SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 1227

# Rješavanje problema izrade rasporeda nadoknada primjenom genetskog algoritma

Siniša Pribil

Zagreb, lipanj 2010.

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

*Zahvaljujem asistentu mr.sc. Marku Čupiću na ustupljenim materijalima, strpljenju, savjetima i velikoj pomoći koju mi je pružio tijekom izrade ovoga rada.*

*Zahvaljujem kolegama Mislavu Bobesiću, Filipu Boltužiću, Ivanu Ferdelji, Tinu Franoviću, Denisu Ćutiću, Domagoju Kusaliću i Vladimiru Uzunu na suradnji prilikom razvoja sustava za izradu rasporeda nadoknada.*

*Veliko hvala Matei na lektoriranju rada, korisnim savjetima i beskrajnom razumijevanju za moje probleme tijekom rada na ovom projektu.*

*Naposlijetku, najveće hvala mojim roditeljima na razumijevanju i podršci tijekom cijelog dosadašnjeg školovanja.*

# SADRŽAJ

# 1. Uvod

Svakodnevno se u životu susrećemo s različitim oblicima problema čija su nam rješenja potrebna, ali do njih nije uvijek lako doći. Razvitkom računalne tehnologije mnogi poslovi su nam olakšani zbog činjenice da računala mogu u kratkom vremenu obaviti veliku količinu izračuna. Tako su se mnogi dugi i dosadni proračuni, u zadnjih nekoliko desetaka godina, pretvorili u svega nekoliko klikova mišem. Ipak, postoje problemi koje, zbog njihove složenosti, niti današnja najbrža računala nisu u stanju riješiti u razumnom vremenskom roku. Takvi problemi nazivaju se NP-teški[1] problemi.

Jedan od takvih problema je problem izrade rasporeda, a on se ubraja u razred najtežih problema koje definira računarska znanost. Bit problema je u njegovoj složenosti. Izrada rasporeda i slični problemi imaju toliko mogućih rješenja da ih čak niti računalo nije u mogućnosti sve pretražiti i pronaći najbolje.

Sredinom prošloga stoljeća počelo se intenzivnije razvijati evolucijsko računarstvo koje je inspiraciju za novi način rješavanja problema pronalazilo u evolucijskim procesima. Ubrzo su evolucijski i slični algoritmi postali rješenje za do tada nerješive probleme na računalu. Pokazalo se kako za rješavanje nekih problema nije dovoljna samo „sirova snaga“ računala, nego i odgovarajuća strategija. Evolucijski algoritmi ne pretražuju cijeli prostor stanja za određeni problem, pa tako niti ne mogu jamčiti da će uvijek pronaći najbolje rješenje. Unatoč tome, najčešće je pronađeno rješenje sasvim prihvatljivo u okviru problema koje rješavaju i što je najvažnije, nalaze ga unutar vremena koje smo voljni čekati. Algoritmi koji daju zadovoljavajuće dobra rješenja, ali ne uvijek i optimalna, zovu se približni algoritmi ili heuristike (Čupić (2009a)).

U ovom radu je opisan problem i rješenje izrade rasporeda nadoknada (laboratorijskih vježbi, ispita i predavanja) na Fakultetu elektrotehnike i računarstva u Zagrebu korištenjem genetskog algoritma. Navedene su i opisane vrste evolucijskih algoritama i njihove komponente. Opisano je i programsko rješenje problema a navedeni su i analizirani rezultati mjerenja ovisnosti učinka algoritma o ulaznim parametrima.

[1] ne-polinomijalan, nije rješiv u polinomijalnom vremenu

# 2. Problem

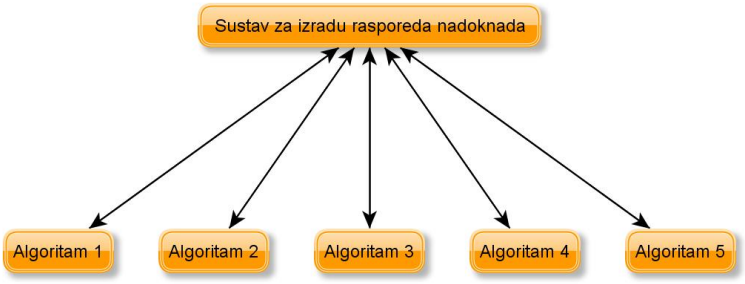
Fakultetski raspored nije u potpunosti statičan sustav. Profesori ponekad ne mogu održati zakazano predavanje prema rasporedu, a studenti iz različitih razloga ne mogu pristupiti laboratorijskim vježbama ili ispitima. Zbog tog razloga se često moraju organizirati nadoknade spomenutih događaja. U rasporedu koji za svaki dan organizira obaveze za nekoliko tisuća studenata, određivanje termina nadoknade koji će odgovarati svim studentima je iznimno težak zadatak.

U ovom poglavlju su detaljno opisani programski zadatak i zahtjevi koje ostvarenje mora zadovoljavati. Pobliže su opisani građa sustava za izradu rasporeda nadoknada i način na koji su potencijalna rješenja vrednovana.

## 2.1. Formalna definicija

Algoritam za pronalazak termina nadoknada pomoću genetskog algoritma treba biti dio većeg sustava za rješavanje problema nadoknada, prikazanog na slici 2.1. Sustav će kombinirati rješenja nekoliko različitih algoritama kako bi dobio najbolje moguće rješenje. Kombiniranje algoritama podrazumijeva:

- naizmjenično izvršavanje jednog od algoritama u određenom broju koraka
- dohvaćanje i prosljeđivanje trenutno najboljeg rješenja sljedećem algoritmu.

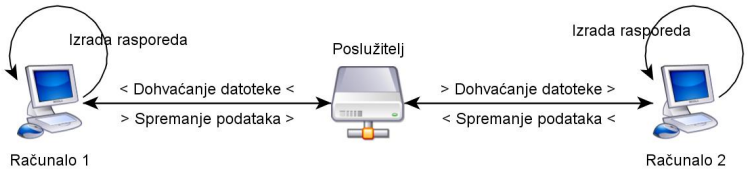



**Slika 2.1:** Organizacija sustava za izradu rasporeda nadoknada

Sustav mora moći ispravno raditi i sa samo jednim algoritmom na raspolaganju. To znači da svaki algoritam treba na temelju ulaznih podataka izgraditi vlastiti skup mogućih rješenja i biti u mogućnosti u potpunosti samostalno napredovati prema najboljem rješenju. Također, kada sustav od njega to zatraži, algoritam mora biti u mogućnosti predati svoje trenutno najbolje rješenje u unaprijed dogovorenom obliku ili vanjsko rješenje uključiti u vlastiti skup.

Cijeli sustav za izradu rasporeda nadoknada predviđen je za pokretanje na lokalnom računalu korisnika, a s internetskog poslužitelja će se dohvaćati u obliku izvršne datoteke. Ta datoteka se sastoji od programskog koda i podataka potrebnih za izvođenje. Cijeli sustav se ne izvršava na poslužitelju zbog toga što je to izuzetno složen zadatak koji zahtijeva veliku količinu računalnih sredstava. Pokretanje više izradi rasporeda na poslužitelju uvelike bi utjecalo na njegov normalan rad, a ni učinak svake pojedine izrade ne bi bio na zadovoljavajućoj razini.

Pokretanjem izrade na lokalnom računalu oslobađa se središnji sustav na internetskom poslužitelju. Osoba koja izvršnu datoteku dohvati s poslužitelja ima potpunu kontrolu nad izvođenjem — sama bira kada će izradu pokrenuti i zaustaviti, na kojem će se računalu izvršavati te u svakom trenutku ima uvid u stanje (najbolje pronađeno rješenje) izvođenja. Kada je raspored izrađen, potrebno ga je još samo poslati na poslužitelj koji će ažurirati zauzetost studenata i dvorana. Jednostavan pregled rada sustava prikazan je na slici 2.2. Za više informacija o sustavu za izradu rasporeda nadoknada pogledati Franović et al. (2009).




**Slika 2.2:** Pregled rada sustava za izradu rasporeda nadoknada

### 2.1.1. Ulazni podaci

Svi podaci potrebni za ispravan rad algoritma nalaze se zajedno s izvršnim kodom u datoteci koju korisnik dohvaća s poslužitelja. Sastoje se od dvije komponente. Prva komponenta je *XML* datoteka u kojoj su zapisani parametri rasporeda, dok drugu čine datoteke s podacima o raspoloživosti dvorana i studenata u određenom vremenskom razdoblju.

Svaki raspored određen je imenom i skupom događaja (engl. *event*) od kojih se sastoji. Događaj je određen imenom, identifikatorom i iznosom koji predstavlja vrijeme trajanja termina unutar tog događaja. Termin (engl. *term*) je jedna rezervacija dvorane u određenom vremenskom razdoblju. Svi termini jednog događaja imaju jednako trajanje. Svaki događaj sadrži informaciju o raspodjeli termina, koja može biti dodijeljena (engl. *given*) ili slučajna (engl. *random*).

Ukoliko je raspodjela termina dodijeljena, tada definicija događaja još mora sadržavati nazive i identifikatore svih termina koji su predviđeni za održavanje unutar tog događaja. Ako je distribucija slučajna, potrebno je odrediti najmanji i najveći broj dozvoljenih termina. U tom slučaju će algoritam tražiti rješenja rasporeda s brojem termina unutar tog raspona.

Osim imena i broja događaja i termina, u definiciji rasporeda je potrebno odrediti vremenski raspon unutar kojega će algoritam tražiti rješenja, mjesta (dvorane) u kojima će se nadoknade održavati i studente koji trebaju biti prisutni. Sva tri podatka mogu se odrediti na tri razine — na razini cijeloga rasporeda, na razini događaja ili na razini termina. Ukoliko se podaci odrede na višoj razini, podrazumijeva se da su određeni i na svim nižim razinama koje pripadaju navedenoj višoj. Na primjer, ukoliko je vremenski raspon određen na razini rasporeda, on vrijedi za sve događaje i termine. Ako su dvorane određene na razini događaja, podaci se primjenjuju samo na termine tog događaja, dok svi drugi događaji moraju određivati vlastite skupove dvorana.

Korisnik može odrediti i međusobnu povezanost događaja na način da se svi termini jednog događaja moraju odraditi u određeno vrijeme, odnosno prije početka termina drugog događaja. Ta povezanost se naziva preduvjet, a može je sadržavati svaki događaj. Da bi se zadao preduvjet, među podacima koji opisuju događaj mora se nalaziti naziv preduvjeta koji se treba završiti prije i vrijeme koje mora proći između završetka jednog i početka drugog događaja. U tom slučaju, algoritam će nuditi samo rješenja koja zadovoljavaju takav uvjet (ukoliko je to moguće).

U okviru 2.1 naveden je primjer datoteke s parametrima rasporeda. Vidljivo je da se raspored *PrimjerRasporeda* sastoji od dva događaja, *Event1* i *Event2*. Vremenski

raspon zadan je na razini rasporeda, što znači da se odnosi na oba događaja, jednako kao i dvorane u kojima se nadoknade održavaju. Studenti koji trebaju sudjelovati u nadoknadama su navedeni na razini događaja. Događaj *Event1* ima zadanu razdiobu termina, pa su unutar definicije događaja zadana dva termina, svaki sa svojim imenom i identifikatorom. Događaj *Event2* ima slučajnu razdiobu s parametrima $1$ i $3$, što znači da korisnik daje algoritmu mogućnost da sam procijeni optimalan broj termina (najmanje $1$, a najviše $3$) u kojima će svi studenti moći prisustvovati nadoknadi. Termini u događaju *Event1* traju $120$ minuta, a u *Event2* $240$ minuta. Naposljetku, *Event2* ima zadan *Event1* kao preduvjet s razmakom od jednog dana. To znači da će se prvi termin događaja *Event2* moći održati tek $1$ dan nakon zadnjeg termina događaja *Event1*.

Za svaki događaj određen u parametrima rasporeda među ulaznim podacima postoje još dvije datoteke. Jedna datoteka sadrži podatke o raspoloživosti svih studenata koji su dodijeljeni tom događaju unutar vremenskog razdoblja u kojem se događaj može održati. Druga sadrži jednake podatke o dvoranama u kojima se održavaju termini događaja.

**Okvir 2.1:** Primjer datoteke s podacima o rasporedu

```
<plan name="PrimjerRasporeda" id="3">
  <def>
    <time>2009-11-02 08:00#2009-11-10 20:00</time>
    <rooms>A101$FER/A101$20,A102$FER/A102$20</rooms>
  </def>
  <event name="Event1" id="E3.1" termDuration="120">
    <distribution>
      <type>GIVEN</type>
    </distribution>
    <def>
      <people>
        <jmbags>0000002307,0000002469,0000002477</jmbags>
      </people>
    </def>
    <term name="Termin1" id="T3.1.1"/>
    <term name="Termin2" id="T3.1.2"/>
  </event>
  <event name="Event2" id="E10.6" termDuration="240">
    <distribution>
      <type>RANDOM</type>
      <min>1</min>
      <max>3</max>
    </distribution>
    <preconditions>
      <precondition eventName="Event1" timeDistance="1d"/>
    </preconditions>
    <def>
      <people>
        <jmbags>0000001115,0000001226,0000001343</jmbags>
      </people>
    </def>
  </event>
</plan>
```

### 2.1.2. Izlazni podaci

Nakon što algoritam pronađe zadovoljavajuće rješenje, potrebno je izrađeni raspored vratiti na poslužitelj kako bi se rezervirale dvorane i studenti. Datoteka s izrađenim rasporedom sintaksno je jednaka datoteci s parametrima za izradu rasporeda, uz izostanak podataka o preduvjetima ili slučajnoj raspodjeli termina.

Ukratko, datoteka s izrađenim rasporedom zapravo je lista događaja od kojih svaki ima zadanu razdiobu i točno navedene termine. Svaki termin ima zadani vremenski okvir i dvoranu u kojoj se održava, te studente koji su u njemu raspoređeni. Primjer izrađenog rasporeda prikazuje okvir 2.2.

Svaki raspored trebao bi zadovoljavati sljedeća ograničenja:

1. niti jedan termin ne smije biti zakazan izvan zadanog vremenskog okvira,
2. svi termini moraju biti zakazani samo u dvoranama koje su dane na raspolaganje navedenim terminima,
3. svi zadani studenti moraju biti raspoređeni unutar razine na kojoj su zadani,
4. ukoliko je moguće rasporediti studente u manji broj dvorana, to treba učiniti (ako je kapacitet neke dvorane manji od ukupnog broja slobodnih mjesta u rezerviranim dvoranama),
5. svi termini jednog događaja moraju imati jednako trajanje koje je zadano u ulaznim parametrima,
6. broj termina u događaju sa zadanom raspodjelom termina ne smije se mijenjati,
7. broj termina u događaju sa slučajnom raspodjelom termina ne smije biti manji od minimalnog zadanog, ni veći od maksimalnog zadanog,
8. događaj koji ima zadane preduvjete mora ih sve ispunjavati,
9. dva termina ne smiju biti zakazana u isto vrijeme i u istim dvoranama,
10. dvorane ne smiju biti popunjene preko kapaciteta i
11. student rezerviran u nekom terminu ne smije u isto vrijeme već biti zauzet nekom drugom zakazanom obavezom.

**Okvir 2.2:** Primjer gotovog rasporeda

```
<schedule planid="3">
  <plan name="IzradjenRaspored">
    <event name="Event1" id="E3.1" termDuration="15">
      <term name="Termin1" id="3.1.2">
        <def>
          <people>
            <jmbags>0000002499,0000002517,0000002618</jmbags>
          </people>
          <time>2009-11-02 12:00#2009-11-02 14:00</time>
          <rooms>FER/A209$FER/A209$20</rooms>
        </def>
      </term>
      <term name="Termin2" id="3.1.3">
        <def>
          <people>
            <jmbags>0000002005,0000002149,0000001909</jmbags>
          </people>
          <time>2009-11-02 18:00#2009-11-02 20:00</time>
          <rooms>FER/A102$FER/A102$20</rooms>
        </def>
    </event>
  </plan>
</schedule>
```

Unatoč tome, često se može dogoditi da zbog krivo zadanih parametara ili „gustog“ postojećeg rasporeda do takvog rješenja nije moguće doći. Svaki raspored mora zadovoljavati točke od 1 do 6, odnosno algoritam mora vratiti odgovarajuću informaciju ukoliko te uvjete ne može zadovoljiti. Za ostale slučajeve može se nastaviti tražiti najbolje rješenje.

## 2.2. Vrednovanje rješenja

Kvaliteta rješenja, odnosno izrađenog rasporeda se vrednuje kroz nekoliko komponenti. Neke od njih su više, a neke manje važne. Komponente koje utječu na izračun kvalitete rješenja (od najvažnije prema najmanje važnoj) su:

1. broj nezadovoljenih preduvjeta,

2. broj konfliktnih dvorana,

3. broj prenapučenih dvorana,

4. broj konfliktnih studenata,

5. broj termina i

6. broj slobodnih mjesta u dvoranama.

Prve četiri komponente čine čvrsta ograničenja (engl. *hard constraints*). Raspored koji ne zadovoljava čvrsta ograničenja nije u potpunosti zadovoljavajući. Broj nezadovoljenih preduvjeta i konfliktnih dvorana pritom su posebno važni, jer je raspored s tim značajkama potpuno neizvediv u obliku u kojemu je zadan. Ipak, takvo rješenje može korisniku ukazati na moguću pogrešku u zadavanju problema ili nemogućnost pronalaska rješenja koje bi zadovoljavalo prva dva čvrsta ograničenja.

Broj konfliktnih studenata i prenapučenih dvorana predstavljaju čvrsta ograničenja, ali su daleko prihvatljivija od prva dva. Ukoliko će pokoji student možda imati drugu obavezu za vrijeme zakazane nadoknade, ponuđeni termin može odgovarati značajnoj većini ostalih studenata pa se spomenuti problem lako može riješiti na neki drugi način (na primjer dolaskom na konzultacije). Također, manji postotak prenapučenih dvorana je moguće prihvatiti kao rješenje (osobito za predavanja).

Posljednje dvije komponente se nazivaju mekim ograničenjima (engl. *soft constraints*). Neovisno o njihovom iznosu raspored je u potpunosti valjan i prihvatljiv. Meka ograničenja služe za bolju usporedbu rješenja i odabir najboljega.

Tako se kvaliteta svakog rješenja (rasporeda) može prikazati $n$-komponentnim vektorom (u ovom slučaju sa $6$ komponenti) u obliku:

*[nezadovoljeni preduvjeti, konfliktne dvorane, konfliktni studenti, prenapučene dvorane, broj termina, broj slobodnih mjesta].*

Kvalitetnije rješenje je ono s manjim vrijednostima komponenti višeg prioriteta. Zato se usporedba dvaju rješenja vrši na način da se prvo usporede vrijednosti prve komponente. Ako su jednake, uspoređuju se vrijednosti druge komponente i tako dalje. Tako, na primjer, vrijedi slijedećih nekoliko nejednakosti (pri čemu znak „manje“ ukazuje na bolje rješenje):

$$[0, 1, 72, 6, 8, 32] < [1, 0, 56, 4, 10, 85],$$
$$[0, 0, 12, 0, 13, 269] < [0, 0, 12, 0, 13, 123],$$
$$[0, 0, 15, 0, 7, 101] < [0, 0, 16, 0, 7, 0] \text{ itd.}$$

# 3. Evolucijski algoritmi

Sami počeci evolucijskog računarstva sežu još u daleke 1930-te godine. Međutim, tek je pad cijena elektroničkih računala u 1960-ih katalitički djelovao na ovo područje. Rasprostranjenošću nove tehnologije omogućeno je korištenje računalne simulacije kao alata za analizu sustava puno složenijih od onih koje je analizirala matematika.

Među znanstvenicima posebno zainteresiranim za ovo područje bili su evolucijski biolozi koji su u njemu vidjeli mogućnost razvoja i proučavanja novih prirodnih evolucijskih sustava i inženjeri iz područja računarske znanosti koji su moć evolucije željeli iskoristiti za razvoj boljih rješenja postojećih problema, uključujući i područja umjetne inteligencije.

U ovom poglavlju navedena je kratka povijest razvoja evolucijskih algoritama i objašnjeni su osnovni pojmovi i princip rada. Navedene su tri glavne skupine evolucijskih algoritama i njihove međusobne razlike. Na kraju su pobliže objašnjeni operatori mutacije i križanja u genetskom algoritmu.

## 3.1. Povijest

Tridesetih godina prošloga stoljeća, ideje Sewella Wrighta (opisane u Wright (1932)) pomogle su u popularizaciji evolucijskog računarstva. On je opisao kako evolucijski sustav istražuje prostor s više lokalnih ekstrema[1] i dinamički oblikuje nakupine jedinki oko takvih područja. Time se evolucijski sustav lako može zamisliti kao optimizacijski proces, što je stajalište koje i danas ima svoje pobornike i protivnike. Unatoč tome, gledanje na evolucijski sustav kao optimizaciju pridonio je razvoju algoritama koji su automatizirali mnoge mukotrpne optimizacijske procese. Kao rezultat toga, optimizacijski problemi su i danas prevladavajuće područje upotrebe evolucijskog računarstva.

U šezdesetima je započeo intenzivniji razvitak različitih upotreba evolucijskih sustava. Nekoliko grupa ljudi je uvidjelo kako se i jednostavni evolucijski modeli mogu

[1] tzv. višemodalni sustav, prema Golub (2004)

predstaviti u računalnom obliku i iskoristiti za rješavanje složenih problema.

Na Tehničkom sveučilištu u Berlinu su, prema De Jong (2006), Rechenberg i Schwefel počeli oblikovati ideju kako bi evolucijski procesi mogli biti iskorišteni za rješavanje optimizacijskih problema s realnim parametrima (više u Rechenberg (1965)). Iz tih prvih ideja nastale su evolucijske strategije, jedna od podskupina evolucijskih algoritama koja i danas predstavlja najmoćniji alat za optimiziranje funkcija.

Na UCLA-u je u isto vrijeme Fogel uočio potencijal evolucije u rješavanju problema iz područja umjetne inteligencije (Fogel et al. (1966)). Njegove ideje bile su vezane uz korištenje evolucijskih sustava za unaprjeđenje inteligentnih agenata, koji su bili oblikovani kao strojevi s konačnim brojem stanja. Tada razvijen programski okvir, nazvan „evolucijsko programiranje“, se unaprjeđuje i danas, te se osim evolucije strojeva s konačnim brojem stanja koristi za rješavanje mnogih drugih problema.

Holland je, na Sveučilištu u Michiganu, u evolucijskom sustavu vidio ključ oblikovanja samoprilagođavajućih sustava sposobnih za snalaženje u nedeterminističkom okruženju (što je opisano u Holland (1962) i Holland (1967)). On je naglasio važnost potrebe da se takvi sustavi s vremenom prilagođavaju svojoj okolini na temelju informacija koje od nje primaju. To je dovelo do nove podskupine evolucijskih algoritama nazvane „reproduktivni planovi“ koji čine osnovu današnjih genetskih algoritama.

Sva ta rješenja su bila vrlo idealizirani modeli evolucijskih procesa u prirodi. Cilj nije bio modelirati vjerne kopije bioloških procesa, već prilagoditi ostvarenje na način da se modeliraju one komponente koje su važne za rješavanje problema. Kao posljedica toga danas većina evolucijskih algoritama radi s poprilično jednostavnim pretpostavkama (utvrđena veličina populacije, jednospolne jedinke, statički okoliš dobrote i druge, o kojima će više riječi biti u sljedećim odlomcima).

## 3.2. Jednostavni evolucijski sustav

Prije ulaska u detalje oko konkretnih primjena evolucijskih algoritama, potrebno je shvatiti osnovne pojmove vezane uz svaki evolucijski sustav.

Charles Darwin je, kako piše Čupić (2009b), teoriju evolucije predstavio pomoću pet pretpostavki na kojima se temelje evolucijski algoritmi:

- plodnost vrsta — potomaka ima uvijek više nego što je potrebno,
- veličina populacije je približno stalna,
- količina hrane je ograničena,

– kod vrsta koje se seksualno razmnožavaju nema identičnih jedinki nego postoje varijacije i

– najveći dio varijacija prenosi se nasljeđivanjem.

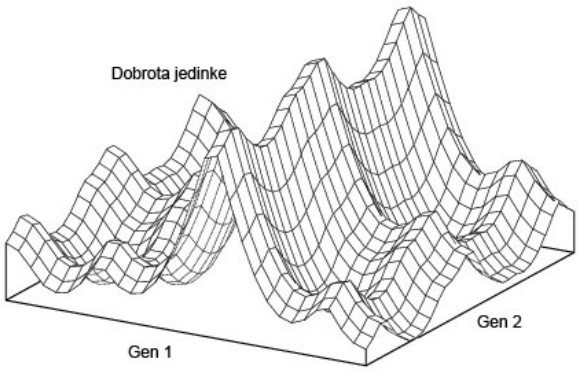
Iz tih razloga će u svakoj populaciji postojati borba za preživljavanje — ukoliko je količina hrane ograničena, preživjet će i razmnožavati se samo one jedinke koje će biti jače.

Osnovno pitanje koje se postavlja je: na koji način predstaviti jedinku (ili organizam) koja će biti dio populacije. Najjednostavniji način bi bio odrediti $L$ osobina od kojih bi svaka bila izabrana zbog svojeg svojstva da pomogne pri određivanju kvalitete jedinke — njezine dobrote (engl. *fitness*). Kasnije će biti prikazano kako će taj faktor utjecati na sposobnost opstanka jedinki u populaciji.

Svaka jedinka će biti predstavljena vektorom osobina sa sebi svojstvenim vrijednostima. Na primjer:

<*visina, težina, boja kože, boja očiju, boja kose*>

Navedeni vektor možemo zamisliti kao skup genetskih osobina (genotip), kao skup vanjskih, stečenih osobina (fenotip) ili kao njihovu kombinaciju. On razapinje petodimenzionalni prostor značajki u kojemu svakoj točki možemo pridružiti neku vrijednost, vrijednost dobrote za jedinku koja poprima takve značajke. Takav prostor zove se okoliš dobrote, a grafički prikaz prostora dobrote za genotip od dvije značajke predstavljen je na slici 3.1.




**Slika 3.1:** Okoliš dobrote za dvije značajke

S tako opisanim pojmovima, moguće je odrediti zakone kretanja evolucijskog sustava odnosno osnovni evolucijski algoritam, koji je predstavljen algoritmom 3.1.

Važno je uočiti kako je algoritam nedeterministički, odnosno da će u različitim pokretanjima davati različite rezultate. To ujedno može biti i pozitivno i negativno.

Pozitivno može biti u kontekstu testiranja jer se brzo može ispitati ponašanje sustava u različitim uvjetima. S druge strane, ukoliko dođe do pogreške, vrlo ju je teško ponoviti kako bi se ispravila. Upravo zbog nedeterminizma, važno je ne donositi zaključke o ponašanju algoritma na temelju jednog ili dva pokretanja.

**Algoritam 3.1** Osnovni evolucijski algoritam

Stvori početnu populaciju od $M$ jedinki
- koristeći jednoliku razdiobu nad cijelim prostorom značajki, te izračunaj vrijednost dobrote za svaku jedinku

**ponavljaj**

Odaberi jednog člana populacije za roditelja
- koristeći jednoliku razdiobu nad cijelom populacijom (svaka jedinka ima jednaku vjerojatnost da bude izabrana)

Koristeći genotip izabranog roditelja, stvori dijete
- na način da kloniraš roditelja i nad tako stvorenom jedinkom izvrši mutaciju

Izračunaj dobrotu stvorene jedinke

Izaberi člana populacije koji će odumrijeti
- na način da nasumično (jednolikom razdiobom nad cijelom populacijom roditelja) izabereš jednog kandidata iz roditeljske populacije koji će se boriti za opstanak sa stvorenom jedinkom. U roditeljskoj populaciji ostaje jedinka s većom dobrotom.

**kraj ponavljanja**

Potrebno je pobliže opisati postupak mutacije. Mutacija u ovom kontekstu predstavlja izmjenu jednog gena (jedne vrijednosti u vektoru karakteristika). Izbor gena za mutaciju vrši se nasumično, pri čemu svaki gen ima jednaku vjerojatnost da bude izabran, a postupak mutiranja predstavlja mala promjena vrijednosti gena. Na primjer, ako bi vrijednosti bile realni brojevi, mutacija bi mogla biti dodavanje ili oduzimanje nekog iznosa $\delta$.

Važno je uočiti kako evolucijski algoritam ne pretražuje temeljito cijeli prostor značajki, nego „skače“ iz točke u točku vršenjem mutacije nad jedinkama populacije. Kako se nova jedinka nakon stvaranja odmah natječe za mjesto u populaciji s jednom od slučajno izabranih članica i u populaciji ostaje „jača“, srednja vrijednost dobrote populacije se svakim korakom algoritma penje prema jednom od vrhova okoliša dobrote.

## 3.3. Evolucijsko programiranje

Promatrajući pseudokod osnovnog evolucijskog algoritma možemo uočiti više načina na koje bi ga mogli unaprijediti. Na primjer, kako se svaki puta nasumično odabire roditelj koji će stvoriti dijete, moguć je slučaj u kojem se najbolja ili iznimno dobra jedinka u populaciji nikad ne izabere za reprodukciju. Kako se novo dijete natječe za opstanak s jednim trenutnim pripadnikom populacije, moguće je da neka loša jedinka opstane jer je „ždrijeb“ nikad nije izabrao za natjecanje.

Evolucijsko programiranje rješava navedeni problem. Jednako kao i osnovni algoritam, svodi se na model koji uključuje populaciju roditelja utvrđene veličine $N$ jedinki. Razlika je u tome što svaki od roditelja stvara po jedno dijete. Nova generacija roditelja dobiva se tako da se obje generacije (roditelja i djece) spoje u jednu i sortiraju po dobroti. Iz tako dobivene populacije veličine $2N$ se izabere prvih $N$ jedinki. Reprodukcija, odnosno stvaranje djece, može biti aseksualna (kloniranje roditelja i mutacija djeteta) ili seksualna (kombiniranje značajki oba roditelja). Više o evolucijskom programiranju pišu Fogel et al. (1966).

De Jong (2006) je testovima pokazao da pri jednakim parametrima evolucijsko programiranje daje nešto bolje rezultate nego osnovni algoritam. To se može objasniti činjenicom da evolucijsko programiranje ima više „elitističku“ selekciju, budući da samo $50\%$ najboljih opstaje u sljedećoj generaciji. Samim time se povećava srednja dobrota populacije, ali i smanjuje raznolikost, što može rezultirati konvergiranju prema lokalnom optimumu. Općenito je vrlo važno naći ravnotežu između dijelova sustava koji povećavaju raznolikost (npr. mutacija) i onih koji je smanjuju (npr. selekcija).

## 3.4. Evolucijske strategije

Davanje jednake mogućnosti svakom članu populacije da se na temelju njegovog genetskog materijala stvori nova jedinka nije slučaj koji često možemo susresti u prirodi. I sam Darwin je u svojoj teoriji evolucije naglasio kako potomaka ima uvijek više nego je potrebno. Evolucijske strategije leže upravo na toj pretpostavci. Postupci osnovnog algoritma i evolucijskog programiranja lako se mogu prepraviti da stvaraju veću populaciju djece. Ipak, uz tu mogućnost postavlja se pitanje kakav bi trebao biti odnos veličina roditeljske i dječje populacije.

Sedamdesetih godina prošloga stoljeća paralelno su se razvijala dva modela evolucijskih strategija. Jedan je bio $(\mu + \lambda)$ model, pri čemu je $\mu$ definiran kao veličina roditeljske populacije, dok je $\lambda$ predstavljao veličinu populacije djece. Drugi, up-

ečatljiviji model $(1 + \lambda)$ predviđa samo jednog roditelja koji stvara cijelu novu populaciju.

Treba imati na umu da takav model gubi usporednost specifičnu za evolucijske algoritme. Počevši od samo jedne roditeljske jedinke, algoritam se svodi samo na niz njezinih mutacija (od cijele populacije djece bira se jedna koja se dalje mutira i razmnožava). Sveden na samo jedan početni genotip, ishod algoritma uvelike ovisi o dobroti prve stvorene jedinke. S druge strane, $(\mu + \lambda)$ model pokazao se uspješnijim.

## 3.5. Genetski algoritam

Značajka svih do sada navedenih algoritama je da jedinke u populaciji izumiru tek kada ih zamijene nove jedinke s većom dobrotom. U takvim populacijama nema prirodnog starenja, a vrijednost prosječne dobrote cijele populacije je monotono rastuća odnosno padajuća funkcija koja se kreće prema jednom od optimuma okoliša dobrote. Važno je pritom naglasiti da prosječna dobrota teži jednom od optimuma, jer to može predstavljati problem. Dopuštajući jedinkama da žive beskonačno dugo smanjuje se raznolikost populacije. Unutar nje postupno prevladava jedna „obitelj“ jedinki koja ne mora nužno težiti globalnom optimumu okoliša dobrote, a naposljetku će cijela populacija biti „zarobljena“ na lokalnom optimumu.

Postoji nekoliko načina kako se spomenuti problem može riješiti. Jedan od njih je da se s vremena na vrijeme dopusti da nova jedinka s nižom dobrotom zamijeni stariju, bolju jedinku. Općenito se evolucijski modeli u kojima se u jednom koraku izmjenjuju jedna ili nekoliko jedinki zovu modeli stacionarnog stanja (engl. *steady-state*). Potpuno drugačiji pristup bio bi da se cijela roditeljska populacija u svakom koraku u potpunosti zamijeni populacijom djece. Takav model naziva se generacijski, a primjenjuju ga genetski algoritam i neke vrste ($\mu$, $\lambda$) evolucijskih strategija (više o generacijskim genetskim algoritmima u Holland (1975), De Jong (1975) i Goldberg (1989)).

Uz to, genetski algoritam primjenjuje pravi biološki značaj dobrote jedinke, a to je sposobnost da preživi i razmnožava se. Osnovni evolucijski algoritam, evolucijsko programiranje i evolucijske strategije koriste model u kojemu je dobrota jedinke važna pri opstanku unutar populacije. Jednom kada se nađe u populaciji, svaka jedinka ima jednaku mogućnost za razmnožavanje. Genetski algoritam pri odabiru jedinki za razmnožavanje uzima u obzir dobrotu, na način da će jedinka s većom dobrotom češće biti izabrana za stvaranje potomstva. Pseudokod s primjerom generacijskog genetskog algoritma prikazan je algoritmom 3.2.

Lako je uočiti posljedice koje sa sobom nose novosti uvedene u genetskom algoritmu. Kako se cijela postojeća populacija zamjenjuje novom, funkcija srednje vrijednosti dobrote populacije više neće biti monotona nego će oscilirati. Takvo ponašanje je poželjno, jer ne dopušta da algoritam zastane na lokalnom optimumu.

**Algoritam 3.2** Primjer generacijskog genetskog algoritma

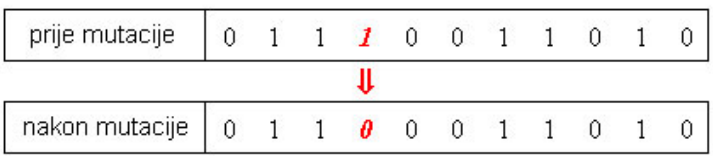
Stvori početnu populaciju od $M$ jedinki
**ponavljaj**
  Izračunaj i spremi vrijednost dobrote $u(i)$ za svakog člana populacije $i$
  Odredi vjerojatnosti $p(i)$ selekcije jedinki za razmnožavanje
  – pri čemu je $p(i)$ proporcionalan $u(i)$
  Stvori $M$ novih jedinki
  – tako da koristiš prije određenu vjerojatnost selekcije roditelja
  U potpunosti zamijeni staru populaciju novom
**kraj ponavljanja**

### 3.5.1. Evolucijski operatori

Evolucijski operatori su „alati“ evolucije. Pomoću njih se populacija mijenja i napreduje. Do sada su već spomenuta dva evolucijska operatora — selekcija i mutacija.

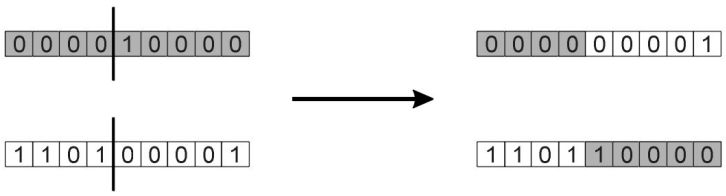
*Selekcija* služi za odabir jedinki koje će stvarati potomstvo. U prošlom odlomku je opisano kako genetski algoritam uvodi selekciju na temelju dobrote, što znači da će jedinka s većom dobrotom imati veću vjerojatnost biti izabrana za razmnožavanje.

*Mutacija* djeluje na jednu jedinku, mijenjajući je u određenom postotku kako bi utjecala na raznolikost. U odlomku o osnovnom evolucijskom algoritmu naveden je jedan od načina kako se može izvoditi mutacija. Međutim, ona se može provoditi i na puno nižoj razini koja može biti jednaka za velik broj problema (iako nisu svi problemi rješivi takvom metodom). Ako se pretpostavi da je cijela jedinka, odnosno njezin genotip, pretvoren u binarni zapis, tada bi mutacija predstavljala izmjenu određenog broja bitova – 0 u 1 i obrnutom kao što je prikazano na slici 3.2.




**Slika 3.2:** Mutacija jednog bita

Genetski algoritam uvodi još jednu novost s obzirom na ostale dosad spomenute modele – razmnožavanje s više roditelja. Osnovna ideja svodi se na to da dijete nasljeđuje genetski materijal od više roditelja, dok se pritom ne treba ograničiti na samo dvoje. Stvaranje nove jedinke na temelju dvije ili više roditeljskih naziva se *križanje* (engl. *crossover*). Križanje predstavlja vrlo važnu komponentu genetskog algoritma jer s obzirom na jednospolno razmnožavanje rezultira raznovrsnijim jedinkama. Ukoliko se ovdje jedinka predstavi samo nizom bitova, križanje se može izvršiti na način da se nasumično odabere pozicija oko koje će se niz podijeliti na dva dijela. Zatim se lijevi dio kopira od jednog roditelja, a desni od drugog, što prikazuje slika 3.3. Takvo križanje naziva se križanje oko jedne točke (engl. *1 point crossover*). Na sličan način može se izvesti i križanje oko više točaka (engl. *2 point crossover, n point crossover*).




**Slika 3.3:** Križanje oko jedne točke

# 4. Primjena genetskog algoritma na problem izrade rasporeda nadoknada

U ovom poglavlju pobliže je objašnjeno rješenje ostvareno u programskom jeziku *Java* koje koristi genetski algoritam za izradu rasporeda nadoknada. Opisan je način na koji je algoritam prilagođen radu unutar većeg sustava i strukture podataka kojima je predstavljeno rješenje. Navedene su izvedbe evolucijskih operatora, a na kraju je opisana potreba za algoritmima lokalne pretrage i način na koji su ti algoritmi ostvareni.

## 4.1. Ostvarenje rješenja

Programsko rješenje se sastoji od struktura podataka koje predstavljaju primjerak rasporeda i samog genetskog algoritma koji koristi navedene strukture. Zbog složenosti rješenja i usmjerenosti ovog rada na genetski algoritam, ovdje su spomenuti i opisani samo važniji razredi i metode.

### 4.1.1. Strukture podataka

Jedan od najvažnijih razreda je onaj koji predstavlja jedinku populacije (jedno moguće rješenje), a naziva se `Chromosome`. Zbog bolje kontrole nad konvergencijom populacije i učinkovitijih operacija križanja i mutacije, jedinka nije predstavljena kao niz bitova (kako je to opisano u poglavlju 3.5.1.), nego je sastavljena od složenih tipova podataka. Od metoda razreda `Chromosome` ističu se sljedeće:

- `void importResult(ISchedulingResult)` — stvara novu jedinku na temelju rješenja drugog algoritma,
- `void generateEvents()` — slučajnim odabirom pokušava stvoriti novu jedinku poštivajući pritom sva zadana pravila,
- `void crossover(Chromosome parent1, Chromosome parent2)` — stvara novu jedinku (dijete) na temelju provedbe križanja dvaju roditelja,

- `void mutate()` — provodi algoritme mutacije nad trenutnom jedinkom,
- `void optimize()` — optimira trenutnu jedinku prema ograničenju 4 navedenom u poglavlju 2.1.2. i
- `void localSearch()` — provodi algoritme lokalne pretrage nad trenutnom jedinkom.

Svaka jedinka sadrži listu događaja koji su predstavljeni razredom `GenEvent`, dok je svaki događaj sastavljen od termina predstavljenih razredom `GenTerm`. Razredi `GenEvent` i `GenTerm` sadrže nazive i identifikatore događaja odnosno termina, a `GenTerm` još dodatno sadrži podatke o rezervaciji termina, poput vremena i dvorane u kojoj se termin održava, studenata koji su dodijeljeni navedenom terminu i slično.

Razred `GenEvent` ima slične metode kao i razred `Chromosome`, jer se većina operacija provodi na razini događaja i termina, pa se pozivi metoda samo prosljeđuju razredom `Chromosome` na sve događaje od kojih se jedinka sastoji. Zanimljivo je samo istaknuti metodu `organizeStudents`, koja u slučaju zadane raspodjele termina jednoliko raspoređuje studente u sve termine, uzimajući u obzir raspoloživost studenata u zadanom vremenu. Kod slučajne razdiobe takva raspodjela studenata nije potrebna jer se novi termini rezerviraju prema potrebi.

Važno je istaknuti još jedan razred (`GenResMan`) koji predstavlja upravitelja rezervacijama. Razred na jednom mjestu sadrži sve informacije o rezervacijama dvorana i studenata, te razlikuje fiksne i lokalne rezervacije. Fiksne rezervacije su zadane u ulaznim podacima. To su termini koji su već zauzeti u glavnom rasporedu i koje nije moguće mijenjati (redovna nastava, ispiti i laboratorijske vježbe). Lokalne rezervacije su one koje su zadane unutar izrade rasporeda, odnosno one koje je privremeno rezervirao sam algoritam. U slučaju potrebe, te rezervacije se mogu izbrisati i na isto mjesto rezervirati neki drugi termin.

Osim metoda za rezervaciju i oslobađanje određenog termina i ispitivanje zauzetosti studenta ili dvorane za zadani vremenski raspon, `GenResMan` nudi i metode za dohvat svih slobodnih termina za dvoranu u određenom vremenskom rasponu. To je posebno korisno pri inicijalizaciji novih termina jer se vrlo lako može nasumično odabrati vrijeme za rezervaciju koje je unutar dozvoljenih okvira.

### 4.1.2. Prilagodba algoritma sustavu za izradu nadoknada

U poglavlju 2.1. opisana je potreba za ostvarenjem algoritma koji će biti u mogućnosti raditi kao dio većeg sustava. Navedeno je kako algoritam mora moći koristiti rješenja drugih algoritama kako bi poboljšao svoje rezultate. Iz tog razloga oblikovano je sučelje koje svaki algoritam mora ostvariti.

Naziv sučelja je `ISchedulingAlgorithm`, a značajnije javne metode koje određuje su:

- `void prepare(IPlan plan, Map<String, ISchedulingData> eventsSchedulingData)` — priprema strukture na temelju predanih podataka,
- `void step()` — izvršava jedan korak algoritma,
- `void use(ISchedulingResult result)` — uključuje rješenje drugog algoritma u vlastitu populaciju i
- `ISchedulingResult getResult()` — vraća najbolju jedinku iz populacije u predviđenom obliku.

Metodom `prepare` algoritmu se predaju svi podaci potrebni za izradu rasporeda. Detaljniji opis ulaznih podataka naveden je u poglavlju 2.1.1. Objekt `plan` sadrži parametre rasporeda, dok objekt `eventsSchedulingData` sadrži podatke o zauzećima dvorana i studenata za razdoblje u kojemu se raspored izrađuje.

Uključivanje vanjskog rješenja u vlastitu populaciju, određeno metodom `use`, ostvaruje se tako da se jedna od jedinki populacije zamijeni vanjskim rješenjem. To pridonosi bržoj konvergenciji rješenja i predstavlja dodatno osiguranje da algoritam neće zastati na lokalnom optimumu.

## 4.2. Evolucijski operatori

Nakon opisa osnovnih komponenti programskog rješenja, moguće je objasniti na koji način se nad populacijom provode operatori križanja odnosno mutacije jedinki. Kako jedinka, odnosno kromosom, u programskom rješenju nije predstavljena kao niz bitova, tako se niti evolucijski operatori ne mogu realizirati kao jednostavne operacije nad bitovima, već su nešto složenije prirode.

### 4.2.1. Križanje

Križanje je evolucijski operator kojim se iz dvije ili više jedinki stvara nova koja ima obilježja svih roditelja. U ovom programskom rješenju ostvareno je križanje s dva roditelja. Postupkom se željelo postići da jedinka dijete naslijedi (koliko je to moguće) najpozitivnija obilježja oba roditelja. Križanje se provodi na razini događaja, jer na razini rasporeda sve jedinke moraju izgledati jednako (imati iste događaje).

Pseudokod postupka križanja prikazan je algoritmom 4.3. Algoritmom je prikazano križanje dva događaja, jer se križanje jedinki svodi na međusobno križanje istovjetnih događaja. Razlikuju se dva slučaja koja određuju na koji način se križanje provodi, slučaj kad je razdioba termina dodijeljena i kada je slučajna.

Na početku se izdvajaju termini jednog, odnosno drugog događaja, i priprema se lista događaja nove jedinke. Dohvaća se i lista studenata koji trebaju biti raspoređeni u termine. U ovoj točki algoritam se dijeli na dva slučaja.

U slučaju da je raspodjela termina u događaju zadana, termini se međusobno ne smiju miješati, nego se od svakog pojedinog termina treba uzeti jedan primjerak. To se radi tako da se prolazi po nizovima termina jednog i drugog događaja uz uspoređivanje istovjetnih. Termin koji je bolji odabire se za prelazak na novu jedinku. Dobrota termina se procjenjuje na temelju omjera prenapučenosti dvorane i broja konfliktnih studenata. Ukoliko su studenti zadani na razini događaja, nakon odabira termina vrši se jednolika raspodjela studenata po svim terminima. Ako su studenti zadani na razini termina, oni su već ispravno dodijeljeni u svoje termine i raspodjelu nije potrebno vršiti.

Ukoliko je raspodjela termina slučajna, termini jednog i drugog događaja se grupiraju, pa se zatim niz sortira prema dobroti. Novoj jedinki se dodjeli minimalan broj termina i pokuša se rasporediti sve studente u postojeće termine. Ukoliko postupak ne uspije, dodaje se jedan po jedan termin sve dok se ne dosegne maksimalan dopušteni broj termina i pokušava se rasporediti sve studente. Ako svi studenti ne budu uspješno raspoređeni u maksimalan broj termina, raspoređuje ih se na silu (što može imati za posljedicu konflikte studenata ili prenapučenost dvorana).

**Algoritam 4.3** Križanje dva događaja

$T1 \leftarrow terminiDogadaja1$
$T2 \leftarrow terminiDogadaja2$
$DIJETE \leftarrow \emptyset$
$S \leftarrow DOG.studenti$
**ako** (raspodjela $GIVEN$) **tada**
  **za** $(i = 0; i < T1.duljina; i++)$ **radi**
    **ako** $(T1_i < T2_i)$ **tada**
      $DIJETE \leftarrow DIJETE + T1_i$
    **inače**
      $DIJETE \leftarrow DIJETE + T2_i$
    **završi ako**
  **završi za**
  **ako** $(\neg studentiNaRaziniTermina)$ **tada**
    $organizirajSveStudente(DIJETE, S)$
  **završi ako**
**inače**
  $TA \leftarrow T1 + T2$
  $sortiraj(TA)$
  **za** $(i = 0; i < DOG.minBrTermina; i++)$ **radi**
    $DIJETE \leftarrow TA.ukloni(0)$
  **završi za**
  $organizirajStudente(DIJETE, S)$
  **dok** $(DIJETE.vel <= DOG.maksBrTermina \;\&\&\; S \neq \emptyset)$ **radi**
    $DIJETE \leftarrow TA.ukloni(0)$
    $organizirajStudente(DIJETE, S)$
  **završi dok**
  **ako** $(S \neq \emptyset)$ **tada**
    $organizirajSveStudente(DIJETE, S)$
  **završi ako**
**završi ako**

### 4.2.2. Mutacija

Cilj mutacije je osigurati raznolikost. Posebno je važna jer ima sposobnost „izbacivanja“ algoritma iz lokalnog optimuma, što je vrlo česta situacija u kojoj se može naći. Ovo programsko rješenje sastoji se od tri različite metode mutacije:

1. pronalazak novog termina,
2. potpuna zamjena studenata između dvaju termina te
3. djelomična zamjena studenata između termina.

Prva metoda, pronalazak novog termina, radi na principu odbacivanja jednog slučajno odabranog termina i pronalazak novog (dvorane i vremena održavanja). Svi studenti koji su bili dodijeljeni odbačenom terminu dodjeljuju se i novom terminu, neovisno o činjenici odgovara li im novi termin ili ne. Pritom se obraća pozornost da se za novi termin odabere vrijeme održavanja unutar zadanog okvira i dvorana raspoloživa u zadanom vremenu.

Pri potpunoj zamjeni studenata oba termina određuju se slučajnim odabirom. Zatim se svi studenti prvobitno dodijeljeni prvom terminu presele u drugi i obrnuto. Djelomična zamjena studenata između termina, osim po činjenici da se ne sele svi studenti iz jednog termina, razlikuje se i po tome što ne moraju nužno svi studenti jednog termina biti preseljeni u isti odredišni termin, već se on također određuje slučajnim odabirom.

Metoda mutacije koja će se izvršavati uglavnom se odabire slučajno, pri čemu svaka metoda ima jednaku vjerojatnost da bude izabrana. U posebnim slučajevima (kada je raspodjela studenata određena na razini termina ili događaj čini samo jedan termin) je moguće odabrati i izvršiti samo prvu metodu, kako je prikazano algoritmom 4.4.

**Algoritam 4.4** Odabir metode mutacije

**ako** ($studentiNaRaziniTermina \parallel TERMINI.vel < 2$) **tada**
  $mutacija1()$
**inače**
  $ODABIR \leftarrow slucajno(3)$
  **ako** ($ODABIR == 0$) **tada**
    $mutacija1()$
  **završi ako**
  **ako** ($ODABIR == 1$) **tada**
    $mutacija2()$
  **završi ako**
  **ako** ($ODABIR == 2$) **tada**
    $mutacija3()$
  **završi ako**
**završi ako**

### 4.2.3. Selekcija

U poglavlju 3.5.1. opisana je funkcija i značaj operatora selekcije za bolji rad algoritma. Ovdje je opisan samo mehanizam koji koristi programsko rješenje.

Poželjno je da rad operatora selekcije bude usko povezan s dobrotama jedinki u smislu da jedinka s većom dobrotom ima i veću vjerojatnost da bude izabrana za razmnožavanje. Jedan od rješenja navedenog problema je proporcionalna selekcija. Proporcionalna selekcija znači da će vjerojatnost odabira jedinke za razmnožavanje biti proporcionalna njezinoj dobroti. Problem nastaje u slučaju kad se dobrota jedinki u populaciji minimalno razlikuje, što znači da će i vjerojatnosti odabira biti približno jednake. Time se gubi svojstvo češćeg odabra bolje jedinke. Druga prepreka pri ostvarenju proporcionalne selekcije je činjenica da je u slučaju izrade rasporeda nadoknada dobrota jedinke predstavljena $6$-dimenzionalnim vektorom. Iz navedenih razloga se u ovom slučaju koristi selekcija rangiranjem.

U selekciji rangiranjem vjerojatnost odabira jedinke je određena njezinim položajem u populaciji. Ako se pretpostavi da cijelu populaciju od $n$ jedinki sortiramo prema dobroti tako da na prvom mjestu bude jedinka s najvećom dobrotom, a na posljednjem jedinka s najmanjom dobrotom, tada se svakoj jedinki populacije može dodijeliti određen broj bodova s kojima će sudjelovati u odabiru. Prva (najbolja) jedinka dobit će $n$ bodova, druga $n - 1$ i tako dalje sve do posljednje jedinke koja će dobiti $1$ bod.

Omjer dodijeljenih bodova jedinkama u populaciji od $10$ članova prikazan je kružnim dijagramom na slici 4.1.


**Slika 4.1:** Podjela bodova prema rangu u populaciji

Može se vidjeti kako bi se odabirom slučajne točke na kružnici prikazanog dijagrama lako mogla odrediti jedinka kojoj točka pripada. Računski se isti postupak može provesti na način da se prema jednadžbi za sumu prvih $n$ prirodnih brojeva, a ona glasi:

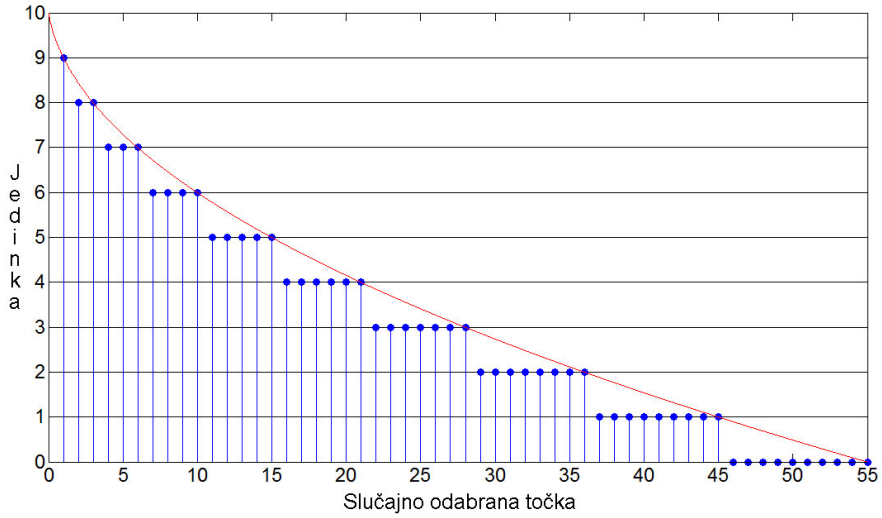
$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{4.1}$$

izračuna ukupna suma bodova za sve članove populacije. Ta suma predstavlja sve točke kružnice. Generatorom slučajnih brojeva određuje se jedna točka $x$ na kružnici. Jedino što još preostaje je odrediti kojoj jedinki točka pripada. To se izravno može učiniti uz pomoć formule:

$$i = n - \frac{(\sqrt{8 * x + 1} - 1)}{2} \tag{4.2}$$

koja za zadanu točku (slučajni broj $x$) i veličinu populacije $n$ vraća područje jedinke u kojem se točka nalazi (jedinke su sortirane od najbolje do najlošije). Preostaje još samo zaokružiti dobiveni iznos na manji cijeli broj. Graf funkcije s izračunatim područjima za populaciju od $10$ jedinki prikazan je na slici 4.2.

Na x-osi grafa prikazane su točke od kojih se jedna odabire slučajnim odabirom. Točaka ima onoliko koliko iznosi suma bodova dodijeljenih svim jedinkama u populaciji. Ucrtana funkcija na grafu određuje broj točaka koje će biti dodijeljene pojedinoj jedinci. Tako će se jedinci na poziciji $0$ dodijeliti $10$ točaka, jedinci na poziciji $1$ će se dodijeliti $9$ točaka itd. Što više točaka određuje pojedinu jedinku, veća je vjerojatnost da će slučajnim odabirom biti odabrana jedna od tih točaka, odnosno ta jedinka.



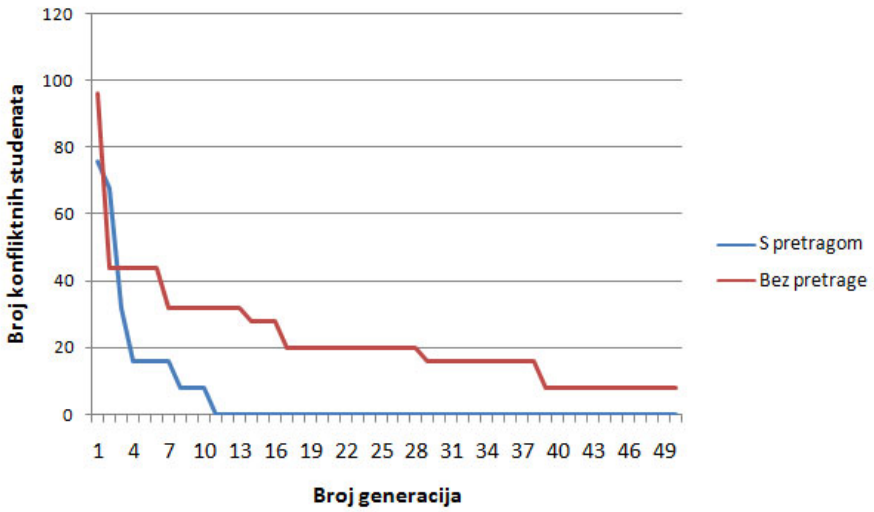

**Slika 4.2:** Graf funkcije selekcije za populaciju od 10 jedinki

Programsko rješenje koristi i poseban oblik selekcije zvan *elitistička selekcija*. Ona odabire najboljih $k$ članova populacije koji izravno prelaze u sljedeću populaciju. Time je osigurano očuvanje najbolje jedinke (maksimalna dobrota populacije ne pada), dok se istovremeno provode standardni operatori koji potiču raznolikost jedinki.

## 4.3. Lokalne pretrage

Za razliku od ostalih evolucijskih algoritama, genetski algoritam ima učinkovite mehanizme koji ga sprječavaju da se nađe u lokalnom optimumu bez mogućnosti oporavka. S druge strane, pokazalo se da genetski algoritam daje slabe rezultate pri finom podešavanju rješenja, odnosno da sporo konvergira. Iz tog razloga je Holland predložio da se genetski algoritam koristi kao predprocesor u procesu pretrage prostora prije nego što se pokrene glavni mehanizam pretrage — funkcije lokalne pretrage (prema Golub (2004)).

Funkcija lokalne pretrage pokušava rješenje dobiveno genetskim algoritmom dodatno poboljšati koristeći ljudska znanja o navedenom problemu. Slika 4.3 prikazuje postupak rada genetskog algoritma s istim parametrima, kroz jednak broj generacija i s uključenom, odnosno isključenom lokalnom pretragom. Na grafu je jasno vidljiv problem spore konvergencije genetskog algoritma i kako lokalna pretraga rješava taj problem. Ono što na grafu nije vidljivo je cijena lokalne pretrage u smislu procesorskog vremena koje ona iziskuje, o čemu je više napisano u sljedećem poglavlju.




**Slika 4.3:** Graf rada algoritma sa i bez funkcija lokalne pretrage

U ovom programskom rješenju ostvarene su dvije funkcije lokalne pretrage: pretraga po studentima i pretraga po terminima. S obzirom da su funkcije lokalne pretrage poprilično složene, ne izvršavaju se nad cijelom populacijom nego samo nad određenim brojem slučajno odabranih jedinki. Jedno pokretanje lokalne pretrage nad jednom jedinkom sastoji se od 10 pokušaja optimizacije jednom od dvije navedene metode, s time da se jedan pokušaj nastavlja sve dok je optimizacija uspješna.

### 4.3.1. Pretraga studenata

Lokalna pretraga studenata svodi se na pokušaje razmještaja konfliktnih studenata između termina. Algoritam odabire jedan slučajni termin i iz njega jednog slučajnog konfliktnog studenta. Zatim se za sve termine u koje student može biti smješten ispituje odgovaraju li studentu. Cilj funkcije je smanjiti broj konfliktnih studenata u jedinki. Pseudokod koji prikazuje postupak lokalne pretrage za jednog studenta prikazan je algoritmom 4.5.

**Algoritam 4.5** Lokalna pretraga za jednog studenta

$STUD \leftarrow TERMIN.konfliktniStudenti$
**ako** ($STUD \neq \emptyset$) **tada**
  $S \leftarrow STUD[slucajno(STUD.duljina)]$
  **za** ($i = 0; i < DOG.termini.duljina; i++$) **radi**
    **ako** ($\neg studentRezerviran(S, DOG.temini[i].vrijeme)$) **tada**
      $TERMIN.oslobodiStudenta(S)$
      $DOG.temini[i].rezervirajStudenta(S)$
      **prekini**
    **završi ako**
  **završi za**
**završi ako**

### 4.3.2. Pretraga termina

Pretraga termina za jedan slučajno odabran termin pokušava naći bolje rješenje. To se obavlja tako da se od upravitelja rezervacijama zatraži lista svih slobodnih termina (kombinacija dvorane i vremena početka) koji zadovoljavaju parametre. Funkcija zatim za svaki ponuđeni termin ispituje je li bolji ili lošiji od trenutnog. Ukoliko se nađe bolji termin, stari termin se oslobađa, a svi studenti se premještaju u novi termin, kao što prikazuje algoritam 4.6.

Slično kao i kod odabira metode mutacije, pri odabiru funkcija lokalne pretrage treba uzeti u obzir koje se u kojim slučajevima mogu izvršavati nad jedinkom. Kako pretraga po studentima mijenja raspodjelu studenata među terminima, ona se neće moći koristiti kao mehanizam optimizacije događaja kod kojih su studenti određeni na razini termina. U tom slučaju jedinka će se optimizirati jedino funkcijom pretrage termina.

**Algoritam 4.6** Lokalna pretraga termina

$T \leftarrow ORG.dohvatiTermine(TERMIN.trajanje)$
$TDOB \leftarrow TERMIN.dobrota$
$NAJDOB \leftarrow TDOB$
$NAJTER \leftarrow \emptyset$
**za** $(i = 0; i < T.duljina; i++)$ **radi**
  $PRIVDOB \leftarrow T_i.dobrota$
  **ako** $(PRVDOB < NAJDOB)$ **tada**
    $NAJDOB \leftarrow PRIVDOB$
    $NAJTER \leftarrow T_i$
  **završi ako**
**završi za**
**ako** $(NAJDOB < TDOB)$ **tada**
  $S \leftarrow TERMIN.oslobodiStudente()$
  $TERMIN.oslobodiTermin()$
  $NAJTER.rezervirajTermin(TERMIN.parametri)$
  $NAJTER.rezervirajStudente(S)$
**završi ako**

## 4.4. Algoritam

Nakon svih opisanih osnovnih ideja i pojmova vezanih uz ostvarenje, algoritam 4.7 prikazuje pseudokod genetskog algoritma koji se koristi za rješavanje problema izrade rasporeda nadoknada. Pseudokod prikazan navedenim algoritmom je radnja koju vrši metoda `step()`, odnosno predstavlja jedan korak algoritma.

Varijable u algoritmu predstavljaju sljedeće parametre:

- $n$ — veličina populacije,
- $j$ — vjerojatnost mutacije,
- $k$ — broj *elitističkih* jedinki i
- $l$ — postotak lokalne pretrage.

**Algoritam 4.7** Programsko ostvarenje genetskog algoritma

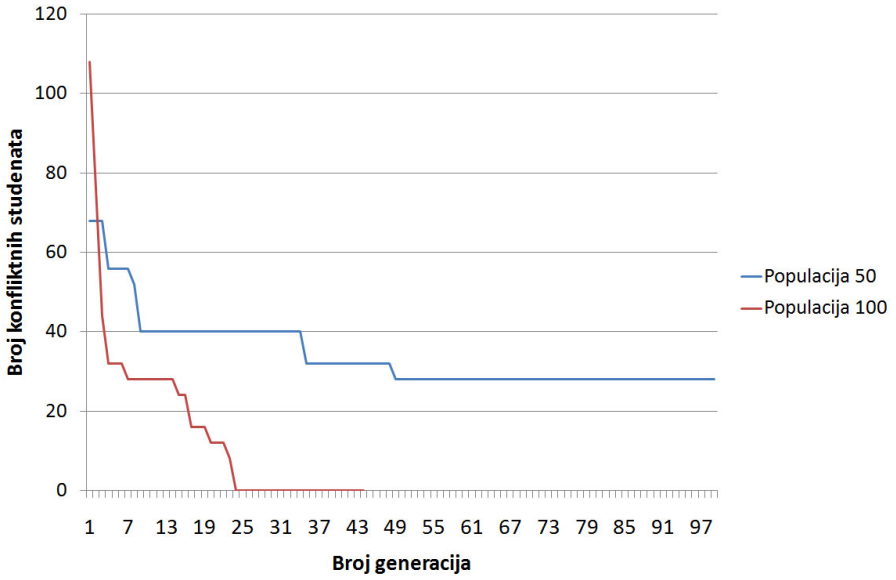
Pripremi memorijsku lokaciju za novu populaciju
Najboljih $k$ članova stare populacije izravno prenesi u novu
**za** $(i = 0; i < n - k; i++)$ **radi**
    Operatorom selekcije opisanom u poglavlju 4.2.3. odaberi dvije jedinke iz stare populacije za roditelje
    Stvori novu jedinku na temelju genetskog materijala roditelja i dodaj je u novu populaciju
**završi za**
Izvrši lokalnu pretragu nad $n * l$ slučajno odabranih jedinki nove populacije
Mutiraj $n * j$ slučajno odabranih jedinki nove populacije
    – s time da se *elitističke* jedinke ne mutiraju
Sortiraj jedinke unutar nove populacije
    – tako da je na prvom mjestu najbolja jedinka, a na posljednjem najlošija
Zamijeni staru populaciju novom

# 5. Utjecaj parametara na rad algoritma

U ovom poglavlju opisana je analiza utjecaja parametara na rad algoritma. Navedeni su parametri koji su ispitivani u testnim primjerima i opisani su zadaci koji su korišteni za testiranje. Prikazani su i analizirani rezultati testova.

## 5.1. Testni parametri

Rad genetskog algoritma općenito, pa tako i programskog ostvarenja izrade rasporeda nadoknada, podložan je mnogim parametrima. Neki od parametara spomenutih u prethodnim poglavljima su: veličina populacije, postotak mutacije, korištenje funkcija lokalne pretrage, broj pokušaja optimizacije funkcijama lokalne pretrage, broj *elitističkih* jedinki i mnogi drugi.




**Slika 5.1:** Rad algoritma u jednakim uvjetima za različite veličine populacije

Na slici 5.1 prikazan je graf rada algoritma u ovisnosti o veličini populacija. U oba slučaja svi ostali parametri su jednaki i rješava se isti problem u jednakom vremenu. Na grafu je zbog jednostavnosti prikazana samo jedna komponenta vektora dobrote jedinke (broj konfliktnih studenata). Može se primijetiti kako je algoritam s duplo većom populacijom u istom vremenu uspio izmijeniti upola manje generacija, ali je već u trećoj izmjeni „pretekao“ algoritam koji je radio s manjom veličinom populacije. Uz to, algoritam s većom populacijom je dosta prije isteka vremena uspio svesti broj konfliktnih studenata na $0$, dok onome s manjom to nije uspjelo.

U sklopu ovog rada provedena su dva kruga testiranja utjecaja parametara na rad algoritma. U prvom krugu testiran je utjecaj sljedećih četiriju parametra:

- korištenje lokalne pretrage po studentima (s i bez),
- korištenje lokalne pretrage po terminima (s i bez),
- veličina populacije ($20$, $50$, $100$, $200$ i $500$ jedinki) i
- vjerojatnost mutacije ($0.3$, $0.5$, $0.7$, $0.8$, $0.9$).

Testne primjere čine sve kombinacije navedenih parametara, njih ukupno $100$, a tablica svih primjera nalazi se u dodatku A.

Najbolja dobivena kombinacija parametara iz prvog kruga korištena je kao osnova za drugi krug. U drugom krugu testirana je ovisnost kvalitete rješenja o veličini populacije. Veličine populacija korištene u testiranjima sastoje se od: $50$, $100$, $150$, $200$, $250$, $300$, $350$, $400$, $450$ i $500$ jedinki.

## 5.2. Provođenje pokusa

Prvi krug testova sastoji se od serije koju čini ukupno $3000$ pokusa, što znači da je za svaki testni primjer algoritam pokretan $30$ puta. Osim parametara testnih primjera, ostali parametri algoritma jednaki su za sva pokretanja te time ne mogu utjecati na međusobni odnos rezultata. Za jedno pokretanje algoritmu je dodijeljeno $10$ minuta, nakon čega je prekinut njegov rad i zabilježeno najbolje rješenje. Algoritam pri svakom pokretanju ponovno priprema strukture podataka, tako da su rezultati u potpunosti neovisni.

Testni zadatak koji se rješava u svim primjerima prvog kruga sastavljen je od $4$ događaja, od kojih je za prva dva događaja određen vremenski raspon od $5$ dana, a za druga dva $4$ dana. Svaki od događaja ima slučajnu razdiobu termina u koje treba rasporediti ukupno $674$ studenta. Jasno je kako ovakav slučaj ne predstavlja realan problem izrade nadoknada (nadoknade se inače organiziraju za puno manji broj studenata),

ali je izabran zbog bolje raspodjele rezultata i lakše analize. Datoteka s parametrima rasporeda priložena je u dodatku B.

Za drugi krug testova korišteni su zadaci temeljeni na zadatku iz prvog kruga, s jedinim razlikama u broju studenata i veličini raspoloživih dvorana. Testirano je rješavanje problema koji se sastoje od $100$, $150$, $200$, $250$, $300$, $350$, $400$, $450$, $500$, $550$, $600$ i $650$ studenata. Veličine dvorana su proporcionalno smanjene u odnosu na smanjenje broja studenata u svakom pojedinom zadatku drugog kruga.

Drugi krug testova čini ukupno $3600$ eksperimenata. Kao i u prvom krugu, svaki testni primjer pokretan je $30$ puta, a vrijeme trajanja jednog pokusa je $2$ minute.

## 5.3. Rezultati

U sljedećim odlomcima navedeni su i komentirani rezultati provedenih mjerenja. Dobiveni rezultati prikazani su tablicama i grafovima.

### 5.3.1. Prvi krug testova

Po završetku prvog kruga testova, dobiveni rezultati grupirani su prema testnim primjerima kako bi se za svaki primjer analizirala aritmetička sredina i standardna devijacija. Graf aritmetičkih sredina i devijacija za svaki testni primjer prikazan je slikom 5.4, dok su detaljni podaci za najboljih 20 rezultata prikazani tablicom 5.1.

**Tablica 5.1:** Detaljni podaci o najboljih 20 rezultata prvog kruga testiranja

| | | Aritmetička sredina | | | | Standardna devijacija | | | |
|---|---|---|---|---|---|---|---|---|---|
| Rang | Test | Kon. st. | Pre. dv. | Br. t. | Br. sl. m. | Kon. st. | Pre. dv. | Br. t. | Br. sl. m. |
| 1 | 42 | 14,93 | 0,23 | 13,23 | 347,4 | 10,08 | 0,76 | 0,99 | 127,39 |
| 2 | 41 | 16 | 0 | 13,17 | 343,6 | 10,37 | 0 | 1,07 | 118,56 |
| 3 | 38 | 16,87 | 0 | 13 | 296,8 | 12,65 | 0 | 1,1 | 125,99 |
| 4 | 92 | 16,93 | 0 | 13,1 | 331,27 | 10,59 | 0 | 1,11 | 126,32 |
| 5 | 37 | 17,07 | 0,03 | 12,97 | 303,53 | 15,61 | 0,18 | 0,98 | 141,99 |
| 6 | 43 | 17,6 | 0,03 | 13,03 | 346,4 | 11,86 | 0,18 | 1,08 | 144,04 |
| 7 | 89 | 17,8 | 0,03 | 12,8 | 288,33 | 13,37 | 0,18 | 1,08 | 128,92 |
| 8 | 40 | 17,8 | 0,13 | 13,1 | 318,6 | 12 | 0,56 | 0,98 | 137,03 |
| 9 | 86 | 18,27 | 0,13 | 13,23 | 333,3 | 10,91 | 0,56 | 1,23 | 150,71 |
| 10 | 88 | 19,27 | 0 | 12,87 | 302 | 14,77 | 0 | 1,12 | 141,8 |
| 11 | 44 | 19,53 | 0,1 | 12,83 | 292,53 | 9,93 | 0,54 | 0,78 | 125,56 |
| 12 | 39 | 21,2 | 0 | 12,63 | 258,27 | 16,64 | 0 | 0,84 | 122,63 |
| 13 | 90 | 21,53 | 0 | 12,77 | 270,6 | 16,41 | 0 | 0,76 | 121,19 |
| 14 | 14 | 21,6 | 0,1 | 12,23 | 273,33 | 14,32 | 0,54 | 0,42 | 85,69 |
| 15 | 87 | 21,87 | 0,03 | 13,2 | 299,07 | 17,54 | 0,18 | 1,28 | 153,81 |
| 16 | 91 | 22,47 | 0 | 13,07 | 336,93 | 14,66 | 0 | 0,93 | 136,43 |
| 17 | 82 | 22,67 | 0,03 | 13 | 290,33 | 17,72 | 0,18 | 0,97 | 115,08 |
| 18 | 36 | 23,27 | 0 | 13,07 | 300,93 | 15,8 | 0 | 0,96 | 134,7 |
| 19 | 34 | 23,33 | 0 | 12,87 | 250,2 | 15,99 | 0 | 0,72 | 103,23 |
| 20 | 18 | 23,47 | 0 | 12,27 | 284,8 | 12,89 | 0 | 0,51 | 89,83 |

Niti jedno rješenje nije imalo nezadovoljenih preduvjeta niti konfliktnih dvorana, pa su te dvije komponente vektora dobrote u daljnjoj analizi rezultata zanemarene. Ostale komponente navedene su u tablici i to po sljedećim stupcima:

- *Kon. st.* — broj konfliktnih studenata,
- *Pre. dv.* — broj prenapučenih dvorana,
- *Br. t.* — broj termina i
- *Br. sl. m.* — broj slobodnih mjesta u dvoranama.

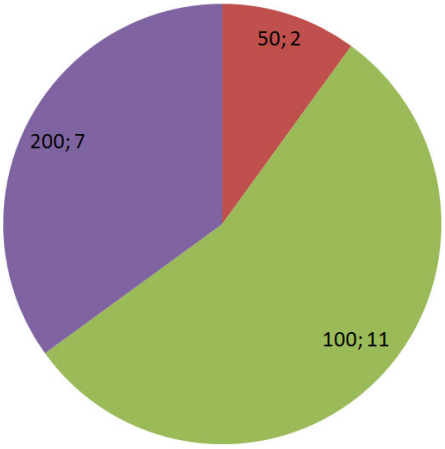
Zbog više komponenti od kojih se sastoji rješenje, aritmetičku sredinu i standardnu devijaciju grupe testova nije moguće prikazati jednim brojem, nego su one izračunate posebno za svaku komponentu. Zato su i u tablici stupci grupirani na one koji pripadaju aritmetičkoj sredini i one koji se odnose na standardnu devijaciju.

Iz tablice je vidljivo da su aritmetičke sredine prvih $20$ rezultata poprilično slične a standardne devijacije osciliraju s većim vrijednostima, pa se osim prvog rezultata koji ima obje najniže vrijednosti i time se opravdano može nazvati najboljim rješenjem, ostale rezultate ne bi trebalo čvrsto razdvajati prema rangu.

Rezultati pokusa pokazuju kako se najbolja rješenja postižu koristeći parametre iz bloka testnih primjera $26$ – $50$. Zajedničko obilježje tih testnih primjera je korištenje lokalne pretrage po terminima uz isključenu pretragu po studentima. Ni jedan rezultat iz bloka $51$ – $75$, gdje se koristi samo pretraga po studentima, nije ušao među najboljih $20$. Dapače, kao što se može vidjeti iz grafa na slici 5.4, upravo se u ovom bloku nalaze najlošija rješenja. Izostanak dobrih rezultata uz korištenje pretrage po studentima pokazuje kako funkcija pretrage nije dovoljno uspješna da bi značajnije pridonijela rezultatu. Štoviše, najbolji rezultati uz korištenje samo pretrage po terminima ukazuju kako je cijena pretrage po studentima previsoka s obzirom na njezinu učinkovitost. To bi se moglo objasniti prevelikim brojem operacija koje funkcija mora izvršiti da bi optimizirala stanje samo jednog studenta. S druge strane, pretraga po terminima odjednom optimira stanja više studenata, što je pokazalo pozitivan učinak.

U kontekstu veličine populacije, na slici 5.2 prikazana je blaga dominacija rezultata s veličinom populacije od $100$ jedinki naprema onima od $200$, iako je najbolje rješenje temeljeno na populaciji od $200$ jedinki. Stoga je moguće pretpostaviti kako bi se još bolji rezultati postigli veličinom populacije od oko $150$ jedinki. Rezultati druge serije pokusa će pokazati utječe li opseg danog problema (broj studenata koje je potrebno organizirati) na optimalnu veličinu populacije.

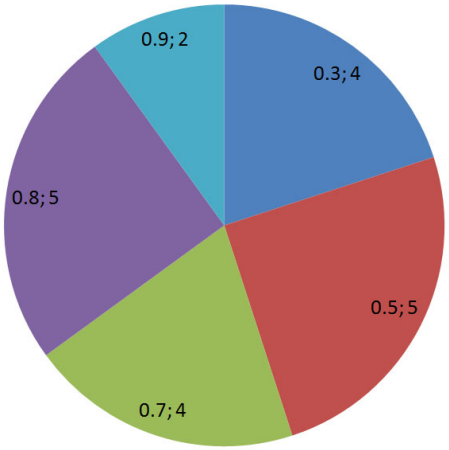
Svega dva rezultata su dobivena populacijama od $50$ jedinki, a ni jedno rješenje s $20$ jedinki se ne nalazi među prvih $20$, što ukazuje na činjenicu kako takve populacije



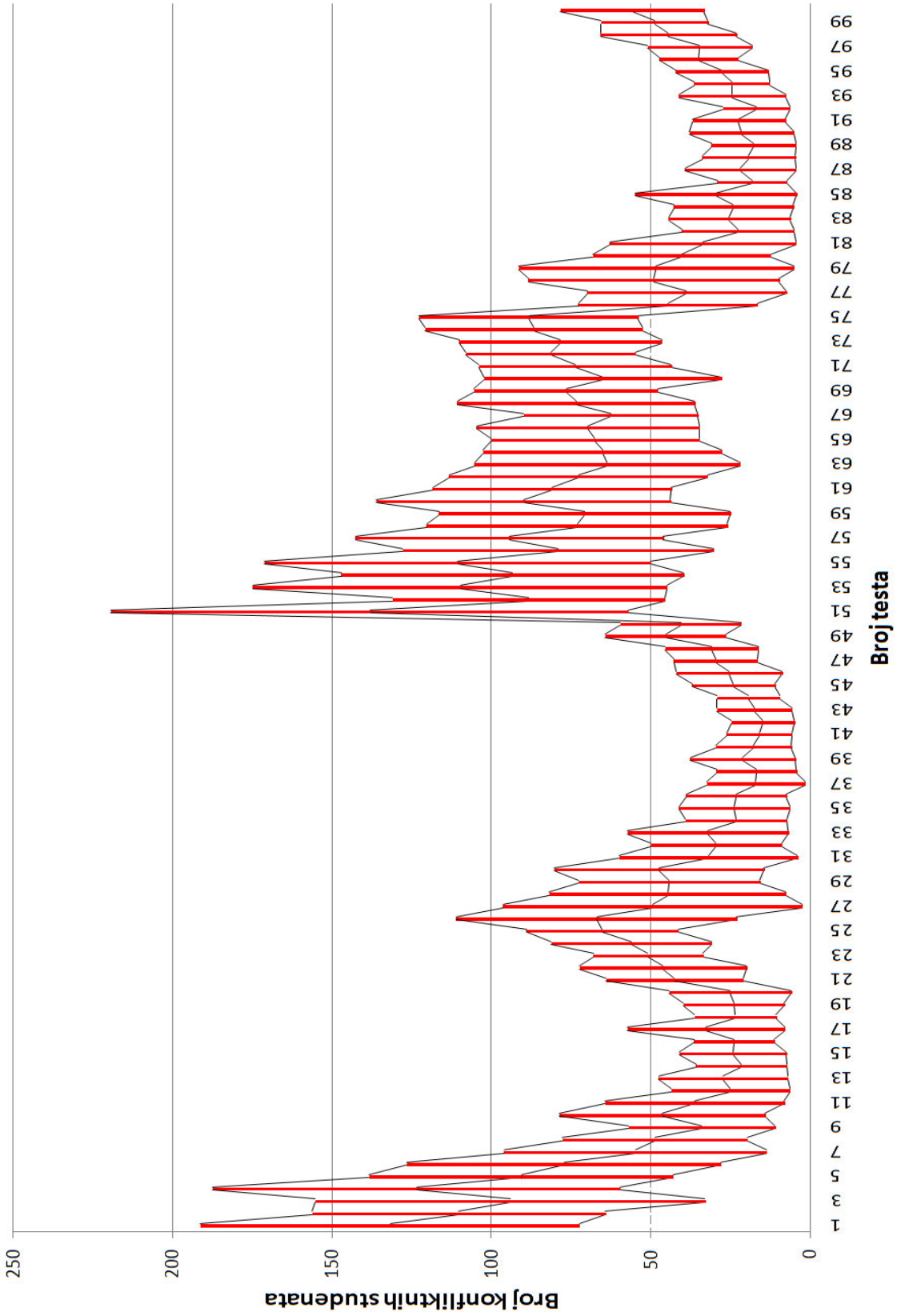

**Slika 5.2:** Udio rješenja s određenim veličinama populacije među najboljih 20

nude premalu raznolikost i mogućnost paralelnog pretraživanja. Zanimljiv je i podatak kako u najboljim rješenjima nema ni jednog temeljenog na populaciji od 500 jedinki, iz čega se da zaključiti kako ne treba pretjerati s veličinom populacije. Veća populacija iziskuje više procesorskog vremena u jednom koraku te se populacije rjeđe izmjenjuju, čime se umanjuje utjecaj križanja.

O utjecaju postotka mutiranih jedinki u populaciji rezultati nažalost, kao što se vidi na slici 5.3, ne govore puno. Iz ovih podataka može se jedino zaključiti kako bi se o ovom parametru trebalo provesti još nekoliko testova na problemima različite složenosti, što bi možda moglo ukazati na povezanost postotka mutiranih jedinki i broja studenata u problemu. Osim toga, trebao bi se ostvariti mehanizam prilagodbe jačine mutacije pa testirati ovisnost rješenja o tom parametru.




**Slika 5.3:** Udio rješenja s određenim postocima mutacije među najboljih 20




**Slika 5.4:** Rezultati prvog kruga testova (aritmetička sredina i standardna devijacija broja konfliktnih studenata)

### 5.3.2. Drugi krug testova

Rezultati drugog kruga testova prikazani su tablicom 5.2 i grafom 5.5. Grupirani su prema složenosti zadatka koji se rješavao (određen brojem studenata koje je potrebno razmjestiti u termine). U tablici su zbog detaljnije analize navedene veličine populacija dva najbolja rješenja.

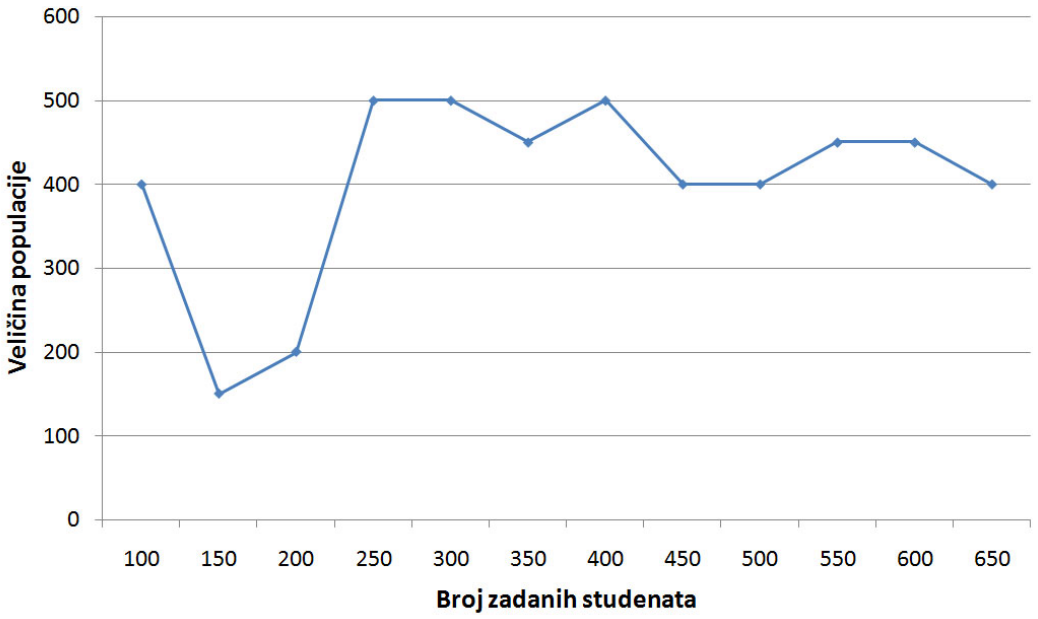
**Tablica 5.2:** Ovisnost veličine populacije o opsegu zadanog problema

| | Veličina populacije | |
|---|---|---|
| Broj zadanih studenata | Prvo najbolje rješenje | Drugo najbolje rješenje |
| 100 | 400 | 250 |
| 150 | 150 | 300 |
| 200 | 200 | 400 |
| 250 | 500 | 200 |
| 300 | 500 | 350 |
| 350 | 450 | 400 |
| 400 | 500 | 350 |
| 450 | 400 | 500 |
| 500 | 400 | 500 |
| 550 | 450 | 500 |
| 600 | 450 | 500 |
| 650 | 400 | 500 |

Kao što je i vidljivo iz tablice, rezultati ne pokazuju značajnu povezanost između opsega problema i veličine populacije. Posebno je zanimljiv slučaj međusobno sličnih problema s $200$ i $250$ studenata, gdje veličine populacija najboljih rješenja iznose $200$ odnosno $500$ jedinki. Na temelju navedenih rezultata moguće je jedino zaključiti kako su za probleme s više od $300$ studenata pogodne populacije s $350$ ili više jedinki. U području problema sa $100$ do $300$ studenata razultati poprilično osciliraju i ne može se izvući sličan zaključak.

Uzrok ovakvim rezultatima mogli bi biti neodgovarajući testni primjeri. Kako svi testni primjeri imaju jednak omjer kapaciteta raspoloživih dvorana i broja studenata za rasporediti, moguće je da takav skup ne daje dovoljnu raznolikost koja bi pokazala određenu ovisnost. U svakom slučaju, prije donošenja bilo kakvog zaključka vezanog uz ovisnost veličine populacije o opsegu zadanog problema, bilo bi dobro ponoviti

navedeno testiranje s većim brojem raznovrsnih problema. Uz to, trebalo bi ispitati i ovisnost veličine populacije o omjeru kapaciteta dvorana i broja studenata. Osim u sklopu testiranja, moguće je razviti sustav koji će imati mogućnost analize rezultata i prilagodbe veličine populacije algoritma pri stvarnom korištenju.




**Slika 5.5:** Ovisnost veličine populacije o opsegu zadanog problema (najbolja rješenja)

# 6. Zaključak

Tehnologija svakim danom sve više napreduje, računala postaju sve brža, ali još uvijek postoje problemi koji su toliko složeni da ih niti današnja najbrža računala ne mogu riješiti u zadovoljavajućem vremenskom periodu. Složenost problema leži u ogromnom broju stanja od kojih se takvi problemi sastoje, pa pokušaj pretraživanja svih stanja „sirovom“ računalnom snagom, jednostavno nije moguć. Jedan od takvih problema je problem izrade i izmjene rasporeda, s kojim se na Fakultetu[1] svakodnevno susrećemo. Kako zbog velikog broja kombinacija nije moguće ispitati kvalitetu svakog mogućeg rasporeda i time odrediti najbolji, potrebno je pronaći manje računalno zahtjevan način kojim će se postići jednak ili približno jednak ishod.

Sredinom prošloga stoljeća znanstvenici su uvidjeli kako se simuliranje procesa biološke evolucije na računalu može koristiti kao mehanizam rješavanja optimizacijskih problema. To je predstavljalo početak razvoja evolucijskog računarstva. Jedna vrsta evolucijskih algoritama je i genetski algoritam.

U ovom radu je opisana primjena genetskog algoritma za rješavanje problema izrade rasporeda nadoknada na Fakultetu. Opisan je sustav koji kombinira rezultate nekoliko heuristika kako bi korisniku ponudio što bolje rješenje. Opisane su strukture podataka koje genetski algoritam koristi pri radu i postupci evolucijskih operatora kojima oblikuje populaciju rješenja. Naveden je nedostatak genetskog algoritma u obliku spore konvergencije prema optimalnom rješenju. Opisani su još uloga i konkretno ostvarenje funkcija lokalne pretrage.

U sklopu rada provedeno je istraživanje kako parametri algoritma utječu na njegov učinak. Istraživanje je provedeno u dva kruga. Prvi krug testova temelji se na $100$ testnih primjera s različitim vrijednostima promatranih parametara i svim ostalim jednakim uvjetima. Rezultati su pokazali kako funkcije lokalne pretrage zaista mogu pridonijeti bržoj konvergenciji algoritma, ali da iziskuju velike količine računalnih sredstava i da treba pronaći ravnotežu između složenosti pretrage i učinka na kvalitetu rješenja. Analiza utjecaja veličine populacije pokazala je kako manje populacije ne

[1] Fakultet elektrotehnike i računarstva

nude dovoljno raznolikosti u smislu nasljednog materijala važnog za nastavak evolucije. Drugi krug testova, temeljen na 120 testnih primjera, trebao je pokazati kako veličina populacije koja rezultira najboljim rješenjem ovisi o broju studenata koje je potrebno razmjestiti. Rezultati nisu pokazali takvu ovisnost, ali bi trebalo provesti još testiranja prije nego se zaključi da ona uopće ne postoji.

Osim poboljšanja funkcije lokalne pretrage studenata u cilju manje složenosti, moguće su i ostale promjene u kodu s ciljem bolje učinkovitosti. Jedna od mogućnosti je oblikovanje metoda mutacije na način da se pomoću parametra može mijenjati stupanj u kojem mutacija mijenja jedinku. Zatim bi se moglo ispitati ponašanje sustava i pronaći najbolji faktor mutacije. Dodatna nadogradnja takvog sustava je uvođenje postupka adaptivne mutacije (opisao Golub (2004)) koji bi mijenjao stupanj mutacije jedinke ovisno o stanju u kojem se algoritam nalazi.

U kontekstu nadogradnje cijelog sustava za izradu rasporeda nadoknada pojavila se ideja razvoja alata koji bi ovisno o složenosti problema određivao optimalne veličine populacija algoritama. Osim početnih podataka, takav sustav imao bi mogućnost samoprilagodbe na temelju rezultata pri redovnom korištenju.

# LITERATURA


Kenneth A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. University Microfilms International, 1975.

Kenneth A. De Jong. *Evolutionary computation: a unified approach*. The MIT Press, 2006.

L.J. Fogel, A.J. Owens, i M.J. Walsh. *Artificial intelligence through simulated evolution*. John Wiley & Sons Inc, 1966.

Tin Franović, Mislav Bobesić, Filip Boltužić, Denis Ćutić, Domagoj Kusalić, Pribil Siniša, i Vladimir Uzun. *FerSched, Tehnička dokumentacija programskog rješenja u sklopu predmeta Projekt*. Fakultet elektrotehnike i računarstva, Zagreb, 2009.

D.E. Goldberg. Genetic algorithms in search, optimization, and machine learning. *Addision Wesley, MA, USA*, 1989.

Marin Golub. *Genetski algoritam, prvi dio, skripta*. Fakultet elektrotehnike i računarstva, Zagreb, 2004.

J.H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314, 1962.

J.H. Holland. Nonlinear environments permitting efficient adaptation. U *Computer and information sciences, II: proceedings*. Academy Press, 1967.

JH Holland. Adaptation in natural and artificial systems. *Ann Arbor MI: University of Michigan Press*, 1975.

I. Rechenberg. Cybernetic solution path of an experimental problem. *Library translation*, 1122, 1965.

Marko Čupić. *Evolucijsko računanje i problem izrade rasporeda, rad za kvalifikacijski doktorski ispit*. Fakultet elektrotehnike i računarstva, Zagreb, 2009a.

Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi, skripta u sklopu predmeta Umjetna inteligencija*. Fakultet elektrotehnike i računarstva, Zagreb, 2009b.

S.P. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. U *Proc. XI Internat. Congr. Genetics*, svezak 1, stranice 356–366, 1932.

# Dodatak A
# Tablica testnih slučajeva prvog kruga testiranja

**Tablica A1:** Parametri zadani u pojedinim testnim slučajevima prvog kruga testiranja

| Test | Pretraga studenti | Pretraga termini | Populacija | Mutacija |
|---|---|---|---|---|
| 1 | ne | ne | 20 | 0.3 |
| 2 | ne | ne | 20 | 0.5 |
| 3 | ne | ne | 20 | 0.7 |
| 4 | ne | ne | 20 | 0.8 |
| 5 | ne | ne | 20 | 0.9 |
| 6 | ne | ne | 50 | 0.3 |
| 7 | ne | ne | 50 | 0.5 |
| 8 | ne | ne | 50 | 0.7 |
| 9 | ne | ne | 50 | 0.8 |
| 10 | ne | ne | 50 | 0.9 |
| 11 | ne | ne | 100 | 0.3 |
| 12 | ne | ne | 100 | 0.5 |
| 13 | ne | ne | 100 | 0.7 |
| 14 | ne | ne | 100 | 0.8 |
| 15 | ne | ne | 100 | 0.9 |
| 16 | ne | ne | 200 | 0.3 |
| 17 | ne | ne | 200 | 0.5 |
| 18 | ne | ne | 200 | 0.7 |
| 19 | ne | ne | 200 | 0.8 |

Nastavak na sljedećoj stranici...

Tablica A1 – Nastavak

| Test | Pretraga studenti | Pretraga termini | Populacija | Mutacija |
|---|---|---|---|---|
| 20 | ne | ne | 200 | 0.9 |
| 21 | ne | ne | 500 | 0.3 |
| 22 | ne | ne | 500 | 0.5 |
| 23 | ne | ne | 500 | 0.7 |
| 24 | ne | ne | 500 | 0.8 |
| 25 | ne | ne | 500 | 0.9 |
| 26 | ne | da | 20 | 0.3 |
| 27 | ne | da | 20 | 0.5 |
| 28 | ne | da | 20 | 0.7 |
| 29 | ne | da | 20 | 0.8 |
| 30 | ne | da | 20 | 0.9 |
| 31 | ne | da | 50 | 0.3 |
| 32 | ne | da | 50 | 0.5 |
| 33 | ne | da | 50 | 0.7 |
| 34 | ne | da | 50 | 0.8 |
| 35 | ne | da | 50 | 0.9 |
| 36 | ne | da | 100 | 0.3 |
| 37 | ne | da | 100 | 0.5 |
| 38 | ne | da | 100 | 0.7 |
| 39 | ne | da | 100 | 0.8 |
| 40 | ne | da | 100 | 0.9 |
| 41 | ne | da | 200 | 0.3 |
| 42 | ne | da | 200 | 0.5 |
| 43 | ne | da | 200 | 0.7 |
| 44 | ne | da | 200 | 0.8 |
| 45 | ne | da | 200 | 0.9 |
| 46 | ne | da | 500 | 0.3 |
| 47 | ne | da | 500 | 0.5 |
| 48 | ne | da | 500 | 0.7 |
| 49 | ne | da | 500 | 0.8 |
| 50 | ne | da | 500 | 0.9 |
| 51 | da | ne | 20 | 0.3 |

Nastavak na sljedećoj stranici...

Tablica A1 – Nastavak

| Test | Pretraga studenti | Pretraga termini | Populacija | Mutacija |
|---|---|---|---|---|
| 52 | da | ne | 20 | 0.5 |
| 53 | da | ne | 20 | 0.7 |
| 54 | da | ne | 20 | 0.8 |
| 55 | da | ne | 20 | 0.9 |
| 56 | da | ne | 50 | 0.3 |
| 57 | da | ne | 50 | 0.5 |
| 58 | da | ne | 50 | 0.7 |
| 59 | da | ne | 50 | 0.8 |
| 60 | da | ne | 50 | 0.9 |
| 61 | da | ne | 100 | 0.3 |
| 62 | da | ne | 100 | 0.5 |
| 63 | da | ne | 100 | 0.7 |
| 64 | da | ne | 100 | 0.8 |
| 65 | da | ne | 100 | 0.9 |
| 66 | da | ne | 200 | 0.3 |
| 67 | da | ne | 200 | 0.5 |
| 68 | da | ne | 200 | 0.7 |
| 69 | da | ne | 200 | 0.8 |
| 70 | da | ne | 200 | 0.9 |
| 71 | da | ne | 500 | 0.3 |
| 72 | da | ne | 500 | 0.5 |
| 73 | da | ne | 500 | 0.7 |
| 74 | da | ne | 500 | 0.8 |
| 75 | da | ne | 500 | 0.9 |
| 76 | da | da | 20 | 0.3 |
| 77 | da | da | 20 | 0.5 |
| 78 | da | da | 20 | 0.7 |
| 79 | da | da | 20 | 0.8 |
| 80 | da | da | 20 | 0.9 |
| 81 | da | da | 50 | 0.3 |
| 82 | da | da | 50 | 0.5 |
| 83 | da | da | 50 | 0.7 |

Nastavak na sljedećoj stranici...

Tablica A1 – Nastavak

| Test | Pretraga studenti | Pretraga termini | Populacija | Mutacija |
|---|---|---|---|---|
| 84 | da | da | 50 | 0.8 |
| 85 | da | da | 50 | 0.9 |
| 86 | da | da | 100 | 0.3 |
| 87 | da | da | 100 | 0.5 |
| 88 | da | da | 100 | 0.7 |
| 89 | da | da | 100 | 0.8 |
| 90 | da | da | 100 | 0.9 |
| 91 | da | da | 200 | 0.3 |
| 92 | da | da | 200 | 0.5 |
| 93 | da | da | 200 | 0.7 |
| 94 | da | da | 200 | 0.8 |
| 95 | da | da | 200 | 0.9 |
| 96 | da | da | 500 | 0.3 |
| 97 | da | da | 500 | 0.5 |
| 98 | da | da | 500 | 0.7 |
| 99 | da | da | 500 | 0.8 |
| 100 | da | da | 500 | 0.9 |

# Dodatak B
# Parametri testnog problema

**Okvir B.1:** Parametri rasporeda korišteni u pokusima

```
<plan name="MATLABraspored">
  <event name="Predavanje1a" id="E3.1" termDuration="60">
    <distribution>
      <type>RANDOM</type>
      <min>1</min>
      <max>4</max>
    </distribution>
    <def>
      <people>
        <groups>Grupe za predavanja#270#0</groups>
      </people>
      <time>2009-11-02 08:00#2009-11-06 20:00</time>
      <rooms>B1$FER/B1$180,B4$FER/B4$180,D1$FER/D1$252,
        D2$FER/D2$252</rooms>
    </def>
  </event>
  <event name="Predavanje1b" id="E3.2" termDuration="60">
    <distribution>
      <type>RANDOM</type>
      <min>1</min>
      <max>4</max>
    </distribution>
    <def>
      <people>
        <groups>Grupe za predavanja#270#0</groups>
      </people>
```

```
      <time>2009-11-02 08:00#2009-11-06 20:00</time>
      <rooms>B1$FER/B1$180,B4$FER/B4$180,D1$FER/D1$252,
        D2$FER/D2$252</rooms>
    </def>
  </event>
  <event name="Predavanje2a" id="E3.3" termDuration="60">
    <distribution>
      <type>RANDOM</type>
      <min>1</min>
      <max>4</max>
    </distribution>
    <def>
      <people>
        <groups>Grupe za predavanja#270#0</groups>
      </people>
      <time>2009-11-16 08:00#2009-11-19 20:00</time>
      <rooms>D1$FER/D1$252,D2$FER/D2$252,B4$FER/B4$180,
        B1$FER/B1$180</rooms>
    </def>
  </event>
  <event name="Predavanje2b" id="E3.4" termDuration="60">
    <distribution>
      <type>RANDOM</type>
      <min>1</min>
      <max>4</max>
    </distribution>
    <def>
      <people>
        <groups>Grupe za predavanja#270#0</groups>
      </people>
      <time>2009-11-16 08:00#2009-11-19 20:00</time>
      <rooms>B1$FER/B1$180,B4$FER/B4$180,D1$FER/D1$252,
        D2$FER/D2$252</rooms>
    </def>
  </event>
</plan>
```

## Rješavanje problema izrade rasporeda nadoknada primjenom genetskog algoritma


### Sažetak

Izrada rasporeda iznimno je složen kombinatorički problem kojeg nije moguće riješiti pretraživanjem cijelog prostora stanja. Genetski algoritam koristi metode biološke evolucije kako bi u relativno kratkom vremenu pronašao približno optimalno rješenje, koje je najčešće zadovoljavajuće. U ovom radu opisan je način rada evolucijskih algoritama, prednosti i nedostaci genetskog algoritma te načini na koje se on može unaprijediti. Razvijen je programski sustav koji pomoću genetskog algoritma izrađuje raspored nadoknada i provedeno je testiranje utjecaja parametara algoritma na njegov učinak.



**Ključne riječi:** genetski algoritam, problem izrade rasporeda, GA, evolucijski algoritmi, lokalna pretraga, križanje, mutacija


## Solving the problem of compensation schedules using genetic algorithms


### Abstract

The scheduling problem is a highly complex combinatorial problem which cannot be solved by searching the entire state space. The genetic algorithm uses methods that mimic biological evolution to find a solution close to an optimal one (and, as such, usually a satisfactory one) in a relatively short amount of time. This paper describes the principles of evolutionary algorithms, the advantages and disadvantages of the genetic algorithm, and the ways in which it can be improved. A software system which uses the genetic algorithm to solve the problem of generating compensation schedules is developed. Experiments are performed to study the influence of parameters on the algorithm's performance.



**Keywords:** genetic algorithm, scheduling problem, GA, evolutionary algorithms, local search, crossover, mutation