SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Podešavanje i testiranje parametara evolucijskih algoritama kod problema rasporeda međuispita

Đorđe Grbić
Voditelj: Doc. dr. sc. Domagoj Jakobović

Zagreb, svibanj, 2010.

Sadržaj

1. Uvod

Na Fakultetu elektrotehnike i računarstva ima preko 3000 studenata smještenih u preko 125 kolegija. Za gotovo svaki od kolegija se održavaju međuispiti u tri ciklusa. Svaki ciklus međuispita traje približno deset dana smještenih u dva tjedna. Međuispiti moraju biti tako posloženi da student može prisustvovati svim međuispitima kolegija koje je upisao. Za prethodno ograničenje ali i još neka koja će biti spomenuta kasnije mora se definirati problem rasporeda međuispita.

Problem rasporeda međuispita je vrlo čest na sveučilištima i smatra se NP-potpunim problemom [1]. Općenito, problem opisuje slaganje međuispita u termine i prostorije. U ovom radu će biti opisano samo slaganje međuispita u termine. Raspored će biti ocjenjivan po svom zadovoljavanju ograničenja koja su mu zadana, a zadane su dvije skupine ograničenja. Čvrsta ograničenja (engl. *Hard constraints*) - ograničenja koja moraju biti u svakom trenutku zadovoljena ako želimo smatrati raspored zadovoljivim. Meka ograničenja (engl. *Soft constraints*) - ograničenja koja ne moraju nužno biti zadovoljena da bi se raspored smatrao zadovoljivim ali je bolji onaj raspored koji više udovoljava tim ograničenjima.

Isprva se ovaj problem rješavao ručno, no kako je rastao broj studenata i kolegija na Bolonjskom procesu na FER-u ovaj zadatak je postalo nemoguće rješavati na taj način. Raniji pokušaji rješavanja problema rasporeda su se temeljili na heurističkim algoritmima. Jedna skupina su algoritmi koji rješavaju problem bojanja grafa. Takvi algoritmi su se pokazali učinkovitima samo za probleme s malim brojem studenata i kolegija [2].

Za stvaranje upotrebljivih rasporeda korišteni su manje egzaktni algoritmi, tzv. metaheuristički algoritmi. Metaheuristički algoritmi su skupina generičkih algoritama koji iterativno poboljšavaju kandidate za rješenje problema. Koriste se u slučajevima kada su jedine egzaktne metode za pronalaženje optimalnog rješenja nekog problema temeljene na iscrpnoj pretrazi. Velik dio skupine metaheurističkih algoritama čine algoritmi evolucijskog računarstva. Korištenje

metaheurističkih algoritama se pokazalo vrlo učinkovitim za probleme s velikim brojem studenata i kolegija. U ovom radu će biti opisana tri algoritma iz skupine evolucijskih algoritama: Genetski algoritam (engl. *Genetic algorithm*) [8], Jednostavni imunološki algoritam (engl. *Simple Immunological Algorithm, SIA*) [7] i Algoritam harmonijske pretrage (engl. *Harmony search algorithm*) [9]. Bit će egzaktno definiran problem generiranja rasporeda, detaljno opisan rad svakog od algoritama, navedeni eventualni nedostatci i prednosti svakog od algoritama i statistički obrađeni pokusi nad parametrima algoritama.

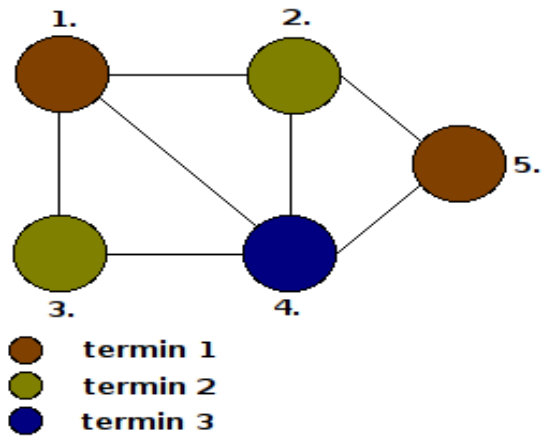
2. Definiranje problema rasporeda međuispita

Problem rasporeda međuispita (engl. *Examination Scheduling Problem*) je vrlo sličan kao Problem rasporeda kolegija (engl. *Course Scheduling Problem*) [3]. Problem se javlja kad obrazovna ustanova nudi učenicima određenu slobodu kod izbora kolegija koje će slušati. Svaki student izabire određen broj iz skupa njemu dostupnih kolegija.

Svakom kolegiju mora biti dodijeljen termin u kojem će se održati međuispit. Bilo bi vrlo nezgodno za studenta i nastavnike kad bi student imao dva međuispita u isto vrijeme. Tako se dolazi do prvog zahtjeva koji raspored mora zadovoljiti (čvrsto ograničenje). Definiran je kao: “Ne postoji student koji ima dva međuispita u istom terminu” tj. “Za sve kolegije koji dijele isti termin međuispita, presjek studenata koji pohađaju te kolegije mora biti prazan skup”.

Radi jednostavnosti će u nastavku teksta kolegiji biti izjednačeni s njihovim međuispitima jer svaki kolegij održava samo jedan međuispit u jednom ciklusu. Svi termini su dva sata dugački i nema preklapanja termina tj. termini su neovisni jedan o drugom. Problem s ovim ograničenjima se može prikazati pomoću grafa s obojanim čvorovima [3] gdje susjedni čvorovi ne smiju biti obojani istom bojom. Svaki kolegij je prikazan pomoću čvora grafa, povezani su oni čvorovi (kolegiji) koji međusobno dijele studente. Termini u kojima mogu biti međuispiti su su prikazani bojom čvorova. Na slici 2.1. su prikazani kolegiji pomoću čvorova grafa, a termini bojama kojima su čvorovi obojani. Čvorovi koji su međusobno susjedni dijele studente koji ih pohađaju.

Sljedeće ograničenje je da neki kolegiji mogu biti samo u određenim, definiranim terminima koji su pravi podskup svih termina. To znači da neki čvorovi grafa mogu poprimiti samo neke boje.



Slika2.1.

Prikaz rasporeda međuispita pomoću obojanog grafa. Susjednost čvorova znači da postoje studenti koji su u oba kolegija.

Treće jako ograničenje kaže da zbroj studenata sa svih međuispita koji se održavaju u jednom terminu ne smije premašiti kapacitet tog termina. Fakultet u jednom terminu ima ograničen broj prostorija u kojima se međuispiti mogu održavati, pa zbroj kapaciteta svih prostorija daje kapacitet termina.

Neki kolegiji dolaze u grupama, što znači da se međuispiti iz kolegija koji pripadaju jednoj grupi moraju održati u istom terminu. To daje zadnje jako ograničenje. Sva jaka ograničenja prikazana su u tablici 2.1**.**

Oznaka ograničenja	Opis čvrstog ograničenja
H1	Isti student ne smije imati dva ispita istovremeno
H2	Kolegij smije biti samo u dopuštenom, definiranom terminu
H3	U jednom terminu ne smije biti studenata više od kapaciteta termina
H4	Grupe kolegija imaju međuispit u isto vrijeme

Tablica 2.1.

Popis svih jakih ograničenja

Slaba ograničenja raspored ne mora nužno ispunjavati da bi bio provodiv u stvarnosti ali razlikovati će se oni rasporedi koji više udovoljavaju slabim ograničenjima kao kvalitetniji od onih koji im manje udovoljavaju. Ta ograničenja prvenstveno služe da bi se raspored optimirao tako da stvara što manje opterećenje što većem broju studenata. Uveden je pojam “kazna” koji dodjeljuje numeričku vrijednost rasporedu kao mjeru kršenja slabih ograničenja. Postoje tri slaba ograničenja s različitim težinama kazne.

Prvo ograničenje je minimizirati broj studenata koji imaju više od jednog ispita u istom danu. Ovo ograničenje ima najveću težinu kazne među mekim ograničenjima.

Ograničenje je opravdano time što bi slaganje rasporeda, bez ovog ograničenja, moglo jednoj skupini studenata staviti preveliki razmak između međuispita, dok bi drugima moglo staviti više ispita u jedan dan. Takav razvoj događaja bi bio vrlo nepogodan ali nije sigurno je li moguće napraviti raspored u kojem nitko nema više ispita u istom danu. Također, relativno mali broj studenata bi i mogao imati više ispita u jednom danu radi optimalnog iskorištavanja fakultetskih resursa.

Sljedeće ograničenje je vrlo slično kao prošlo s razlikom da se u ovom slučaju pokušava što većem broju studenata omogućiti minimalno jedan dan odmora između dva ispita. Ovo ograničenje ima manju težinu od prošlog zbog toga što nepoštivanje ne čini raspored mnogo lošijim. Također, kazna za više ispita u istom danu istom studentu mora doći više do izražaja nego kazna za dva ispita dan za danom istom studentu. Tijek algoritma računanja kazne je prikazan na slici 2.2. Važno je napomenuti da je kazna u ovom slučaju obrnuto proporcionalna dobroti. Veličine faktorSusjedniDan(c) i faktorIstiDan(c) ovise o veličinama definiranim u konfiguracijskoj datoteci.

Oznaka ograničenja	Opis oganičenja
S1	Što manje studenata ima ispit isti dan
S2	Što manje studenata ima ispit dan za danom

Tablica 2.2.
Popis svih slabih ograničenja

Algoriram 1: Računanje kazne

$kazna \leftarrow 0$
for $t_i, t_j \in T$, $redniBrojDana(t_i) = redniBrojDana(t_j)$ ***do***
  $faktorKazneUDanu \leftarrow 1 + \frac{1}{\left| redniBrojUDanu(t_i) - redniBrojUDanu(t_j) \right|}$
  for $c_k \in C, c_l \in C \setminus \{c_k\}, \{(c_k, t_i), (c_l, t_j)\} \subseteq S$ ***do***
    $brojStudenata \leftarrow brojZajednickihStudenata(c_k, c_l)$
    $faktorKazne \leftarrow min(faktorIstiDan(c_k), faktorIstiDan(c_l))$
    $kazna \leftarrow kazna + faktorKazne \cdot faktorKazneUDanu \cdot brojStudenata$
  end for
end for
$kazna \leftarrow kazna \cdot KOEFICIJENT\_ZA\_ISTI\_DAN$
for $t_i, t_j \in T, redniBrojDana(t_i) = redniBrojDana(t_j) + 1$ ***do***
  for $c_k \in C \setminus \{c_k\}, \{(c_k, t_i), (c_l, t_j)\} \subseteq S$ ***do***
    $brojStudenata \leftarrow brojZajednickihStudenata(c_k, c_l)$
    $faktorKazne \leftarrow min(faktorSusjedniDan(c_k), faktorSusjedniDan(c_l))$
    $kazna \leftarrow kazna + faktorKazne \cdot brojStudenata$
  end for
end for

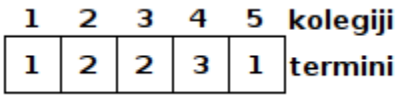
Slika 2.2.
Algoritam za računanje kazne

3. Opis rada algoritama evolucijskog računarstva i implementacija za problem rasporeda međuispita

Algoritmi evolucijskog računarstva su skupina metaheurističkih populacijskih optimizacijskih algoritama. Svi algoritmi skup rješenja pronalaze pomoću operatora često inspiriranih biološkim procesima: selekcija, mutacija, rekombinacija, reprodukcija, itd. Evolucija rješenja se odvija ponavljanjem spomenutih operatora kroz generacije. Svi evolucijski algoritmi upotrijebljeni za rješavanje problema rasporeda međuispita imaju neke zajedničke mehanizme koje upotrebljavaju. Jedan od njih je računanje dobrote rješenja koje je opisano u poglavlju ranije.

Drugi, vrlo važan mehanizam, je stvaranje inicijalne populacije. Populacija je stvorena tako da u početku poštuje jaka ograničenja. Iako je problem stvaranja populacije koja poštuje jaka ograničenja NP težak problem [1], eksperimentalno je ustanovljeno da priroda problema s kojom se susrećemo u ovom slučaju je takva da se taj problem vrlo uspješno rješava pomoću algoritma s povratkom unazad (engl. *Backtracking algorithm*)[4]. Kako već inicijalna populacija poštuje jaka ograničenja, možemo izgraditi evolucijske operatore tako da čuvaju to obilježje.

Da bi bilo moguće upotrijebiti evolucijske algoritme za rješavanje bilo kojeg problema, taj problem moramo prikazati na način koji operatorima dopušta jednostavnu manipulaciju nad njima. Taj bi prikaz, također, trebao jednoznačno opisivati rješenje. Takav prikaz se zove još i kromosom (engl. *Chromosome*) Svakom kolegiju i svakom terminu je dodijeljen identifikacijski redni broj. Rješenje je prikazano pomoću jednodimenzionalnog polja brojeva koje je veliko onoliko koliko ima kolegija. Indeks polja nam označava redni broj kolegija. Elementi polja su brojevi koji označavaju redni broj termina u kojem se element na indeksu *i* održava. Ovaj prikaz kromosoma je samo jedan od mogućih. Kromosom je prikazan na slici 3.1. i prikazuje graf sa slike 2.1.



Slika 3.1.
Prikaz rasporeda pomoću kromosoma

3.1 Genetski algoritam

Genetski algoritam je metaheuristička metoda optimiranja koja pretražuje prostor stanja inspirirana biološkom evolucijom organizama sa sposobnošću seksualne reprodukcije. Genetski algoritam spada u skupinu populacijskih evolucijskih algoritama jer optimiranje vrši uz pomoć populacije mogućih rješenja. Za optimiranje genetski algoritam koristi tri operatora: selekciju, križanje i mutaciju.

Tijek genetskog algoritma je sljedeći: prvo se generira i evaluira početna populacija, zatim se ulazi u petlju koja se ponavlja dok se ne pokaže potreba za zaustavljanjem. Zaustavljanje petlje je se odvije nakon zadanog vremenskog razdoblja. U petlji se popunjava nova generacija *mladim* jedinkama. Prvo se odabiru jedinke iz populacije po nekom kriteriju. Odabrane jedinke se križaju, tako da rezultat bude novo rješenje koje je kombinacija dva rješenja koja su ušla u križanje. Operatorom križanja se vrši lokalna pretraga prostora. Nakon što je dobivena nova jedinka nad njom se vrši mutacija. Mutacija je nasumična izmjena nekoliko elemenata rješenja i služi za proširivanje prostora pretrage. Bez operatora mutacije rješenja bi brzo konvergirala u neki od lokalnih optimuma. Sad je stvorena nova jedinka koja se zove *dijete*. Svako novo *dijete* će biti evaluirano. Provjerava se u novoj populaciji postoji li već takva identična jedinka i ako ne postoji, stavlja se je u novu populaciju. Ako identična jedinka već postoji u novoj populaciji, tada će *dijete* biti odbačeno i proizvedeno novo. Ovaj korak osigurava da populacija neće izgubiti raznolikost zbog klonova, što bi smanjilo prostor pretrage algoritma. Kao zadnja jedinka u novoj populaciji prelazi najbolja jedinka iz prošle populacije. Tako se čuva najbolje do tada pronađeno rješenje. Kada se popuni nova populacija stara populacija nestaje, umjesto nje stavi se nova i petlja kreće ispočetka.
Dijagram toka genetskog algoritma je prikazan na slici 3.2**.**

Implementirane su dvije vrste selekcije od kojih se jedna izabere u konfiguracijskoj datoteci: troturnirsku (engl. *3-tournament selection*) i slučajnu proporcionalnu selekciju (engl. *Roulette Wheel selection*) [5].

Troturnirska selekcija uzima nasumično tri jedinke iz populacije i izabire dvije s najboljom dobrotom. Odabrane jedinke se križaju.

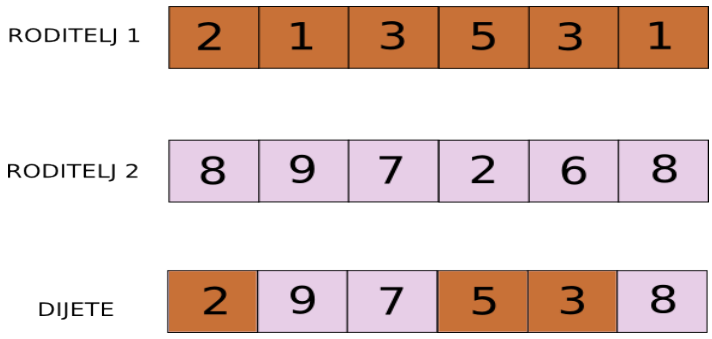
Slučajna proporcionalna selekcija simulira kotač ruleta. Na intervalu [0, 1] se svakoj jedinki dodjeli podinterval obrnuto proporcionalan njezinoj kazni. Podintervali se ne preklapaju jedan s drugim. Veličina podintervala koji jedinka x_i dobije računa se po formuli (2.1).

$$p(x_i) = \frac{k_{max} - k_i}{N \cdot k_{max} - \sum_{n=1}^{N} k_n} \qquad (2.1)$$

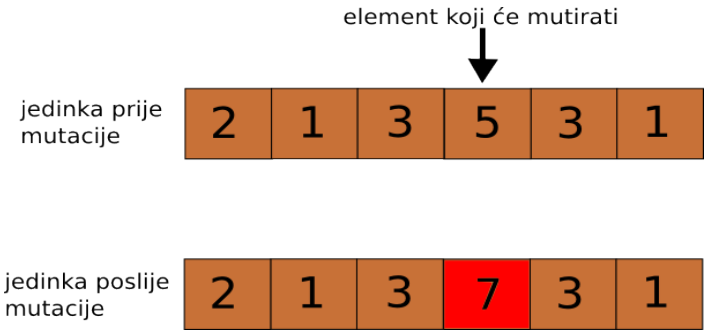
U formuli (2.1) k_{max} označava kaznu najlošije jedinke, a k_i kaznu jedinke za sa indeksom *i*. Generiraju se dva nasumična broja u intervalu cijelog "kotača" i izaberu se one jedinke na čiji su podinterval "pali" nasumični brojevi.

Križanje se obavlja tako da se kromosom djeteta najprije u cijelosti popuni s elementima prvog roditelja, a nakon toga se nasumični elementi kromosoma popune s rednim brojevima termina drugog roditelja i to samo ako se time neće prekršiti jaka ograničenja. Vjerojatnost izbora svakog elementa iz kromosoma drugog roditelja je 50%. (Slika 3.2)

Mutacija radi tako da se kroz petlju uzimaju redom elementi kromosoma, za svaki element se generira nasumični broj u intervalu od 0 do 1 i ako je dobiveni broj manji od vjerojatnosti mutacije, element kromosoma se popuni nasumičnim terminom koji neće prekršiti jaka ograničenja rasporeda. Vjerojatnost mutacije se definira u konfiguracijskoj datoteci.



Slika 3.2
Križanje dvije jedinke.
Dijete se stvori kopiranjem 1. rodielja, zatim
se nasumično odabrani elementi drugog roditelja preslikaju
u dijete. Dijete mora poštavati jaka ograničenja



Slika 3.3
Mutacija jedinke.
Izabere se nasumično element jedinke,
zatim se umjesto njega stavi neka nasumična
vrijednost koja poštuje čvrsta
ograničenja

Slika 3.4.
Dijagram toka genetskog algoritma

3.2 Jednostavni imunološki algoritam

Jednostavni imunološki algoritam(engl. *Simple Immunological Algorithm, SIA*) je vrlo jednostavan metaheuristički populacijski algoritam [6]. Inspiriran je imunološkim sustavom sisavaca. Rješenja predstavljaju antitijela, a dobrota rješenja afinitet antitijela prema antigenu (funkciji koju se optimira). U ovom slučaju afinitet antitijela je veći što je funkcija kazne manja. SIA algoritam u ovom slučaju koristi dva evolucijska operatora: reprodukciju i mutaciju.

Tijek SIA algoritma je sljedeći: Generira se i evaluira početna populacija rješenja. Zatim se ulazi u glavnu petlju. Iterira se kroz sve članove populacije i svaki se klonira *faktorNapuhivanja* puta. Svaki klon se zatim mutira na isti načina kao i u genetskom algoritmu s vjerojatnosti *vjerojatnostMutacije*. Za svaki klon se ispituje: ako je klon bolji od najgore jedinke u populaciji, onda se umjesto nje u populaciju umeće taj klon. Glavna petlja se vrti dok nije ispunjan zahtjev za zaustavljanjem. *faktorNapuhivanja* kao i *vjerojatnostMutacije* se uzimaju iz konfiguracijske datoteke. Pseudokod algoritma je prikazan na slici 3.3.

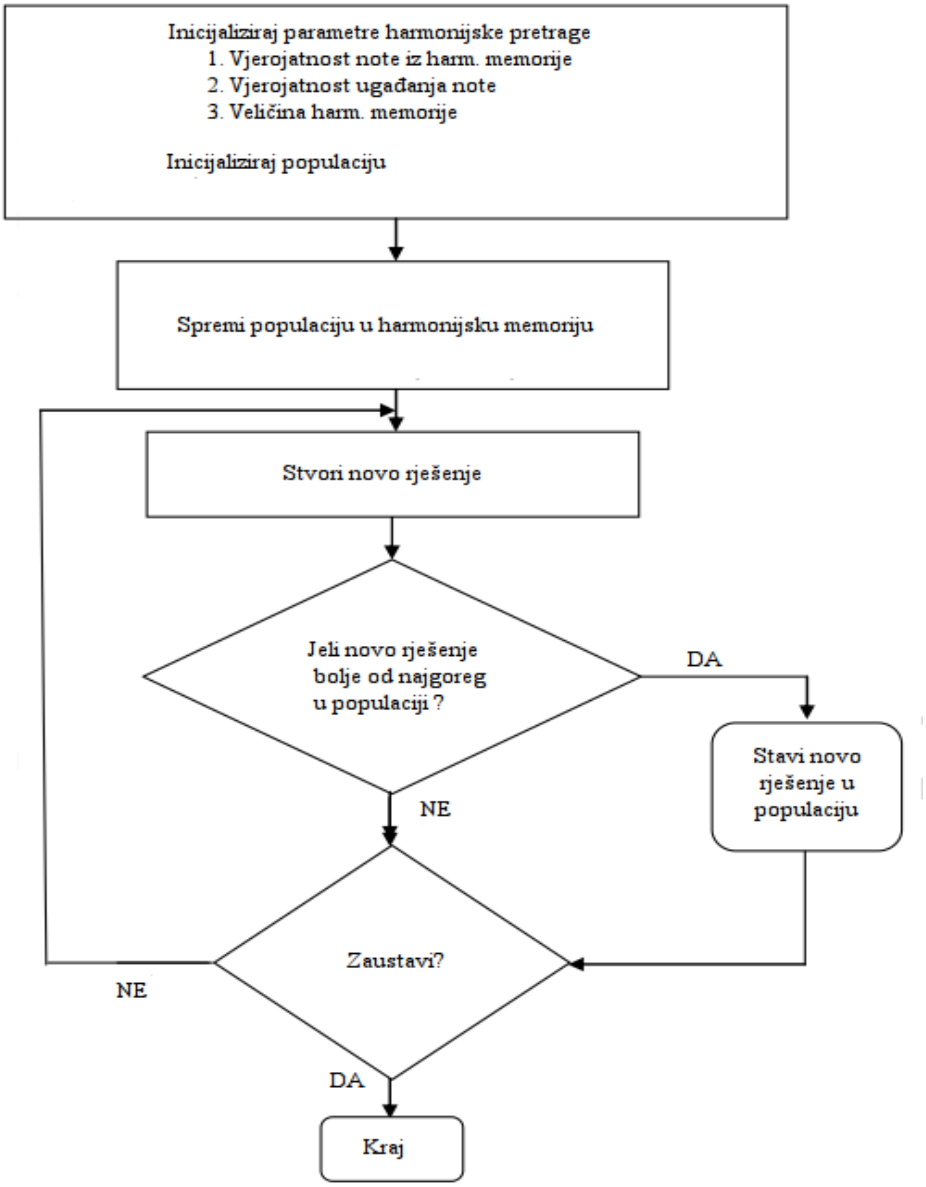
Algoritam 2: Jednostavni imunološki algoritam

$x_i = rješenje$
$P = \{x_1, x_2, \ldots, x_{velicinaPopulacije}\}$
$inicijalizirajPopulaciju(P)$
$P_{nova} \leftarrow P$
repeat until $\neg ispunjenZahtjevZaZaustavljanje$
  for all $x_i \in P$
    for $\forall\ j \in \{1, 2, \ldots, faktorNapuhivanja\}$ ***do***
      $x_{novi} \leftarrow x_i$
      $x_{novi} \leftarrow mutiraj(x_{novi}, vjerojatnostMutacije)$
      if $kazna(x_{novi}) < kazna(x_k), x_k \in P_{nova}, kazna(x_k) \geq kazna(x_p), \forall x_p \in P_{nova} \setminus \{x_k\}$ ***do***
        $P_{nova} \leftarrow P_{nova} \setminus \{x_k\}$
        $P_{nova} \leftarrow P_{nova} \cup \{x_{novi}\}$
      end if
    end for
  end for all
  $P \leftarrow P_{nova}$
end repeat

Slika 3.3.
Pseudokod SIA algoritma.

3.3 Algoritam harmonijske pretrage

Algoritam harmonijske pretrage (engl. *Harmony Search algorithm, HS*) algoritam je relativno nov metaheuristički optimizacijski algoritam koji su predložili Geem i Lee 2004. godine [7]. Inspiriran je ponašanjem skupine muzičara koji kolektivno sviraju pritom ponekad improvizirajući. Svaki član benda (član populacije) svira instrument i zajedno pokušavaju odsvirati fantastičnu melodiju. Melodija se estetski ocjenjuje (funkcija dobrote).



Slika 3.3
Dijagram toka algoritma harmonijske pretrage

Algoritam harmonijske pretrage se provodi u pet koraka koji se ponavljaju dok se ne ispuni uvjet zaustavljanja.

Prvi korak: inicijalizira se početna populacija jedinki (opisano u 3. poglavlju) i evaluira se svako rješenje u inicijalnoj populaciji. Populacija jedinki se u ovom algoritmu zove *harmonijska memorija*, a rješenje *melodija*. Iz konfiguracijske datoteke se pročitaju sljedeći parametri:

1. *harmony_memory_probability (HMP)* – vjerojatnost da se komponenta (termin, nota) nove melodije preuzme iz harmonijske memorije.
2. *pitch_adjusting_probability (PAP)* – vjerojatnost da se nota preuzeta harmonijske memorije ugodi, tj. umjesto nje uzme neka druga iz okoline trenutne.
3. *harmony_memory_size* – veličina harmonijske memorije. Slično kao veličina populacije kod genetskog algoritma

Drugi korak: *improvizira* se nova melodija (slika 3.4). Nova melodija se stvara notu po notu (termin po termin). Svaka nova nota se nasumično odabere iz harmonijske memorije s vjerojatnošću HMP (obično 0.7 < HMP < 1). Kolegij se smješta u jedan od termina u koji je smješten isti taj kolegij u nekom od nasumično odabranih rješenja iz memorije. Naravno, mogu se odabrati samo one komponente koje ne krše jaka ograničenja rasporeda. Na primjer, ako je HMP = 0.9, to znači da će 90% komponenti u novoj melodiji biti preuzeto iz harmonijske memorije. Kako sve nove komponente melodije moraju zadovoljavati jaka ograničenja, a poštivanje tih uvjeta je NP-težak problem, opet mora biti iskorišten algoritam s povratkom unazad.

One note koje su ostale nepopunjene (vjerojatnost: 1-HMP), nakon što je melodija izgrađena pomoću harmonijske memorije, se moraju popuniti nasumično iz skupa dozvoljenih termina. Ovaj operator je sličan kao operator mutacije kod genetskog algoritma. Proširuje prostor pretrage za optimalnim rješenjem da se algoritam ne bi zaglavio u nekom od lokalnih optimuma.

Nakon što se proizvede nova melodija provuče se kroz *ugađanje* nota tj. svaka nota u novoj melodiji preuzeta iz harmonijske memorije ima vjerojatnost

PAP da se *ugodi*. Ugađanje znači da će se postojeća nota u novoj melodiji zamijeniti s nekom iz neposredne blizine. Konkretno, *ugađanje* znači da će se kolegij koji je termin dobio iz harmonijske memorije smjestiti u neki drugi termin koji je u istom danu kao i prvotno odabrani i ne krši jaka ograničenja.

Treći korak: Tek proizvedena melodija se evaluira. Dobrote nove melodije i melodije koja je u harmonijskoj memoriji, a ima najmanju dobrotu, se uspoređuju. Ako nova melodija ima bolju dobrotu, onda se stavlja u harmonijsku memoriju, a najgora se briše. U suprotnom se nova melodija odbacuje.

Četvrti korak: Nove se melodije proizvode sve dok algoritam ne dobije zahtjev za zaustavljanje.

Algoritam 3: Stvaranje nove melodije

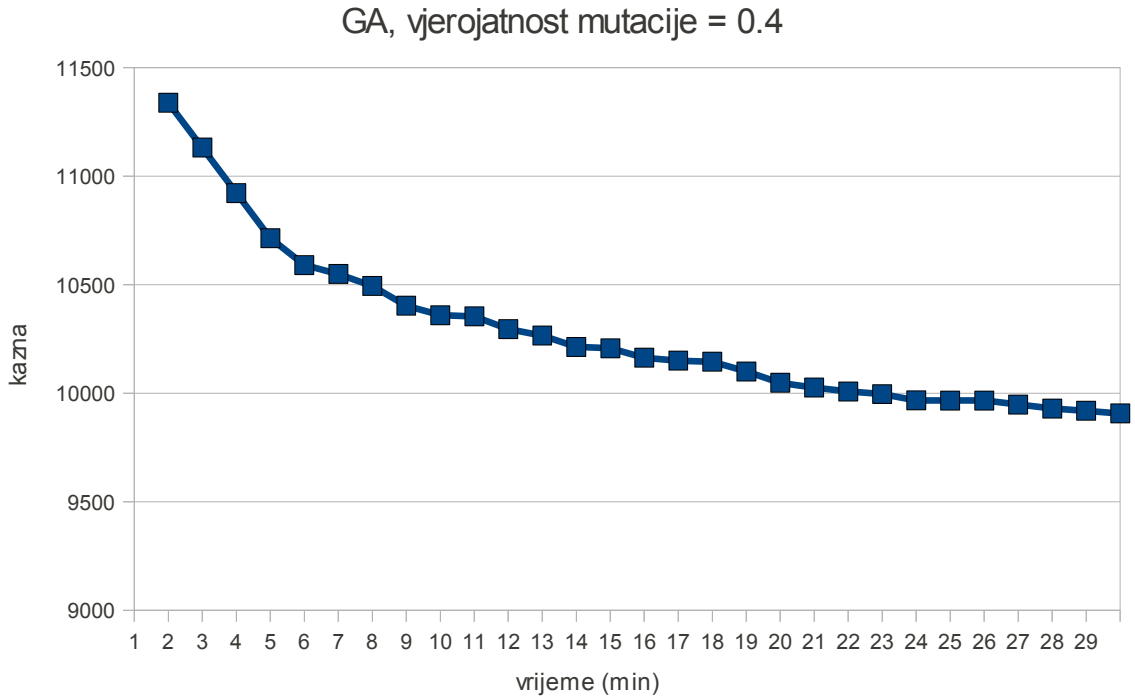
```
for index ∈ {1, 2, ..., velicina_jedinke} do
    x ← generirajNasumicanBroj([0, 1])
    if x < HMP do
        randJedinka ← nasumična jedinka iz memorije koja poštuje čvrsta ograničenja nove melodije
        novaMelodija[index] ← randMelodija[index]
    else do
        novaMelodija[index] ← nasumičan element koji poštuje čvrsta ograničenja nove melodije
    end if else
    y ← generirajNasumicanBroj([0, 1])
    if y < PAP ∧ x < HMP do
        novaMelodija[index] ← nasumicanTerminIzIstogDanaKaoI(novaMelodija[index])
    end if
end for
```

Slika 3.4
Pseudokod stvaranja nove melodije
kod algoritma harmonijske pretrage.

4. Utjecaj parametara na dobivene rezultate

U ovom poglavlju su izneseni eksperimentalni rezultati za parametar vjerojatnost mutacije kod genetskog algoritma i jednostavnog imunološkog algoritma, te za parametre *harmony_memory_probability* i *pitch_adjustment_probability.* Pokusi su pokrenuti na Intel Pentium 4 računalima takta 2.4Ghz s 512 MB radne memorije. Za svaku instancu parametra pokus je pokrenut 30 puta. Svaki pokus je trajao 30 minuta. Algoritmi su testirani na problemu rasporeda međuispita za ljetni semestar akademske godine 2009./2010. na Fakultetu elektrotehnike i računarstva.

Utjecaj izbora parametra je kod metaheurističkih algoritama vrlo izražen jer program s pogrešno odabranim parametrima najčešće daje vrlo loša rješenja. Nasumično je odabrana vjerojatnost mutacije od 0.4 za genetski algoritam, a konvergencija srednje vrijednosti najboljih rješenja za taj parametar je prikazana grafom na slici 5.1.



slika 5.1
Genetski algoritam s vjerojatnošću mutacije od 0.4.
Odnos kazne i proteklog vremena

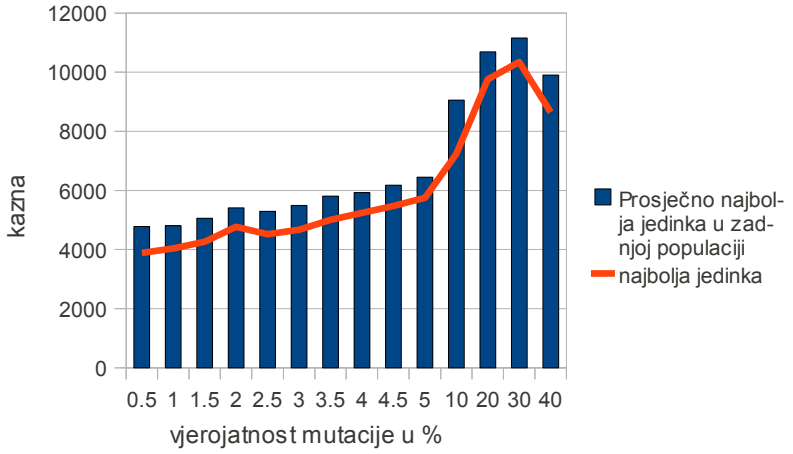
Može se primijetiti da za ovako odabran parametar mutacije algoritam vrlo brzo ulazi u stagnaciju s relativno lošim rezultatom.

Pokusi su izvedeni na genetskom algoritmu s parametrima: populacija od 40 jedinki, troturnirska selekcija. Mijenja se vjerojatnost mutacije.

Vjerojatnost mutacije u %	Prosječno najbolja jedinka u zadnjoj generaciji (kazna)	Devijacija prosječno najbolje jedinke (kazna)	Nrosječna devijacija zadnje generacije (kazna)	Najbolja jedinka (kazna)
0.5	4780.444444	457.418739	561.894682	3885.666667
1.0	4808.944444	334.574312	804.005962	4039.666667
1.5	5055.866667	462.564347	1132.89633	4266
2.0	5412.155556	377.617219	1259.212907	4772.666667
2.5	5297.5	390.392726	1388.393186	4515.333333
3.0	5491.755556	316.357658	1547.485392	4673
3.5	5808.566667	461.603935	1542.995173	5003.333333
4.0	5929.577778	335.853050	1739.692516	5244.333333
4.5	6172.944444	439.849940	1763.996701	5478.666667
5.0	6445.766667	398.586296	1968.854149	5752
10.0	9059.344444	581.571971	3368.444705	7232
20.0	10685.8	506.143095	3583.746411	9760.666667
30.0	11155.333333	429.796183	3633.284814	10350
40.0	9907.922222	497.002345	3536.904115	8653

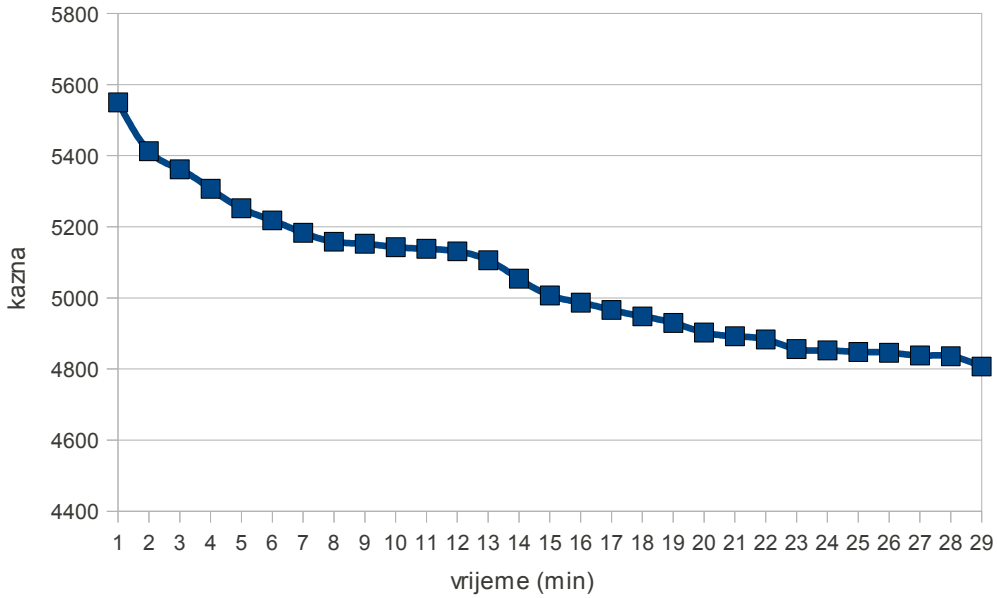
Tablica 5.1

Odnos vjerojatnosti mutacije i kvalitete rješenja za genetski algoritam (GA)



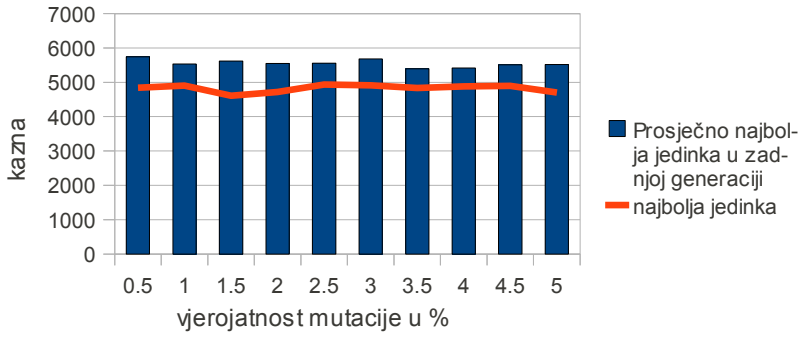
Slika 5.2

Odnos vjerojatnosti mutacije i kvalitete rješenja za genetski algoritam (GA)



Slika 5.3
Genetski algoritam s vjerojatnošću mutacije 0.005.
Odnos proteklog vremena i kazne

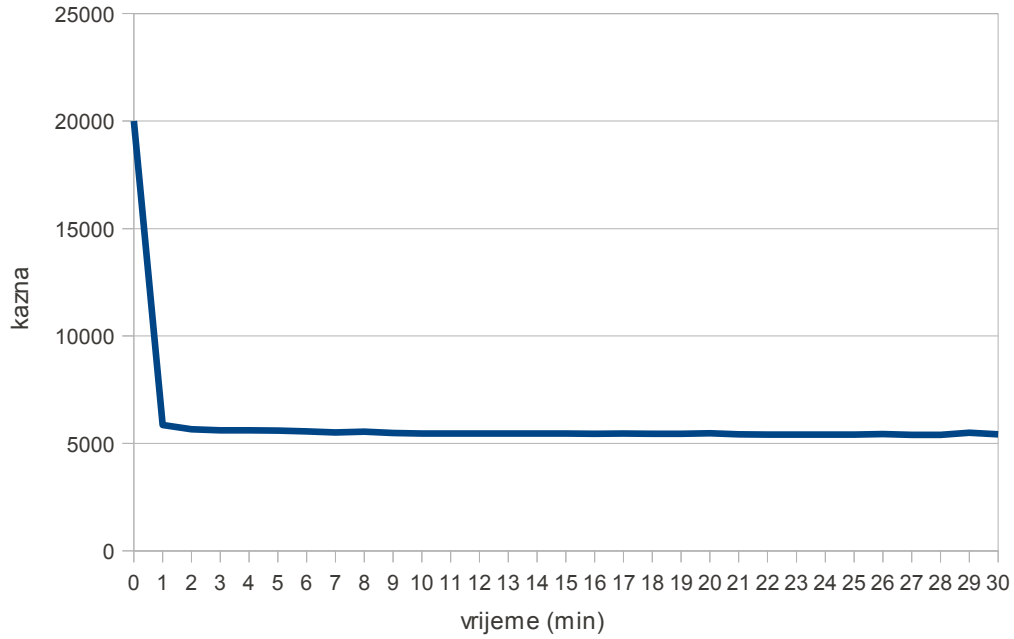
Sljedeća grupa pokusa je izvedena za jednostavni imunološki algoritam. Algoritam je pokrenut sa stalnom vrijednosti parametara: *faktorNapuhivanja* je 10, a veličina populacije je 100 jedinki. Ovdje je također prikazan odnos kvalitete rješenja i vjerojatnosti mutacije. Podaci prikazani u tablici 5.2 pokazuju da kvaliteta rješenja kojeg stvori imunološki algoritam (SIA) implementiran za problem rasporeda međuispita nije značajno ovisna o parametru mutacije u području od 0.005 do 0.05. Može se vidjeti utjecaj vjerojatnosti mutacije na raznolikost populacije.



Slika 5.4
Odnos vjerojatnosti mutacije i kvalitete rješenja za
jednostavani imunološki algoritam (SIA)

Vjerojatnost mutacije u %	Prosječno najbolja jedinka u zadnjoj generaciji (kazna)	Devijacija srednje najbolje jedinke (kazna)	Prosječna devijacija zadnje generacije (kazna)	Najbolja jedinka (kazna)
0.5	5744.866667	474.227637	49.440805	4846
1.0	5534.511111	405.25101	89.254678	4902
1.5	5620.711111	525.827214	187.13641	4608.333333
2.0	5549.866667	613.568118	134.140272	4717.333333
2.5	5556.055556	510.392592	178.787408	4936.333333
3.0	5683.2	751.477822	339.216781	4911.666667
3.5	5394.7	352.717033	251.045941	4835.666667
4.0	5416.077778	379.647121	550.66851	4879
4.5	5510.622222	337.15052	364.298602	4896.666667
5.0	5521.277778	436.103068	461.03832	4702

Tablica 5.2
Odnos vjerojatnosti mutacije i kvalitete rješenja
za jednostavni imunološki algoritam (SIA)

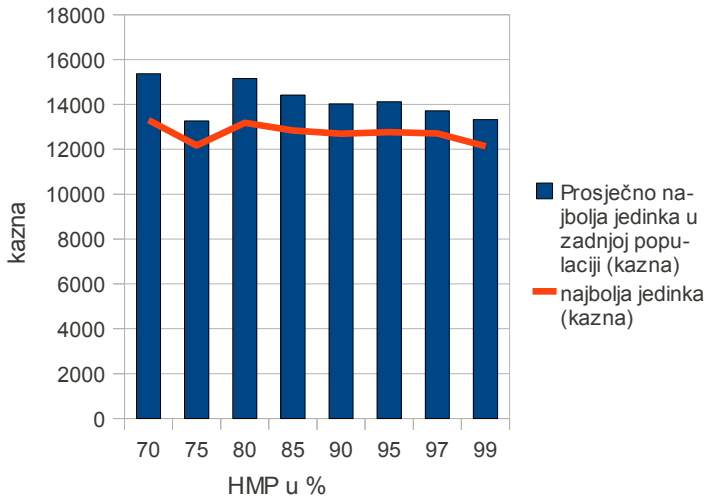


Slika 5.5
Jednostavni imunološki algoritam
s vjerojatnošću mutacije 0.035.
Odnos proteklog vremena i kazne

Grupa pokusa koja je izvedena za algoritam harmonijske pretrage se odnosi na dva parametra: vjerojatnost uzimanja elementa iz harmonijske memorije (HMP) i vjerojatnost *ugađanja* elementa (PAP).

Vjerojatnost mutacije u %	Prosječno najbolja jedinka u zadnjoj populaciji (kazna)	Devijacija srednje najbolje jedinke	Prosječna devijacija zadnje populacije	Najbolja jedinka (kazna)
70	15368	1312.001662	1121.973817	13306.333333
75	13269.7	1004.289438	574.650628	12173.666667
80	15154.944444	1516.843847	862.73708	13181
85	14420.922222	1060.787493	625.931802	12841.333333
90	14026.711111	857.095884	604.391967	12695
95	14125	1266.596477	568.58972	12769
97	13723.366667	812.236619	492.773447	12714
99	13332.422222	759.824279	449.041893	12144.333333

Tablica 5.3
Odnos vjerojatnosti uzimanja vrijednosti
iz harmonijske memorije i kvalitete rješenja za
algoritam harmonijske pretrage (HS)

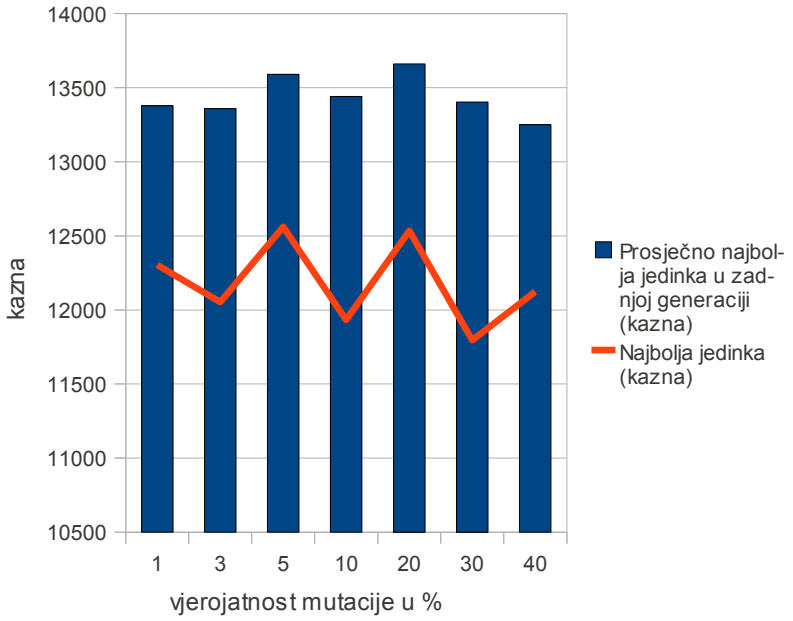


Slika 5.6
Odnos vjerojatnosti uzimanja vrijednosti
iz harmonijske memorije i kvalitete rješenja za
algoritam harmonijske pretrage (HS)

Vjerojatnost mutacije u %	Prosječno najbolja jedinka u zadnjoj generaciji (kazna)	Devijacija prosječne najbolje jedinke (kazna)	Najbolja jedinka (kazna)	Prosječna devijacija zadnje generacije (kazna)
1	13377.911111	680.061593	12304	582.279168
3	13358.944444	686.430866	12053	449.516495
5	13589.588889	663.549013	12561	494.396742
10	13441.077778	742.668244	11933.333333	496.848405
20	13660.111111	660.934543	12533	442.166782
30	13403.2	613.686484	11797	417.099922
40	13249.811111	526.012504	12123	417.57534

Tablica 5.4
Odnos vjerojatnosti *ugođavanja* elemenata
i kvalitete rješenja za algoritam harmonijske pretrage (HS)

Prvo je testirano ponašanje kvalitete rješenja ovisno o parametru HMP sa stalnim vrijednostima: veličina populacije od 40 jedinki, PAP je 1%. Rezultati su u tablici 5.3. Nakon toga je testirano ponašanje kvalitete rješenja ovisno o parametru PAP sa stalnim vrijednostima: veličina populacije od 40 jedinki, HMP je 98%. Rezultati pokazuju da algoritam harmonijske pretrage implementiran za problem rasporeda međuisipita naparameru način prikazan u ovom radu ne pronalazi zadovoljavajuća rješenja. Rješenja dobivena ovim algoritmom su za red veličine lošija od rješenja koja su pronašli jednostavni imunološki algoritam i genetski algoritam. Mogući uzrok lošoj kvaliteti rješenja je korištenje algoritma s povratkom unatrag kod stvaranja nove jedinke. Algoritam s povratkom unatrag iscrpno pretražuje prostor rješenja, pa je zbog toga vrlo spor i nije pogodan za korištenje u algoritmima kod kojih je bitna brzina izvođenja.



Slika 5.6
Odnos vjerojatnosti *ugođavanja* elemenata
i kvalitete rješenja za algoritam harmonijske pretrage (HS)

5. Zaključak

Metaheuristički algoritmi iz područja evolucijskog računarstva su pokazali dobre performanse kod rješavanja problema rasporeda međuispita. Međutim, taj tip algoritama može biti vrlo osjetljiv na radne parametre. Od iznimne važnosti je odrediti parametre za koje će algoritmi pronaći dobra rješenja. Ispitani su parametri za genetski algoritam, jednostavni imunološki algoritam i algoritam harmonijske pretrage. Rezultati su pokazali da je od navedenih algoritama genetski algoritam pronašao najbolje rješenje. Jednostavni imunološki algoritam je pronašao nešto lošije rezultate od genetskog algoritma ali je pokazao veću otpornost na promjenu vjerojatnosti mutacije. Algoritam harmonijske pretrage je pronašao najlošija rješenja. Razlika između algoritma harmonijske pretrage i ostalih je za red veličine. Mogući uzrok tome je neadekvatna implementacija nekih dijelova algoritma harmonijske pretrage za problem rasporeda međuispita.

6. Sažetak

Problem rasporeda međuispita je NP-potpun problem koji se često javlja u obrazovnim ustanovama. Osmišljeni su mnogi algoritmi za rješavanje ovog problema zasnovani na heuristikama. Vrlo dobre rezultate u rješavanju nekih NP-potpunih problema pronalaze algoritmi evolucijskog računarstva. Prilikom rješavanja problema rasporeda međuispita na Fakultetu elektrotehnike i računarstva korištena su tri algoritma evolucijskog računarstva: genetski algoritam, jednostavni imunološki algoritam i algoritam harmonijske pretrage. Opisana je implementacija operatora algoritama i ispitana kvaliteta rješenja u odnosu na vjerojatnost mutacije kod genetskog i jednostavnog imunološkog algoritma, a u odnosu na parametre uzimanja iz harmonijske memorije i *ugađanja* kod algoritma harmonijske pretrage. Rezultati su pokazali da genetski algoritam pronalazi najbolja rješenja za vjerojatnost mutacije u području oko 1.5%. Lošiji od njega je jednostavni imunološki algoritam. Algoritam harmonijske pretrage u trenutnoj implementaciji ne pronalazi zadovoljavajuća rješenja.

7 .Literatura

[1] Garey, Michael R.; Johnson, David S., “Computers and Intractability: A Guide to the Theory of NP-Completeness”, W.H. Freeman, 1979., ISBN 0-7167-1045-5

[2] M.A. Al-betar, A.T. Khader, and T.A. Gani, “A harmony search algorithm for university course timetabling”, *Network, 2008.*

[3] D. de Werra, “An introduction to timetabling”, *European Journal of Operational Research*, 1985, pp. 151-162.

[4] Thomas H. Cormen; Charles E. Leiserson, Ronald R. Rivest, Cliff Stein., “Introduction to Algorithms”, McGraw-Hill, 1990., ISBN 0-262-03141-8.

[5] M. Golub, “Genetski algoritam: 1. skripta”, 2004., http://www.zemris.fer.hr/~golub/ga/ga.html (10.05.2010.)

[6] Vincenzo Cutello and Giuseppe Nicosia, “An Immunological Approach to Combinatorial Optimization Problems”, Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence, Seville, Spain, pp. 361-370, 2002.

[7] Z.W. Geem, K.S. Lee, “A new structural optimization method based on the harmony search algorithm”, *Computers and Structures (2004.)*, 82(9-10),pp.781-798.

[8] Holland, J. H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Ann Arbor, MI: University of Michigan Press, 1975.

[9] Z.W. Geem, K.S. Lee, “A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice”, Comput. Methods Appl. Mech. Engrg. 194 (2005) 3902–3933 www.elsevier.com/locate/cma.