SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 000

Izrada rasporeda laboratorijskih vježbi uporabom genetskog algoritma

Filip Pankretić

Zagreb, lipanj 2023.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljujem svojoj obitelji na svoj pružanoj podršci. Hvala im što su vjerovali u mene i bodrili me na mom putu.

SADRŽAJ

Popis slika **vi**

1. Uvod **1**

2. Problem raspoređivanja studenata u grupe **2**

2.1. Stroga ograničenja 2

2.2. Blaga ograničenja 3

3. Genetski algoritam **4**

4. Dodjela termina laboratorijskih vježbi **7**

4.1. Reprezentacija rasporeda u obliku jedinke 7

4.1.1. Niz termina s listom studenata 8

4.1.2. Niz studenata s dodijeljenim terminom 9

4.2. Ocjena rasporeda 10

4.2.1. Dobrota i kazna 10

4.2.1.1. Dobrota 11

4.2.1.2. Kazna 11

4.2.1.3. Kazna s kombinacijom dobrote 11

4.2.2. Ograničenje broja studenata 12

4.2.3. Izračun preklapanja 12

4.2.4. Slijedni termini 13

4.2.5. Pauza nastala nakon dodijeljenog termina 14

4.2.6. Termin dodijeljen u danu bez nastavnih obveza 14

4.3. Elitizam 15

4.4. Odabir roditelja 16

4.4.1. Proporcionalna selekcija 16

4.4.2. Turnirska selekcija 18

4.5. Križanje roditelja 20

4.5.1. Nasumično križanje 20
4.5.2. Alternirajuće k-križanje 20
4.6. Mutacija 21
4.6.1. Promjena termina za fiksan broj mjesta 22
4.6.2. Nasumičan odabir termina 23
4.7. Veličina populacije 23

5. Usporedba rezultata **24**
5.1. Vrijednosti odabrane za izračun kazne 24
5.2. Elitizam i odabir roditelja 27
5.3. Mutacija 30
5.4. Križanje 30
5.5. Vremenska komponenta i veličina populacije 31

6. Zaključak **34**

Literatura **35**

POPIS SLIKA

3.1. Podjela evolucijskog računanja na glavne grane uz odabrane algoritme (Čupić, 2011). 4

4.1. Prikaz zapisa nastavnih obveza studenta u memoriji. 8
4.2. Prikaz niza termina s listom studenata u memoriji. 9
4.3. Prikaz niza termina s listom studenata u memoriji. 10
4.4. Provjera preklapanja dvaju termina. 13
4.5. Generirani raspored s uključenim elitizmom. 15
4.6. Generirani rasporedi s isključenim elitizmom. 16
4.7. Proporcionalna selekcija roditelja. 18
4.8. Turnirska selekcija roditelja. 19
4.9. Nasumično križanje. 20
4.10. Alternirajuće križanje s K = 3. 21
4.11. Alternirajuće križanje s K = 1. 21
4.12. Alternirajuće križanje s K = N/2. 22
4.13. Mutacija uvečanjem termin za fiskan broj K = 3. 22
4.14. Mutacija dodjelom nasumičnog termina. 23

5.1. (a) kazna = 25 (b) kazna = 150 (c) kazna = 250 26
5.2. Rasporedi s uključenim elitizmom. 27
5.3. Rasporedi s isključenim elitizmom. 27
5.4. Poboljšana verzija proprocionalne selekcije s isključenim elitizmom. . 28
5.5. Osnovna verzija proprocionalne selekcije s isključenim elitizmom. . . 28
5.6. Poboljšana verzija proprocionalne selekcije s uključenim elitizmom. . 29
5.7. Osnovna verzija proprocionalne selekcije s uključenim elitizmom. . . 29
5.8. Mutacija promjenom termina za fiksni broj mjesta. 30
5.9. (a) nasumično (b) naizmjenično s K=1 (c) naizmjenično s K=3 (d) pola rasporeda 30

5.10. Mutacija sa stopom mutacije 0. 32
5.11. Mutacija sa stopom mutacije 0.1. 32
5.12. Referentne postavke algoritma. 33

1. Uvod

Izrada rasporeda problem je koji je prisutan svakodnevno, kako u obrazovnim institucijama poput škola, fakulteta, učilišta tako i u raspoređivanju satnice u industriji. Svaki raspored ima eksponencijalan broj konfiguracija i rješenje problema jednostavno nije izvedivo iscrpnim pretraživanjem kombinacija grubom silom.

Problem koji će se obrađivati u ovom radu podskup je problema, potrebno je generirati raspored laboratorijskih vježbi za 831 studenta u 37 grupa laboratorijskih vježbi bez preklapanja u njihovim rasporedima.

Cilj ovog rada je opisati rad metaheuristika, konkretno generacijskog genetskog algoritma koji pretražuje prostor stanja simuliranjem procesa evolucije. Potrebno je pomoću eksperimentalnih rezultata pronaći najbolju konfiguraciju algoritma koji će u konačnosti generirati kvalitetan raspored iskoristiv za neometano izvođenje nastave.

2. Problem raspoređivanja studenata u grupe

U obrazovnim institucijama, dva najčešća problema slaganja rasporeda su dodjela grupa studentima i dodjela termina ispita. Raspored se smatra poželjnim ako profesor nema istovremeno dva predavanja, dva predmeta se istovremeno ne drže u istoj učionici, studenti nemaju više predavanja istovremeno i zadovoljena su sva stroga i što više blagih ograničenja (Ganguli i Roy, 2017).

Za većinu problema raspoređivanja pokazano je da su NP-teški[1] i da ne mogu biti riješeni u polinomnom vremenu koristeći deterministički algoritam. To vrijedi zbog kombinatorne eksplozije do koje dolazi zbog velikog broja, ograničenja, zahtjeva i velikog broja studenata, predmeta, nastavnika i slično.

Prilikom izrade rasporeda potrebno je paziti na dvije vrste ograničenja: *stroga ograničenja* (engl. *hard constraints*) i *blaga ograničenja* (engl. *soft contraints*).

2.1. Stroga ograničenja

Stroga ograničenja odnose se na probleme koji su fizički nemogući. Poput toga da je profesor ili student na dva ili više mjesta istovremeno (Achini Kumari, 2017). Student mora imati dodijeljen samo jedan termin vježbi, a taj termin vježbi ne smije se preklapati s drugim terminima. Profesori koji održavaju vježbe ne smiju istovremeno održavati dva ili više termina, ali mogu imati dodijeljeno više termina u istom danu. U jednoj učionici istovremeno se ne može nalaziti više od jedne grupe studenata. Dodatne probleme mogu zadavati kapaciteti učionica, prilagođenost studentima s posebnim potrebama i slično.

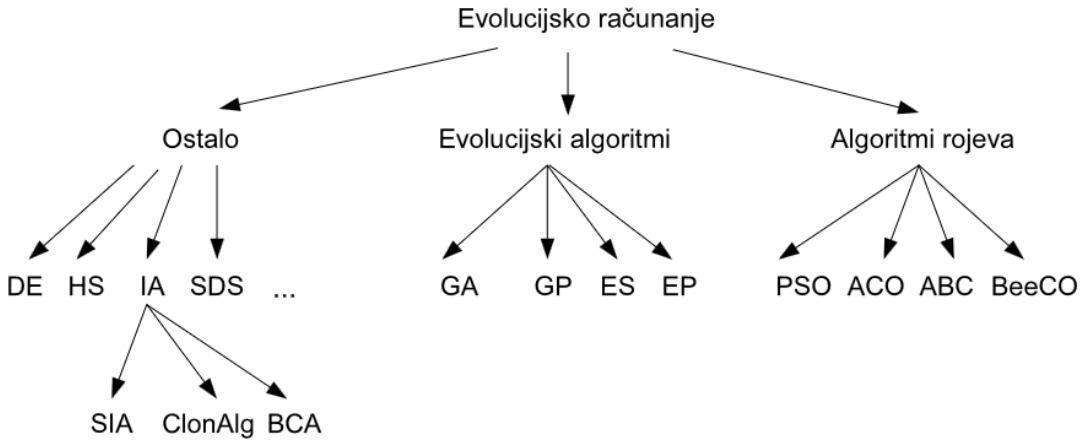
[1] Ne postoji algoritam u polinomijalnom vremenu.

2.2. Blaga ograničenja

Blaga ograničenja su ona koja se smatraju poželjnima, npr: jutarnji termini, želje profesora za određenim predavaonama, termini predavanja slijedno bez višesatnih pauza između. Često se stroga ograničenja zbog specifičnih situacija mogu tretirati kao blaga. Na primjer, kod predmeta s neobaveznom prisutnošću broj studenata smanji se nakon prvih nekoliko predavanja pa kršenjem maksimalnog kapaciteta ne nastaje veliki problem.

3. Genetski algoritam

Evolucijsko računanje je grana umjetne inteligencije koja se, najvećim dijelom, bavi rješavanjem optimizacijskih problema (Čupić, 2011). Evolucijsko računanje je familija metaheuristika[1] inspiriranih prirodnim procesima i biološkom evolucijom. Dijeli se na 3 grane: *evolucijske algoritme*, *algoritme rojeva* i *ostalo*.



Slika 3.1: Podjela evolucijskog računanja na glavne grane uz odabrane algoritme (Čupić, 2011).

Evolucijski algoritmi inspirirani su Darwinovom teorijom o postanku vrsta koja se temelji na 5 pretpostavka:

1. potomka uvijek ima više nego je potrebno
2. veličina populacije je prillbižno stalna
3. količina hrane je ograničena
4. kod vrsta koje se seksualno razmnožavaju, nema identičnih jedinki već postoje varijacije

[1] Skup algoritamskih koncepta primjenjivih na širok spektar problema

5. najveći dio varijacija prenosi se nasljeđem.

Genetski algoritam jedan je od evolucijskih algoritama stvorenih za rješavanje optimizacijskih problema, a koji izravno utjelovljuje navedene postavke (Čupić, 2019). Klasični evolucijski algoritam sadrži slijedeće dijelove:

- **Populacija** - podskup svih mogućih rješenja danog problema.
- **Jedinka** - jedno od rješenja u populaciji.
- **Funkcija dobrote** - numerička oznaka kvalitete neke jedinke.
- **Operator odabira roditelja** - operator odabira roditelja za daljnje križanje.
- **Operator križanja** - operator kombiniranja dviju jedinki kako bi se generirao jedan ili dva potomka.
- **Operator mutacija** - operator koji se koristi kako bi se održala raznolikost jedinki.

Sekvencijski[2] genetski algoritam može se podijeliti na dvije tipične izvedbe: *eliminacijski genetski algoritam* (engl. *steady-state genetic algorithm*) i *generacijski genetski algoritam* (engl. *generational genetic algorithm*) (Čupić, 2011). Kako se *generacijski genetski algoritam* koristi za rješenje zadanog problema, njegov pseudokod je dan u nastavku:

Algorithm 1 Generacijski genetski algoritam

```
populacija := new jednika[N]
dobrota := new ocjena[N]
for i := 0; i < N; i++ do
    dobrota[i] := ocijeni(populacija[i])
end for
while nije kraj do
    pom_populacija := new jednika[N]
    pom_dobrota := new ocjena[N]
    for i := 0; i < N; i++ do
        roditelj1 := odabirRoditelja(populacija)
        roditelj2 := odabirRoditelja(populacija)
        dijete := križaj(roditelj1, roditelj2)
    end for
```

[2]Rade u jednodretvenim sustavima

```
    populacija := pom_populacija
    dobrota := pom_dobrota
  end while
```

Kod generacijskog genetskog algoritma eventualno dobro dijete ne može se odmah iskoristiti; ono mora pričekati dok se ne završi čitava generacija.

Generacijski algoritam može taokođer biti elitistički[3]. Za razliku od neelitističkih inačica, kroz evoluciju graf jedinki bit će nazubljen te najbolja jedinka nove generacije ne mora biti bolja od prethodne (Čupić, 2019).

[3]Svojstvo algoritma da sadrži najbolju jedinku

4. Dodjela termina laboratorijskih vježbi

Problem dodjele termina laboratorijskih vježbi znatno je lakši od izrade cjelovitog rasporeda. Nasuprot originalnog problema izrade cjelovitog rasporeda, sad postoji već prethodno izrađen raspored i isti je potrebno popuniti dodatnim terminima laboratorijskih vježbi. Također, nije potrebno uzeti u obzir cijeli raspored već samo jedan tjedan u semestru, što znatno olakšava problem i sami pristup problemu. Nadalje, nije potrebno pratiti niti sljedeće stavke: željeni termini profesora ili asistenata, profesori ili asistenti dodijeljeni terminu, održavaju li se dva ili više termina istovremeno itd. Općenito je potrebno paziti na navedena ograničenja, ali trenutno ne spadaju u opseg ovog rada.

4.1. Reprezentacija rasporeda u obliku jedinke

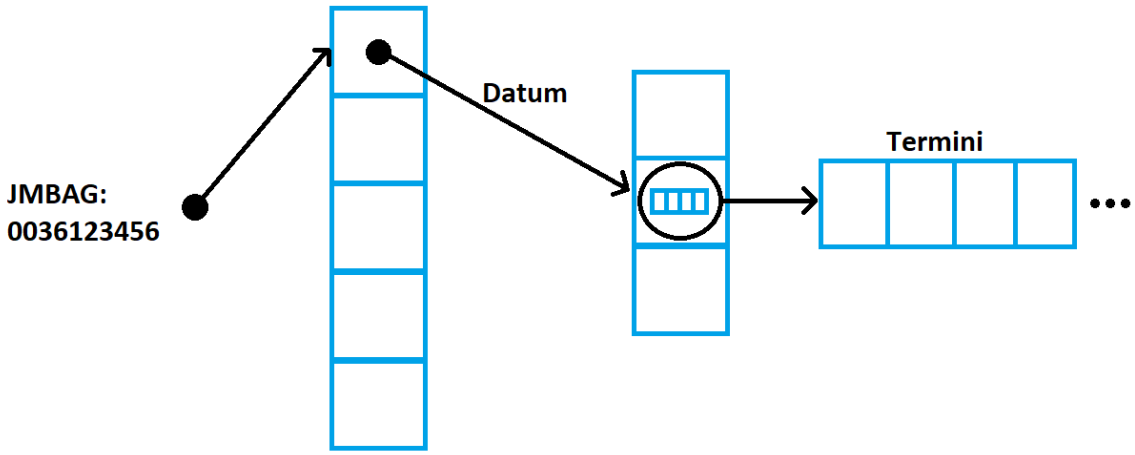
Svaki generirani raspored (također istoznačno: jedinka) potrebno je ispravno prikazati u memoriji. Ovisno o zapisu koji se koristi, problem se može znatno olakšati ili otežati. Osim zapisa same jedinke, program prima niz datoteka potrebnih za rad algoritma: popis studenata, popis dostupnih termina laboratorijskih vježbi, maksimalan i minimalan broj studenata po terminu te prethodna zauzetost studenata.

Prvo je potrebno definirati univerzalne podatke koji se koriste neovisno o zapisu samog rasporeda. Zapis termina laboratorijskih vježbi dan je formatu:

šifra_termina;kapacitet;datum;vrijeme_početaka;vrijeme_kraja;učiona, gdje je dodatno *kapacitet* u formatu: *makimalan_kapacitet/minimalan_kapacitet*. Svaki zapis spremamo u četiri odvojena niza: niz s cijelovitim zapisom, niz s početkom termina, niz s krajem termina i niz s datumom termina. Nadalje se termini referenciraju indeksom na kojem su raspoređeni.

Slijedi prikaz termina studenata. Moguće je da se u tablici zauzetosti studenata nalaze studenti kojima nije potrebno dodijeliti termin. Stoga se za prikaz rasporeda

koristi tablica raspršenog adresiranja[1] koja kao ključ prima JMBAG[2] studenta, kao vrijednost sadrži još jednu tablicu koja sada prima kao ključ datum termina, a kao vrijednost ima strukturu podataka vektor[3] s popisom prethodnih termina. Slijedni termini se spajaju u jedan zajednički termin kako bi se kasnije ubrzala provjera kolizija. Kolizije već prethodno dodijeljenih termina prilikom učitavanja podataka ne provjeravaju se.



Slika 4.1: Prikaz zapisa nastavnih obveza studenta u memoriji.

Popis studenata kojima je potrebno dodijeliti termin mogu sadržavati termin koji ne postoji u tablici prethodno dodijeljenih termina U tom slučaju pretpostavlja se da student u tom tjednu nema ostalih nastavnih obveza. Studenti se prikazuju kao niz JMBAG-ova kojemu se kasnije pristupa preko indeksa pojedinog elemenata.

Dva načina zapisa rasporeda koja će biti obrađena su *niz termina s listom studenata* i *niz studenata s dodijeljenim terminom*.

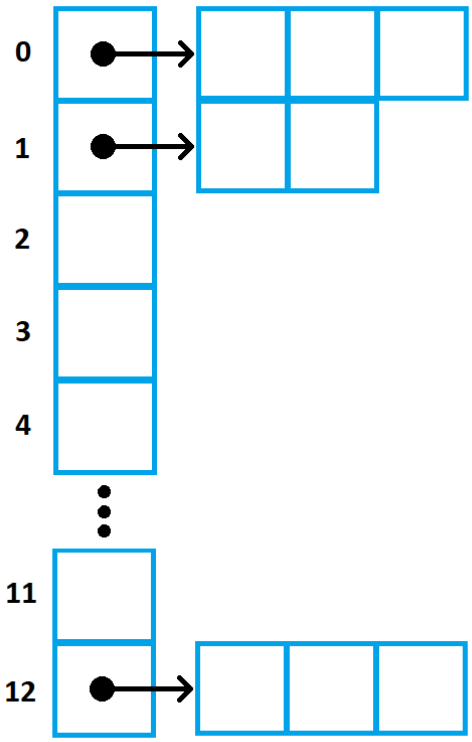
4.1.1. Niz termina s listom studenata

Prvi način zapisa je niz termina gdje svaki element tog niza sadrži vektor sa studentima kojima je dodijeljen taj termin.

Prednosti ovog zapisa su lakši ispis podataka, jer svaki termin odmah sadrži popis studenata i jednostavnija provjera zauzetog kapaciteta nekog termina. Prednosti je vrlo malo, dok mana ima puno više.

Neki od nedostataka su da je potrebno paziti da je svaki student dodijeljen samo jednom terminu, teže je implementirati operatore mutacije i operator križanja jer je-

[1] Struktura podataka koja u pretincima adresiranim nekim ključem sadrži određeni tip podatka.
[2] Jedinstveni matični broj akademskog građanina
[3] Struktura podataka koji automatski alocira i dealocira memoriju, a ponaša se kao niz



Slika 4.2: Prikaz niza termina s listom studenata u memoriji.

dinka ima dvije osi od kojih je jedna varijabla. Također je zbog toga otežan izračun preklapanja. Dodatno, ovaj prikaz je i memorijski i vremenski zahtjevan zbog korištenja dodatnih struktura podataka.

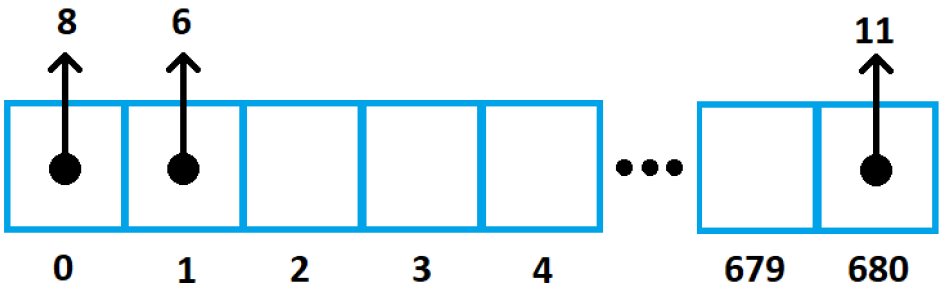
4.1.2. Niz studenata s dodijeljenim terminom

Drugi način prikaza je niz gdje indeks svakog elementa predstavlja jednog studenta, a element je indeks dodijeljenog termina.

Ovaj način pruža jednodimenzionalni niz što znatno olakšava definiranje operatora križanja i mutacije. Kako svaki student predstavlja jedan element niza, odmah je riješeno tvrdo ograničenje da student može pristupiti samo jednom terminu laboratorijskih vježbi. Lakše je izračunati pojedina preklapanja jer je za svakog studenta jednostavno pronaći dodijeljeni termin. Memorijski i vremenski je efikasniji od prvog načina zbog jednostavnije strukture.

Otežan je izračun zauzetog broja mjesta po terminu i ispis podataka. Za provjeru zauzeća potrebno je jednom proći kroz cijeli niz kako bi se za svaki termin prebrojala zauzeta mjesta, a za ispis rasporeda potrebno je za svaki termin proći kroz cijeli niz i ispisati studenta kako se naiđe na njega ili je potrebno u zasebnoj strukturu držati

popis studenata slično kao prvi zapis. Kako se raspored ispisuje samo povremeno, to ne predstavlja problem.



Slika 4.3: Prikaz niza termina s listom studenata u memoriji.

Zbog velikog broja prednosti druge strukture naspram prve, ta struktura odabrana je za prikaz rasporeda.

4.2. Ocjena rasporeda

Ocjena rasporeda rezultat je funkcije dobrote (engl. *fitness*). Točnije, to je numerička oznaka koja se koristi za ocjenu kvalitete rasporeda. Ukupna ocjena nekog rasporeda jednaka je sumi funkcija dobrote za svakog studenta iz nekog rasporeda.

$$\sum_{i=0}^{n} dobrota(student[i]) \tag{4.1}$$

4.2.1. Dobrota i kazna

Osim dobrote, raspored može koristiti i kaznu (engl. *punishment*). Kazna je mjera koliko je neki raspored loš. Ponekad je za problem lakše definirati ukupnu kaznu, nego ukupnu dobrotu. U praksi se pokušava ili minimizirati kazna ili maksimizirati dobrota. Kazna, kao i dobrota, zapravo je heuristika koja aproskimira koliko je daleko, odnosno blizu, dobiveno rješenje od cilja, tj. najboljeg rješenja.

U sklopu ovog rada, najbitnije je svojstvo rasporeda da nema preklapanja, stoga će bolji raspored biti onaj koji ima manje preklapanja, a potom onaj koji ima bolju ocjenu.

Prilikom izračuna ocjene rasporeda, sljedeća svojstva uzeta su kao pretpostavke:

- studenti preferiraju imati vježbe u istom danu kad već imaju nastavne aktivnosti
- bolji je termin koji stvara manju rupu do najbliže nastave aktivnosti nego onaj koji stvara veću

– broj nastalih pauza u rasporedu nije bitan i
– bolji je termin koji je slijedno s nastavnim aktivnostima bez pauze neovisno o opterećenju taj dan.

4.2.1.1. Dobrota

Kad bi raspored koristio dobrotu za svoju ocjenu, neke od heuristika koje bi se mogle koristiti su broj studenata koji nemaju preklapanje i broj studenata kojima je dodijeljen termin slijedno s ostalim nastavnim aktivnostima.

Teško je definirati kvalitetnu funkciju dobrote za raspored jer je vrlo malo svojstva kojima se može dodijeliti smislena numerička vrijednost. Čak i ako se uspiju pronaći smislene karakteristike za funkciju dobrote, teško je namjestiti vrijednosti tako da algoritam preferira sva svojstva istovremeno.

Na primjer, ako se za svakog studenta bez preklapanja funkcija dobrote poveća za 10, a za svaki slijedni raspored se poveća za 5, algoritam će vrlo vjerojatno preferirati studente koji nemaju preklapanja, a neće preferirati da studenti imaju slijedna predavanja. Opet, može se dogoditi da algoritam sve studente stavi u nekoliko termina gdje svi studenti imaju slijedna predavanja, ali s puno preklapanja i puno prekršenih ograničenja.

4.2.1.2. Kazna

Rasporedi sati esencijalno imaju jako puno ograničenja koja ne smiju biti prekršena ili smiju, ali do neke granice. Stoga, lakše je implementirati funkciju kazne rasporeda. Za kaznu, isto kao i za dobrotu, vrijedi da ako se postavi prevelika kazna za bilo koje svojstvo, vjerojatno je da će dominirati rasporedi koji imaju minimiziranu kaznu s tim svojstvom.

Naime, kako je cilj pronaći raspored koji nema preklapanja, to ne predstavalja preveliki problem, ali ako se koristi kazna umjesto dobrote za ocjenu algoritma, dostupan je puno veći broj heuristika koje mogu potencijalno pronaći kvalitetniji raspored.

Neke od dostupnih heuristika su: broj preklapanja, broj pauza, vrijeme preklapanja, koliko su prekršeni zahtjevi kapaciteta nekog termina i vremenski period najmanje pauze nastale nakon dodavanja termina.

4.2.1.3. Kazna s kombinacijom dobrote

Kao najbolja opcija ipak se pokazala kazna s kombinacijom dobrote. Osim heuristika kazne, uvodi se mali broj heuristika koje rasporedu daju nagradu, to jest smanjuju

kaznu za iznos pomnožen malim koeficijentom ako su zadovoljena neka dobra svojstva u rasporedu. Kada bi nagrada bila velika, onda bi algoritam zapravo pokušavao maksimizirati nagradu umjesto da minimizira kaznu.

4.2.2. Ograničenje broja studenata

Kod izračuna cijene prilikom kršenja maksimalnog ili minimalnog kapaciteta koriste se sljedeće pretpostavke:

- minimalni broj studenata u terminu je uvijek 15
- maksimalan broj studenata u terminu je uvijek 16
- gore je kada broj studenata premaši maksimalan kapacitet, nego kad je broj studenta manji od minimuma.

Nakon što je dostupna informacija o broju studenata za svaki termin, dodjeljuje se kazna ovisno o prekršenim ograničenjima. Kazna za termine kojima je dodijeljen veći broj studenata od maksimalnog jednaka je razlici broja dodijeljenih studenata i maksimalnog broja studenata za taj termin pomnožena s nekim koeficijentom.

$$(brojStudenata - 16) \cdot koeficijentZaViseStudenata \tag{4.2}$$

Istovjetno, termini kojima je dodijeljen manji broj studenata od minimalnog doprinose kaznom jednakom razlici minimalnog broja studenata i dodijeljenog broja studenata pomnoženom s nekim koeficijentom.

$$(15 - brojStudenata) \cdot koeficijentZaManjeStudenata \tag{4.3}$$

Koeficijent za manji i veći broj studenata nisu jednaki, što se temelji na trećoj zadanoj pretpostavci. Nastavniku je očigledno jednostavnije ispitati manji broj studenata od minimuma, nego veći od maksimuma, zato je koeficijent za manje studenata manji od onog za više. Dobro svojstvo za ta dva koeficijenta je da njihova razlika ne bude prevelika. Jer, ako bi koeficijent za manji broj studenata bio premali, onda bi algoritam mogao dominirati rasporedima koji nemaju popunjena sva mjesta u učionicama, a isto vrijedi i obratno.

4.2.3. Izračun preklapanja

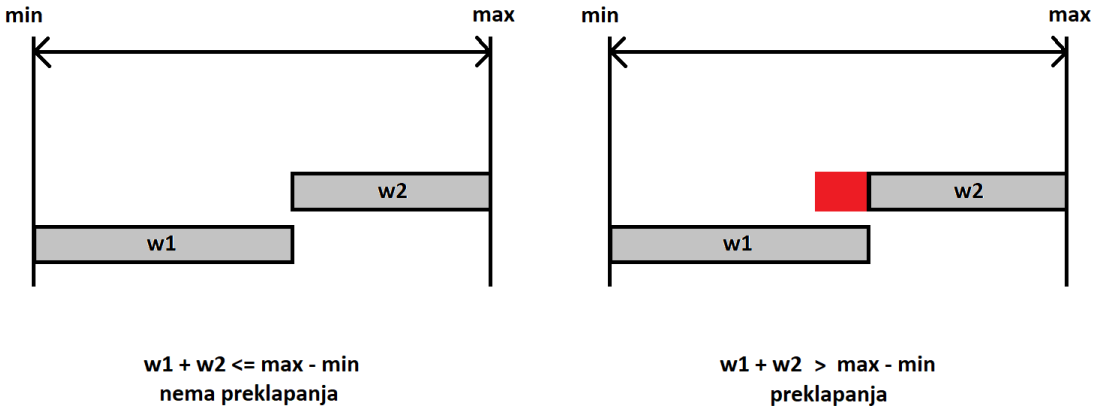
Student može imati preklapanja u rasporedu prije nego li mu je dodijeljen termin laboratorijskih vježbi. Kod takvih situacija računaju se zasebno kolizije dodijeljenog termina sa svakim prethodnim terminom zasebno, tako da teoretski jedan student može

proizvesti više od jedne kolizije. Kazna se dodjeljuje svakoj od tih kolizija kako bi se čak i u slučaju kolizija što prije odbacio raspored gdje pojedinim studentima gotovo pa nije moguće pristupiti terminu laboratorijskih vježbi.

Preklapanje dvaju termina lako se može provjeriti formulom:

$$max(t1, T1) - min(t0, T0) < (t1 - t0) + (T1 - T0) \tag{4.4}$$

T0 i *T1* predstavljaju redom početak i kraj dodijeljenog termina, a *t0* i *t1* redom početak i kraj nekog od termina ostalih nastavnih obveza. *max(t1, T1) - min(t0, T0)* predstavlja interval na kojem se nalaze termini koji se trenutno uspoređuju i ako je taj interval manji od zbroja duljina zasebnih intervala, onda postoji preklapanje.



Slika 4.4: Provjera preklapanja dvaju termina.

Najvažnije svojstvo rasporeda je da ima minimalni broj preklapanja, zato se umnožak preklapanja množi s koeficijentom koji je veći od ostalih. Razlog tome je kako bi se što prije pronašao raspored koji ima minimalan broj preklapanja i onda se krenulo u dodatno poboljšanje ostalih svojstva rasporeda.

4.2.4. Slijedni termini

Kao što je spomenuto u poglavlju 4.2.1.3, zbog nekih poželjnih svojstava rasporedu se može dodijeliti nagrada. Nagrada je implementirana tako da ukupnu kaznu rasporeda smanji za neku brojčanu konstantu. Manje konstante osiguravaju, pogotovo ako je implementirano više nagrada, da se ne preferiraju rasporedi s više nagrada i manje prekršenih ograničenja.

Jedno od tih svojstva je slijedno održavanje termina. Sljednost se provjerava samo ako se navedeni termin ne preklapa s niti jednim ostalim terminom. Za provjeru sljed-

nosti može se koristiti formula 4.4, ali tako da se znak nejednakosti promijeni u znak jednakosti.

$$max(t1, T1) - min(t0, T0) = (t1 - t0) + (T1 - T0) \tag{4.5}$$

Ali, zbog toga što provjera sljednosti ponovo uspoređuje dobiveni termin sa svim ostalim terminima, poželjno je, ako je moguće, ubrzati provjeru. Zato se za provjeru preklapanja dvaju termina koristi sljedeća formula:

$$t0 = T1 \vee T0 = t1 \tag{4.6}$$

gdje *T0, T1, t0* i *t1* imaju isto značenje kao i formula 4.4. Dovoljno je samo provjeriti počinje li ili završava dodijeljeni termin onda kada bilo koji drugi redom završava ili počinje.

4.2.5. Pauza nastala nakon dodijeljenog termina

Broj pauza u rasporedu nije bitan, već se za izračun kazne koristi samo minimalna udaljenost dodijeljenog termina do svih ostalih termina. Provjera pauza se ne vrši u slučaju da su već pronađena dva slijedna termina ili ako postoji kolizija. Za pronalazak najmanjeg perioda pauze u rasporedu koristi se sljedeća formula:

$$min(|T0 - t1|, |T1 - t0|) \tag{4.7}$$

gdje *T0, T1, t0* i *t1* imaju isto značenje kao i formula 4.4. Potom se kazna povećava za iznos izračunat formulom:

$$koeficijent \cdot najmanjiRazmak^2 \tag{4.8}$$

Koeficijent služi za podešavanje doprinosa kazne. Prema pretpostavci bolji je raspored onaj koji nema pauze što znači da će rasporedi s većom pauzom biti gori, tj. imat će veću kaznu. Kako bi se ta pauza što više minimizirala, najmanji razmak se kvadrira kako bi se studentima dodijelili termini sa što manjim vremenskim razmakom.

4.2.6. Termin dodijeljen u danu bez nastavnih obveza

Temeljeno na prvoj pretpostavci izračuna ocjene rasporeda, studenti preferiraju termine laboratorijskih vježbi u istom danu kada imaju neku drugu nastavnu aktivnost,

stoga ako student nema drugih nastavnih aktivnosti u danu održavanja laboratorijskih vježbi, dodjeljuje se kazna relativno velikog iznosa.

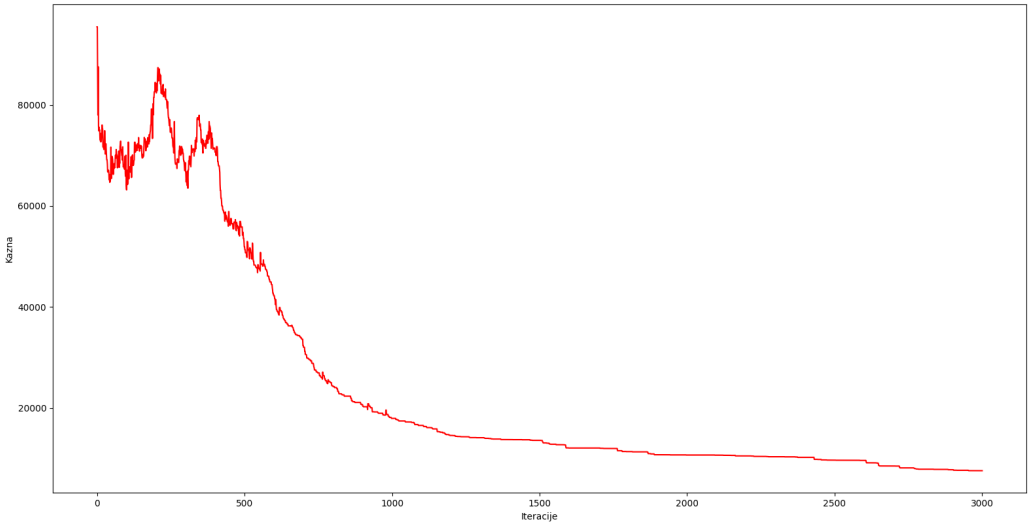
Provjera ima li student neke druge nastavne aktivnosti tog dana održava se na samom početku prije provjera preklapanja, sljednosti termina i pauze s najmanjim vremenskim razmakom. Ako je ta provjera istinita, onda se ostala svojstva ne provjeravaju.

4.3. Elitizam

Elitizam (engl. *elitism*) je svojstvo algoritma da uvijek sadržava najbolji raspored.

Odabir elitnog rasporeda vrši se tako da se prolazi kroz kazne svih jedinki i gleda se ima li raspored manji broj preklapanja od prethodnoga. U slučaju da ima, raspored na kojeg se naišlo bira se kao elitni, a u slučaju da je broj preklapanja jednak, onda se odabire onaj s manjom kaznom.

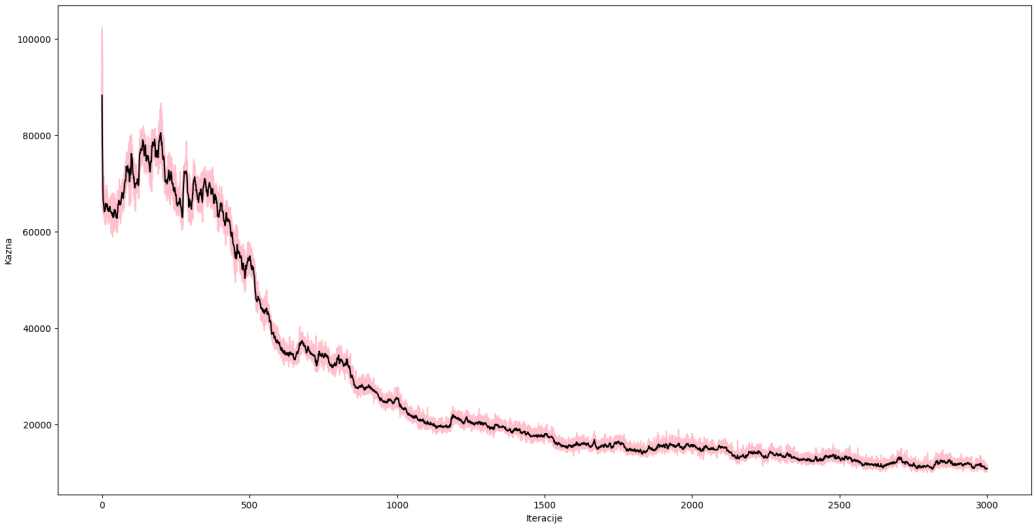
Kod elitističkih algoritama graf koji prikazuje kretanje funkcije kazne kroz evoluciju je monoton. Zbog načina odabira elitinog rasporeda, ovdje će to svojstvo monotonosti biti vidljivo oko 1000 iteracije kada broj preklapanja postane nula.



Slika 4.5: Generirani raspored s uključenim elitizmom.

Kod algoritama koji nisu elitistički taj će graf biti nazubljen: prosječne vrijednosti funkcije kazne asimptotski će padati, ali je sasvim moguće da u nekon koraku algoritma kazna najbolje jedinke bude veća od kazne najbolje jedinke koju je algoritam imao u nekom prethodnom koraku (Čupić, 2019).

Prilikom izrade rasporeda, očuvanje elitnog rasporeda je pokazalo dobra svojstva



Slika 4.6: Generirani rasporedi s isključenim elitizmom.

poput brže konvergencije i manjeg raspršenja. A kod nekih postavka algoritma, jedino su s uključenim elitizmom rasporedi počeli konvergirati. To je zato što elitni raspored pruža algoritmu točku za koju se može držati kako bi se odupro divergenciji zbog raznih loših utjecaja na algoritam. Tako sada algoritam postaje striktno konvergentan.

4.4. Odabir roditelja

Kod odabira roditelja važno je da oba roditelja budu različita kako bi se održala raznolikost rasporeda. Kada bi oba odabrana roditelja bila jednaka, raspored nastao operatorom križanja bi bio jednak roditeljima, a to nije poželjno svojstvo genetskog algoritma.

Operator odabira roditelja sužava prostor pretrage riješenja jer temeljem različitih algoritama roditelji koji imaju bolju ocjenu vrjerojatnije će biti odabrani. Primjer tih operatora su *Proporcionalna selekcija* (engl. *Roulette-Wheel Selection*) i *K-turnirska selekcija* (engl. *K-torunament Selection*).

4.4.1. Proporcionalna selekcija

Proprcionalna selekcija svakom roditelju dodjeljuje brojčanu vrijednost postotka sa šansom da upravo on bude odabran. Vrijednost je veća što je raspored bolji, a manja što je raspored gori.

Osnovna izvedba algoritma izgleda ovako:

Algorithm 2 Proporcionalna selekcija

```
raspon := min(kazna) + max(kazna)

vjerojatnosti := new float[N]
for i := 0; i < N; i++ do
    vjerojatnosti[i] := raspon - kazna[i]
end for
zbroj_vjerojatnosti := sum(vjerojatnosti)
for i := 0; i < N; i++ do
    vjerojatnosti[i] := vjerojatnosti[i] / zbroj_vjerojatnosti
end for

broj := generiraj(0, 1)
podrucje := 0
for i := 0; i < N; i++ do
    podrucje := podrucje + vjerojatnosit[i]
    if broj <= podrucje then
        odaberiIStani(i)
    end if
end for
```

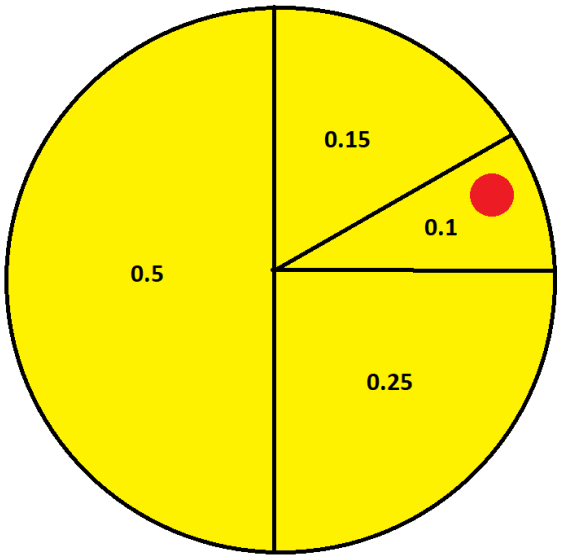
Proporcionalni odabir tradicionalno je implementiran tako da svaki element podijelimo sa sumom elemenata, te će tako element s najvećom vrijednošću imati najveću šansu odabira, no kako je sada bolji onaj raspored koji ima manju kaznu, potrebno je ipak podatke transformirati.

Transformirane vrijednosti dobiju se tako da se nad svakim elementom provede transformacija:

$$\forall i \; vjerojatnost[i] = max(kazna) + min(kazna) - kazna[i] \tag{4.9}$$

Ovom transformacijom zamjenjuje se veličina svih elemenata tako da najmanji elementi postaju najveći i obrnuto. Sada je samo potrebno svaki element podijeliti sumom svih elemenata kako bi se svi elementi razdijelili po intervalu od 0 do 1.

Veliki brojevi predstavljaju problem, kod populacije s 5 rasporeda od kojih nakon transformacije svi imaju redom vrijednosti 10001, 10002, 10003, 10004 i 10005 razlike vjerojatnosti svakog rasporeda bit će zanemarive i svi rasporedi će efektivno imati vjerojatnost od 20%. Kako bi se taj efekt ublažio, moguće je sve kazne iz popula-



Slika 4.7: Proporcionalna selekcija roditelja.

cije umanjiti vrijednošću za jedan manjom od najmanje vrijednosti. Tako sada redom vrijednosti iznose 1, 2, 3, 4, 5.

Sljedeći problem predstavljaju negativne vrijednosti, zbog nagrada koje rasporedi mogu poprimiti, kazna rasporeda može teoretski poprimiti negativnu vrijednost, zato je potrebno pronaći minimalnu vrijednost i u slučaju da je ta vrijednost negativna, svaki element se uveća za apsolutu vrijednost najmanje kazne.

Nadalje, vrijednosti svih kazni rasporeda u svakom krugu iteracije su skoro podjednake, stoga ako se u nizu nalazi velik broj rasporeda, svaki raspored imat će otprilike podjednaku šansu da bude odabran i tako se efektivno dobije nasumični odabir.

Nadalje, boljim rasporedom smatra se onaj koji ima manje kolizija, zbog toga je teško dodijeliti ispravnu vjerojatnost odabira, jedan način je da se umjesto kazne gleda broj kolizija, ali kada bi se broj kolizija smanjio na 0, svi rasporedi bi imali podjednaku vjerojatnost odabira.

Kada bi se sve ove adaptacije uvele u algoritam, jako bi usporio zbog velikog broja transformacija i kalkulacija prije samog odabira.

4.4.2. Turnirska selekcija

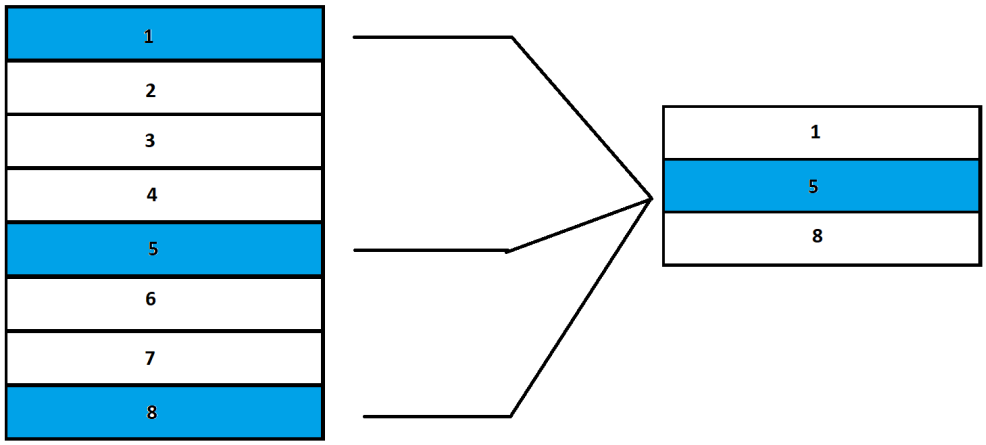
Turnirska selekcija kao ulaz u algoritam prima parametar k koji označava broj jedinki koji ulaze u turnir. Vrlo jednostavnim postupkom, iz populacije se nasumično odabere

k rasporeda nakon čega se jednostavno vrati onaj koji ima najmanji broj preklapanja ili, u slučaju jednakog broja preklapanja, onaj s najmanjom kaznom.

Algorithm 3 Proporcionalna selekcija

```
populacija := new raspored[k]
while size(populacija) != k do
    jedinka := odaberiNasumicno(rasporedi)
    if populacija ne sadrži jedinku then
        dodaj(populacija, jedinka)
    end if
end while
roditelj := odaberiNajbolji(populacija)
```

Kod punjenja pomoćne populacije, potrebno je provjeravati da se novo odabrani raspored već ne nalazi u populaciji.



Slika 4.8: Turnirska selekcija roditelja.

Ako se za vrijednost *k* u algoritmu postavi 1, onda je to efektivno odabir nasumičnog roditelja, a ako se za *k* postavi veličina populacije, onda se odabire elitna jedinka. Za broj odabranih rasporeda postavlja se manja vrijednost poput 2 ili 3 kako bi se omogućilo barem minimalno svojstvo kompetencije.

Turnirska selekcija rješava problem velikih brojeva jer odabir rasporeda ne ovisi o razlici kazni pojedinih rasporeda. Dodatno, negativne vrijednost isto ne utječu na ispravnost algoritma kao niti veliki broj rasporeda u populaciji.

Zbog svoje jednostavnosti, efikasnosti i robusnosti, turnirska selekcija s parametrom tri je odabrana kao algoritam za odabir roditelja.

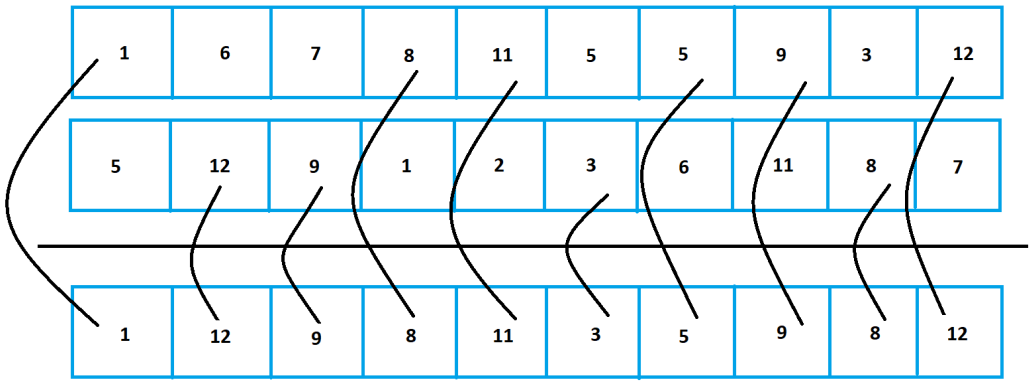
4.5. Križanje roditelja

Operator križanja roditelja, kao i operator odabira roditelja, sužava prostor pretrage. Križanjem dvaju roditelja nastaje jedno ili dva nova rješenja ovisno o konkretnim implementacijama. Odabrano je da operator križanja kao rezultat vraća jedno dijete.

Različite implementacije daleko najmanje utječu na kvalitetu dobivenih rasporeda čak i kod ekstremnih postavka operatora.

4.5.1. Nasumično križanje

Nasumičnim križanjem, za svaki element iz rasporeda postoji 50% šanse da bude odabran bilo koji od oba roditelja za dohvat tog elementa.



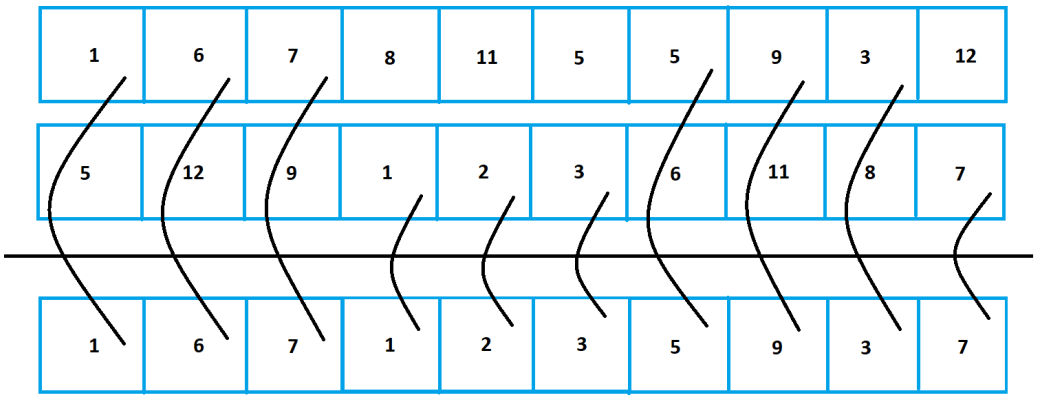
Slika 4.9: Nasumično križanje.

4.5.2. Alternirajuće k-križanje

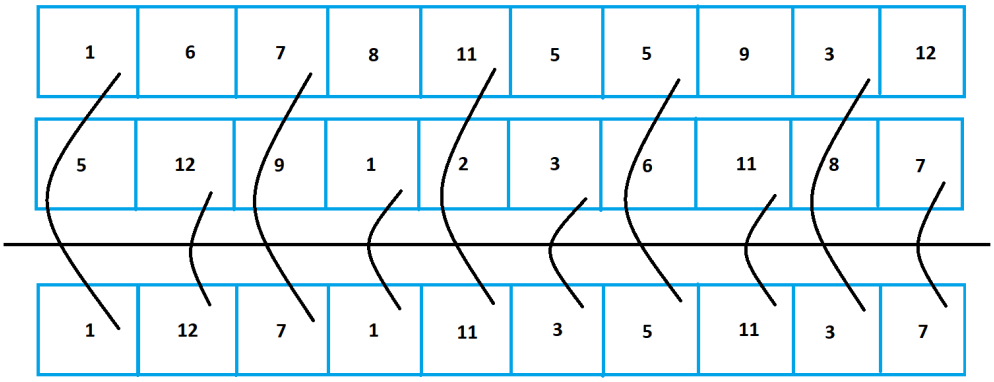
Različito od nasumičnog križanja, ovdje je slučajnost pristupna samo jednom, za odabir roditelja od kojeg počinje križanje. Nadalje, ovisno o parametru k koji je predan algoritmu, djetetu se postavlja idućih k vrijednosti odabranog roditelja te se potom roditelj mijenja.

Ako se kao parametar k postavi jedinica, onda se redom uzima termin jednog, pa termin drugog roditelja naizmjence.

Sljedeći posebni slučaj je ako se za parametar k postavi vrijednost N/2, onda algoritam jednostavno uzme polovicu svakog roditelja. Također, ispravno je nazvati ovaj slučaj križanje s polovičnom točkom prekida.



Slika 4.10: Alternirajuće križanje s K = 3.



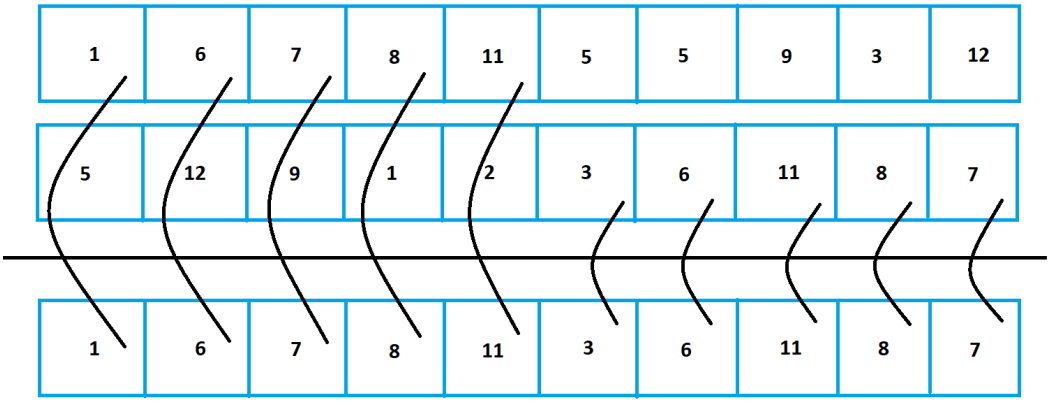
Slika 4.11: Alternirajuće križanje s K = 1.

Posljednje, ako se za parametar *k* odabere vrijednost veća od N/2, onda ovo postaje križanje s točkom prekida. Jedina razlika na sami algoritam križanja točkom prekida je što nisu dostupne točke prekida manje od N/2.

4.6. Mutacija

Osim operatora koji sužavaju područje pretrage, mutacija ga proširuje. Bez operatora mutacije, vrlo brzo bi sva nova rješenja počela stagnirati, tj. ne bismo dobili više novih rješenja. Operatoru je dodijeljena stopa mutacije, postotak studenata za koji će se promijeniti dodijeljeni termin.

Optimalna vrijednost mutacije iznosi oko 0.002 ili 0.2%. Raspored se sastoji od termina za 831 studenta, prema tome, nakon svake mutacije će se u prosjeku promijeniti grupa za 1.66 studenata što pruža dovoljno jako svojstvo širenja. Uz veće vrijednosti stope mutacije, konvergencija je sporija bez poboljšanja u kvaliteti rješenja.

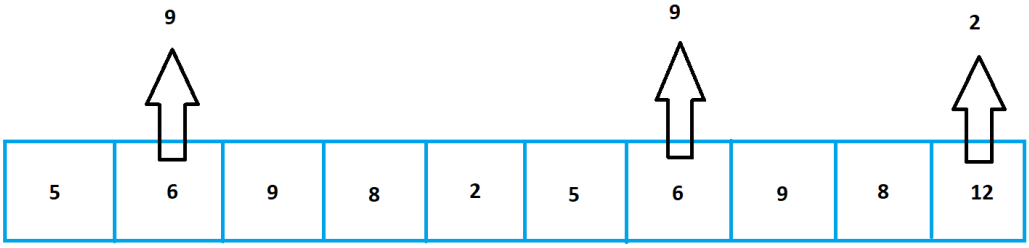


Slika 4.12: Alternirajuće križanje s K = N/2.

Kod ovog problema nije potrebno pronaći najbolje moguće rješenje, već dovoljno dobro. Zato i s jako malim stopama promjene, kad algoritam ima veću vjerojatnost zapeti u lokalnom, rješenje se smatra prihvatljivim.

4.6.1. Promjena termina za fiksan broj mjesta

Jedna od varijanti mutacije je promjena termina za fiksan broj mjesta. Na primjer, studenta se prebaci s termina 8 na termin 11.



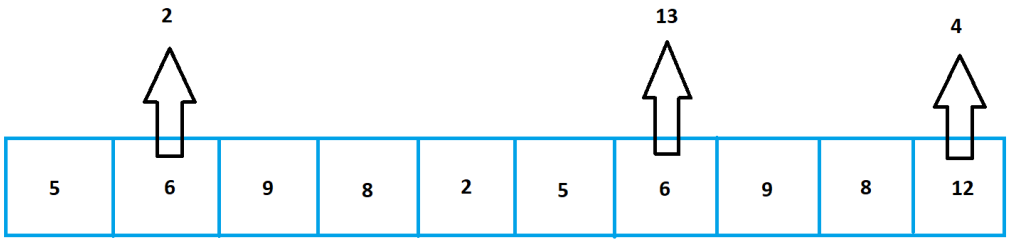
Slika 4.13: Mutacija uvečanjem termin za fiskan broj K = 3.

Ovako definiran operator kasnije u ciklusu pretrage, kad se raspored stabilizira i većina termina imaju popunjen kapacitet od 15 ili 16 mjesta, radi velik broj promjena što destabilizira ponašanje. Ako se napravi promjena termina za 2 studenta, pogotovo ako su oni u istoj grupi, novi raspored će vrlo vjerojatno imati veću kaznu. Prilikom promjene termina primjenjuje se formula:

$$\forall i \; termin[i] = (termin[i] + N) \mod brojTermina \tag{4.10}$$

4.6.2. Nasumičan odabir termina

Nasumičan odabir termin radi tako da se studentu dodijeli nasumično generiran broj termina između 0 i *brojTermina* gdje svaki termin ima jednaku šansu biti odabran.



Slika 4.14: Mutacija dodjelom nasumičnog termina.

Dodjelom nasumičnog termina umjesto pomaka termina za fiksni broj mjesta bolje rješava problem potencijalnog nakupljanja termina na jednom mjestu pa je odabran kao operator mutacije.

4.7. Veličina populacije

Veličina populacije je broj rasporeda koji se u svakom trenu čuva u memoriji. S manjom veličinom populacije povećava se broj iteracija po sekundi, ali je zato manja raznolikost rješenja. No s većom veličinom populacije smanjit će se broj iteracija po sekundi, ali će se povećati raznolikost rješenja te će se potencijalno pronaći bolje rješenje.

Operacije računanja kazne su skupe zbog velikog broja provjera kroz koje je potrebno proći pa će se s većom populacijom broj iteracija po sekundi drastično smanjiti. Potrebno je pronaći dobar balans kvalitete i brzine. Kroz testiranje, vrijednosti oko 25 su se pokazale jako dobre.

5. Usporedba rezultata

U ovom poglavlju bit će potkrijepljeni razni parametri odabrani kroz ostatak rada, razlike između različitih algoritama, rad konfigurabilnih algoritama ovisno o parametrima i brzina izvođenja ovisno o parametrima. Prilikom testiranja uzimaju se sljedeće pretpostavke, osim ako je navedeno drugačije:

- iznosi kazne:
  - previše studenata u terminu je kvadriran broj studenata pomnožen s 95
  - premalo studenata u terminu je kvadriran broj studenata pomnožen s 45
  - preklapanje iznosi 150
  - slijedni termin iznosi -5
  - pauza između termina nastala zbog odabranog termina je 2 pomnoženo s kvadriranim brojem sati
  - termin dodijeljen u danu bez ostalih nastavnih obveza iznosi 100
- algoritam je elitistički
- odabir roditelja je turnir s parametrom 3
- dijete nastaje nasumičnim križanjem roditelja
- mutacija nasumično odabire termin sa stopom mutacije 0.002
- bolji je onaj raspored koji ima manje preklapanja i onaj koji ima manju kaznu
- algoritam staje nakon 3000 iteracija
- 831 student se dijeli u 43 termina.

5.1. Vrijednosti odabrane za izračun kazne

Smanjivanjem kazne, algoritam je u ranijim iteracijama više sklon praznijim grupama ili grupama od nekoliko studenata. Provodi se ispitivanje o broju termina s prekršenim

Tablica 5.1: Popunjenost termina ovisno o kazni nezadovoljavanja minimlanog kapaciteta.

kazna	14 ili manje	17 ili više
15	3	22
30	4	19
45	3	18
55	5	16

ograničenjem minimalnog kapaciteta i maksimalnog kapaciteta. Ispitivanje se radi na 1000 iteracija.

Iz tablice je vidljivo kako kazne s vrijednošću 45 i 55 imaju najbolja svojstva, kazna od 55 je odabrana kako svejedno pri većem broju iteracija ne bi toliko bili preferirani termini s manjim brojem studenata.

Isti postupak se provodi za kaznu s prekršenim maksimalnim kapacitetom s 1500 iteracija.

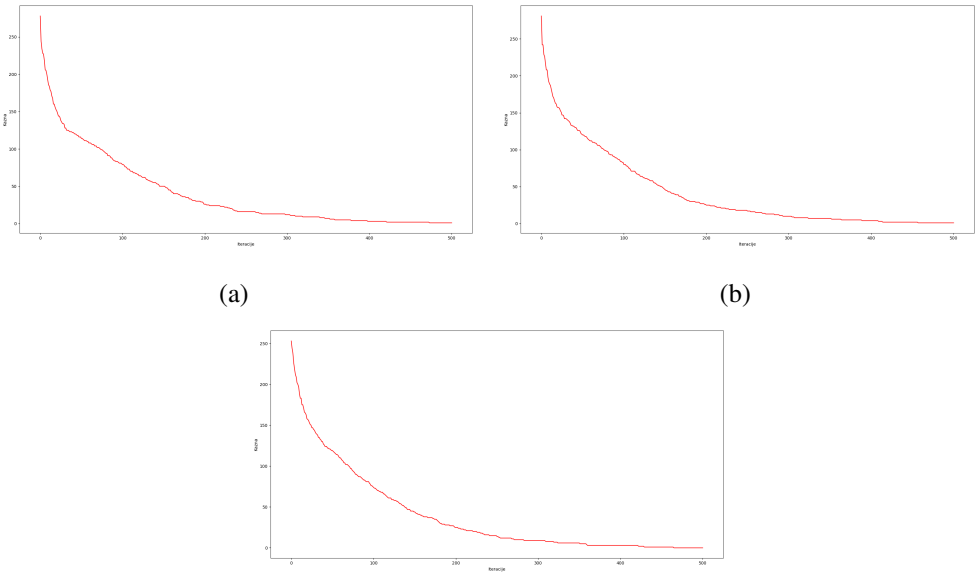
Tablica 5.2: Popunjenost termina ovisno o kazni nezadovoljavanja maksimalnog kapaciteta.

kazna	14 ili manje	17 ili više
50	6	11
75	5	12
95	2	6
110	2	13

Vrijednost kazne od 95 konzistentno pokazuje najbolja svojstva te je upravo iz tog razloga odabrana.

Nadalje slijedi ispitivanje o brzini minimizacije preklapanja ovisno o kazni. Prilikom ispitivanja broj iteracija je postavljen na 100.

Iz grafova je moguće primijetiti kako kazna preklapanja nema veliks utjecaj na algoritam, zbog turnirske selekcije dominira odabir rasporeda s manjim brojem preklapanja.



(a) (b)

(c)

Slika 5.1: (a) kazna = 25 (b) kazna = 150 (c) kazna = 250

Također, ispituje se i nagrada, teško je ispitati razliku u kazni rasporeda jer će rasporedi s većom nagradom prirodno imati manju kaznu i obrnuto, ali raspored manje kazne u tom slučaju ne mora biti bolje kvalitete. Stoga se ispituje broj prekršenih ograničenja kapaciteta termina laboratorijskih vježbi kako bi se pokazalo da su rasporedi s manjom nagradom skloniji boljim rješenjima.

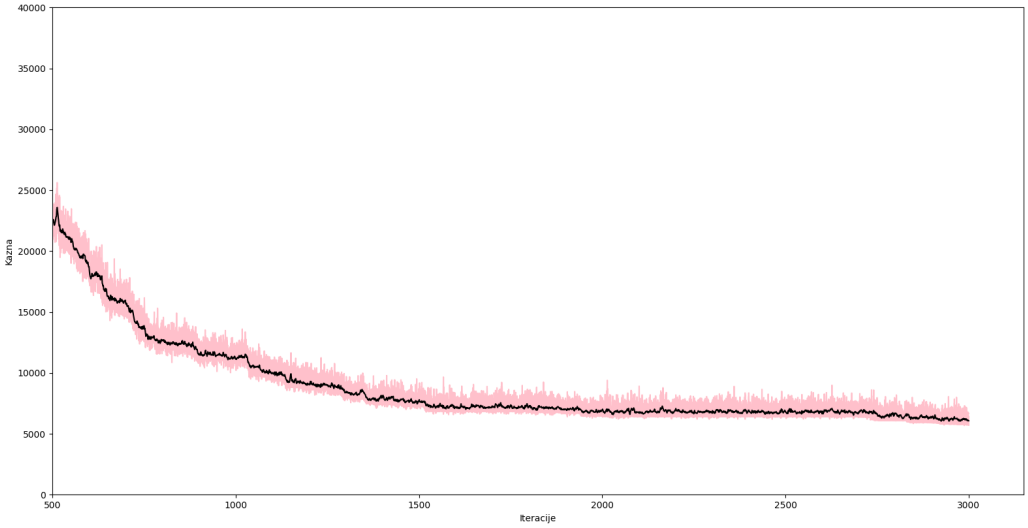
Tablica 5.3: Prekršena ograničenja kapaciteta ovisno o nagradi slijednih temina.

nagrada	prekršena ograničenja
1	4
5	5
15	7
25	6

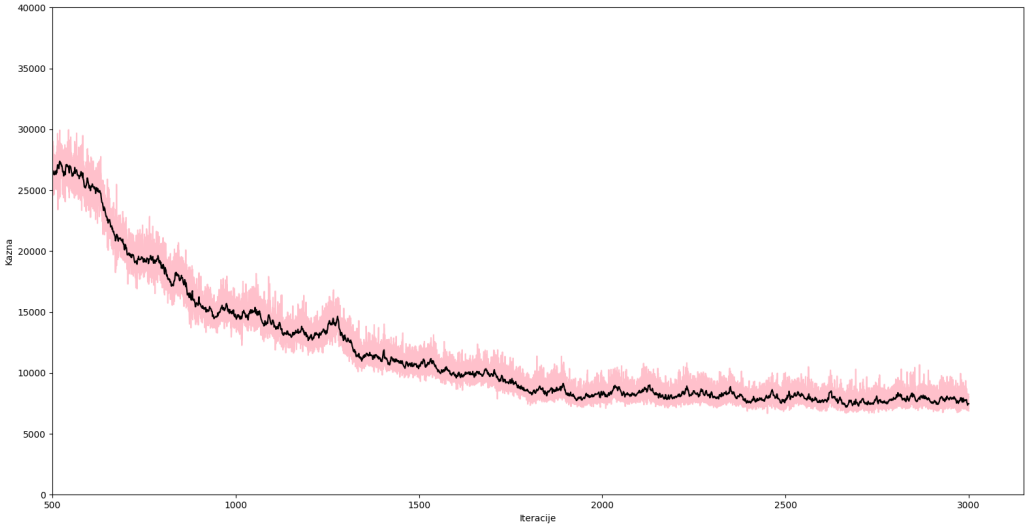
U tablici se nalaze prosječne vrijednosti termina s prekršenim ograničenjima, nagrada iznosa 5 je odabrana jer pruža mali broj termina s prekršenim ograničenjima, a rasporedima ipak dodjeljuje dobra svojstva.

5.2. Elitizam i odabir roditelja

Kod promatranja elitizma, razlike između elitističke i neelitističke varijante su relativno male, ipak s uključenim elitizmom rasporedi imaju manje varijacije i brže konvergiraju što je vidljivo na grafovima. Prvih 500 iteracija je odrezano na prikazima.



Slika 5.2: Rasporedi s uključenim elitizmom.



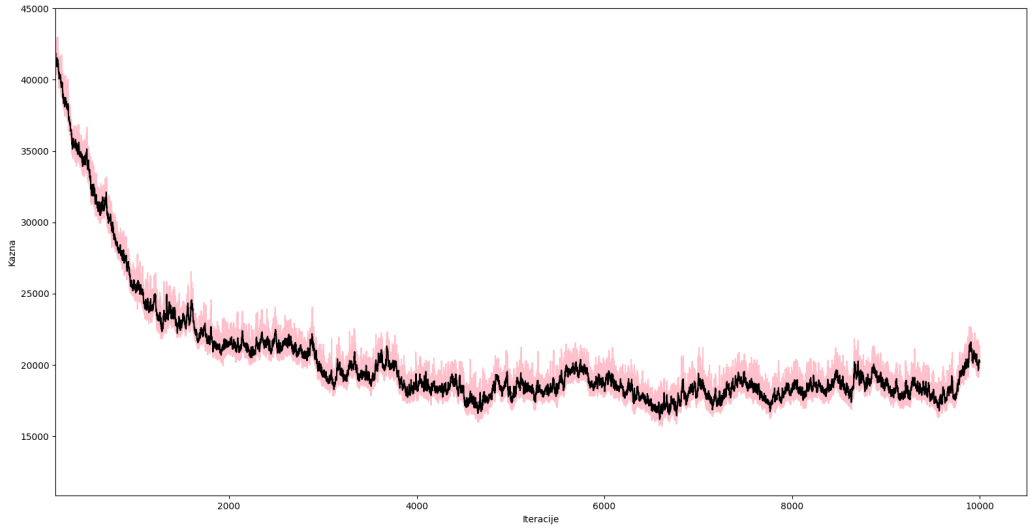
Slika 5.3: Rasporedi s isključenim elitizmom.

Elitistički algoritam u 3000 iteracija dostigne kaznu od oko 5500 dok neelitistička inačica završi na oko 7500.

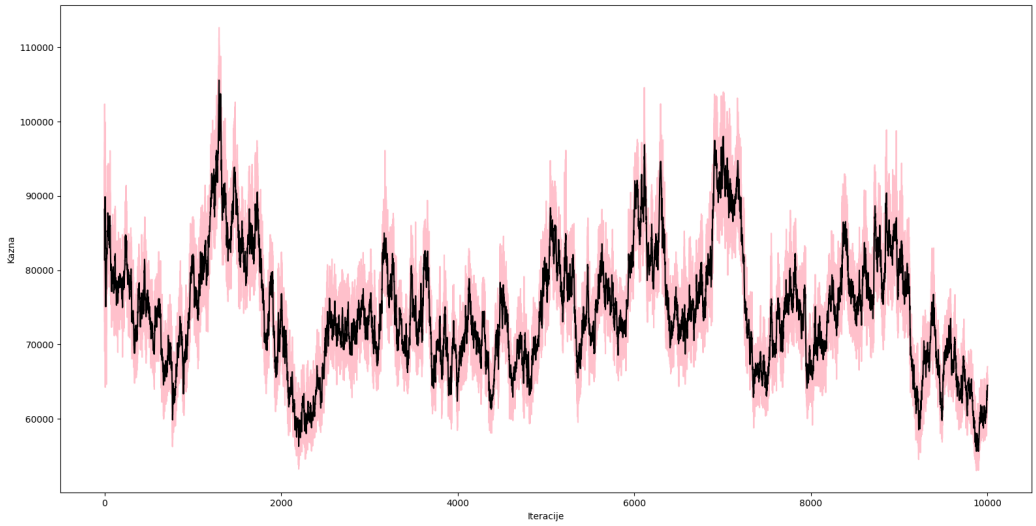
Bez elitizma odabir roditelja s verzijom proporcionalnog odabira sa svim modifikacijama kako bi odabir bio što bolji, populacijom veličine 10 što pospješuje dodatno

uspjeh proporcionalne selekcije i 10000 iteracija i dalje ne pronalazi raspored bez preklapanja. Broj preklapanja varira oko 10 preklapanja te rasporedi zapinju u lokalnom optimumu s kaznom koja iznosi oko 20000.

Ako se umjesto unaprijeđene verzije proporcionalne selekcije koristi osnovna verzija, bez korekcije velikih i negativnih brojeva te također bez elitizma, rezultati su još gori, prosječan broj preklapanja nakon 10000 iteracija iznosi oko 250 i kazna poprima vrijednosti oko 60000.



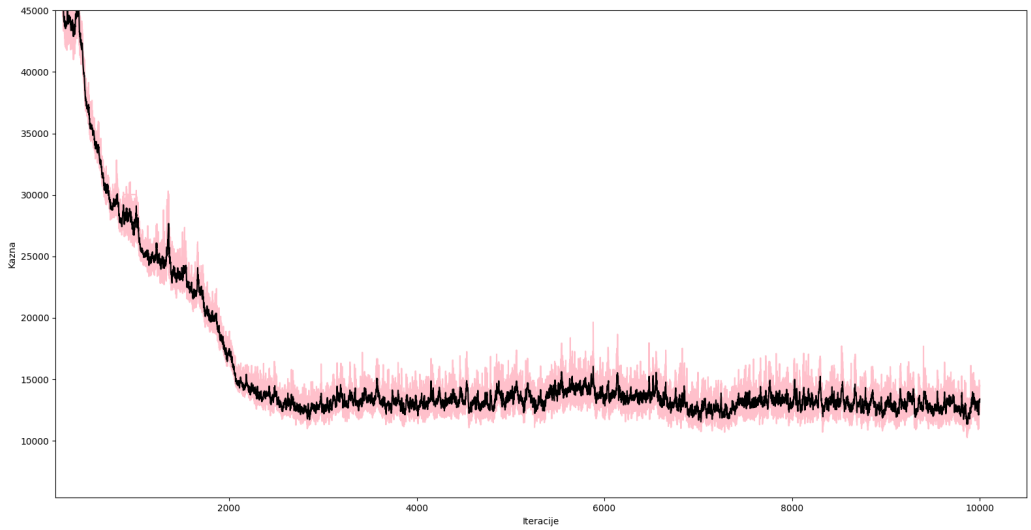
Slika 5.4: Poboljšana verzija proprocionalne selekcije s isključenim elitizmom.



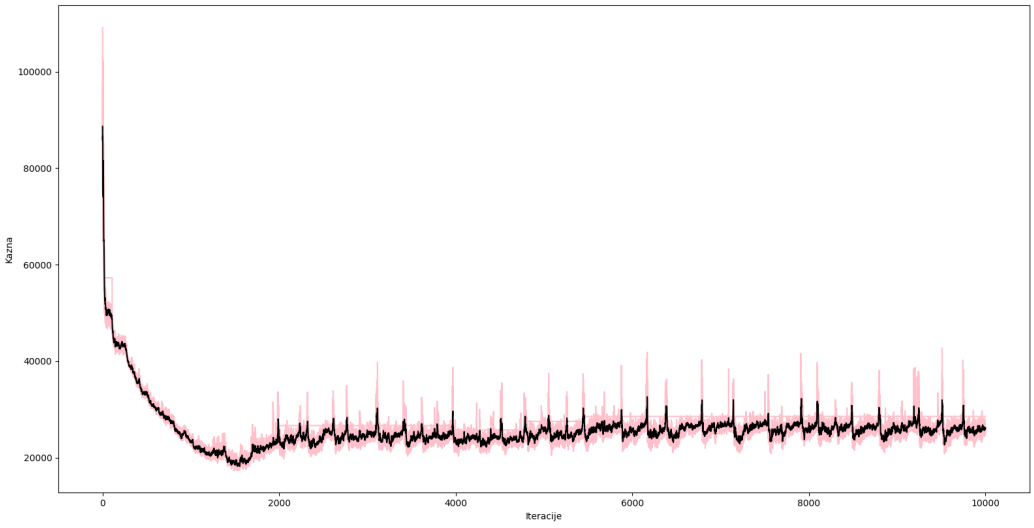
Slika 5.5: Osnovna verzija proprocionalne selekcije s isključenim elitizmom.

Kod poboljšane verzije odabira proporcijom uključivanjem elitizma vrijednosti se smanjuju nakon 10000 iteracija s 20000 na vrijednosti oko 12000, dok broj kolizija ostaje relativno jednak. Osnovna verzija postiže drastično bolje vrijednosti, nakon 1500 iteracija se broj kolizija smanjuje na oko 50, ali tu algoritam zapne u lokalnom optimumu i dalje zastaje na kazni od oko 25000.

Usporedba rada turnirske selekcije s elitizmom i bez njega već je prethodno prikazana na slikama 4.5 i 4.6.



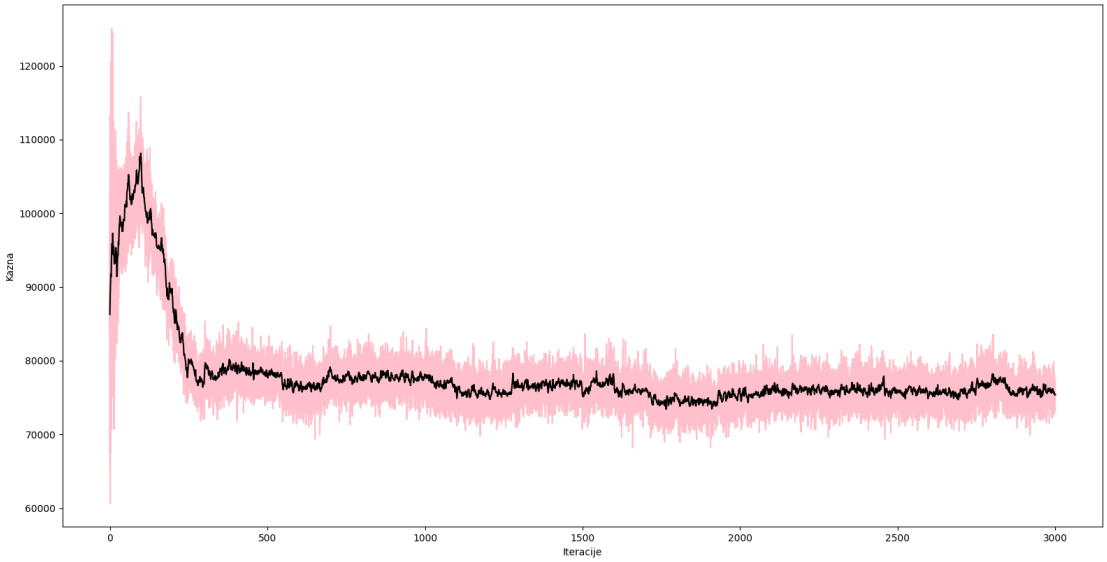
Slika 5.6: Poboljšana verzija proprocionalne selekcije s uključenim elitizmom.



Slika 5.7: Osnovna verzija proprocionalne selekcije s uključenim elitizmom.

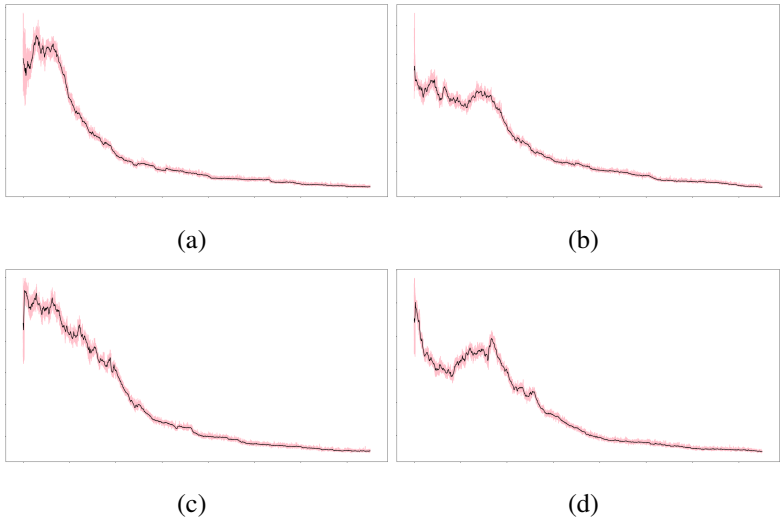
5.3. Mutacija

Rezultati mutacije odabirom nasmičnog termina vježbi vidljivi su na slici 4.5, dok promjena termina za fiksnu vrijednost jednakoj 3 zapinje na oko 30 kolizija i na kazni oko 75000.



Slika 5.8: Mutacija promjenom termina za fiksni broj mjesta.

5.4. Križanje



Slika 5.9: (a) nasumično (b) naizmjenično s K=1 (c) naizmjenično s K=3 (d) pola rasporeda

Sve vrste križanja rasporeda testirane su s 1500 iteracija, razlike su vidljive u početnim iteracijama algoritma, kada svaki operator postiže drugačije rezultate. Nakon već 1000 iteracija, svi operatori djeluju otprilike jednako bez primjetnih razlika. Zbog toga je odabran operator nasumičnog križanja jer može potencijalno pružati jače svojstvo otpora konvergenciji od ostalih zbog komponente nasumičnosti koju sadrži.

5.5. Vremenska komponenta i veličina populacije

Elitizam i operatori križanja ne mijenjaju značajno vrijeme izvođenja. Izračun kazne je konstantan, stoga neovisno o konfiguraciji ima jednak utjecaj. Operator odabira roditelja, veličina populacije i stopa mutacija utječu na vrijeme potrebno za izvršavanje jedne iteracije i pronalazak kvalitetnog rasporeda.

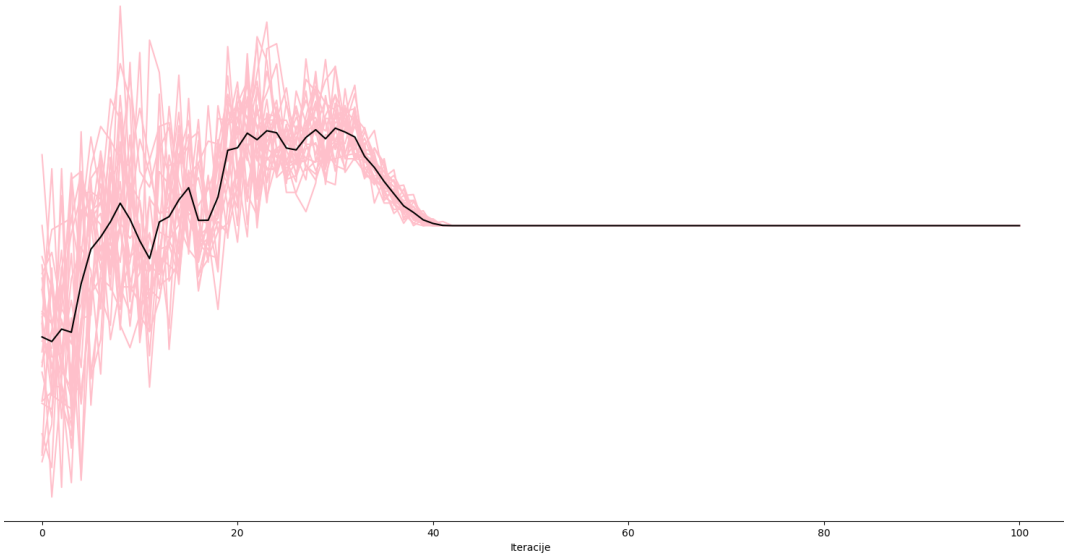
Testiranje vremena izvođenja testira se na 300 iteracija sa 100 rasporeda u populaciji.

Tablica 5.4: Vrijeme izvođenja ovisno o operatoru odabira roditelja.

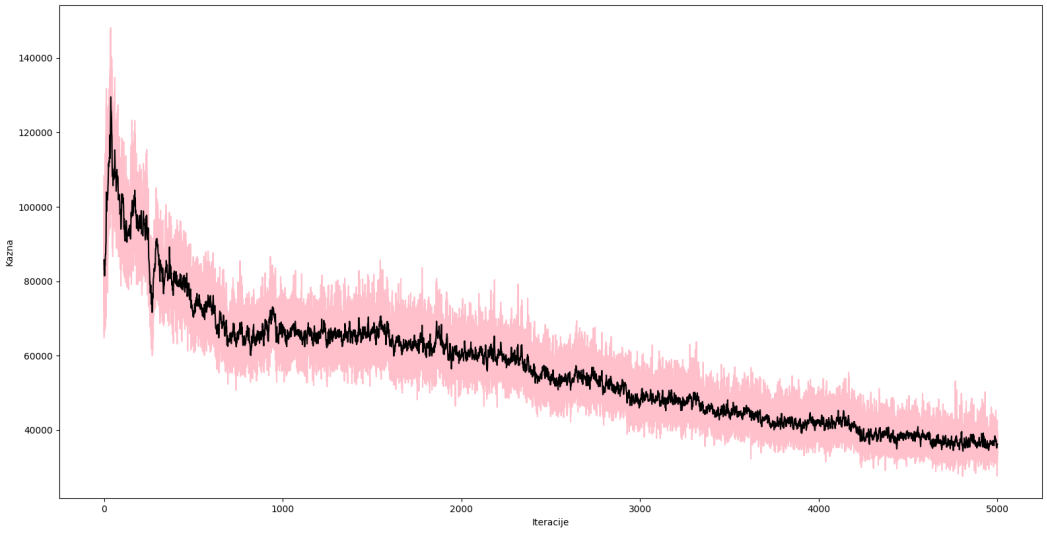
proporcionalna selekcija	proporcionalna selekcija (poboljšana)	turirska selekcija (K = 3)
13.87	14.53	15.05

Vrijeme izvođenja poboljšane verzije proporcionalne selekcije je veće kao što je očekivano, ali operator turnirske selekcije, iako je sporiji od ostalih, pruža puno bolje rezultate zbog čega se i dalje smatra najboljim operatorom za ovaj zadatak.

Nadalje, s većom stopom mutacije, vrijeme izvođenja algoritma će se povečati jer je potrebno generirati i primijeniti više vrijednosti, no to nije fokus, već brzina konvergencije ovisno o stopi mutacije. Sa stopom mutacije nula, nakon svega nekoliko iteracija učenje prestaje, sa stopom iteracije oko 1% učenje uspori, ali ne prestaje, a sa stopom oko 0.2% učenje je najbrže.



Slika 5.10: Mutacija sa stopom mutacije 0.



Slika 5.11: Mutacija sa stopom mutacije 0.1.

S visokom stopom mutacije od 1% u prosjeku se mijenja termin za 8.3 studenta, zbog toga raspored jako sporo konvergira i generirana djeca su jako različita što je vidljivo na grafu.

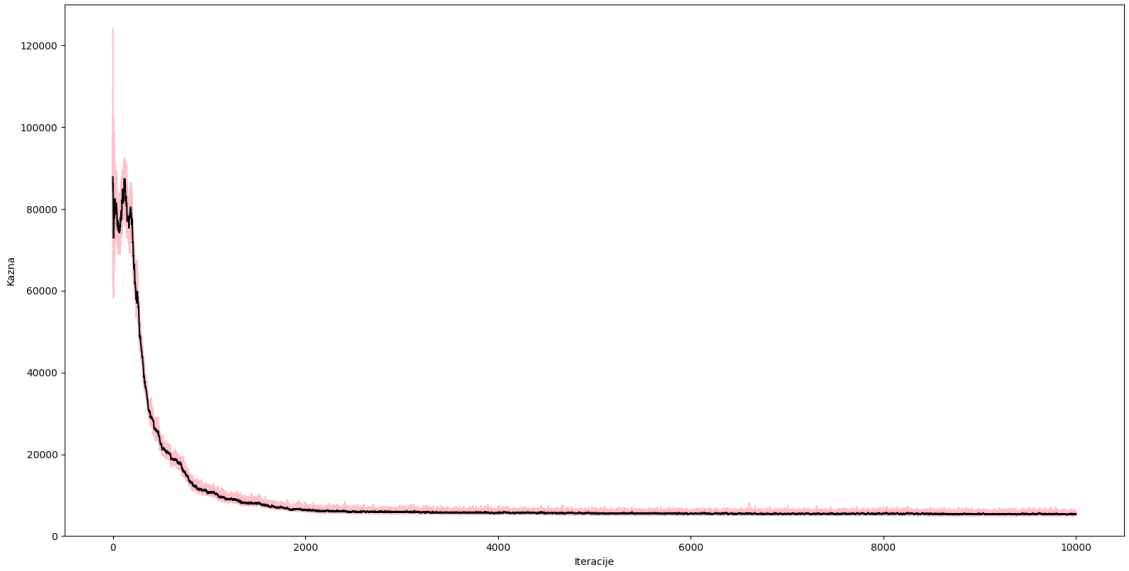
Posljednje je preostalo usporediti brzinu pronalaženja rasporeda određene kvalitete.

Uspoređuje se pronalazak rasporeda s kaznom 20000, 1000 i 7500 s populacijom 15, 25, 50, 150 i 250, sva vremena su izražena u sekundama.

Tablica 5.5: Vrijeme izvođenja ovisno o operatoru odabira roditelja.

populacija/kazna	20000	10000	7500
15	5.81	13.74	19.22
25	8.14	10.01	15.6794
50	7.69	15.52	24.39
150	13.76	25.09	40.63

Iz prikazane tablice vidljivo je kako je upravo 25 najbolja veličina populacije, iako na početku su opcije poput 50 i 15 brže, kasnije u algoritmu je potreban velik broj iteracija kako bi se prošao što veći broj iteracija i opet što veća raznolikost. Za kraj pružam prikaz referentnih pretpostavki, ali uz povećanje broja iteracija na 10000.



Slika 5.12: Referentne postavke algoritma.

Nakon 10000 iteracija algoritam je pronašao raspored bez preklapanja i s potrebnim kapacitetima za svaki termin s konačnom kaznom od 5012.

6. Zaključak

U sklopu ovog rada predstavljen je problem izrade raspodjele studenata u nastavne i ispitne grupe te niz prepreka i ograničenja koja postoje prilikom izrade rasporeda. Problem se svodi na pametnu pretragu velikog prostora stanja.

Jedna od metoda pretrage rješenja je primjena mateheursitke, točnije generacijskog genetskog algoritama gdje se modeliranjem biološkog procesa evolucije simulira proces opstanka najjačih i izumiranja najslabijih jedinki.

Opisan je rad algoritma kao i svih njegovih dijelova. Predložen je prikaz rasporeda u memoriji kao niz studenata s dodijeljenim terminom, takva reprezentacija pruža jednostavan i efikasan zapis. Umjesto izračuna dobrote, predložena je uporaba kazne.

Ispostavilo se kako elitizam pozitivno utječe na dobivene rezultate. Za rasporede je predloženo da se vrednuju prvotno po broju preklapanja, a tek on po kazni. Predloženo je da se kazni pribrajaju broj preklapanja, udaljenost dodijeljenog termina od sljedećeg najbližeg i razlika između dostupnog i iskorištenog kapaciteta učionice.

Kao operator odabira roditelja najbolje se pokazala turnirska selekcija, a za operator križanja nije pronađena bitna razlika, međutim odabran je operator nasumičnog križanja. Za mutaciju je pokazano da je najbolja relativno mala stopa mutacije takva da u prosjeku mutira selekcija za jedan do dva studenta.

Posljednje, populacija od dvadesetak rasporeda pruža najbolja svojstva raznolikosti uz veliku brzinu prolaska kroz velik broj iteracija.

Kao krajnji rezultat ovog rada definirana je konfiguracija algoritma koja u konačnici generira iskoristiv raspored s terminima laboratorijskih vježbi bez preklapanja i s takvim svojstvima koja odgovaraju stunentima.

LITERATURA

H. Achini Kumari. Genetic algorithm for university course timetabling problem. Magistarski rad, University of Mississippi, 2017.

R. Ganguli i S. Roy. A study on course timetable scheduling using graph coloring approach. *International Journal of Computational and Applied Mathematics*, 12(2): 469–485, 2017.

M. Čupić. *Raspoređivanje nastavnih aktinosti evolucijskim računanjem*. Doktorska disertacija, Fakultet elektrotehnike i računarstva, 2011.

M. Čupić. Evolucijsko računarstvo, 2019. URL `http://java.zemris.fer.hr/nastava/ui/evo/evo-20190604.pdf`.

.

Izrada rasporeda laboratorijskih vježbi uporabom genetskog algoritma

Sažetak

Genetski algoritmi su primjenjivi na širok spektar problema, a izrada rasporeda sati jedan je od konkretnih i primjenjivih problema. U sklopu ovog rada opisan je rad generacijskog genetsko algoritma s mogućim rješenjima raznih operatora potrebnih za rad algoritma. Uspoređeni su utjecaji različitih konfiguracija i rezultata dobivenih iz provedenih eksperimenata kako bi se ispitala kvaliteta generiranih rasporeda. Usporedbom rezultata odabrana je konfiguracija koja je predložena kao konačno rješenje problema.

Ključne riječi: Operatori, generacijski, eksperimenti, kvaliteta, konfiguracija.

Generating a schedule of laboratory exercises using a genetic algorithm

Abstract

Genetic algorithms are applicable to a wide range of problems, and class scheduling is one of concrete and applicable problems. As part of this paper, the operation of the generational genetic algorithm is described with the possible solutions of various operators required for the operation of the algorithm. The effects of different configurations and the results obtained from the conducted experiments were compared in order to examine the quality of the generated schedules. By comparing the results, the configuration that was proposed as the final solution to the problem was selected.

Keywords: Operators, generational, experiments, quality, configuration.