SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 000

Izradra 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 se svojoj obitelji na svoj pružanoj podršci.

SADRŽAJ

POPIS SLIKA

1. Uvod

Uvod rada. Nakon uvoda dolaze poglavlja u kojima se obrađuje tema.

2. Problem raspoređivanja studenta u grupe

U obrazovnim institucijama, dva najčešća problema slaganja rasporeda su dodjela grupa studentima te 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 te su zadovoljena 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 rješeni u polinomnom vremenu koristeći detrministički algoritam. To vrijedi zbog kombinatorne eksplozije do koje dolazi zbog velikog broja, ograničenja, zahtjeva i velikog broja studenata, predemta, 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 se odnose na probleme koji su fizički nemogući. Poput toga da je ili profesor ili student na dva ili više mjesta istovremeno. (Achini Kumari, 2017). Student mora imati dodjeljen samo jedan termin vježbi, taj termin se ne smije preklapati s drugim terminima. Profesori koji održavaju vježbe ne smiju istovremeno održavati dva ili više termina ali mogu imati dodjeljeno više termina u istom danu. U jednoj učionici se istovremeno može nalaziti samo jedna grupa studenata. Dodatne probleme mogu zadavati kapaciteti učionica, prilagođenost studentima s posebnim potrebama i slično.

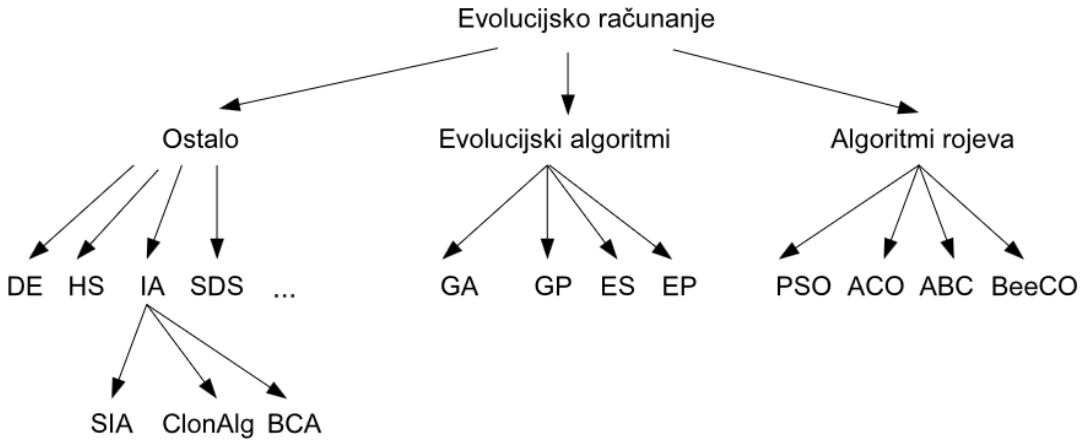
[1] Ne postoji algoritam u polinomijalnom vremenu.

2.2. Blaga ograničenja

Blaga ograničenja su ona koja se smatraju poželjnima, 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 studenta se smanji 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 su inspirirani 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 identičcnih jedinki već postoje varijacije te

[1] Skup algoritamskih koncepta primjenjivih na širok spektar problema

5. najveći dio varijacija prenos 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 sljedeće 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*) te *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 taokđer biti elitistički[3]. Za razliku od ne elitističkih inačica, kroz evoluciju graf jedinki će biti 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 te je isti 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 sada ne spadaju u opseg ovog rada.

4.1. Reprezentacija rasporeda u obliku jedinke

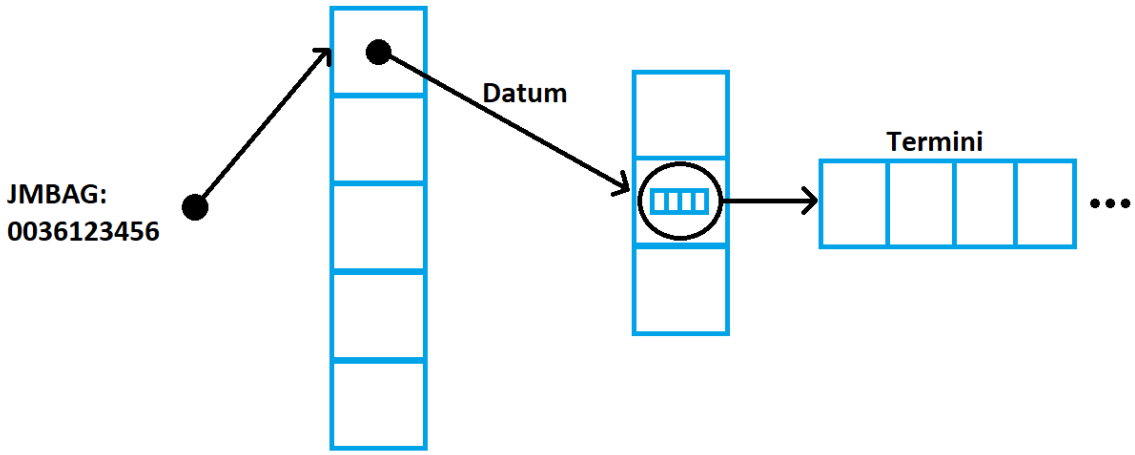
Svaki generirani raspored (također istoznačno: jedinka) potrebno je ispravno prikazati u memoriji. Ovisno o zapisu koji se koristi problem može znatno olakšati ili otežati. Osim zapisa same jedinke program prima niz datoteka potrebnih za rad algoritma: popis studenata, popis dostupnih termina laboratorijskih 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 zapiso, 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 u tablici zauzetosti studenata se nalaze studenti kojima nije potrebno dodijeliti termin. Stoga se za prikaz rasporeda

koristi tablica raspršenog adresiranja[1] koja kao ključ prima JMBAG[2] studenta, a kao vrijednost sadrži istu prima koja sada kao ključ prima 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 se ne provjeravaju.



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 se pretpostavlja da student u tom tjednu nema ostalih nastavnih obveza. Studenti se prikazuju kao niz JMBAG-a kojemu se kasnije pristupa preko indeksa pojedinog elemenata.

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

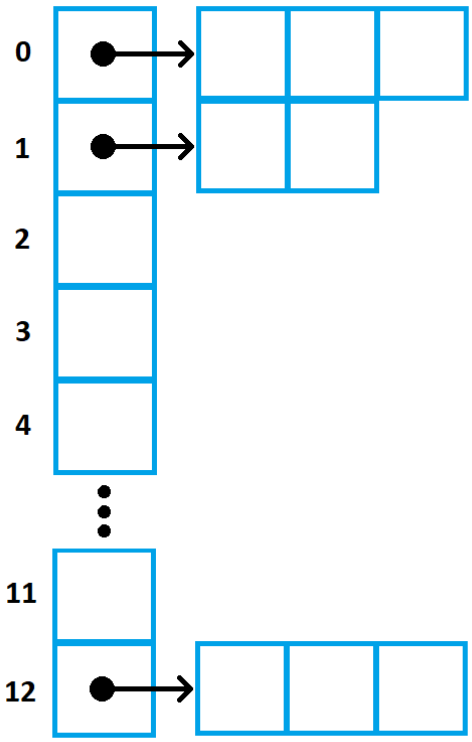
4.1.1. Niz termina s listom studenata

Prvi način zapisa je niza 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 je paziti da je svaki student dodijeljen samo jednom terminu, teže je implementirati operatore mutacije i operator križanja jer jedinka ima dvije osi od kojih je jedna varijabila, također je zbog toga otežan izračun

[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.

preklapanja. Dodatno, ovaj prikaz je i memorijski i vremenski zahtjevan zbog dodatnog korištenja struktura podataka.

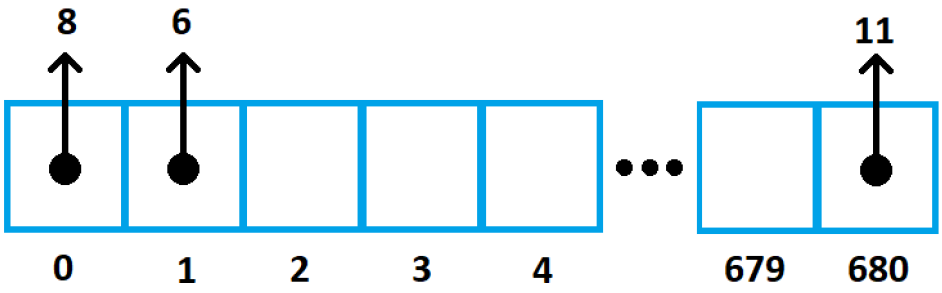
4.1.2. Niz studenta s dodijeljenim terminom

Druga 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 jednostavno pronaći za svakog studenta 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 je potrebno za svaki termin proći kroz cijeli niz te 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

onda više nije problem.



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

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

4.2. Ocijena rasporeda

Ocjena rasporeda je rezultat funkcije dobrote (engl. *fitness*). Točnije to je numerička oznaka koja se koristi za ocjenu kvaliteta rasporeda. Ukupna ocjena nekog rasporeda jednaka je sumi funkcija dobrote za svakog studenata 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 koliko je neki raspored loš. Ponekad je za neki problem lakše definirati ukupnu kaznu, nego ukupnu dobrotu. U praksi se pokušava ili minimizirati kazna ili maksimizirati dobrota. Kazna isto kao i dobrota je zapravo heuristika koja aproksimira koliko je daleko odnosno blizu dobiveno rješenje od cilja tj. najboljeg rješenja.

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

Prilikom izračuna ocijene rasporeda, sljedeća svojstva su uzeta 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

Ako bi raspored koristio dobrotu za svoju ocjenu, neke od heuristike koje bi mogli 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, 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 te neće preferirati da studenti imaju slijedna predavanja. Opet može se dogoditi da algoritam sve studente stavi u par 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 je lakše implementirati funkciju kazne rasporeda. Za kaznu isto kao i za dobrotu vrijedi da ako se psotavi prevelika kazna za bilo koje svojstvno, 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 heuristika koje su dostupne su: broj preklapanja, broj rupa, vrijeme preklapanja, koliko je studenata premalo ili previše u nekom terminu i vremenski period najmanje rupe nastale nakon dodavanja termina.

4.2.1.3. Kazna s kombinacijom dobrote

Kao najbolja opcija se ipak 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. Kada bi nagrada bila velika, onda bi efektivno algoritam 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,
- lošije 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.

Nakon što je dostupna informacija o broju studenata za svaki termin, dodijeljuje 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 pomožena 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 za obrat.

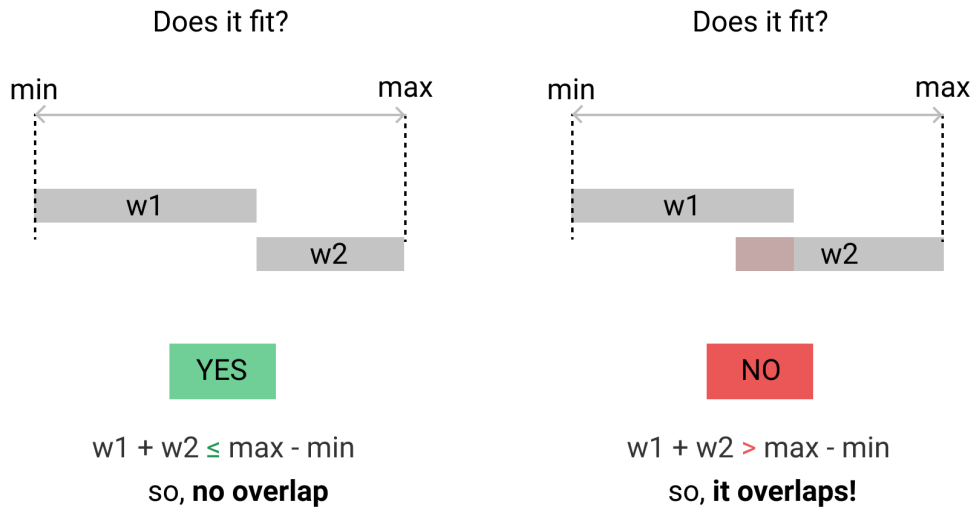
4.2.3. Izračun preklapanja

Student može imati preklapanja u rasporedu prije nego li mu je dodijeljen termin laboratorijskih vježbi. Kod takvih situacija se računaju 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 na kojem se nalaze termini koji se trenutno uspoređuju te ako je taj interval manji od zbroja duljina zasebnih intervala, onda postoji preklapanje.



Slika 4.4: Provjera preklapanja dva termina.

Najvažnije svojstveno 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 te onda krenulo u dodatno poboljšanje ostalih svojstva rasporeda.

4.2.4. Slijedni termini

Kao što je spomenuto u odjeljku 4.2.1.3, zbog nekih poželjnih svojstva rasporedu se može dodijeliti nagrada. Nagrada je implementirano 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 sljednosti može se koristiti formula 4.4 ali tako da se znak nejednakosti promjeni 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. Pa se zato za provjeru preklapanje dva 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 li 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 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 dodjeljen u danu bez nastavnih obveza

Temeljeno na prvoj pretpostavci izračuna ocjene rasporeda, studenti preferiraju termine laboratorijskih vježbi u istom dana kada imaju neku drug nastavnu aktivnost, stoga ako student nema drugih nastavnih aktivnosti u danu održavanja laboratorijskih vježbi, dodjeljuje se kazna relativno velikog iznosa.

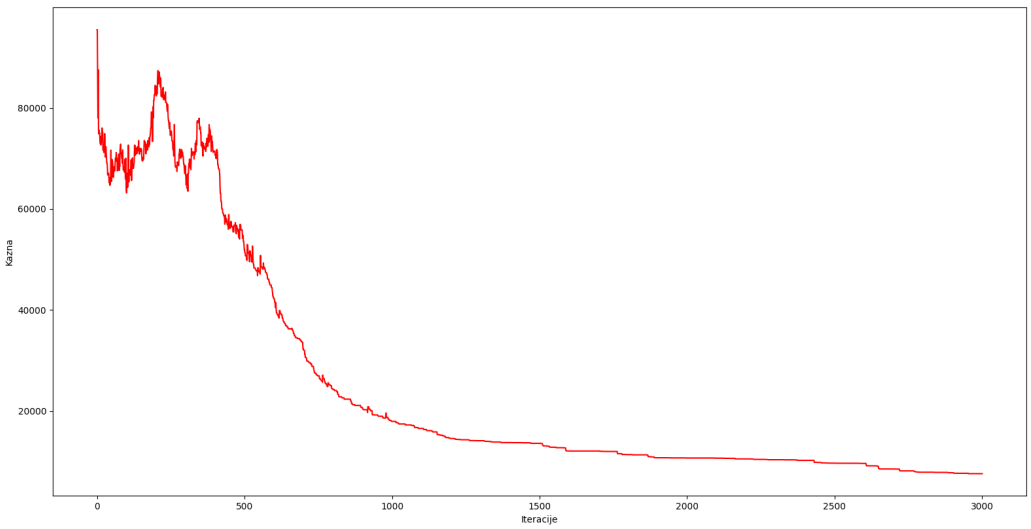
Provjera ima li student neke druge nastavne aktivnosti taj da ili ne održava se na samom početku prije provjera preklapanja, sljednosti termina i pauze s najmanjim vremenskim razmakom. Ako je ta provjera istina onda se ostala svojstva ne provjeravaju.

4.3. Elitizam

Elitizam (engl. *elitism*) je svojstvno algoritma da uvijek sadržava najbolju jedinku.

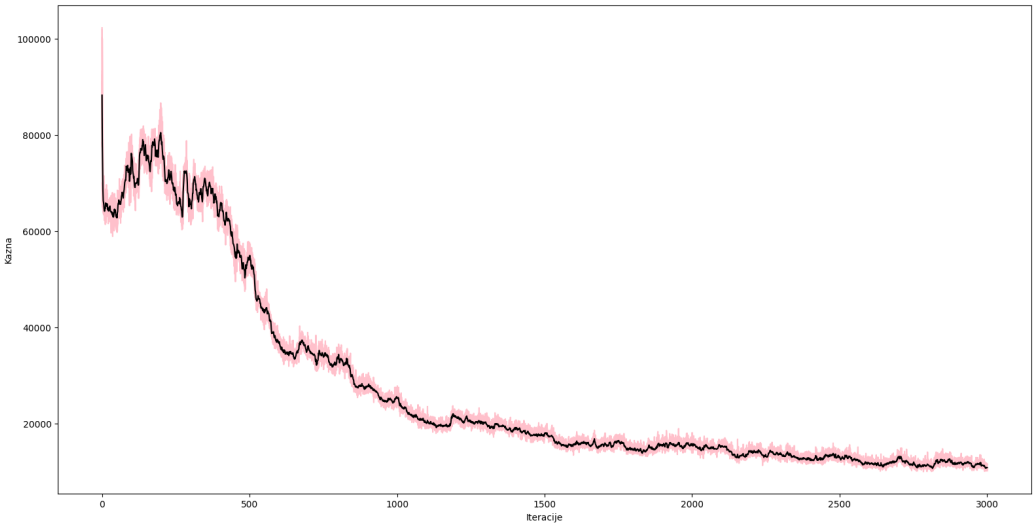
Odabir elitnog rasporeda vrši se tako da se prolazi kroz kazne svih jedinki i gleda se ima li raspored manji broj preklapanja od prethodnoga potom. U slučaju da ima, raspored na kojeg se naišlo se bira kao elitni, a u slučaju da je broj preklapanja jedinak onda se odabire onaj s manjom kaznom.

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-te iteracije kada broj preklapanja postane 0.



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 kazne najbolje jedinke koju je algoritam imao u nekom prethodnom koraku (Čupić, 2019).



Slika 4.6: Generirani rasporedi s isključenim elitizmom.

Prilikom izrade rasporeda očuvanje elitnog rasporeda je pokazalo dobra svojstva 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. Time algoritam postaje striktno konvergentan.

4.4. Odabir roditelja

Kod odabira roditelja važno je da oba roditelja budu različita kako bi se održala raznolikost raposreda. 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 suzuje prostor pretrage riješenja, jer temeljem različitih algoritama roditelja koji imaju bolju ocjenu će vrjerojatnije 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

Proporcionalna selekcija se temelji na međusobnoj proporciji ocjena svih rasproeda iz populacije. Ideja je svakom rasporedu dodijelitio dio intervala između 0 i 1 uključujući i nasumičnog odabirom broja u tom intervalu odabire se roditelj u ćij dio intervala taj broj upada. Roditelji sa manjom kaznom će imati veći dio intervala a roditelji s većom kazon manji.

Osnovna izvedba algoritma izgleda ovako:

Algorithm 2 Proporcionalna selekcija

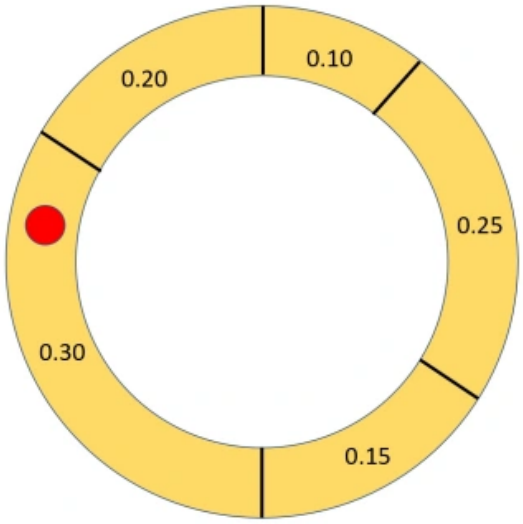
```
raspon := min(kazna) + max(kazna)

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 implemenetiran tako da svaki element podjelimo sa sumom elemenata, te će tako element s najvećom vrjednošću imati najveću šansu odabira, no kako je sada bolji onaj raspored koji ima manju kaznu potrebno je ipak podatke transformirati.

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

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

Ovom transformacijiom zamjenjuje se veličina svih elemenata tako da najmanji elementi postaju naveći i obrnuto. Sada je samo potrebno svaki element podjeliti sumom svih elemenata kako bi se svi elementi razdjelili po intervalu od 0 do 1.



Slika 4.7: Proporcionalna selekcija roditelja.

Veliki brojevi predstavljaju problem problem, kod populacije s 5 rasporeda od kojih nakon transformacije svi imaju redom vrijednosti 10001, 10002, 10003, 10004 i 10005 razlike vjerojatnost svakog rasporeda će biti zanimarive i svi rasporedi će efektivno imati vjerojatnost od 20%. Kako bi se taj efekt ublažio, moguće je sve kazne iz populacije umanjiti vrijednoću za jedan manjom od najmanje vrijednosti. Tako sada redom vrijednosti iznose 1, 2, 3, 4, 5.

Sljedeći problem predstavljaju negativne vrijdnosti, zbog nagrada koje rasporedi mogu poprimiti, kazna rasproeda 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 iteraciji nalazi veliki broj rasporeda, svaki raspored će imati podjednaku šansu da bude odabran i tako se efektivno dobije nasumični odabir.

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

podjenaku vjerojatnost odabira.

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

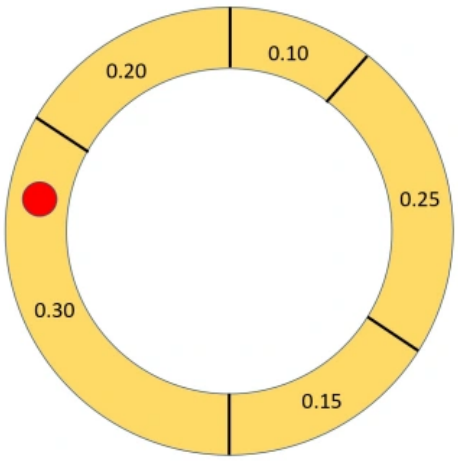
4.4.2. K-turnirska selekcija

Turniska selekcija kaulaz u algoritam prima parametar *k* koji označava broj jedinki koji ulaze u turnir. Vrlo jednostavnim postupkom, iz populacije se nasumično odabere *k* raspored 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 pomocne populacije, potrebno je provjeravati da novo odabrani raspored se već ne nalazi u populaciji.



Slika 4.8: Ovo ce biti turnirska selekcija.

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 jednika.

Za broj odabranih raspored postavlja se manja vrijednost poput dvij ili tri kako bi se omogućila barem minimalno svojstvo kompetencije.

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

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

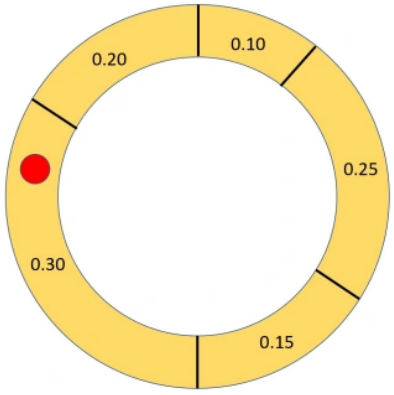
4.5. Križanje roditelja

Operator križanja roditelja kao i uperator odabira roditelja suzuje prostor pretrage. Križanjem dvaju roditelja nastaje jedno ili dva nova riješenja ovisno o konkretnim implementacijama. Odabrano je da operator križanja kao rezultat vraća jedno dijete. To dijete se ne odabire na osnovu svoje kazne već nasumično.

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

4.5.1. Nasumično križanje

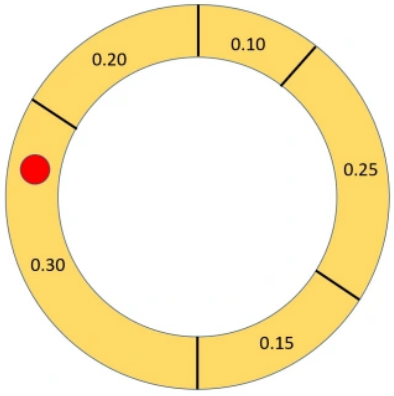
Nasmučno križanje jednostavno redom iterira kroz sve studente iz rasporeda i sa šansom od 50% odabire zadaje li se studentu termin prvog ili drugog roditelja. Nasumično križanje se koristi za operator križanja roditelja.



Slika 4.9: Ovo ce biti turnirska selekcija.

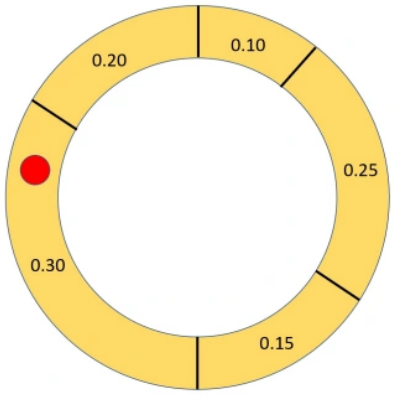
4.5.2. Alternirajuće k-križanje

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



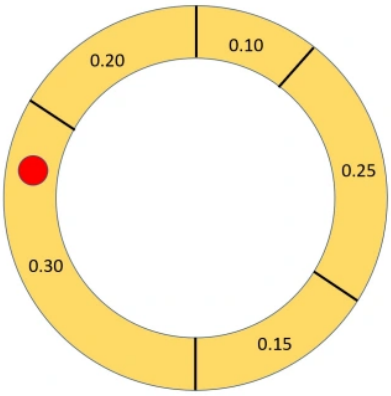
Slika 4.10: Ovo ce biti turnirska selekcija.

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



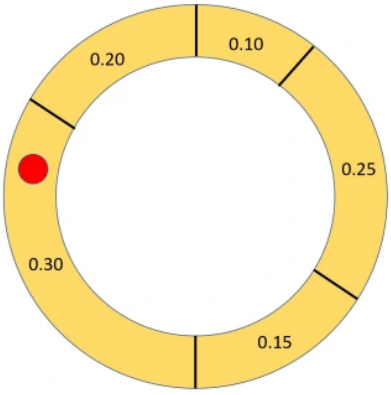
Slika 4.11: Ovo ce biti turnirska selekcija.

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



Slika 4.12: Ovo ce biti turnirska selekcija.

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.



Slika 4.13: Ovo ce biti turnirska selekcija.

4.6. Mutacija

Osim operatora koji suzuju područje pertrage, mutacija ga proširuje. Bez operatora mutacije, vrlo brzo bi sva nova rješenja počela stagnirati, tj. ne bi dobili više novih rješenja. Operatoruje je dodjeljena stopa muta, postotak studenata za koji će se promjeniti dodjeljeni termin.

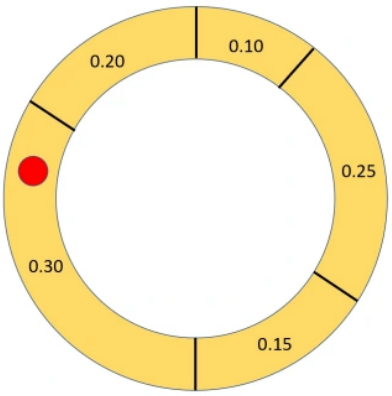
Optimalna vrijednost mutacie iznosi oko 0.002 ili 0.2%. Raspored se sastoji od termina za 831 studenta, prema toma nakon svake mutacije će se u prosjeku promjeniti grupa za 1.66 studenta što pruža dovoljno jako svojstvo širenja. Uz veće vrijednosti

konvergencija je sporija a rješenja nisu bolja.

Kod ovog problema, nije potrebno pronaći najboljelje moguće rješenje već dovoljno dobro. Zato i s jako malim stopama promjene kad algoritam ima veću vjerojatnost zapesti u lokalnom optimumu to ne predstavlja problem.

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 sa termina 8 na termin 11.



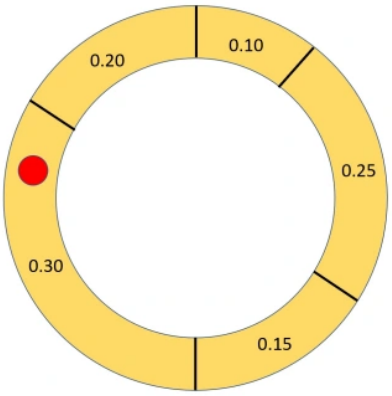
Slika 4.14: Ovo ce biti turnirska selekcija.

Ovako definiran operator kasnije u ciklusu pretrage kad se raspored stabilizira i većina termina imaju popunjen kapacitet od 15 ili 16 mjesta može napraviti više štete nego koristi. Ako se napravi promjena termina za 2 studenta, pogoto ako su oni u istoj grupi, novi raspored će vrlo vjerojatno imati veću kaznu.

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.

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



Slika 4.15: Ovo ce biti turnirska selekcija.

4.7. Veličina populacije

Veličina populacije je broj rasporeda koji se u svakom trenu čuva u memoriji. S manjom veličinom populacije povečava se broj iteracija po sekuni ali je zato manja raznolikost rješenja. Pak, s većom veličinom populacije smanjit će se broj iteracija po sekundi ali će se povečati raznolikos 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 večom populacijom broj iteracija po sekundi drastično smanjiti. Potrebno je pronači dobar balans kvalitete i brzine. Kroz testiranje vrijednosti oko 25 su se pokazale jako dobre.

5. Usporedba rezultata

Kroz rad je već izrečena odabrana konnfiguracija algorita. U ovom poglavlju će biti prikazana dodatne usporedbe rezultata zbog kojih su baš ta svojstva odabrana.

U ulaznim podacima, dostupno je više termina vježbi nego je potrebno, kod treniranja veće broja termina vidljivo je kako pno vište termina ima manji broj dodjeljenih termina, a nema gotov niti jedan termin koji ima više nego max. Kada se kazna za manji broj mjesta još više smanjima, ovaj efekt postaje akstremniji i sve je manje rasporeda koji imaju termine s popunjeni svih 16 mjesta.

Kod nagrada s većim tj. manjim vrijednostima počinju se izrazito preferirati rasporedi koji zadovoljavalju ta svojstva, čak iako se raspored smatra "lošim". Izrazito se usporava prolazak rasporeda s ne prekršenim ograničenjima.

Usporedba elitizma, dati primjer proporcionalnog rasporeda sa i bez elitizma.

pokazati proporciju sa velikim i malim brojem jedinki., pokazati opcenito k turnir vs proporcija.

Proporcija jako sporo konvergira k 0 zato što nema u algoirtmu kolizije

pokazati neke ekstremne situacije odabira roditelja. pokazati da i onda, rezultati su otprilike jednaki.

Pokazati primjer kada je mutacija jednaka 0

Tablice vremena dolaska do kazne: 50k, 20k, 10k, ako je populacija 10, 15, 25, 50, 150, 250

6. Zaključak

Zaključak.

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`.

.

Izradra rasporeda laboratorijskih vježbi uporabom genetskog algoritma

Sažetak

Ovo je sažetak mojeg rada.

Ključne riječi: Ključne riječi, odvojene zarezima.

Creating a schedule of laboratory exercises using a genetic algorithm

Abstract

Ovo je sažetak mojeg rada.

Keywords: Keywords, spaced apart by commas.