dva predavanja odjednom
- u svakoj učionici se u jednom terminu može održati najviše jedno predavanje.

Problem rasporeda za fakultete također je problem obrazovnog rasporeda, no zbog svoje težine izdvojen je kao zasebni problem. Za razliku od škole, na fakultetu postoje smjerovi koji otežavaju slaganje rasporeda - jedan predmet može biti u više smjerova i studenti sa svih smjerova moraju imati mogućnost prisustvovati predavanju. Problem može



Slika 1.1: Familija problema rasporeda

biti slaganje rasporeda sati ili slaganje rasporeda pisanja ispita. Jedna od razlika ova dva problema je to što se kod pisanja testova u jednoj predavaonici može pisati test iz više predmeta, dok kod predavanja u nekom trenutku može biti samo jedno predavanje u predavaonici. U radu [2] se spominje da su satničarima, radeći ručno, trebali tjedni ili čak mjeseci da bi složili valjani raspored sati za semestar.

Problemi rasporeda imaju skup strogih i skup blagih uvjeta. Svi uvjeti navedeni u prethodna tri problema spadaju u stroge uvjete. Da bi rješenje problema bilo prihvatljivo, moraju biti zadovoljeni svi strogi uvjeti. Blagi uvjeti ne moraju biti zadovoljeni, ali njihovim zadovoljavanjem dobivamo na kvaliteti rješenja.

1.2 Matematička definicija problema

Definicija 1.2.1. *Neka je C skup svih predmeta, R skup svih predavaonica, d broj dana u tjednu za koje radimo raspored i p broj perioda u danu. Definiramo skupove Terms i Class na sljedeći način:*

$$Periods := \{(x, y) | x = 1 \dots d, y = 1 \dots p\}$$
$$Terms := \{(c, t, r) | c \in C, t \in Periods, r \in R\}.$$

Svaka funkcija $F : (C, R, Terms) \to \mathcal{P}(Terms)$ naziva se raspored.

Definicija 1.2.1 je općenita definicija funkcije rasporeda. Ipak, pri zadavanju problema želimo da nam funkcija uzima u obzir još par stvari prilikom izrade rasporeda:

- nisu sve predavaonice jednake veličine
- svaki predmet sluša različiti broj studenata
- predmeti pripadaju određenim smjerovima
- predavanja treba održati nastavnik.

Definicija 1.2.2. *Neka je C skup svih predmeta, T skup svih nastavnika, R skup svih predavaonica, d broj dana u tjednu za koje radimo raspored i p broj perioda u danu. Definiramo skupove:*

$$Courses := \{(c, t, k, l, n) | c \in C, t \in T, k \in \mathbb{N}, l \in \mathbb{N}\ , n \in \mathbb{N}\} \tag{1.1}$$

(k - broj predavanja u tjednu koja se moraju održati)
(t - minimalan raspon dana kroz koja predavanja moraju biti zadana)
(n - očekivani broj studenata na predavanju)

$$Curriculas \subset \mathcal{P}(C) \tag{1.2}$$

$$Rooms := \{(r, n) | r \in R, n \in \mathbb{N}\} \tag{1.3}$$

(n - kapacitet predavaonice)

$$Periods := \{(x, y) | x = 1 \ldots d, y = 1 \ldots p\} \tag{1.4}$$

$$Terms := \{(course, period, room) | course \in Courses, period \in Periods, room \in Rooms\}. \tag{1.5}$$

Svaka funkcija $F : (Courses, Curriculas, Rooms, Periods) \rightarrow \mathcal{P}(Terms)$ *naziva se raspored.*

Iako smo promijenili domenu funkcije, još nismo definirali uvjete koje naše rješenje problema mora zadovoljavati.

Definicija 1.2.3. *Za problem rasporeda sati za fakultete strogi uvjet zadan je formulom logike prvog reda i raspored ga mora zadovoljavati, odnosno formula logike prvog reda mora biti ispunjena. Skup svih strogih uvjeta označavamo sa H.*

Definicija 1.2.4. *Za problem rasporeda sati za fakultete blagi uvjet zadan je formulom logike prvog reda. Raspored ne mora zadovoljavati blagi uvjet. Skup svih blagih uvjeta označavamo sa S.*

Definicija 1.2.5. *Za problem rasporeda sati za fakultete skup svih uvjeta označavamo sa* U *i vrijedi* $U = H \cup S$.

Sada ćemo zapisati stroge uvjete koje problem mora zadovoljavati. Koristit ćemo oznake kao u definiciji 1.2.2. U problemu koji ćemo rješavati imamo sljedeće stroge uvjete:

- H_1: svaki predmet se u rasporedu mora pojaviti onoliko puta koliko je za njega određeno (prema formuli 1.1: $(c, t, k, l, n) \in Courses$ - predmet c se mora pojaviti k puta).
- H_2: u svakom periodu se u nekoj predavaonici može održavati najviše jedno predavanje.
- H_3: u rasporedu ne smije biti konflikata. Dva predavanja koja pripadaju istom smjeru ne smiju se održavati u isto vrijeme. Dva predavanja koja predaje isti nastavnik ne smiju se održavati u isto vrijeme.
- H_4: ako nastavnik nije dostupan u nekom periodu, nijedno predavanje koje on predaje ne smije biti u tom periodu.

Sljedeće navodimo blage uvjete koje želimo da naš problem zadovoljava u što većem broju:

- S_1: broj studenata na predavanju ne bi trebao biti veći od kapaciteta predavaonice u kojoj se predavanje održava.
- S_2: predavanja bi trebala biti zadana kroz barem minimalan broj dana definiran za svaki predmet (prema formuli 1.1: $(c, t, k, l, n) \in Courses$ - predmet c bi se trebao održati barem u l dana).
- S_3: predavanja ne bi trebala biti izolirana - unutar jednog dana nijedno predavanje unutar nekog smjera ne bi smjelo biti okruženo s dvije praznine.
- S_4: sva predavanja nekog predmeta trebala bi biti u istoj predavaonici.

Konačno, možemo neformalno definirati problem rasporeda sati za fakultete kao pronalaženje funkcije, definirane u 1.2.2, čije rješenje zadovoljava sve stroge uvjete i što veći broj blagih.

Teorem 1.2.6. *Problem rasporeda sati za fakultete je NP-potpun.*

Dokaz. Dokaz je dan u članku [3]. □

U nastavku ćemo pokazati ideju dokaza. U članku se potproblem problema slaganja rasporeda za fakultete svodi na poznate NP-teške probleme. Jedan od primjera je svođenje na problem k-obojivosti grafa. Sljedeća definicija uzeta je iz knjige [8].

Definicija 1.2.7. *Kažemo da je je graf* (G, R) *k-obojiv (*$k \in \mathbb{N} \setminus \{0\}$*) ako postoji particija* $B_1, \ldots, B_k$ *skupa G tako da ne postoje* $a, b \in G$*, te* $j \in 1, \ldots, k$ *za koje vrijedi* $\{a, b\} \in R$ *i* $a, b \in B_j$*. Problem k-obojivosti grafa sastoji se od određivanja je li problem graf k-obojiv.*

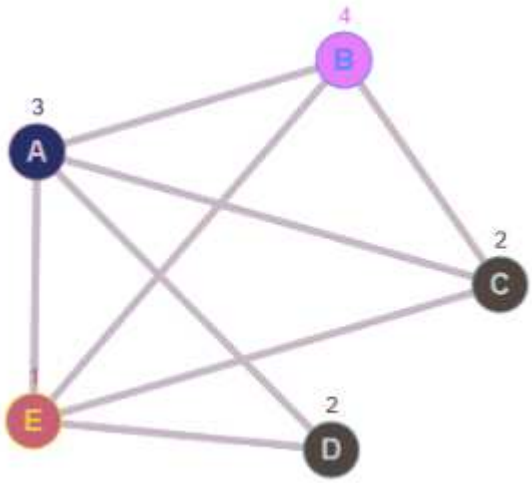
Problem pravljenja rasporeda za fakultete, bez dodjeljivanja predavaonica, možemo prikazati kao problem k-obojivosti grafa, što ćemo demonstrirati na primjeru. U primjeru imamo dva smjera sa po 3 predmeta.

- prvi smjer sadrži predmete A, B i C,
- drugi smjer sadrži predmete A, D i E,
- prvi nastavnik predaje predmet A,
- drugi nastavnik predaje predmete B, C i E,
- treći nastavnik predaje predmet D.

Skup vrhova u grafu je skup svih predmeta za koje radimo raspored. Bridove konstruiramo na dva načina:

- svaka dva predmeta koja se nalaze u istom smjeru povežemo bridom,
- svaka dva predmeta koja predaje isti nastavnik povežemo bridom.

Boje u grafu nam predstavljaju periode u rasporedu, odnosno po definiciji 1.2.2, $k = |Periods|$. Sada je pitanje k-obojivosti grafa izjednačeno sa pitanjem postojanja rasporeda sati za fakultete bez predavaonica. Prepostavimo li da za ovaj primjer imamo 2 dana u kojima imamo po 3 sata za predavanje, dobivamo da je graf 6-obojiv i da raspored za ovaj problem postoji.
Sa Slike 1.2 vidimo da je graf 4-obojiv, odnosno da imamo i više perioda na raspolaganju nego što nam je potrebno.



Slika 1.2: Jedan 4-obojivi graf za demonstracijski primjer

U nekim se radovima upravo na taj način pokušava napraviti raspored sati za fakultete. Prvo se rješava potproblem koji se svede na k-obojivost grafa, gdje se pokušava naći što manji k za koji je graf k-obojiv. Zatim se predmetima dodjeljuju predavaonice, a ako nije moguće pronaći predavaonicu za predmet, period održavanja se promijeni u jedan od neiskorištenih.

Poglavlje 2

Meta-heuristike

Složene probleme najčešće ne možemo riješiti egzaktnim algoritmom, jer bi njegovo izvršavanje trajalo predugo ili tražilo previše resursa. Meta-heuristike nam omogućuju da u kraćem vremenu i s manje resursa nađemo dovoljno dobro rješenje, odnosno rješenje koje dobijemo ne mora biti najbolje moguće. Sljedeće definicije uzete su iz knjige [10].

Definicija 2.0.1. *Algoritme koji pronalaze rješenja koja su zadovoljavajuće dobra, ali ne nude nikakve garancije da će uspjeti pronaći optimalno rješenje, te koji imaju relativno nisku računsku složenost nazivamo približni algoritmi, heurističke metode, heuristički algoritmi ili jednostavno heuristike.*

Definicija 2.0.2. *Metaheuristika je skup algoritamskih koncepata koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema. Možemo reći da je metaheuristika heuristika opće namjene, čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja.*

2.1 Pregled radova

Problem rasporeda sati za fakultete najčešće se rješava meta-heuristikama.
U radu [6] se problem rješava tabu pretraživanjem. Tabu pretraživanje je lokalno pretraživanje u kojem se pamti lista nedavno pogledanih rješenja i brani se vraćanje na njih kako pretraga ne bi zapela u lokalnom optimumu.
U radovima [7] i [5] problem se rješava genetskim algoritmom (2.2).
Metaheuristikom praga prihvaćanja (engl. *threshold accepting metaheuristic*) problem se pokušava riješiti u radu [4].
Radovi [4] i [6] testirani su na istom skupu testnih podataka, koji ćemo koristiti i u ovom radu. Skup testnih podataka koji koristimo uzet je s natjecanja *International Timetabling Competition 2007* ([1]).

2.2 Genetski algoritam

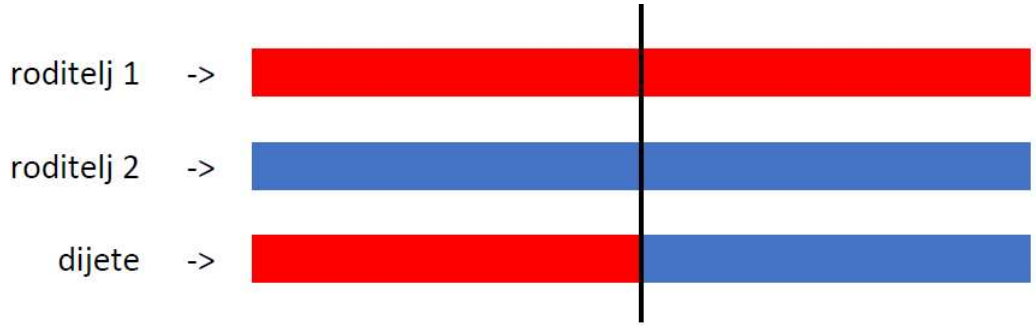
Genetski algoritam pripada prirodom inspiriranim populacijskim algoritmima. Populacijski jer se kombiniranjem i modificiranjem skupa rješenja stvara novi skup rješenja, dok je prirodna inspiracija proces evolucije. Iz Darwinove teorije evolucije možemo izvući sljedećih pet postavki:

- rast populacije je ograničen i veličina populacije je stabilna
- kod vrsta koje se spolno razmnožavaju ne postoje dvije identične jedinke
- jedinka koja je nastala razmnožavanjem ima gene oba roditelja
- unutar populacije postoji nadmetanje za preživljavanje
- preživljavanje najbolje prilagođenih jedinki.

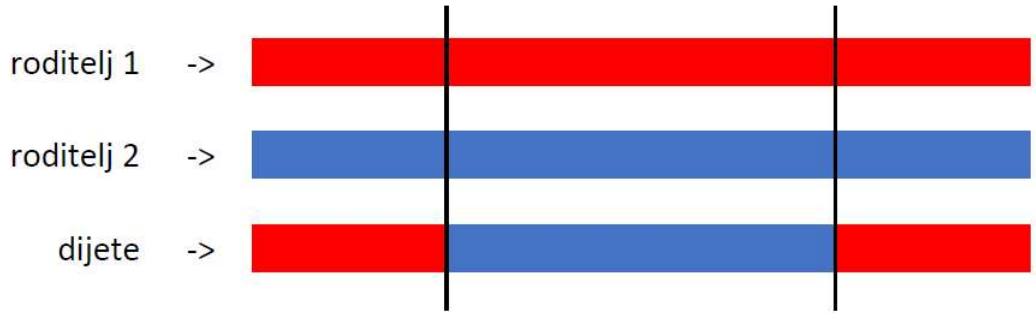
Genetski algoritam radi ponavljanjem iteracija na populaciji rješenja u svrhu poboljšanja rješenja. Za rad algoritma moramo imati početnu populaciju rješenja, koja može biti generirana na potpuno prozvoljan način. Za objašnjavanje rada genetskog algoritma definiramo sljedeće pojmove:

- kromosom - predstavlja jedno potencijalno rješenje problema, tj. jednu jedinku
- populacija - skup kromosoma
- mutacija - promjena koja se može dogoditi na kromosomu (promjena ne mora nužno rezultirati boljim rješenjem)
- križanje - stvaranje novog kromosoma kombiniranjem dva postojeća kromosoma iz populacije
- selekcija - odabir najboljih kromosoma iz populacije u svrhu očuvanja veličine populacije.

Za početak rada genetskog algoritma (1) prvo trebamo generirati početnu populaciju. Zatim ponavljamo iteracije genetskog algoritma dokle god uvjet zaustavljanja, što je najčešće broj iteracija ili valjanost rješenja, nije zadovoljen. Svaka iteracija počinje povećanjem broja kromosoma u populaciji tako da radimo križanje s postojećim kromosomima dok nam populacija ne naraste na zadanu veličinu. Standardni načini križanja su križanje odabirom jedne točke (Slika 2.2) i križanje odabirom dvije točke (Slika 2.2).



Slika 2.1: Križanje odabirom jedne točke



Slika 2.2: Križanje odabirom dvije točke

Nakon tog, s određenom vjerojatnošću, mutiramo svaki kromosom u populaciji. Na kraju selekcijom odaberemo najbolje kromosome koji će ostati u populaciji i reduciramo populaciju na originalnu veličinu.
Svaki kromosom (jedinku) ocjenjujemo funkcijom dobrote, odnosno funkcijom koja nam govori koliko je kromosom prilagođen problemu (okolini). Funkcija dobrote u slučaju problema rasporeda sati za fakultete bila bi zadovoljavanje blagih uvjeta, točnije kažnjavanje njihovog kršenja. Naravno, svi strogi uvjeti također moraju biti zadovoljeni pa njihovo kršenje možemo kažnjavati sa znatno većom kaznom (npr. tisuću puta veća kazna).
Funkcija cilja je naći rješenje problema sa što manjom vrijednosti funkcije dobrote.

Algorithm 1 Osnovni pseudokod genetskog algoritma

```
populacija ← inicijalno punjenje populacije
while uvjet zaustavljanja nije zadovoljen do
    while populacija nije povećana do zadovoljavajuće veličine do
        novi kromosom ← križanje (dva proizvoljna kromosoma iz populacije)
        dodaj novi kromosom u populaciju
    end while
    for kromosom ∈ populacija do
        if vjerojatnost za mutaciju ispunjena then
            mutiraj kromosom
        end if
    end for
    procesom selekcije smanji populaciju do originalne veličine
end while
```

Poglavlje 3

Implementacija algoritma

Genetski algoritam iz poglavlja 2 implementiran je u programskom jeziku Java. Skup testnih podataka uzet je sa natjecanja *International Timetabling Competition 2007* [1], pa će se i pri evaluaciji uzimati pravila s natjecanja. Svi uvjeti koje ćemo koristiti definirani su na kraju poglavlja 1. Kršenje blagih uvjeta boduje se na sljedeći način:

- za svakog studenta koji ne stane u predavaonicu se dodaje 1 kazneni bod
- ako su predavanja u manjem rasponu od definiranog, za svaki dan manje se dodaje 5 bodova kazne
- sva predavanja nekog predmeta trebaju biti u istoj predavaonici - za svaku predavaonicu viška se dodaje 1 bod kazne
- unutar jednog smjera ne smije biti izoliranih predavanja (izolirano predavanje je ono okruženo sa 2 pauze) - za svako izolirano predavanje se dodaje po 1 bod kazne.

Zapis rješenja

Rješenje problema problema zapisujemo kao dvije liste. U prvoj listi pamtimo sve termine, definirane kao element skupa *Terms* (1.5) iz definicije 1.2.2, dok u drugoj listi pamtimo sve predmete koje nismo uspjeli smjestiti u raspored. Ovakvim zapisom rješenja ne dopuštamo kršenje strogih uvjeta u rasporedu, odnosno svi predmeti koji su u prvoj listi termina poštuju sve stroge uvjete, dok oni koje nismo uspjeli smjestiti u raspored idu u listu nesmještenih predmeta. Naravno, da bi rješenje bilo validno lista nesmještenih predmeta mora biti prazna.

3.1 Generiranje početne populacije

Generiranje početne populacije problema rasporeda se može napraviti na više načina. Nasumično stavljanje predmeta u raspored nije dobra opcija, jer bi tako brzo prekršili neki od strogih uvjeta te bi dobili populaciju koja nema niti jedno validno rješenje. Trebamo odrediti poredak kojim ćemo stavljati predmete u raspored. U radu [9] se to radi s težinskim grafovima. Graf je konstruiran kao u ideji dokaza teorema 1.2.6 a težine bridova su broj studenta u konfliktu. U radu su predstavljeni sljedeći načini biranja predmeta:

- L (*Largest degree*) - predmeti s najvećim brojem konflikata imaju prednost
- W (*Weighted degree*) - predmeti s najvećim brojem studenata u konfliktu imaju prednost
- S (*Saturation degree*) - predmeti koji imaju najmanji broj slobodnih perioda u kojima mogu biti imaju prednost.

Za generiranje početne populacije korištena je modificirana verzija prvog pristupa - predmeti su sortirani po zbroju zabrana i broju smjerova u kojima se nalaze. Za svaki predmet prvo stavimo jedno predavanje u svaki dan, dok ne zadovoljimo minimalan broj dana kroz koja se predavanja moraju održati. Zatim, preostala predavanja probamo nasumično ubaciti u raspored. Nasumično pokušamo deset puta pogoditi valjan termin i predavaonicu, a ako ne uspijemo stavljamo ga u listu nesmještenih predmeta.
Nakon što smo sve predmete raširili kroz zadan broj dana (i time zadovoljili jedan blagi uvjet), moramo još ubaciti predmete koje nismo uspjeli nasumičnim odabirom termina. Prvi put za svaki nesmješteni predmet prolazimo kroz sve termine i gledamo postoji li prazan termin i dovoljno velika predavaonica koja mu odgovara. Ako i dalje postoje predmeti koji nisu stavljeni u raspored, prolazimo ponovno kroz listu - sada ga stavljamo u raspored ako postoji termin i slobodna predavaonica bilo koje veličine.
Prvim prolaskom probali smo zadovoljiti blagi uvjet vezan za kapacitet predavaonice, dok smo u drugom prolazu ignorirali blage uvjete te samo pokušali dati validno rješenje. Na slici 3.1 možemo vidjeti usporedbu broja validnih rješenja dobivenih različitim pristupima.

Ovakav pristup u većini slučajeva daje manje validnih rješenja u odnosu na druge. Osim broja validnih rješenja, bitna nam je i njihova kvaliteta. Na slici 3.1 možemo vidjeti da su, unatoč manjem broju, dobivena rješenja kvalitetnija od rješenja dobivenih u radu [9]. Problem broja validnih rješenja u praksi se riješi u prvih pedeset iteracija algoritma, što će biti opisano u poglavljima 3.2 i 3.3.

problem	opisan pristup	L	W	S
comp01	50	50	50	50
comp02	13	50	50	50
comp03	49	49	50	50
comp04	50	50	50	50
comp05	**50**	2	3	2
comp06	48	50	50	50
comp07	34	50	50	50
comp08	50	50	50	50
comp09	40	50	50	50
comp10	43	50	50	50
comp11	50	50	50	50
comp12	16	47	50	46
comp13	50	50	50	50
comp14	50	50	50	50
comp15	45	49	50	50
comp16	50	50	50	50
comp17	29	48	48	48
comp18	50	50	50	50
comp19	50	50	50	50
comp20	29	50	50	50
comp21	27	50	50	50

Slika 3.1: Usporedba broja validnih rješenja u populaciji

problem	opisan pristup	grafovski pristup
comp01	**173**	330
comp02	**290**	769
comp03	**232**	702
comp04	**192**	694
comp05	**476**	1466
comp06	**286**	947
comp07	**333**	1043
comp08	**213**	790
comp09	**268**	847
comp10	**292**	898
comp11	**70**	230
comp12	**547**	1498
comp13	**231**	793
comp14	**211**	745
comp15	**236**	702
comp16	**288**	949
comp17	**290**	902
comp18	**134**	583
comp19	**359**	637
comp20	**337**	1042
comp21	**288**	928

Slika 3.2: Usporedba kvalitete rješenja početne populacije

3.2 Mutacija

Mutacija je promjena određenog kromosoma. U implementaciji algoritma koristimo nekoliko različitih mutacija, ovisno o tome što treba poboljšati u rješenju.

Opća mutacija

Prva mutacija je opća mutacija (2) koja pokušava neispravno rješenje pretvoriti u validno, odnosno pokušava zadovoljiti sve stroge uvjete. Prolazimo kroz listu termina stavljenih u raspored i s određenom vjerojatnošću ih mičemo iz rasporeda te stavljamo na kraj liste predmeta koji nisu u rasporedu. Zatim listu svih predmeta koji nisu u rasporedu probamo smjestiti u raspored.

Algorithm 2 Opća mutacija

smješteni ← lista termina u rasporedu
nesmješteni ← lista predmeta koji nisu smješteni u raspored
for predmet ∈ smješteni **do**
  if vjerojatnost zadovoljena **then**
    makni termin iz liste smještenih i stavi predmet na kraj liste nesmještenih
    oslobodi termin i predavaonicu koju je predmet zauzimao
  end if
end for
for predmet ∈ nesmješteni **do**
  smjesti predmet u neki od slobodnih termina s prostorijom odgovarajuće veličine
  if uspješno smješten **then**
    makni predmet iz liste nesmještenih i stavi termin sa odgovarajućim predmetom, periodom i predavaonom u listu smještenih
  end if
end for
for predmet ∈ nesmješteni **do**
  smjesti predmet u neki od slobodnih termina, ignoriraj veličinu predavaonice
  if uspješno smješten **then**
    makni predmet iz liste nesmještenih u listu smještenih
  end if
end for

Mutacija predavaonica

Mutacija predavaonica (3) je mutacija u svrhu poboljšanja zadovoljavanja blagog uvjeta da svaki predmet koristi samo jednu predavaonicu. Iz liste termina izdvojimo sve termine za nasumično odabran predmet i odaberemo jedan termin iz tog skupa. Predavaonicu koju koristimo u tom terminu stavimo kao predavaonicu svim ostalim predmeta. Ukoliko je predavaonica zauzeta u tom terminu za neki drugi predmet, predavaonice zamijenimo s terminom u konfliktu. Iako mutacijom zadovoljavamo blagi uvjet o jednoj predavaonici po predmetu, možemo prekršiti blagi uvjet o kapacitetu predavaonice.

Algorithm 3 Mutacija predavaonice

```
smješteni ← lista termina u rasporedu
predmet ← nasumično odabran predmet
lista termina ← termini iz liste smješteni za odabran predmet
predavaonica ← predavaonica nasumično odabranog termina iz lista termina
for termin ∈ lista termina do
    if predavaonica termin ≠ predavaonica then
        if predavaonica slobodna u periodu termin then
            promijeni sobu u predavaonica
        end if
        if predavaonica nije slobodna u periodu termin then
            termin-zamjene ← termin koji koristi predavaonica u traženom periodu
            zamijeni predavaonice od termin i termin-zamjene
        end if
    end if
end for
```

Mutacija perioda

Mutacija perioda (4) uzima jedan nasumično odabran termin, zatim prolazi kroz listu svih perioda i pokušava naći slobodan period u koji se termin može prebaciti. Ova mutacija pokušava zadovoljiti dva blaga uvjeta: uvjet o minimalnom broju dana kroz koje se predavanje mora održati i uvjet o izoliranim predavanjima. Mutacija promjene termina može imati i upravo suprotan učinak, na primjer može neizolirani termin promijeniti u izolirani.

Algorithm 4 Mutacija perioda

smješteni ← lista termina u rasporedu
termin ← nasumično odabran termin iz liste *smješteni*
for *period* ∈ skup svih perioda **do**
  if moguće prebaciti *termin* u *period* bez kršenja strogih uvjeta **then**
    novi period od *termin* je *period*
    završi
  end if
end for

Mutacija kapaciteta predavaonica

Mutacija radi na isti način kao i mutacija perioda, samo umjesto novog perioda terminu tražimo novu predavaonicu. Isto kao i kod mutacije perioda nemamo garanciju da će mutacija poboljšati rješenje.

Algorithm 5 Mutacija kapaciteta predavaonica

smješteni ← lista termina u rasporedu
termin ← nasumično odabran termin iz liste *smješteni*
for *predavaonica* ∈ skup svih predavaonica **do**
  if *predavaonica* zadovoljava kapacitet za predavanja iz *termin* **then**
    promijeni predavaonicu u kojoj se održava *termin* u *predavaonica*
    završi
  end if
end for

Svaka se mutacija može dogoditi s određenom vjerojatnošću. Opća mutacija se može dogoditi i na validnom rješenju. Također, opća mutacija može od validnog rješenja napraviti nevalidno. Ostale mutacije (mutacija predavaonica, perioda i kapaciteta predavaonice) će od validnog rješenja uvijek napraviti validno. Iako ostalim mutacijama to nije primarna svrha, i one mogu od nevalidnog rješenja napraviti validno.

3.3 Križanje

Križanjem dva kromosoma dobivamo novi kromosom koji ima značajke oba roditelja. Standardni načini križanja (križanje odabirom jedne ili dvije točke), iako su korišteni u nekim radovima ([7], [5]) ovdje nisu davali zadovoljavajuće rezultate.
Implementiranim križanjem (6) za svaki predmet pokušavamo preuzeti što je više moguće termina iz jednog rasporeda roditelja, tj. za svaki predmet gledamo pripadne termine oba

roditelja te se uzimaju termini roditelja od kojeg više stane u novi raspored. Ako je broj termina koji se mogu staviti u raspored jednak, nasumično se odabire jedan roditelj i uzimaju njegovi termini. Termini koji nisu uspješno smješteni u raspored idu u listu nesmještenih i na kraju križanja se pokušavaju smjestiti u raspored.

Algorithm 6 Križanje

roditelj1 ← nasumično odabran kromosom
roditelj2 ← nasumično odabran kromosom
smješteni ← lista termina u rasporedu
nesmješteni ← lista predmeta koji nisu smješteni u raspored
for *predmet* ∈ skup svih predmeta **do**
    t1 ← broj termina za *predmet* od *roditelj1* koji se mogu staviti u raspored
    t2 ← broj termina za *predmet* od *roditelj2* koji se mogu staviti u raspored
    if *t1* > *t2* **then**
        preuzmi termine od *roditelj1*
    end if
    if *t1* < *t2* **then**
        preuzmi termine od *roditelj*2
    end if
    if *t1* = *t2* **then**
        preuzmi termine od nasumično odabranog roditelja
    end if
end for

Poglavlje 4

Rezultati i moguća poboljšanja

Genetski algoritam opisan u prethodnom poglavlju (3) testirali smo na testnim podacima prikazanima na Slici 4.1, tj. podacima s natjecanja [1].

problem	broj predmeta	broj smjerova	broj zabrana	broj predavonica	broj dana	broj perioda u danu
comp01	30	14	53	6	5	6
comp02	82	70	513	16	5	5
comp03	72	68	382	16	5	5
comp04	79	57	396	18	5	5
comp05	54	139	771	9	6	6
comp06	108	70	632	18	5	5
comp07	131	77	667	20	5	5
comp08	86	61	478	18	5	5
comp09	76	75	405	18	5	5
comp10	115	67	694	18	5	5
comp11	30	13	94	5	5	9
comp12	88	150	1368	11	6	6
comp13	82	66	468	19	5	5
comp14	85	60	486	17	5	5
comp15	72	68	382	16	5	5
comp16	108	71	518	20	5	5
comp17	99	70	548	17	5	5
comp18	47	52	594	9	5	5
comp19	74	66	475	16	5	5
comp20	121	78	691	19	5	5
comp21	94	78	463	18	5	5

Slika 4.1: Instance testnih podataka

Testiranja algoritma provedena su na računalu s Intel i7 procesorom, frekvencije radnog takta 2.8GHz i 8GB radne memorije. Genetski algoritam iterirali smo deset tisuća puta. Usporedbu rezultata s rezultatima iz radova [6] i [4] možemo vidjeti na slici 4.3, dok usporedbu s rezultatima natjecanja [1] možemo vidjeti na slici 4.2.

problem	G.A.	Müller	Lü & Hao	Atsuta	Geiger	Clark
comp01	47	5	5	5	5	10
comp02	146	51	55	50	111	111
comp03	116	84	71	82	128	119
comp04	71	37	43	35	72	72
comp05	**182**	330	309	312	410	426
comp06	122	48	53	69	100	130
comp07	185	20	28	42	57	110
comp08	77	41	49	40	77	83
comp09	112	109	105	110	150	139
comp10	138	16	21	27	71	85
comp11	17	0	0	0	0	3
comp12	**198**	333	343	351	442	408
comp13	81	66	73	68	622	113
comp14	91	59	57	59	90	84
comp15	111	84	71	82	128	119
comp16	114	34	39	40	81	84
comp17	162	83	91	102	124	152
comp18	**31**	83	69	68	116	110
comp19	111	62	65	75	107	111
comp20	142	27	47	61	88	144
comp21	143	103	106	123	174	169

Slika 4.2: Usporedba rezultata genetskog algoritma s rezultatima natjecanja [1]

Na obje slike vidimo da bolje rezultate dobivamo u testnim problemima comp05, comp12 i comp18. Na slici 4.1 možemo vidjeti da su to specifični problemi koji imaju više smjerova nego predmeta, što znači da se predmeti održavaju u više smjerova i tako otežavaju slaganje validnog rasporeda.
Opisan genetski algoritam svoj prostor pretraživanja konstantno proširuje križanjem i općim mutacijama te tako izbjegava zapinjanje u lokalnim optimumima. Iako za neke testne podatke to predstavlja veliku prednost, u drugim slučajevima je to mana - kako se prostor pretraživanja širi, tako se smanjuje lokalno pretraživanje, tj. po jednoj iteraciji algoritma mogu se dogoditi najviše tri mutacije koje simuliraju lokalno traženje boljeg rješenja.

problem	genetski algoritam	*tabu search*	*threshold accepting*
comp01	47	5	5
comp02	146	34	91
comp03	116	70	108
comp04	71	35	53
comp05	**182**	298	359
comp06	122	47	79
comp07	185	19	36
comp08	77	43	63
comp09	112	99	128
comp10	138	16	49
comp11	17	0	0
comp12	**198**	320	389
comp13	81	65	91
comp14	91	52	81
comp15	111	69	-
comp16	114	38	-
comp17	162	80	-
comp18	**31**	67	-
comp19	111	59	-
comp20	142	35	-
comp21	143	105	-

Slika 4.3: Usporedba rezultata genetskog algoritma, tabu pretraživanja [6]

i algoritma praga prihvaćanja [4]

Poboljšanje algoritma možemo postići tako da genetski algoritam pretvorimo u hiper-heuristiku. Hiper-heuristika je meta-heuristika nastala kombinacijom više heuristika. Zamijenimo li mutaciju s nekom nepopulacijskom heuristikom, npr. tabu pretraživanjem, mogli bismo dobiti značajno bolja rješenja. Tabu pretraživanje, koje bismo ubacili umjesto mutacije, trebalo bi također biti namješteno da unutar malog broja iteracija pretražuje susjedstvo rješenja bez velikog širenja.
Drugi način bi bio da algoritam podijelimo u faze, kao što je već napravljeno u nekim radovima. Nakon što genetski algoritam završi s radom, skup dobivenih rješenja pokušavamo poboljšati nekom drugom heuristikom. Naravno, nova heuristika ne koristi dobiven skup rješenja kao početnu populaciju, nego pokušava lokalnim pretraživanjem poboljšati dobivena rješenja.

Iako je standardni pristup da se nepopulacijske meta-heuristike koriste za pravljenje inicijalne populacije, ovdje bismo mogli napraviti suprotno - u prvoj fazi možemo pustiti genetski algoritam da pronađe rješenje i zatim ga u drugoj fazi nepopulacijskom meta-heuristikom pokušati poboljšati.

Poglavlje 5

Zaključak

U ovom je radu opisan genetski algoritam za rješavanje problema rasporeda sati. Opisanim algoritmom se pretražuje veliki prostor rješenja, što daje bolje rezultate na testnim podacima gdje ima više smjerova nego predmeta. Mana algoritma je nedovoljno lokalno pretraživanje - iako se pronađe rješenje koje je blizu lokalnog optimuma, često se ne pronađe put do tog optimuma. Na testnim podacima se vidi da se najviše krše blagi uvjeti o izoliranim predavanjima i broju predavaonica u kojima se predavanja održavaju. Jedno moguće poboljšanje algoritma bi bilo pretvoriti genetski algoritam u hiper-heuristiku, čime bi povećali lokalno pretraživanje.

Svaki fakultet je različit, pa bi tako za primjenu ovog algoritma na stvarnom problemu trebalo prilagoditi uvjete. Predavaonice na fakultetima su prilagođene određenim predmetima (laboratorijske vježbe, računarski praktikumi...), pa bi tako možda trebalo definirati uvjet da se neki predmet može održavati u određenom tipu predavaonice. Također, svi uvjeti koji su definirani u ovom radu nisu bitni za održavanje nastave, npr. nije bitno da se predmet održava u istoj predavaonici.

Bibliografija

[1] *International Timetabling Competition*, (2007), http://www.cs.qub.ac.uk/itc2007/index.htm.

[2] V. A. Bardadym, *Computer-Aided School and University Timetabling: The New Wave*, (1996), https://link.springer.com/chapter/10.1007/3-540-61794-9_50.

[3] T. B. Cooper. i J. H. Kingston, *The complexity of timetable construction problems*, (1995), https://link.springer.com/chapter/10.1007/3-540-61794-9_66.

[4] Martin Geiger, *Applying the threshold accepting metaheuristic to curriculum based course timetabling*, Annals of Operations Research **194** (2008), 189–202.

[5] E. Norgren J. Jonasson, *Investigating a Genetic Algorithm-Simulated Annealing Hybrid Applied to University Course Timetabling Problem*, (2016), 254–260, https://www.diva-portal.org/smash/get/diva2:927039/FULLTEXT01.pdf.

[6] Zhipeng Lü i Jin Kao Hao, *Adaptive Tabu Search for course timetabling*, European Journal of Operational Research **200** (2010), 235–244.

[7] H. Turabieh S. Abdullah, *Generating University Course Timetable Using Genetic Algorithms and Local Search*, Convergence Information Technology, International Conference on **1** (2008), 254–260.

[8] M Vuković, *Složenost algoritama*, 2019.

[9] J. Wahid, *Construction of Initial Solution Population forCurriculum-Based Course Timetabling using Combination of Graph Heuristics*, (2016), https://bit.ly/3q95uJE.

[10] M Čupić, *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.*, 2013.

Sažetak

U ovom radu je predstavljen problem rasporeda sati za fakultete. Instanca problema koju promatramo je NP-teška, pa za njegovo rješavanje koristimo meta-heuristiku genetski algoritam. U radu je opisana svaka faza algoritma - od generiranja početne populacije do implementacije križanja i mutacija. Istaknute su prednosti i mane svake faze algoritma. Rezultati su uspoređeni s rezultatima iz radova koji koriste istu instancu problema te su navedeni mogući načini poboljšanja algoritma.

Summary

In this thesis we present university course timetablinc problem and one way of solving it. The instance of the problem we are observing is NP-hard, so we use the meta-heuristic genetic algorithm to solve it. All phases of the algorithm, starting from generating initial population to crossover and mutations, are described with their advantages and disadvantages. Results are then compared with others in similar papers. Finally, possible algorithm improvements are described as well.

Životopis

Vedran Deduš rođen je 15. 05. 1994. u Zagrebu. Pohađao je osnovnu školu Petar Zrinski i matematičku V. gimnaziju. Preddiplomski sveučilišni studij Matematika upisuje 2013. godine na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. Studij završava 2018. Po završetku preddiplomskog studija, na istom fakultetu upisuje diplomski sveučilišni studij Računarstvo i matematika. Za vrijeme preddiplomskog studija radio je u Privatnoj Gimnaziji Marul, gdje je držao dopunsku nastavu iz matematike. Za vrijeme diplomskog studija kreće raditi za tvrtku CROZ na poziciji softverskog inženjera.