SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 192

Utjecaj parametara algoritama evolucijskog računanja na kvalitetu rješenja za problem rasporeda međuispita

Đorđe Grbić

Zagreb, lipanj 2011.

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

Zahvaljujem mentorima doc. dr. sc. Domagoju Jakobovi ću i mr. sc. Marku Čupi ću na suradnji u izradi diplomskog rada. Posebno zahvaljujem mr. sc. Marku Čupi ću na vođenju, pomoći i podršci.

SADRŽAJ

1. Uvod

Problem izrade rasporeda međuispita (engl. *Exam-timetabling problem*) je jedan od nekoliko problema izrade rasporeda na fakultetima. Ovaj problem je kombinatorni optimizacijski problem i njegova težina rješavanja se smatra NP-potpunom. Postoji nekoliko inačica spomenutog problema. Problem koji se obrađuje u ovom radu je problem smještanja međuispita u predefinirane nepreklapajuće vremenske intervale (termin). Međuispit može biti dodjeljen u samo jedan termin. U jedan termin je moguće staviti više međuispita dok god nije premašen kapacitet tog termina. Kako bi se proizveo kvalitetan raspored mora se zadovoljiti još niz drugih zahtjeva. Cilj je proizvesti raspored koji zadovoljava osnovne predispozicije kvalitetnog studija i nudi određenu fleksibilnost kod davanja zahtjeva prije same izrade, npr. međuispit iz kolegija "Matematika 1" mora biti u ponedjeljak u 9 ujutro.

Najraniji pristupi rješavanja ovog problema su koristili heuristike bojanja grafova (de Werra (1985)) gdje su međuispiti predstavljani čvorovima, termini su predstavljani bojama čvorova, a bridovi između čvorova su predstavljali dijeljene studente između pripadajućih međuispita. Iako se taj pristup pokazao efikasnim za slučajeve kada ima malo studenata i međuispita, većinom su neupotrebljivi kada se broj studenata i međuispita poveća (Al-Betar et al., 2008).

Nakon neuspjeha heuristika za bojanje grafova, problemu izrade rasporeda se pristupilo korištenjem metaheurističkih algoritama. Metaheuristički algoritmi su skupina generičkih algoritama koji su sposobni rješavati raznolike probleme koji se ne mogu riješiti u polinomijalnom vremenu koristeći egzaktne matematičke metode (Talbi, 2009). Metaheuristički algoritmi se mogu podijeliti na dvije skupine: algoritmi koji koriste jedno rješenje (engl. *Single-solution based algorithms*) i algoritmi koji koriste populaciju rješenja tzv. populacijski algoritmi (engl. *Population based algorithms*). Prvi kreću s jednim rješenjem koje se iterativno unaprjeđuje dok se ne postigne globalni optimum. Nedostatak ovakvih algoritama je što se često zaglave u lokalno optimalnim rješenjima i nisu sposobni istražiti ostatak prostora pretrage u potrazi za globalno optimalnim rješenjem. Populacijski algoritmi, s druge strane, koriste više rješenja (tzv.

populacija rješenja). Ovakvi algoritmi pokušavaju pokriti veću površinu prostora pretrage i tako povećati izglede za pronalazak globalno optimalnog. Nedostatak ovakvih algoritama je što su uglavnom koncentrirani na istraživanje prostora pretrage i često ne istraže dovoljno temeljito susjedstvo rješenja koja dobiju pa zbog toga pokazuju svojstvo spore konvergencije.

Dodatni problem stvaraju parametri navedenih algoritama. Pogrešan odabir parametara može dovesti ili do preuranjene konvergencije ka lokalno optimalnim rješenjima ili do prespore konvergencije algoritma tako trošeći računalno vrijeme ne dajući zadovoljavajuće rezultate. Populacijski algoritmi koji se koriste za rješavanje ovog problema su: eliminacijski genetski algoritam (engl. *Steady-state genetic algorithm*), generacijski genetski algoritam (engl. *Generation genetic algorithm*), algoritam harmonijske pretrage (engl. *Harmony search algorithm*), jednostavni imunološki algoritam (engl. *Simple immune algorithm*) i $\mathcal{MAX}$-$\mathcal{MIN}$ mravlji sustav (engl. $\mathcal{MAX}$-$\mathcal{MIN}$ *Ant System*).

Paralelizacija je dodatni pristup često korišten kada se susreću teški optimizacijski problemi. Kako su u današnje vrijeme pristupačni višejezgreni procesori i kako fakulteti posjeduju laboratorije s velikim brojem umreženih računala, moguće je upogoniti velik broj metaheurističkih algoritama u paraleli. Očekuje se da će algoritmi biti uspješniji ako svaki algoritam stvori svoju populaciju rješenja koju koristi za pretragu rješenja i nakon nekog vremena razmijeni svoja rješenja s ostalim algoritmima u mreži. Način na koji algoritmi razmjenjuju rješenja je definiran topologijom razmjene prije nego algoritmi krenu s radom.

Za potrebe rješavanja problema izrade međuispita na Fakultetu elektrotehnike i računarstva u Zagrebu, razvijeno je programsko okruženje u Java programskom jeziku koje nudi platformu za distribuirani pogon više heterogenih algoritama evolucijskog računanja. Implementirano je nekoliko metaheurističkih algoritama.

Iako implementacija sustava omogućuje paralelno pokretanje algoritma u ovom radu će biti opisano samo izolirano ponašanje implementiranih metaheurističkih algoritama. Bit će ispitano ponašanje ovisno o parametrima algoritama i pokušat će se dobiti neke zaključci. Također će biti ispitano ponašanje rada populacijskih metaheurističkih algoritama koji rade uz pomoć algoritama lokalne pretrage, ponašanje algoritama prilikom promjene parametara uslijed stagnacije algoritama, utjecaj dinamičkog mijenjanja evaluacijske funkcije.

U 2 poglavlju će problem izrade rasporeda biti detaljno objašnjen i formalno definiran. Poglavlje 3 detaljno opisuje algoritme koji su upotrijebljeni u implementaciji sustava. Poglavlje 5 iznosi rezultate pokusa i zaključke koji se mogu izvući iz njih.

2. Problem izrade rasporeda međuispita

Problem izrade rasporeda je poznat NP-potpun kombinatorni optimizacijski problem s kojim se susreću sveučilišta i fakulteti s velikom populacijom studenata i kolegija, pogotovo ako su mnogi od kolegija izborni. Problem se sastoji od toga da se međuispiti kolegija smještaju u termine tako da poštuju zadana ograničenja. Postoje dvije vrste ograničenja: čvrsta ograničenja i meka ograničenja. Krši li raspored čvrsta ograničenja, tada je neupotrebljiv. Meka ograničenja su ona koja mogu biti prekršena ali bi kršenja trebalo svesti na minimum. Problem koji se rješava uporabom programske implementacije koja je stvorena u sklopu ovog rada se razlikuje od sličnih problema u načinu na koji se računa kazna rasporeda i odabiru ograničenja koja raspored mora poštivati.

Definiran je fiksni vremenski interval (npr. dva tjedna) podijeljen na fiksne nepreklapajuće termine. Nadalje, postoji fiksni broj kolegija koji moraju održati po jedan međuispit u jednom od termina. Svaki međuispit ima skup studenata koji ga polažu. Studenti su podijeljeni po nekoliko modula, gdje broj upisanih studenata po modulima može veoma varirati. Studenti upisani u određeni modul moraju polagati neke od predmeta tog modula. Različiti moduli mogu imati skup preklapajućih obaveznih kolegija. Također se kod izrade rasporeda vodi računa o subjektivnim težinama međuispita koje se pribavljaju putem anonimne ankete. Trebalo bi kolegijima koji su označeni kao teški i onima koji imaju mnogo zajedničkih studenata termine što više odmaknuti. Dodatna kazna se uvodi ako su kolegiji unutar istog modula ili se slušaju na istoj godini studija. Spomenuta ograničenja spadaju u skup mekih ograničenja i za kršenje istih se računa kazna evaluacijskom funkcijom koja je opisana nešto kasnije. Čvrsta ograničenja su:

• međuspiti dva kolegija koji imaju zajedničke studente koji ih polažu se ne smiju nalaziti u istom terminu te

• ukupni broj studenata koji polažu međuispit u jednom terminu ne smije prelaziti unaprijed definiran kapacitet tog termina.

Definiran je skup kolegija kao $C = \{c_1, \ldots, c_{n_C}\}$, skup termina kao $T = \{t_1, \ldots, t_{n_T}\}$ i skup studenata kao $S = \{s_1, \ldots, s_{n_S}\}$. Skup P_a označava skup svih parova kolegija koji imaju zajedničke studente. Definiran je skup $P = \{c_j, c_k\}$ kao podskup od P_a gdje su kolegiji c_j i c_k obavezni predmeti unutar jednog modula. Definirane su funkcije $cKazna(c_i, c_j)$, $tKazna(t_z, t_w)$, i $dKazna(d_{c_x}, d_{c_y})$. Funkcija $cKazna(c_i, c_j)$ predstavlja kaznu između kolegija c_i i c_j. Ova je kazna veća što su kolegiji teži ili što kolegiji imaju veći broj zajedničkih studenata. Funkcija $tKazna(t_z, t_w)$ vraća kaznu između dva termina t_z i t_w. Funkcija $dKazna(d_{c_x}, d_{c_y})$ vraća dodatnu kaznu između para kolegija c_x i c_y koji pripadaju istom modulu.

Funkcija $cKazna(c_i, c_j)$ je definirana kao suma kazni izračunatih za svakog studenta koji se nalazi u oba kolegija c_i i c_j. Svaki student može povećati kaznu za 1, (ako su kolegiji u njegovom upisanom modulu), za 0.5 (ako su oba kolegija unutar godine koju je student upisao ali nisu unutar istog modula), za 0.25 (ako jedan od kolegija nije s godine koju je student upisao) ili za 0.125. Završna vrijednost funkcije se dobiva tako što se dobivena suma pomnoži s $min(w_i, w_j)$, gdje je w_k subjektivna težina kolegija c_k.

Funkcija $tKazna(t_x, t_y)$ je monotono padajuća funkcija kako se termini međusobno udaljavaju i definirana je kao:

$$tKazna(t_x, t_y) = \alpha^{1-\frac{udaljenost(t_x,t_y)}{24}} \tag{2.1}$$

gdje $udaljenost(t_x, t_y)$ vraća udaljenost između termina t_x i t_y izraženu u satima.

Funkcija $dKazna(d_{c_x}, d_{c_y})$ je izražena kao:

$$dKazna(d_{c_x}, d_{c_y}) = \begin{cases} 200, & \Delta = 0 \\ 100, & \Delta = 1 \\ d(\Delta), & \Delta \geq 2 \end{cases} \tag{2.2}$$

gdje je

$$d(\Delta) = \frac{60}{1 + \exp\left((0.2 \cdot \Delta)^3\right)}. \tag{2.3}$$

Ova funkcija se računa samo za one parove kolegija (c_x, c_y) koji se nalaze u skupu P. Vrijednost Δ označava udaljenost između dana u kojima se održavaju međuispiti kolegija c_x i c_y. Vrijednost d_{c_i} označava dan u kojem se održava međuispit kolegija c_i.

Kazna rasporeda međuispita je označena s p i označava mjeru kršenja mekih ograničenja. Sastoji se od dvije komponente, p_1 i p_2:

$$p = p_1 + p_2. \tag{2.4}$$

Komponenta p_1 je definirana kao:

$$p_1 = \sum_{(c_x, c_y) \in P} dKazna(d_{c_x}, d_{c_y}). \tag{2.5}$$

Kazna p_2 se računa kao:

$$p_2 = \sum_{i=1}^{n_T - 1} \sum_{j=i+1}^{n_T} tKazna(t_i, t_j) \cdot r(t_i, t_j) \tag{2.6}$$

gdje je $r(t_i, t_j)$ definiran kao

$$r(t_i, t_j) = \sum_{c_k \in c(t_i)} \sum_{c_l \in c(t_j)} cKazna(c_k, c_l). \tag{2.7}$$

Pri tome $c(t_i)$ vraća skup kolegija koji se održavaju u terminu t_i. Sve konstante koje se koriste u formulama 2.1, 2.2 i 2.3 su odabrane metodom pokušaja i pogreške i specifične su za problem koji rješavaju. U eksperimentima je korištena vrijednost $\alpha = 7$. Pseudokod izračuna kazne je prikazan u algoritmu 2.1.

Algoritam 2.1 pseudokod izračuna kazne rasporeda

$kazna \leftarrow 0$
za $\forall (t_i, t_j), t_i \; i \; t_j \in T$ **radi**
  $ispiti_{t_i} \leftarrow svi\_ispiti(t_i)$
  $ispiti_{t_j} \leftarrow svi\_ispiti(t_j)$
  za $\forall (c_x, c_y), c_x \in ispiti_{t_i}, c_y \in ispiti_{t_j}$ **radi**
    $kazna \leftarrow kazna + cKazna(c_x, c_y)$
  kraj za
  $kazna \leftarrow kazna + tKazna(t_i, t_j)$
kraj za
za $\forall (c_w, c_z) \in P$ **radi**
  $kazna \leftarrow kazna + dKazna(d_{c_w}, d_{c_z})$
kraj za
vrati $kazna$

3. Algoritmi

3.1. Izgradnja početne populacije

Kod rješavanja problema pomoću evolucijskog računanja prilikom inicijalizacije algoritma potrebno je stvoriti početnu populaciju. Početna populacija se najčešće gradi tako da se jedinke koje pripadaju populaciji izgrade nasumičnim dodjeljivanjem vrijednosti. Kod konkretne implementacije koja se koristi za rješavanje izrade rasporeda postavljeno je ograničenje na nasumično stvaranje populacije. Populacija je stvorena tako da u početku poštuje jaka ograničenja. Iako je problem stvaranja populacije koja poštuje jaka ograničenja NP-težak problem (Garey i Johnson, 1979), eksperimentalno je ustanovljeno da je priroda problema s kojom se susrećemo u ovom slučaju takva da se taj problem vrlo uspješno rješava pomoću algoritma (Cormen et al., 1990) s povratkom unazad (engl. *Backtracking algorithm*). Kako već inicijalna populacija poštuje jaka ograničenja, možemo izgraditi evolucijske operatore tako da čuvaju to obilježje.

3.2. Genetski algoritam

Charles Darwin je u 19. stoljeću izdao knjigu (Darwin, 1859) u kojoj je iznio novu teoriju evolucijskog razvoja organizama. Promatrao je brzinu razmnožavanja jedinki unutar vrste i primijetio je da se jedinke razmožavaju eksponencijalnom progresijom, tj. kada bi svi potomci jedinki preživljavali, veličina populacije bi eksponencijalno rasla iz generacije u generaciju. No broj jedinki je ostajao relativno stabilan i nije pokazivao znakove eksponencijalnog rasta. Također je poznato da u okolišu ima ograničena količina resursa koje organizmi moraju konzumirati kako bi preživjeli (hrana, voda, sklonište, itd.). Vidjevši ovo zaključio je da neke od jedinki zasigurno moraju odumrijeti prije nego dosegnu spolnu zrelost. Odumiranje nekih jedinki je uzrokovano borbom za ograničen broj resursa. Nadalje je primjetio da nisu sve jedinke iste vrste jednake, nego da postoje razlike u izgledu, ponašanju, veličini, itd. Ove različitosti

imaju za posljedicu da su neke jedinke uspješnije u borbi za resurse i samim time imaju priliku razmnožavati se. Ovaj proces je nazvao *prirodna selekcija*.

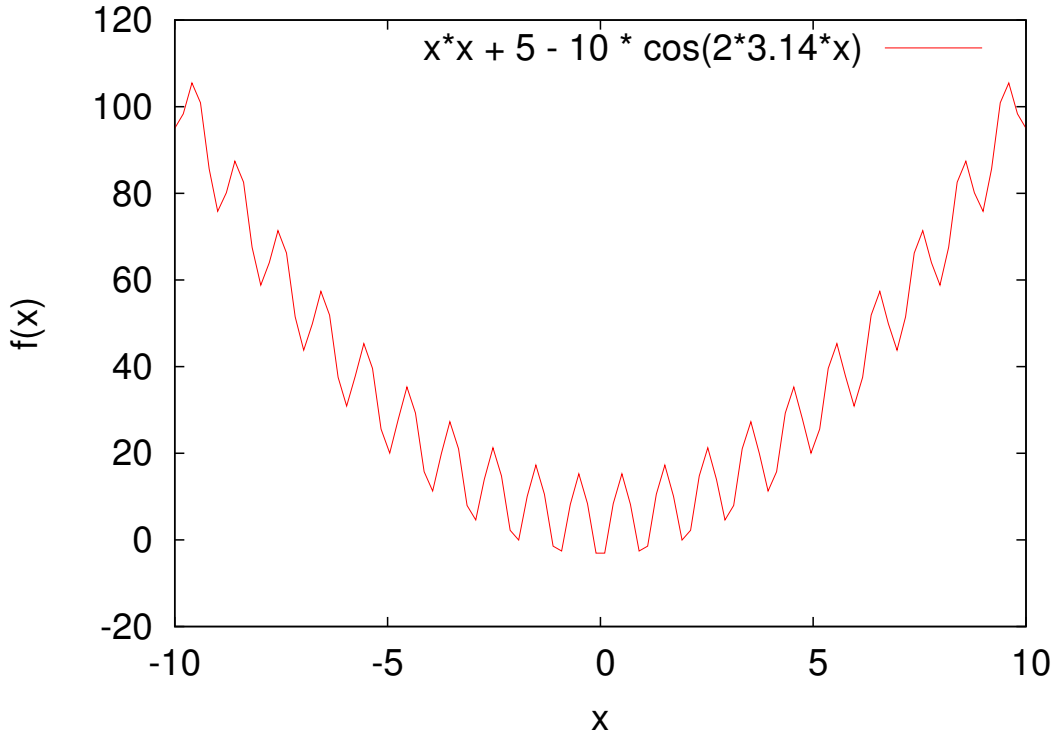
Sljedeći mehanizam koji omogućava evoluciju je *križanje*. Križanje je postupak kojim potomci dobivaju značajke svojih roditelja. Značajke su zapisane u tvorevinama zvanim *kromosomi*. Kromosomi se sastoje od molekule *DNK* koja kemijskim spojevima kodira svojstva jedinke koja je ima u sebi. Jedna cjelina unutar DNK se zove *gen*. Svaki kromosom dolazi u paru gdje jedan iz para dolazi od oca, a drugi od druge jedinke iz odabranog para. Dakle, svako svojstvo dolazi kodirano u dva gena. Geni mogu biti ili ravnopravni ili neravnopravni. Ako su geni ravnopravni, onda će oba gena sudjelovati u nastanku svojstva za koje su zaduženi. Ako su geni neravnopravni, onda će u nastanku svojstva sudjelovati samo dominantni gen. Prilikom križanja dolazi do miješanja gena u kromosomskim parovima i cijepanja, tako da jedna polovina para sudjeluje u proizvodnji potomka. Druga polovina dolazi od majke. Na ovaj način potomak dobiva mješavinu svojstava oba roditelja.

Potomci nisu savršena kombinacija svojih roditelja jer dolazi do nasumičnih grešaka prilikom križanja, a ta se pojava zove *mutacija*. Ove greške su jako rijetke ali se događaju. Mutacija unosi novu informaciju u DNK kod i može djelovati pozitivno, negativno ili neutralno na organizam. Pozitivne mutacije stvaraju svojstva koja pomažu jedinki u borbi za resursima, a negativne mutacije odmažu u tome. Kombinacija prirodne selekcije, križanja i mutacije prilagođava vrste okolišu u kojem žive. Ovo je vrlo pojednostavljeno objašnjenje procesa razmnožavanja jedinki ali je dovoljno da posluži kao osnova za izradu skupine algoritama koji rješavaju optimizacijske probleme.

3.2.1. Jednostavni genetski algoritam

Genetskim algoritmom (De Jong, 2006) se najčešće rješavaju optimizacijski problemi koji se mogu opisati nekom funkcijom. Funkcije koje se optimiraju su oblika $f(\mathbf{x})$ gdje je $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ n-komponentni vektor. Potrebno je pronaći $\mathbf{x}^*$ koji maksimizira funkciju cilja (engl. *objective function*) ili minimizira funkciju troška (engl. *cost function*) . Primjer funkcije troška je $f(x) = x^2 + 5 - 10 \cdot \cos(2\pi x)$ koja je prikazana na slici 3.1.

Alogritam radi s populacijom rješenja $\mathbf{x}$ zvanim jedinke ili kromosomi. Svaka jedinka predstavlja jedno rješenje funkcije. Kvalitetu jedinke prikazujemo njenom dobrotom (engl. *fitness*) ili kaznom. Pošto se u ovom slučaju radi o minimizaciji funkcije, kazna se računa tako da se vrijednost jedinke uvrsti u funkciju. U ovom primjeru je jedinka bolja što je vrijednost funkcije manja (tražimo minimum funkcije). Iz popu-



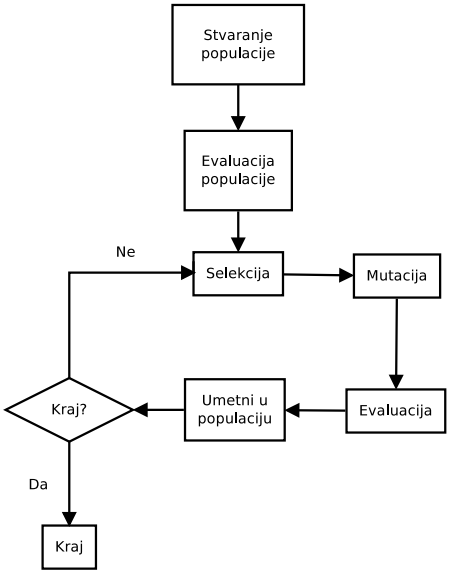
Slika 3.1: Funkcija kojoj treba naći minimum

lacije se zatim odabiru jedinke operatorom selekcije (engl. *selection*) koje će postati roditelji i proizvesti potomstvo. Potomstvo se stvara operatorom križanja (engl. *crossover*). Ideja je napraviti operator križanja tako da zadrži osobine oba roditelja, što bi u ovom primjeru značilo da će se dijete nalaziti negdje između dva roditelja na osi domene funkcije. Ovaj operator služi kako bi se bolje iskoristilo trenutno područje u kojem se nalaze roditelji. Dobiveno dijete prolazi kroz proces mutacije gdje se neki dio djeteta nasumično mijenja. Ovaj operator služi kako bi se dijete prebacilo u možda neki neistraženi dio prostora rješenja i tako proširio prostor pretrage. Nakon što se do kraja stvori dijete ono se stavi u populaciju i dalje tretira kao roditelj za sljedeću generaciju.

3.2.2. Vrste genetskog algoritma

Genetski algoritmi se dijele na dvije skupine ovisno o tome kako se nova rješenja stavljaju u novu generaciju. Po ovoj podjeli algoritme dijelimo na eliminacijski genetski algoritam i generacijski genetski algoritam.

Eliminacijski genetski algoritam radi tako da operator selekcije odabere jedinke za reprodukciju i zatim pomoću operatora križanja i mutacije stvori dijete. Dijete se zatim



Slika 3.2: Dijagram toka eliminacijskog genetskog algoritma

stavi u populaciju, a kako veličina populacije mora biti konstantna odabere se jedinka koju algoritam izbaci iz populacije. Ako želimo da nam prosječna kvaliteta rješenja u populaciji monotono raste, onda možemo dodati uvjet koji ne dozvoljava umetanje novog djeteta u populaciju, ako je lošije od trenutnog najlošije jedinke u populaciji, nego ga odbacuje. Pseudokod je prikazan u algoritmu 3.1 i dijagram toka na slici 3.2.

Algoritam 3.1 pseudokod eliminacijskog genetskog algoritma

$stvori\_populaciju(P)$
$evaluiraj\_populaciju(P)$
dok $i \leq size$ **radi**
  $R \leftarrow odaberi\_roditelje(P)$
  $d \leftarrow krizaj(R)$
  $d \leftarrow mutiraj(d)$
  $evaluiraj(d)$
  $umetni\_umjesto\_neke\_jedinke(P, d)$
kraj dok
vrati najbolja jedinka iz P

Generacijski genetski algoritam radi tako da sve novostvorene jedinke stavi u novu populaciju jednake veličine kao i originalna populacija. Kada popuni novu populaciju staru populaciju zamjeni novom i postupak krene ispočetka. Tako da se stare i nove jedinke ne sreću unutar jedne populacije. Algoritam u ovom obliku ne pamti najbolje pronađeno rješenje pa se može naknadno modificirati tako da se uvede elitizam.

Elitizam znači da će prethodno definiran broj najboljih jedinki biti preseljeno u novu generaciju. Pseudokod algoritma je prikazan u algoritmu 3.2, a dijagram toka je na slici 3.3

Algoritam 3.2 pseudokod Generacijskog genetskog algoritma

$P = stvori\_populaciju(size)$
$evaluiraj\_populaciju(P)$
dok $i \leq size$ **radi**
  $nova\_populacijaP' = \emptyset$
  dok $velicina(P') < size$ **radi**
    $R \leftarrow odaberi\_roditelje(P)$
    $d \leftarrow krizaj(R)$
    $d \leftarrow mutiraj(d)$
    $evaluiraj(d)$
    $umetni(P', d)$
  kraj dok
  $P = P'$
kraj dok
vrati najbolja jedinka iz P

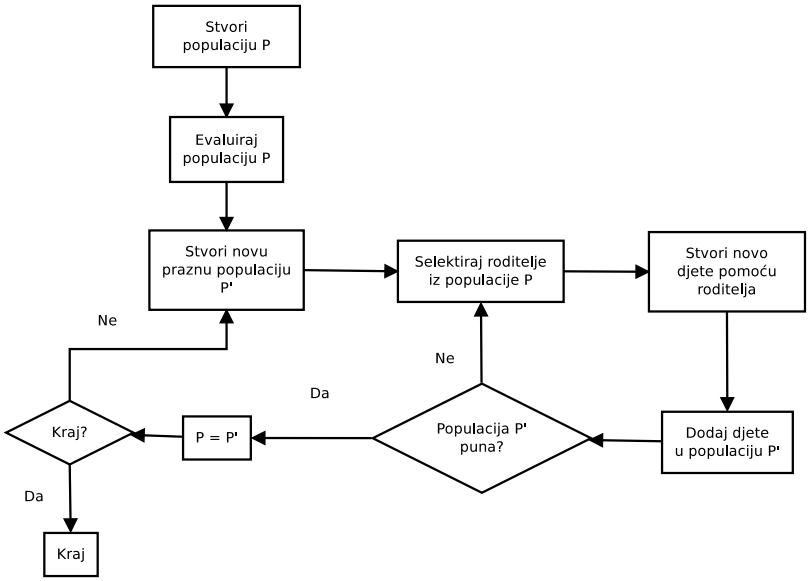
3.2.3. Jednostavni prikaz rješenja

Najjednostavniji prikaz rješenja je binarni prikaz. Binarni prikaz se sastoji od niza binarnih brojeva fiksne duljine kako je prikazano na slici 3.4. Brojevi koji se predstavljaju binarnim kromosomom se mogu kodirati običnim težinskim, Grayevim ili nekim drugim kodom.

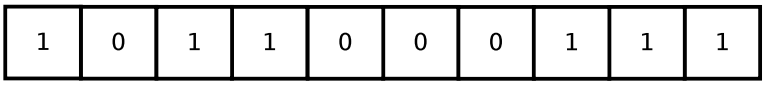
3.2.4. Selekcija

Postoji mnogo vrsta selekcija kod genetskog algoritma ali ovdje će se spomenuti samo dvije najznačajnije, k-turnirska i slučajna proporcionalna selekcija.

k-turnirska selekcija uzima nasumično k jedinki iz populacije i izabire onu s najboljom dobrotom. Postupak se ponavlja dok se ne odabere potreban broj jedinki. Pošto je postupak nasumičnog odabira računalno zahtjevan implementira se modificirana verzija k-turnirske selekcije. Modificirana verzija nasumično odabere k jedinki iz populacije i odabere n najboljih jedinki kao roditelje. Odabrane jedinke se križaju. Verzija koja se najčešće implementira je 3-turnirska selekcija.



Slika 3.3: Dijagram toka generacijskog genetskog algoritma



Slika 3.4: Prikaz binarnog kromosoma

Slučajna proporcionalna selekcija simulira kotač ruleta. Na intervalu [0, 1] se svakoj jedinki dodjeli podinterval obrnuto proporcionalan njezinoj kazni ili proporcionalan njezinoj dobroti. Podintervali se ne preklapaju jedan s drugim. Veličina podintervala koji jedinka $\mathbf{x}_i$ i dobije računa se po formuli 3.1 ako se radi o kazni, a ako se radi o dobroti koristi se formula 3.2.

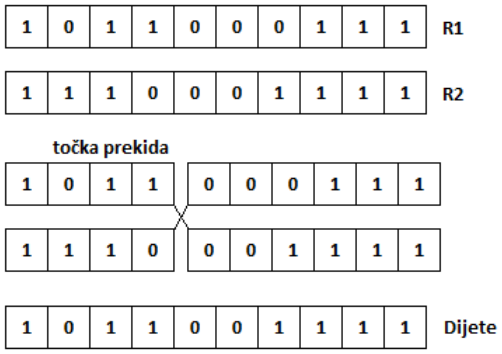
$$p(\mathbf{x}_i) = \frac{k_{max} - k_i}{N \cdot k_{max} - \sum_{n=1}^{N} k_n}, \tag{3.1}$$

$$p(\mathbf{x}_i) = \frac{d_i}{\sum_{n=1}^{N} d_n}, \tag{3.2}$$

gdje k_{max} označava kaznu najlošije jedinke, a k_i kaznu jedinke s indeksom i. d_i označava dobrotu jedinke s indeksom i, a N je broj jedinki u populaciji. Generiraju se dva nasumična broja u intervalu cijelog „kotača“ i izaberu se one jedinke na čiji su podinterval „pali“ nasumični brojevi.

3.2.5. Križanje binarnog kromosoma

Odabrani roditelji prolaze kroz proces križanja kako bi stvorili djecu jedinke. Često se algoritmu postavi vjerojatnost da se obavi križanje (koje se uobičajeno kreću od



Slika 3.5: Križanje s jednom točkom prekida

70% do 90%). Križanje je moguće obaviti na nekoliko načina i ovdje će biti opisana tri: križanje s jednom točkom prekida, križanje s više točaka prekida i uniformno križanje. Križanje će biti pokazano na primjeru dva roditelja, R_1 = 1011000111 i R_2 = 1110001111.

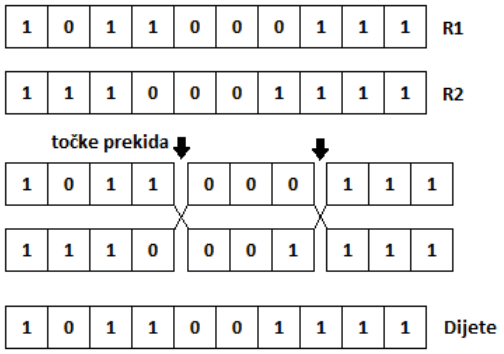
Križanje s jednom točkom prekida se vrši tako da se slučajno odabere točka prekida kromosoma. Dijete se stvori tako da se uzme dio kromosoma prvog roditelja do, a dio drugog roditelja od točke prekida. Ovaj način križanja je rijetko u uporabi zbog toga što ima pristranost pri križanju jer se prvi i zadnji element bilo roditelja nikada ne mogu naći u jednom djetetu. Proces je ilustriran na slici 3.5.

Križanje s više točaka prekida se vrši slično kao križanje s jednom točkom prekida samo što umjesto jedne točke odabere se nekoliko točaka. Broj točaka može biti od 1 do k-1, gdje je k duljina kromosoma. Primjer s dvije točke prekida iza 4. i 7. bita prikazan je na slici 3.6.

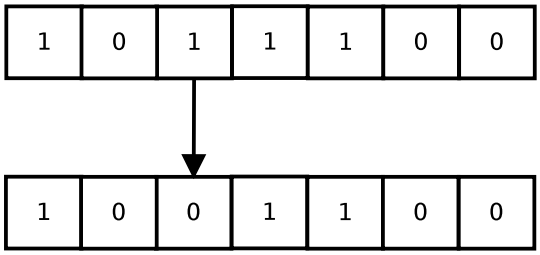
Uniformno križanje se može pokazati kao križanje s k-1 točki prekida. Roditelji se raspadnu iza svakog bita, a dijete nasumično bira koji će bit uzeti.

3.2.6. Mutacija binarnog kromosoma

Operator mutacije na binarnim kromosomom (slika 3.7) djeluje tako da slučajnim procesom odabere koji bit će mutirati i kada ih odabere promjeni im vrijednost. Vjerojatnost da će bit biti mutiran kreće se oko 1% i utvrđuje se eksperimentalno. Veće vjerojatnosti bi destruktivno utjecale na pretragu jer bi anulirale efekt križanja i genetski algoritam pretvorile u nasumičnu pretragu.



Slika 3.6: Križanje s više točkaka prekida



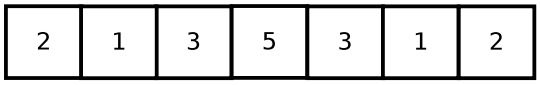
Slika 3.7: Prikaz mutacije koja se dogodila na trećem bitu

3.2.7. Implementacija operatora genetskog algoritma za rješavanje problema izrade rasporeda

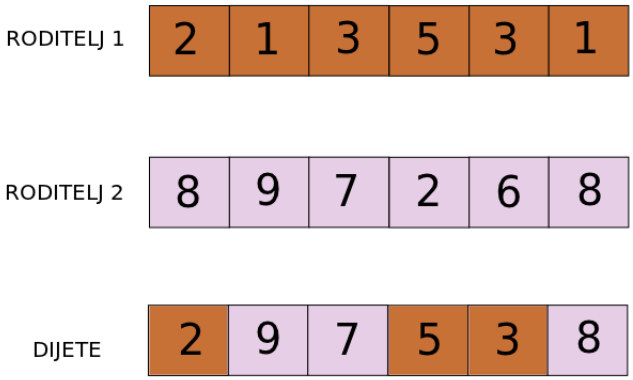
Kako bi genetski algoritam mogli upotrijebiti za rješavanje problema izrade rasporeda neki od operatora će morati biti prilagođeni problemu.

Binarni prikaz nije pogodan za prikaz rasporeda međuispita, zato je potrebno osmisliti zapis rasporeda koji je pogodan za jednostavno i učinkovito križanje i mutiranje, a da zadrži učinak tih operatora. Odabran je jednostavni zapis pomoću vektora cjelobrojnih vrijednosti (slika 3.8). Svaki indeks polja označava i broj pomoću kojega je kodiran međuispit. Dakle, indeks polja 2 označava da to mjesto u vektoru pripada međuispitu koji je kodiran brojem 2. Vrijednost koja je zapisana na pojedinom mjestu u vektoru označava broj kojim je kodiran pojedini termin. Vektor možemo označiti kao $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$. Vrijednost $x_k, 0 < k \leq n$, označava broj termina u kojem je međuispit k smješten.

Operator križanja je vrlo sličan uniformnom križanju. Kako je u ranije opisano, rješenje se prikazuje pomoću vektora brojeva, gdje brojevi unutar vektora označavaju redni broj termina, a indeks pozicije u vektoru označava redni broj međuispita. Križanje se obavlja tako da se kromosom djeteta najprije u cijelosti popuni s elementima



Slika 3.8: Prikaz kromosoma prilagođenog za problem izrade rasporeda. Indeks mjesta u vektoru označava međuispit, a vrijednosti unutar vektora označavaju termine u kojima se ispiti nalaze.



Slika 3.9: Križanje dvije jedinke. Dijete se stvori kopiranjem 1. roditelja, zatim se nasumično odabrani elementi drugog roditelja preslikaju u dijete. Dijete mora poštivati jaka ograničenja.

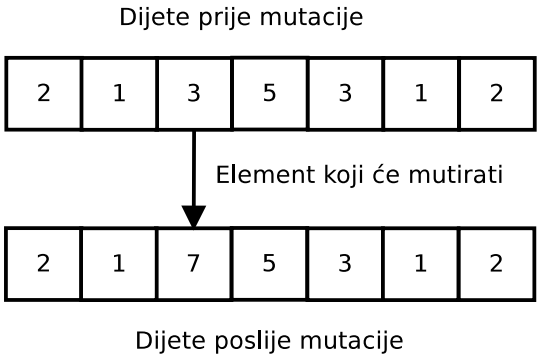
prvog roditelja, a nakon toga se nasumični elementi kromosoma popune s rednim brojevima termina drugog roditelja i to samo ako se time neće prekršiti čvrsta ograničenja. Vjerojatnost izbora svakog elementa iz kromosoma drugog roditelja je 50% (slika 3.9).

Mutacija radi tako da se izgenerira nasumični broj iz binomne razdiobe gdje je parametar razdiobe vjerojatnost mutacije pomnožena s veličinom kromosoma. Ovako generirani broj označava količinu međuispita koji će promijeniti termin. Ispiti koji se sele u drugi termin se odabiru nasumično. Termin se isto odabire nasumično, ali samo ako takva selidba neće prekršiti čvrsta ograničenja rasporeda. Vjerojatnost mutacije se definira u konfiguracijskoj datoteci (slika 3.10).

3.3. Algoritam harmonijske pretrage

3.3.1. Kanonska verzija algoritma harmonijske pretrage

Algoritam harmonijske pretrage (engl. *Harmony Search Algorithm*) je još jedan u nizu algoritama evolucijskog računanja inspiriranih prirodnim pojavama (Geem et al., 2001). Rad algoritma je temeljen na sličnom principu na kojem muzičari improviziraju nove melodije. Nove melodije se stvaraju tako da se dio nota uzme iz skupa ranije poznatih melodija, tzv. harmonijska memorija, a dio nota se improvizira na-



Slika 3.10: Mutacija jedinke. Izabere se nasumično element jedinke, zatim se umjesto njega stavi neka nasumična vrijednost koja poštuje čvrsta ograničenja.

sumično. Nakon što se odsvira nova melodija (rješenje), evaluira se njena estetska kvaliteta (funkcija dobrote) i uspoređuje se s već poznatim melodijama (harmonijska memorija). Ako je nova melodija bolja od najgore iz harmonijske memorije, onda će je nova melodija zamijeniti. Ovakav proces stvaranja glazbe je analogan pronalaženju najboljeg rješenja kod inženjerskih problema tj. pronalaženju maksimuma neke funkcije dobrote. Algoritam spada u skupinu algoritama evolucijskog računanja i vrlo je sličan genetskom algoritmu. Sličan je utoliko što koristi populaciju rješenja stvorenih pretragom prostora rješenja kao temelj za izradu novih rješenja. Također ima operator koji je sličan mutaciji kod genetskog algoritma i služi kako bi se proširio prostor pretrage i kako bi se algoritam oporavio od mogućeg pronalaska lokalnog optimuma. Neke od primjena algoritma harmonijske pretrage su: generalizirani orijentacijski problem (Geem et al., 2005), rješavanje Sudoku slagalice (Geem, 2007), sustavi za vizualno praćenje (Fourie, 2010), problem usmjeravanja vozila (Zong Woo Geem, 2005) te dizajniranje sustava za distribuciju vode (Geem, 2006). Široka uporaba algoritma harmonijske pretrage u različitim inženjerskim problemima je poslužila kao opravdanje za njegovo korištenje kod problema izrade rasporeda međuispita. Slika 3.11 pokazuje jednostavan dijagram toka algoritma harmonijske pretrage. Algoritam 3.3 pokazuje pseudokod osnovnog 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 3.1) 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.

• *vjerojatnost prihvaćanja komponente iz harmonijske memorije* (p_{acc}) – vjerojat-

Algoritam 3.3 pseudokod kanonskog algoritma harmonijske pretrage

Definiraj funkciju dobrote $f(\mathbf{x}), \mathbf{x} = (x_1, x_2, \ldots, x_{size})^T$

Definiraj vjerojatnost preuzimanja iz $harmonijska\_memorija$ (p_{acc})

Definiraj vjerojatnost naknadnog ugađanja note (p_{pa})

Definiraj veličinu populacije $(popSize)$

Generiraj $harmonijska\_memorija$ s nasumičnim harmonijama

$harmonijska\_memorija = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{popSize})$

$evaluiraj(harmonijska\_memorija)$

dok $\neg uvijet\_zaustavljanja$ **radi**

  dok $i \leq size$ **radi**

    ako $rand < p_{acc}$ **tada**

      $x_i^{nova\_harmonija} \leftarrow x_i; x_i \in \mathbf{x}_k; \mathbf{x}_k \in harmonijska\_memorija$

      ako $rand < p_{pa}$ **tada**

        $x_i^{nova\_harmonija} \leftarrow i + \varepsilon; \varepsilon$ je mala proizvoljna vrijednost

      kraj ako

    inače

      izaberi nasumičnu vrijednost za $x_i^{nova\_harmonija}$

    kraj ako

    $\mathbf{x}_{nova\_harmonija} \leftarrow (x_1^{nova\_harmonija}, x_2^{nova\_harmoija}, \ldots, x_{size}^{nova\_harmonija})^T$

    $evaluiraj(\mathbf{x}_{nova\_harmonija})$

  kraj dok

  $\mathbf{x}_k \leftarrow harmonija\_s\_najmanjom\_dobrotom(harmonijska\_memorija)$

  ako $f(\mathbf{x}_{nova\_harmonija}) > f(\mathbf{x}_k);$ **tada**

    stavi $\mathbf{x}_{nova\_harmonija}$ u $harmonijska\_memorija$ umjesto $\mathbf{x}_k$

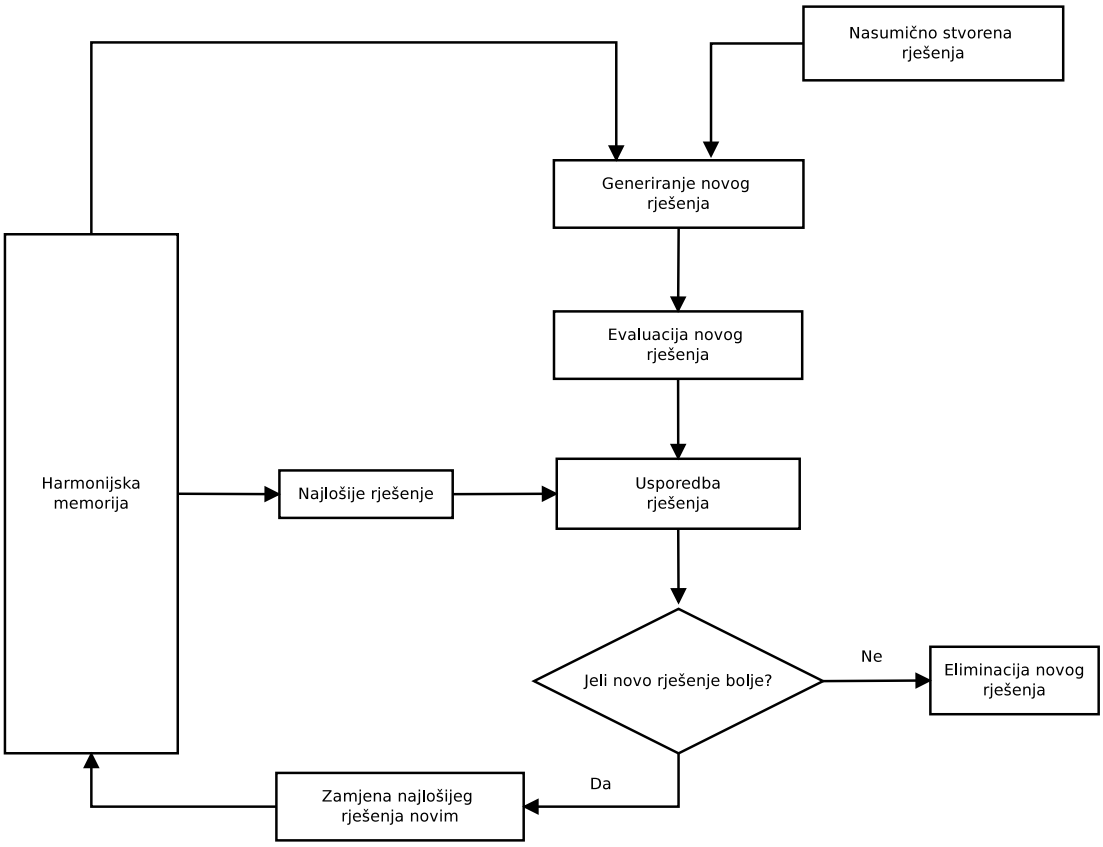
  inače

    odbaci $\mathbf{x}_{nova\_harmonija}$

  kraj ako

kraj dok

vrati $\mathbf{x}_{najbolja}$



Slika 3.11: Dijagram toka algoritma harmonijske pretrage

nost da se komponenta (termin, nota) nove melodije preuzme iz harmonijske memorije.

• *vjerojatnost naknadnog ugađanja komponente* (p_{pa}) – vjerojatnost da se nota preuzeta harmonijske memorije ugodi, tj. umjesto nje uzme neka druga iz okoline trenutne note.

• *veličina harmonijske memorije* $(size)$ – veličina harmonijske memorije. Slično kao veličina populacije kod ostalih populacijskih algoritama

Drugi korak: Improvizira se nova melodija. Nova melodija se stvara notu po notu (komponentu po komponentu). Svaka nova komponenta x_i^{nova} se odabere iz harmonijske memorije $x_i^{odabrana}$ s vjerojatnošću p_{acc} (obično $0.7 < p_{acc} < 1$). Na primjer, ako je p_{acc} = 0.9, to znači da će 90% komponenti u novoj melodiji biti preuzeto iz harmonijske memorije. One note koje su ostale nepopunjene (vjerojatnost: $1 - p_{acc}$), nakon što je melodija izgrađena pomoću harmonijske memorije, moraju se popuniti nasumično. Ovaj operator je sličan operatoru mutacije kod genetskog algoritma. Proširuje prostor pretrage za optimalnim rješenjem da se algoritam ne bi zaglavio u nekom od lokalnih optimuma.

Treći korak: Nakon što se proizvede nova melodija provuče se kroz ugađanje nota

tj. svaka nota u novoj melodiji preuzeta iz harmonijske memorije ima vjerojatnost p_{pa} (najčešće $0 < p_{pa} < 0.1$) da se ugodi. Ugađanje znači da će se postojeća nota u novoj melodiji zamijeniti s nekom iz neposredne blizine.

Četvrti 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.

Peti korak: Nove se melodije proizvode sve dok algoritam ne dobije zahtjev za zaustavljanje, a kada ga dobije, algoritam vrati najbolje pronađeno rješenje do tog trenutka.

3.3.2. Prilagodba algoritma harmonijske pretrage problemu izrade rasporeda

Kako kanonska verzija algoritma harmonijske pretrage nije pogodna za kombinatorne probleme kakav je problem izrade rasporeda bilo je potrebno prilagoditi neke od operatora. Prikaz rješenja u osnovnoj harmonijskog pretrazi se sastoji od vektora **x** jednodimenzionalnih vrijednosti x_i. Kod prilagodbe algoritma za problem izrade rasporeda međuispita prikaz rješenja je ostvaren na isti način kako je opisano u odjeljku 3.2.7.

Početna populacija se u kanonskom algoritmu harmonijske pretrage proizvodi nasumično. U algoritmu harmonijske pretrage koja je implementirana za problem izrade rasporeda se kod izrade početne populacije pazi da se u svakom trenutku poštuju čvrsta ograničenja.

Smatra se da je svaka vrijednost x_i neovisna od bilo koje druge vrijednosti x_j, $i \neq j$. To znači da se iste te vrijednosti mogu mijenjati ne vodeći računa o ostalim vrijednostima. To se vidi kod operacije izrade nove harmonije jer nema posebnih restrikcija kod uzimanja komponenti iz harmonijske memorije. Takav pristup nije moguć ako vrijednosti x_i i x_j, $i \neq j$ nisu neovisne. Kod ove implementacije problema izrade rasporeda postoji uvjet da rješenja u algoritmu uvijek zadovoljavaju čvrsta ograničenja. Može se dogoditi da prethodno smješteni kolegij i u termin t ne dozvoljava da neki drugi kolegij j bude smješten u isti taj termin t jer bi npr. kolegiji i i j imali neke zajedničke studente. To znači da se operacija izrade nove melodije morala implementirati drugačije. Za svaki se kolegij, neposredno prije njegove izgradnje, pronađe skup termina u koje je smješten u nekom od rješenja u harmonijskoj memoriji. Od svih tih termina se uzme podskup termina koji neće kršiti čvrsta ograničenja u novoj melodiji. Iz tog podskupa se onda uzme nasumični termin u koji će kolegij biti smješten. Glavni

problem koji se pojavljivao kod izrade nove melodije je taj što se događalo da se nova melodija izgradi do nekog indeksa i, $i < size$, a da kolegiji na indeksima većim od i nemaju više termina na kojima ne bi kršili čvrsta ograničenja. Indeksi kolegija se, zbog toga, ne izgrađuju redom od najmanjeg prema većima, nego se popunjavaju prvo indeksi onih kolegija koji najčešće ostanu bez termina (algoritam 3.4).

Operacija ugađanja note koja notama izgrađenima pomoću harmonijske memorije dodjeljuje neku vrijednost iz okolice trenutne je također posebno implementirana. Okolica termina t je definirana kao skup svih termina koji se nalaze u istom danu u kojem se nalazi termin t. Kada se kolegiju termin u kojem je smješten „ugađa“, onda se preseli u neki drugi termin u istom danu koji ne krši čvrsta ograničenja.

Oni kolegiji koji se ne smještaju u termine pomoću harmonijske memorije se smjeste u nasumične termine u kojima neće kršiti čvrsta ograničenja.

Uvjet zaustavljanja algoritma je isti kao i kod ranije spomenutih algoritama. Definiraju se prije pokretanja algoritma, a mogu ovisiti o stagnaciji pretrage ili o proteklom vremenu.

Algoritam 3.4 pseudokod izgradnje nove melodije

dok $i \leq size$ **radi**
  sortiraj indekse kolegija po broju puta koliko su ostali bez termina pri izgradnji i stvori $sortirana\_lista\_indeksa$
  $indeks \leftarrow sortirana\_lista\_indeksa(i)$
  $lista\_dozvoljenih\_termina \leftarrow stvori\_listu\_dozvoljenih\_termina(indeks)$
  ako $lista\_dozvoljenih\_termina = \emptyset$ **tada**
    zabilježi na kojem se indeksu dogodio prekid
    pokreni petlju ispočetka
  inače
    $x_{indeks} \leftarrow nasumicni\_indeks(lista\_dozvoljenih\_termina)$
  kraj ako
kraj dok

3.4. Jednostavni imunološki algoritam

Mnoge pojave u prirodi koje pokazuju sposobnost učenja i prilagodbe mogu poslužiti kao inspiracija za razvoj sustava za klasifikaciju i sustava za optimizaciju. Jedna od tih pojava je i imunološki sustav razvijenih životinja. Imunološki sustav ima sposobnost obrane od nekih infekcija s kojima se nikada nije susreo u prošlosti. Područje

razvoja imunoloških optimizacijskih algoritama je započelo s radovima (Bersini i Varela, 1991) i (Farmer et al., 1986). Ovdje će biti opisan samo mali dio sustava za obranu organizma s vrlo pojednostavnjenim opisom interakcija ali dovoljan kao opravdanje i motivacija za daljnji razvoj optimizacijskih algoritama.

Tijelo posjeduje skupinu imunoloških stanica koje dobije rođenjem. Kada strana tijela nastane krvotok aktivira se ovaj urođeni imunološki sustav. Ove stanice tada napadnu uzrok infekcije i ako uspiju ga i unište. Ako osnovni sustav za obranu organizma ne uspije suzbiti infekciju aktivira se sustav koji se pokušava prilagoditi (naučiti) infekciju i na taj način je suzbiti i razviti imunitet protiv slične infekcije. Taj sekundarni sustav se sastoji od stanica, tzv. B-limfocita, koji se aktiviraju prilikom pojave nepoznate infekcije. Strana tijela koja uzrokuju infekciju se zovu antigeni. B-limfociti uništavaju antigene tako što ih receptorima vežu na sebe ili proizvode posebne proteine (antitijela) s receptorima koji vežu antigene i uništavaju ih. Vezanje antitijela za antigene će se dogoditi ukoliko su receptori antitijela kompatibilni s antigenom. Mjera ove kompatibilnosti se zove afinitet. Ukoliko se uspiju povezati antigen i antitijelo, tada se aktivira proces umnožavanja B-limfocita koji proizvode ta antitijela. Prilikom umnožavanja se javljaju nasumične mutacije u genetskom kodu B-limfocita. Neke kopije će, zbog toga, imati jači, a neke slabiji afinitet prema antigenu. Oni B-limfociti s jačim afinitetom prema antigenu će se dalje umnažati. Stanice koje imaju slabiji afinitet će ući u proces apoptoze (programirano samouništenje stanice). Proces prilagođavanja antigenima se nastavlja dok god imunološki sustav ne uništi antigene. Memorijski B-limfociti nastaju nakon infekcije i profilirani su upravo za infekciju koja je netom uništena. Ovakav sustav „pamti“ prijašnju infekciju i u slučaju da se slični antigeni pojave u budućnosti spreman je boriti se protiv takve infekcije bez ponavljanja procesa prilagodbe. Stvaranje memorijskih B-limfocita je osnova za sustav cijepljenja kojim su istrijebljene mnoge bolesti do danas.

Ovaj princip borbe protiv bolesti je poslužio kao osnova za cijeli skup optimizacijskih alogritma koji se zovu Umjetni imunološki sustavi (engl. *AIS - Artificial Immune Systems*). Za potrebe sustava koji rješava problem izrade rasporeda je implementiran jedan iz skupine umjetnih imunoloških sustava zvan Jednostavni imunološki algoritam (engl. *SIA - Simple Immune Algorithm*). Algoritam je opisan u radovima (Cutello i Nicosia, 2002a), (Cutello i Nicosia, 2002b) i (Cutello i Nicosia, 2004).

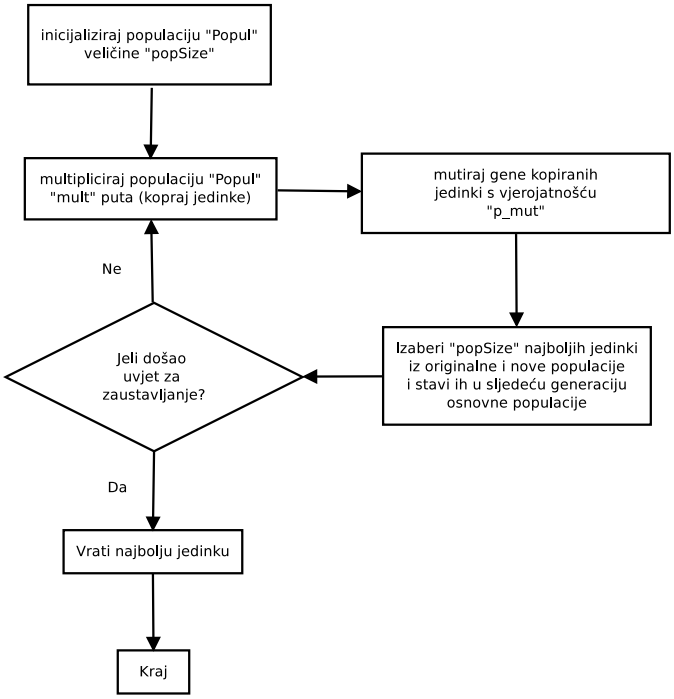
SIA je populacijski algoritam koji sadrži populaciju antitijela. Antitijela u terminologiji imunoloških algoritama znače rješenja (raspored) problema koji se optimira. Antigen u ovom slučaju predstavlja problem ili funkciju koju se pokušava optimirati. Afinitet antitijela prema antigenu predstavlja dobrotu rješenja u odnosu na problem

koji se optimira. Rješenje u kanonskom obliku algoritma predstavlja niz binarnih vrijednosti, no algoritam je lako proširiv na većinu najčešćih reprezentacija rješenja s kojima se susrećemo u praksi. Pseudokod algoritma je prikazan u Algoritmu 3.5.

Algoritam 3.5 pseudokod jednostavnog imunološkog algoritma

Definiraj funkciju afiniteta $f(\mathbf{x}), \mathbf{x} = (x_1, x_2, \ldots, x_{size})^T$
Definiraj faktor umnažanja antitijela $(mult)$
Definiraj vjerojatnost mutacije gena u antitijelu (p_{mut})
Definiraj veličinu populacije antitijela $(popSize)$
Generiraj populaciju $Popul$ s nasumičnim antitijelima
$Popul = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{popSize})$
$evaluiraj(Popul, f(\mathbf{x}))$
dok $\neg uvijet\_zaustavljanja$ **radi**
  $Popul^{clo} = kloniraj(Popul, mult)$
  $Popul^{mut} = mutiraj(Popul^{clo}, p_{mut})$
  $evaluiraj(Popul^{mut}, f(\mathbf{x}))$
  $Popul \leftarrow sortiraj\_i\_odaberi(Popul^{mut} \cup Popul)$
kraj dok

Algoritam prvo definira parametre koji su mu potrebni za rad. U ovom slučaju su to parametri $mult$, p_{mut} i $popSize$. $mult$ je faktor za koji povećavamo populaciju osnovne veličine $popSize$. p_{mut} predstavlja vjerojatnost mutacije gena u novokloniranom antitijelu (rješenju). Nakon što su definirani osnovni parametri, stvara se početna populacija $Popul$. Početna populacija se stvara nasumičnim stvaranjem antitijela. Implementacija za izradu rasporeda međuispita stvara populaciju na isti način kao i svi do sada opisani algoritmi. Tijekom procesa stvaranja populacije sve se jednike evaluiraju funkcijom dobrote (ili kazne u našem slučaju). Glavni dio algoritma je petlja koja se ponavlja dok ne dođe zahtjev za zaustavljanje algoritma. U našem slučaju to je ili stagnacija pretrage algoritma ili proteklo vrijeme. U petlji se odvija nekoliko koraka. Prvo se stvori nova populacija antitijela $Popul^{clo}$ koja je veličine $popSize \cdot mult$ i sadrži $mult$ kopija svakog antitijela iz osnovne populacije $Popul$. Geni antitijela u novoj populaciji se mutiraju s vjerojatnošću p_{mut}. To znači da se kolegiji koji se odluče mutirati stavljaju u nasumičan termin u kojem ne krše čvrsta ograničenja rasporeda. Takva mutirana populacija je označena s $Popul^{mut}$ i još uvijek je veličine $popSize \cdot mult$. Nova antitijela se evaluiraju funkcijom dobrote (kazne). Iz unije populacije novih antigena $Popul^{mut}$ i antitijela iz osnovne populacije $Popul$ se izaberu najboljih $popSize$ antitijela i spreme se u populaciju $Popul$. Ostala antitijela koji nisu odabrani se eliminiraju.



Slika 3.12: Dijagram toka algoritma Jednostavnog imunološkog algoritma

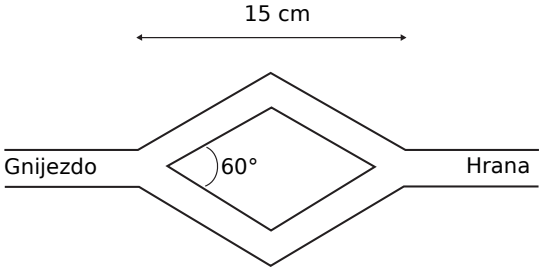
Nakon što algoritam završi s radom vraća antitijelo s najvećim afinitetom prema antigenu tj. rješenje koje ima najbolju funkciju dobrote. Dijagram toka SIA algoritma je prikazan na slici 3.12.

3.5. Optimizacija kolonijom mrava

3.5.1. Uvod

Optimizacija kolonijom mrava (engl. *Ant Colony Optimisation*) je metaheuristički algoritam koji služi za optimizaciju problema koji se mogu prikazati kao putanja kroz težinski graf (Dorigo i Stützle, 2004). Glavna ideja je nastala promatranjem na koji način mravlje kolonije pronalaze najbliži put do izvora hrane. Algoritmi se sastoje od skupa agenata zvanih umjetni mravi koji iterativno pretražuju prostor rješenja u potrazi za optimalnim ili skoro optimalnim rješenjem. Mravi kretanjem kroz graf inkrementalno grade rješenje. Izgradnja rješenja je stohastički proces, a vjerojatnosti se usklađuju uz pomoć feromonskog modela, što znači da se vjerojatnosti koje vode stohastički proces mijenjaju kako mravi izgrađuju rješenja. Algoritmi optimizacije kolonijom mrava također spadaju u populacijske metaheurističke algoritme koji se koriste za rješavanje teških optimizacijskih problema.

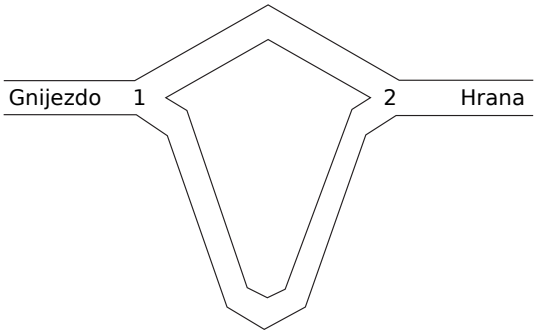
Eksperiment koji je dao opravdanje za uporabu ovog modela je onaj koji su izveli



Slika 3.13: Slika dvokrakog mosta s jednakim krakovima

(Goss et al., 1989). Sastojao se od izgradnje dvokrakog mosta s mravljom kolonijom na jednoj strani i izvorom hrane na drugoj strani mosta, kao što je vidljivo na slici 3.13. U prvom pokusu su oba dva kraka mosta bila jednake duljine. U nekoliko iteracija eksperimenta se pokazalo da će mravi odabrati jedan od krakova u 50% slučajeva. Ovaj rezultat se objašnjava činjenicom da će ostavljeni feromon pozitivno utjecati na odabir puta kojim će se mrav kretati. Mravi se na račvanju mosta moraju odlučiti kojim će putem krenuti. Ako je pokus na početku izvođenja tj. nema feromona, onda će mrav odabrati svoj put nasumično s vjerojatnošću 0.5 za bilo koji od krakova mosta. Svaki mrav će hodajući ostavljati feromonski trag iza sebe. Ako postoji feromonski trag na putu mrava, onda će kretanje mrava biti pod utjecajem tog traga jer mravi imaju tendenciju pratiti taj trag. Fluktuacije u mravljem odabiru lijevog ili desnog puta će učiniti to da će na jednom od krakova biti više feromona nego na drugom. Jači feromonski trag će privući više mrava koji će ostavljati više feromona. Ovaj kružni proces ima za posljedicu da će nakon određenog vremena gotovo svi mravi ići samo jednim krakom mosta. Bitnu ulogu ima i činjenica da feromonski trag ima relativno kratak vijek trajanja, tako da će trag izčeznuti s onog puta kojim se mravi ne kreću ili kreću jako rijetko.

Druga verzija eksperimenta je sadržavala dvokraki most kojem je jedan krak dvostruko dulji od drugog (slika 3.14). Ovdje su mravi svaki put odabirali kraći put za dopremanje hrane u gnijezdo. Ovo ponašanje se pojavilo zato što su mravi koji su nasumično odabrali kraći put do hrane prije stigli na drugu stranu mosta. Na povratku prema gnijezdu (točka 2 na slici 3.14) su morali odabrati krak kojim će se vratiti natrag. Kraći krak u ovom slučaju će imati više feromona zbog toga što se mravi češće kreću po njemu pa će se povratnički mravi vjerojatnije kretati tim putem i tako ostavljati novi trag. Ova pojava će nakon nekog vremena imati za rezultat da će se svi mravi kretati kraćim putem. Takvo ponašanje temelj je onoga što danas znamo pod nazivom izranjajuća inteligencija (engl. *Emerging intelligence*).



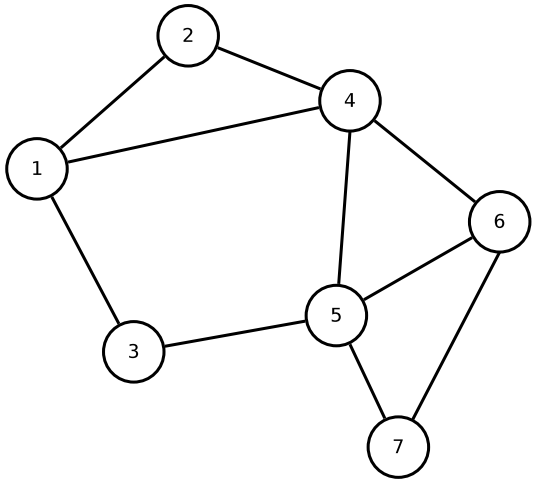
Slika 3.14: Slika dvokrakog mosta s jednim kraćim krakom

3.5.2. Matematički model optimizacije mravljom kolonijom

Sposobnost mrava da pomoću feromonske komunikacije nađe trag je bilo potrebno matematički modelirati (Dorigo i Stützle, 2004). Mravi ostavljaju trag krećući se u oba smjera (od mravinjaka do hrane i obrnuto), a neke vrste mrava ostavljaju jači feromonski trag ako je izvor hrane bogatiji. Ovakva dinamika postavljanja feromona i nestajanja feromona je matematički vrlo složeno modelirati i simulirati. Zakonitosti koje su ovdje uočene našle su, u nešto jednostavnijem obliku, primjenu u nizu mravljih algoritama. Na slici 3.15 je primjer grafa nad kojim treba pronaći najkraći put od čvora 1 do čvora 7. Bridovi grafa predstavljaju dozvoljene (prohodne) prijelaze između čvorova. Ideja algoritma je jednostavna. Inicijalno se vrijednosti feromona postavljaju na neku unaprijed određenu vrijednost. Kada se pusti mrav iz prvog čvora on mora odabrati u koji će čvor prijeći. Kako mu bridovi grafa definiraju dozvoljene prijelaze, skup čvorova u koje može prijeći se sastoji od čvorova 2, 3 i 4. To je označeno s $N_1 = \{2, 3, 4\}$. Odluku o tome u koji čvor će prijeći će učiniti nasumično. Vjerojatnosti odabira pojedinog čvora su označene s p_{ij}, gdje i označava čvor u kojem se mrav nalazi, a j označava čvor u koji može preći. Vrijednost p_{ij} se računa na sljedeći način:

$$p_{ij}^k = \frac{\tau_{ij}}{\sum_{l \in N_i^k} \tau_{il}}, \textbf{ako } j \in N_i^k \tag{3.3}$$

gdje k označava redni broj mrava, τ_{ij} označava jačinu feromonskog traga na prijelazu iz čvora i u čvor j. N_i^k predstavlja skup susjednih čvorova čvora i za mrava k (ne moraju nužno svi mravi imati iste prijelaze između čvorova). Kad se izračunaju vjerojatnosti svih prijelaza na temelju njih se stohastički odabire jedan. Za mrava se postupak ponavlja dok ne dođe do svog cilja. Pri dolasku mrava do odredišta računa se duljina puta koju je obavio i temeljem duljine puta se računa količina feromona koju će postaviti, a količina feromona je obrnuto proporcionalna duljini pređenog puta. Fe-



Slika 3.15: Primjer grafa nad kojim se pronalazi najkraći put

romon se osvježava na sljedeći način:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau^k, \tag{3.4}$$

$$\Delta\tau^k = \frac{1}{L^k}, \tag{3.5}$$

gdje L^k označava duljinu puta mrava k. Potrebno je još i „ispariti“ određenu količinu feromona s bridova grafa, a to se čini na sljedeći način:

$$\tau_{ij} \leftarrow \tau_{ij}(1 - \rho), \tag{3.6}$$

gdje ρ označava brzinu isparavanja i ima vrijednost iz intervala <0, 1>. Ovakav pristup obilaska grafa ima nedostatak da omogućava mravima da rade cikluse što samo usporava rad algoritma. Algoritam se može preraditi da dozvoljava samo prijelaze u dosad ne posjećene čvorove grafa.

Algoritam radi s m mrava i svakog mrava pusti da izgradi rješenje. Nakon što su mravi izgradili svoja rješenja, sva izgrađena rješenja se evaluiraju. Temeljem duljine puta izabire se $n, n \leq m$ najboljih mrava i ti mravi ažuriraju feromone na svom putu (bridove koje su prešli). Nakon ažuriranja provede se isparavanje feromona na svim bridovima.

3.5.3. $\mathcal{MAX} - \mathcal{MIN}$ sustav mrava

Za potrebe izrade rasporeda međuispita implementiran je jedan iz skupine mravljih algoritama imena $\mathcal{MAX} - \mathcal{MIN}$ sustav mrava (engl. *MMAS - $\mathcal{MAX} - \mathcal{MIN}$ ant*

system). Ova verzija algoritma je zapravo unaprijeđena verzija originalnog sustava mrava koju su predložili (**?**). MMAS algoritam se razlikuje od osnovne verzije u dvije stvari. Prvo, samo najbolji mrav ostavlja feromonski trag na bridovima. Drugo, maksimalne i minimalne vrijednosti τ su eksplicitno ograničene od dizajnera algoritma. Formula za ažuriranje feromona izgleda ovako:

$$\tau_{ij} \leftarrow (1-\rho) \cdot \tau_{ij} + \Delta\tau^{najbolji}, \tag{3.7}$$

$$\Delta\tau^{k} = \frac{1}{L^{najbolji}}, \tag{3.8}$$

gdje τ_{ij} označava brid između čvorova i i j koji je najbolji mrav prešao na svom putu. ρ označava brzinu isparavanja, a $L^{najbolji}$ označava duljinu puta najboljeg mrava. Vrijednosti τ_{ij} su ograničene eksplicitno zadanim vrijednostima, τ_{max} s gornje strane i τ_{min} s donje strane. Ako je vrijednost $\tau_{ij} > \tau_{max}$, onda se prepravlja $\tau_{ij} = \tau_{max}$. Ako je vrijednost $\tau_{ij} < \tau_{min}$, onda se prepravlja $\tau_{ij} = \tau_{min}$. Algoritam ima još jednu preinaku koja unosi heurističku procijenu duljine puta kod računanja vjerojatnosti prijelaza između bridova. Preinačena verzija formula za izračun vjerojatnosti prijelaza glasi:

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in N_i^k} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}}, akoj \in N_i^k \tag{3.9}$$

gdje je η_{ij} označava heurističku procjenu duljine puta između čvora i u čvor j. Konstante α i β označavaju važnost heurističkog i feromonskog pretraživanja. Kod implementacije algoritma za rješavanje problema izrade rasporeda se ne koristi ova preinaka jer ne postoji jednostavna heuristička procjena kvalitete odabranog čvora.

3.5.4. Prilagodba $\mathcal{MAX} - \mathcal{MIN}$ sustav mrava za rješavanje problema izrade rasporeda

Kako bi se problem izrade rasporeda prilagodio rješavanju problema izrade rasporeda međuispita trebalo je prikazati problem kao neku vrstu prolaska kroz graf. Mravi koji putuju kroz graf će redom obilaziti međuispite od najmanjeg indeksa i ka većem i dodjeljivati im termine j sukladno vjerojatnosti koju izračunaju. Graf je najjednostavnije prikazati pomoću tablice na slici 3.1.

Vrijednost τ_{ij} označava jačinu feromonskog traga između međuispita i i termina j. Naravno, nisu svi prijelazi između međuispita i termina dozvoljeni. Prije svakog prijelaza mrav provjerava hoće li prijelaz prekršiti čvrsta ograničebnja rasporeda. Ako prijelaz krši čvrsta ograničenja onda $p_{ij} \leftarrow 0$, inače će vjerojatnost p_{ij} biti izračunata sukladno formuli 3.3. Ako se dogodi da mrav nije izgradio raspored do kraja a ne može

	$kolegij_1$	$kolegij_1$	$\cdots$	$kolegij_n$
$termin_1$	τ_{11}	τ_{11}	$\cdots$	τ_{n1}
$termin_2$	τ_{12}	τ_{22}	$\cdots$	τ_{n2}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$termin_m$	τ_{1m}	τ_{2m}	$\cdots$	τ_{nm}

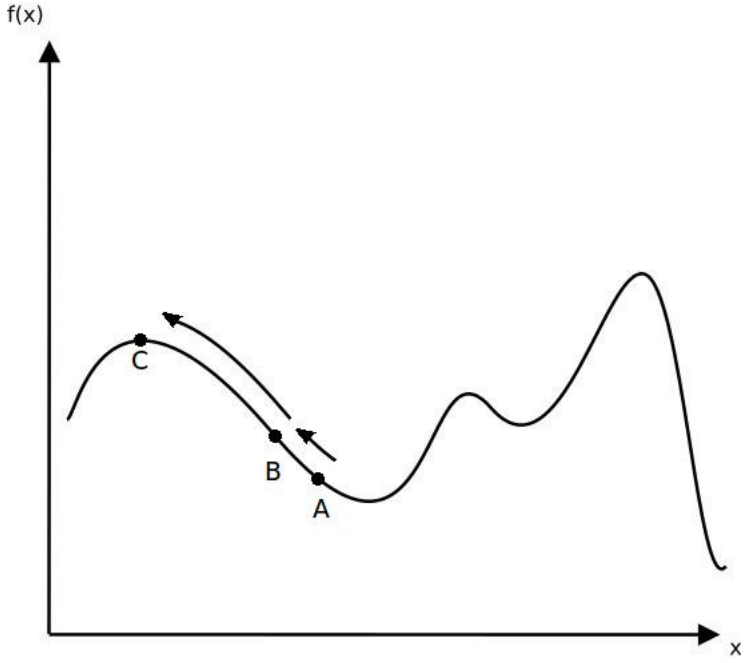
Tablica 3.1: Tablica vrijednosti fermona između međuispita i termina

dodijeliti niti jedan termin sljedećem međuispitu, onda se briše cijeli raspored i kreće graditi raspored ispočetka dok ne uspije. Na ovaj način je prerađen $MAX - MIN$ sustav mrava za rješavanje rasporeda međuispita.

3.6. Lokalne pretrage

Lokalna pretraga je skup metaheurističkih metoda koje služe za rješavanje teških optimizacijskih problema. Koristi se kod problema koji se mogu opisati kao pronalazak najboljeg rješenja među kandidatima iz prostora rješenja. Lokalna pretraga je tip algoritma koji radi s jednim kandidatom rješenja i iterativno pretražuje njegovu neposrednu okolicu u potrazi za boljim rješenjem u nadi da će pronaći najbolje moguće. Neki od algoritama iz ove skupine su algoritam uspona na vrh (engl. *Hill-climbing algorithm*), Tabu pretraga (engl. *Tabu search*) i simulirano kaljenje (engl. *Simulated annealing*). Najjednostavnije je lokalnu pretragu pokazati pomoću algoritma uspona na vrh na problemu pronalaska maksimuma funkcije.

Na slici 3.16 je prikazana jedna takva funkcija. Algoritam odabere jednu točku i počevši s njom krene u potragu. Zatim se provjeravaju sve susjedne točke i ako postoji barem jedna točka s većom dobrotom algoritma se preseli k najboljoj od nađenih. Ako ne postoji takva točka algoritam prestaje s radom ili se postupak ponovi s nekom novom, nasumično odabranom točkom. Na slici je početna odabrana točka označena s A. Kreće se u postupak provjeravanja susjednih točaka tj. rješenja koja se mogu dosegnuti u jednom koraku od početne točke. Recimo da je jedna od tih točaka točka s lijeve strane označena s B. Točka B ima bolju dobrotu od točke A i algoritam se preseli k njoj jer je jedina takva. Postupak se ponavlja dok algoritam iterativno pronalazi sve točke na putu do točke C. Točka C nema susjede koji imaju bolju dobrotu od nje pa algoritam prestaje s radom. Kao što je vidljivo sa slike algoritam nije pronašao globalni optimum jer je lokalno pratio najveći uspon i zaglavio na vrhu. Unatoč nedostatcima algoritma, može pomoći populacijskim algoritmima u bržem pronalasku



Slika 3.16: Primjer funkcije koja se optimira pomoću algoritma uspona na vrh

globalnog optimuma mnogo kompleksnijih problema kao što je problem izrade rasporeda ispita.

Kako bi se algoritmi lokalne pretrage mogli upotrijebiti kod problema izrade rasporeda ispita potrebno je prvo definirati neposredno susjedstvo rasporeda. Neke od implementacija lokalnih pretraga se mogu vidjeti u radovima (Burke et al., 1996) i (Arntzen i Løkketangen). Neposredno susjedstvo $S(\mathbf{x}_k)$ rasporeda $\mathbf{x}_k$ su svi oni rasporedi koji se od rasporeda $\mathbf{x}_k$ razlikuju u jednom i samo jednom elementu. Dakle, raspored iz susjedstva rasporeda $\mathbf{x}_k$ se dobije tako da se jedan međuispit iz tog rasporeda preseli u neki drugi termin različit od onog u kojem se trenutno nalazi.

Dva takva algoritma su implementirana za rješavanje problema izrade rasporeda međuispita: *ovisna lokalna pretraga* i *jednokoračna lokalna pretraga*.

Ovisna lokalna pretraga prije početka pretrage sortira indekse međuispita po količini studenata koje dijeli s ostalim međuispitima, tako da oni međuispiti s najviše dijeljenih studenata završe na početku liste. Zatim se iz sortirane liste uzimaju redom međuispiti i provjeravaju se termini u koje se mogu preseliti. Nađe li se barem jedan termin u koji se međuispit može preseliti, a da smanjuje kaznu rasporeda, onda se međuispit preseli u termin koji je najviše smanjuje. Nakon što algoritam prođe cijelu listu međuispita provjerava se napredak smanjenja kazne. Ako je kazna u međuvremenu smanjena, postupak se ponavlja, inače algoritma završava s radom. Algoritam u jednom prolazu liste međuispita može izvršiti nekoliko preseljenja u druge termine. Sva

preseljenja međuispita u drugi termin moraju poštivati čvrsta ograničenja rasporeda. Složenost ovog algoritma je $O(c,t) = k_o \cdot (c^2 \cdot t^2)$ gdje c označava broj međuispita, a t označava broj termina. Faktor k_o označava broj potrebnih iteracija koje algoritam obavi i ovisi o prirodi problema. Pseudokod se nalazi u algoritmu 3.6.

Algoritam 3.6 pseudokod Ovisne lokalne pretrage

```
sortiraj_po_broju_dijeljenih_studenata(lista_ispita)
size ← velicina(sortirana_lista_ispita)
dok napredak_potrage radi
  za i < size; i = i + 1 radi
    ispit ← lista_ispita(i)
    lista_termina ← dozvoljeni_termini(ispit)
    termin ← nadi_termin_koji_najvise_smanjuje_kaznu(lista_termina)
    preseli_u(ispit, termin)
  kraj za
kraj dok
```

Jednokoračna lokalna pretraga prolazi listu međuispita redom i pokušava pronaći ono preseljenje međuispita u drugi termin koje najviše pridonosi smanjenju kazne. U jednom prolazu liste je moguće samo jedno preseljenje međuispita u drugi termin. Ako se postigne napredak u smanjenju kazne postupak se ponavlja, inače algoritam prestaje s radom. Ovaj algoritam je mnogo sporiji od prethodno opisanog ali pronalazi bolja rješenja. Sva preseljenja međuispita u drugi termin moraju poštivati čvrsta ograničenja rasporeda. Složenost ovog algoritma je $O(c,t) = k_j \cdot (c^2 \cdot t^2)$ gdje c označava broj međuispita, a t označava broj termina. Faktor k_j označava broj potrebnih iteracija koje algoritam obavi i ovisi o prirodi problema. Faktor k_j se pokazao mnogo većim nego faktor k_o što čini ovaj algoritam sporijim nego prethodno opisani. Jednokoračna lokalna pretraga daje mnogo bolje rezultate nego ovisna lokalna pretraga. Pseudokod se nalazi u algoritmu 3.7.

3.7. Simbioza populacijskog algoritma i lokalne pretrage

Kako bi se poboljšale performanse kod rješavanja teških optimizacijskih problema, u zadnje vrijeme se često spajaju koncepti populacijskih metaheurističkih algoritama i lokalnih pretraga. Populacijski metaheuristički algoritmi su dobri u paralelnoj pretrazi

Algoritam 3.7 pseudokod Jednokoračne lokalne pretrage

```
stvori(lista_ispita)
size ← velicina(sortirana_lista_ispita)
dok napredak_potrage radi
  za i < size; i = i + 1 radi
    ispit ← lista_ispita(i)
    lista_termina ← dozvoljeni_termini(ispit)
    termin ← nadi_termin_koji_najvise_smanjuje_kaznu(lista_termina)
    ako najvise_smanjuje_kaznu(termin, ispit tada
      ispit_max ← ispit
      termin_max ← termin
    kraj ako
  kraj za
  preseli_u(ispit_max, termin_max)
kraj dok
```

prostora rješenja i grubom istraživanju regija unutar područja istraživanja. Često se događa da populacijski algoritmi imaju problema s pronalaženjem najboljeg rješenja unutar regije ili im za to treba mnogo vremena. Zato se lokalne pretrage uključuju kao pomoć populacijskim algoritmima kako bi bolje istražile neposredno susjedstvo pronađenih rješenja. U konkretnoj implementaciji algoritma korištene su obadvije verzije lokalne pretrage opisane u prethodnom odjeljku. Ovisna lokalna pretraga se koristi kako bi se unaprijedila svaka novostvorena jedinka zato jer je brži ali daje lošija rješenja. Zato se algoritam jednokoračne lokalne pretrage upotrebljava kako bi se unaprijedila novostvorena jedinka koja je najbolja pronađena jedinka do tog trenutka. Ponašanje populacijskih algoritama evolucijskog računanja koji rade zajedno s algoritmima lokalne pretrage je ispitano u pokusima i rezultati su prikazani u poglavlju 5.

Ovaj koncept se u literaturi može pronaći pod nekoliko imena: Memetički algoritam (engl. *Memetic algorithm*), Baldwinijski evolucijski algoritmi (engl. *Baldwinian EAs*), Lamarckijanski evolucijski algoritmi (engl. *Lamarckian EAs*), kulturološki algoritmi (engl. *cultural algorithms*) ili genetska lokalna pretraga (engl. *Genetic local search*). Pseudokod ovakvog pristupa je prikazan u algoritmu 3.8.

Algoritam 3.8 pseudokod implementacije memskog algoritma

Inicijaliziraj početnu populaciju P
dok $\neg uvijet\_zaustavljanja$ **radi**
$evaluiraj(P)$
P' = Evoluiraj novu populaciju koristeći operatore i P
Ω = Odaberi jedinke iz P' koje će unaprijediti lokalna pretraga
za $\forall jed \in \Omega$ **radi**
$lokalna\_pretraga(jed)$
kraj za
Umetni P' u P
kraj dok

4. Naprednije tehnike poboljšavanja potrage za rješenjima

U nadi da će sustav stvoriti bolje rasporede ispita dodano je nekoliko naprednijih tehnika za pretragu prostora rješenja. Rast mutacije kod pojave stagnacije se pokazao korisnim u rješavanju nekoliko problema prisutnih u našoj ustanovi. Dinamička promjena funkcije evaluacije je inspirirana biološkom evolucijom i postupnim mijenjanjem okoliša kojem se organizmi prilagođavaju. Ideja za rad lokalnih pretraga u ovisnosti o vremenu ili stagnaciji algoritma se pojavila se kao rezultat internih rasprava unutar tima koji je implementirao sustav.

4.1. Rast mutacije kod pojave stagnacije rješenja

Mutacija je operator koji služi da bi se proširio prostor pretrage za rješenjima. Glavna funkcija mu je smanjiti vjerojatnost da algoritma zaglavi u nekom od lokalnih optimuma, tj. povećati vrijeme pretrage u nadi da će dobiti globalni optimum. No vjerojatnost mutacije može utjecati negativno na pretragu ako je previsoka jer ometa iskorištavanje pronađenog prostora u pretrazi. Neizbježna činjenica je da će se nakon nekog vremena ipak pojaviti stagnacija pretrage i u tom stanju su male vjerojatnosti da će se pojaviti veći napredak. Ako se pojavila stagnacija, onda nema više koristi od iskorištavanja nađenog prostora. U tom slučaju ne bi trebalo biti velike štete od povećanja mutacije jer bi se moglo dogoditi da se pretraga algoritma oslobodi eventualnog lokalnog optimuma. Ponašanje ovog pristupa je testirano i rezultati su prikazani u poglavlju 5.

4.2. Rad lokalnih pretraga u ovisnosti o vremenu ili stagnaciji algoritma

Lokalne pretrage ubrzavaju pretragu lokalnog prostora za optimalnim rješenjem, no ovakav pristup bi mogao potaknuti preranu konvergenciju k lokalnom optimumu. Mogući pristupi kako bi se riješio ovaj problem su sljedeći. Dopustiti algoritmu da vrši pretragu bez lokalne pretrage i uključiti je tek kada se pojavi stagnacija kako bi temeljitije ispitala nađeni prostor. Također je moguće uključiti algoritam s lokalnom pretragom te je isključiti ako se dogodi stagnacija u nadi da će se povećati različitost populacije na taj način. Na kraju se lokalnoj pretrazi mogu dodijeliti fiksni vremenski intervali u kojima će joj biti dozvoljeno raditi.

4.3. Dinamička funkcija evaluacije

Funkcija evaluacije je element koji usmjerava rad algoritama u potrazi za rješenjima. U statičkoj verziji funkcija se ponaša jednako tokom čitavog vremena u kojem algoritmi traže rješenja. Dinamička verzija funkcije je osmišljena tako da mijenja skup međuispita koje uzima u obzir kada računa kaznu rasporeda. Skup međuispita za koje računa kaznu ovisi o vremenu proteklom u pretrazi i zahtjevnosti ispita. Zahtjevnost ispita je definirana kao udio koji pojedini međuispit ima u kazni. Na ovaj način se očekuje od algoritma da će postupno optimirati raspored ispita uzimajući u obzir sve veći broj ispita. Npr. dopusti li evaluacijska funkcija računaje kazne samo između zahtjevnih ispita, algoritam bi trebao uzimati u obzir samo međuodnose zahtjevnih ispita i tako naći optimalne pozicije za te ispite. Kasnije se evaluacijska funkcija uključi za sve ispite i manje zahtjevne ispite algoritam pokuša opitimalno složiti između već smještenih zahtjevnih ispita. Rezultati ovoga pristupa su prikazani u poglavlju 5.

5. Utjecaj parametara na dobivene rezultate

U ovom poglavlju su izneseni eksperimentalni rezultati za nekoliko parametara koji mogu utjecati na kvalitetu dobivenih rezultata.

U prvom odlomku se analiziraju rezultati parametara koji utječu na istraživačke sposobnosti algoritama. To se prije svega odnosi na vjerojatnost operatora mutacije (p_{mut}), vjerojatnost prihvaćanja komponente iz harmonijske memorije (p_{acc}), vjerojatnost naknadnog ugađanja komponenete (p_{pa}) te veličina populacije ($size$). Ispitano je još i ponašanje algoritama uslijed povećavanja p_{mut} i smanjivanja p_{acc} koje je uzrokovano stagnacijom pretrage algoritama.

U drugom odlomku se analizira ponašanje populacijskih algoritma u simbiozi s algoritmima lokalne pretrage i faktoru rasta populacije ($mult$). Obavljeni su pokusi u kojima je sudjelovanje lokalne pretrage uvjetovano s nekoliko parametara.

Na kraju su prikazani rezultati pokusa u kojima je dinamički mijenjana funkcija evaluacije.

Pokusi su obavljeni na računalima s procesorima Intel(R) Core (TM)2 QUAD CPU Q8200 brzine 2.33 GHz i radnom memorijom od 3544 MBajta. Verzija programskog jezika Java je 1.6.0_21. Okruženje za pokretanje je Java(TM) SE Runtime Environment verzija 1.6.0_21-b07. Virtualni stroj je Java HotSpot(TM) Client VM verzija 17.0-b17. Za svaku vrijednost parametra pokus je pokrenut 56 puta. Svaki pokus je trajao 30 minuta. Algoritmi su tesirani na problemu rasporeda međuispita za zimski semestar akademske godine 2010./2011. na Fakultetu elektrotehnike i računarstva. Svi pokusi su provedeni na početnoj, unaprijed pretpostavljenoj konfiguraciji algoritama gdje je varijabilan samo onaj parametar koji se u pokusima istražuje. Ovaj način ispitivanja je odabran zbog toga što prostor različitih parametara raste eksponencijalnom progresijom što je parametara više. Početni parametri su prikazani u tablicama: 5.1, 5.2, 5.3 i 5.4.

$parametar$	$vrijednost$
veličina populacije	50
p_{mut}	0.005

Tablica 5.1: Generacijski GA. Vrijednost početnih parametara

$parametar$	$vrijednost$
veličina populacije	50
p_{mut}	0.005

Tablica 5.2: Eliminacijski GA. Vrijednost početnih parametara

$parametar$	$vrijednost$
veličina populacije	10
p_{acc}	0.99
p_{pa}	0.05

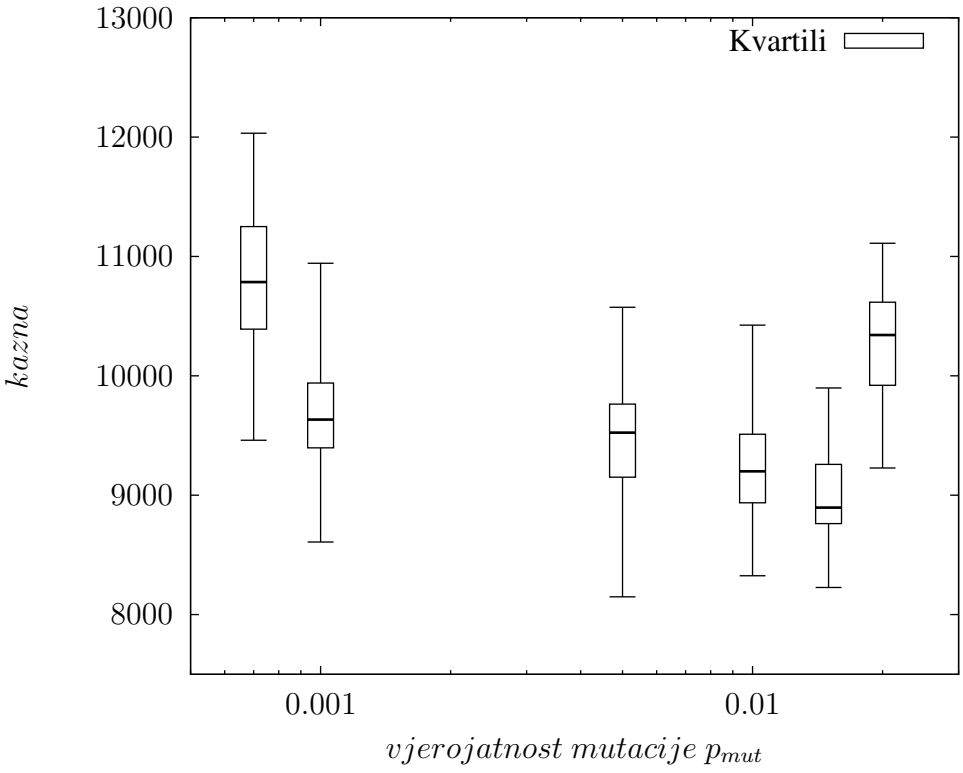
Tablica 5.3: Algoritam harmonijske pretrageonijska pretraga. Vrijednost početnih parametara

$parametar$	$vrijednost$
veličina populacije	100
faktor rasta	10
p_{mut}	0.01

Tablica 5.4: Jednostavni imunološki algoritam. Vrijednost početnih parametara

5.1. Ponašanje algoritama ovisno o parametrima istraživačkih operatora

Pokusi provedeni na operatoru mutacije daju naslutiti da su svi algoritmi osjetljivi na vrijednost vjerojatnosti mutacije ili sličnog istraživačkog operatora (Grafovi: 5.1, 5.3, 5.8 i 5.5. Tablice: 5.5, 5.8, 5.12 i 5.9). Ako je vjerojatnost mutacije preniska, onda je pretraga prostora prekonzervativna i najviše je fokusirana na iskorištavanje prostora u kojem se jedinke iz populacije već nalaze. Ako je vjerojatnost previsoka, onda operator djeluje predestruktivno na pronađena rješenja i pretvara se u nasumičnu pretragu.

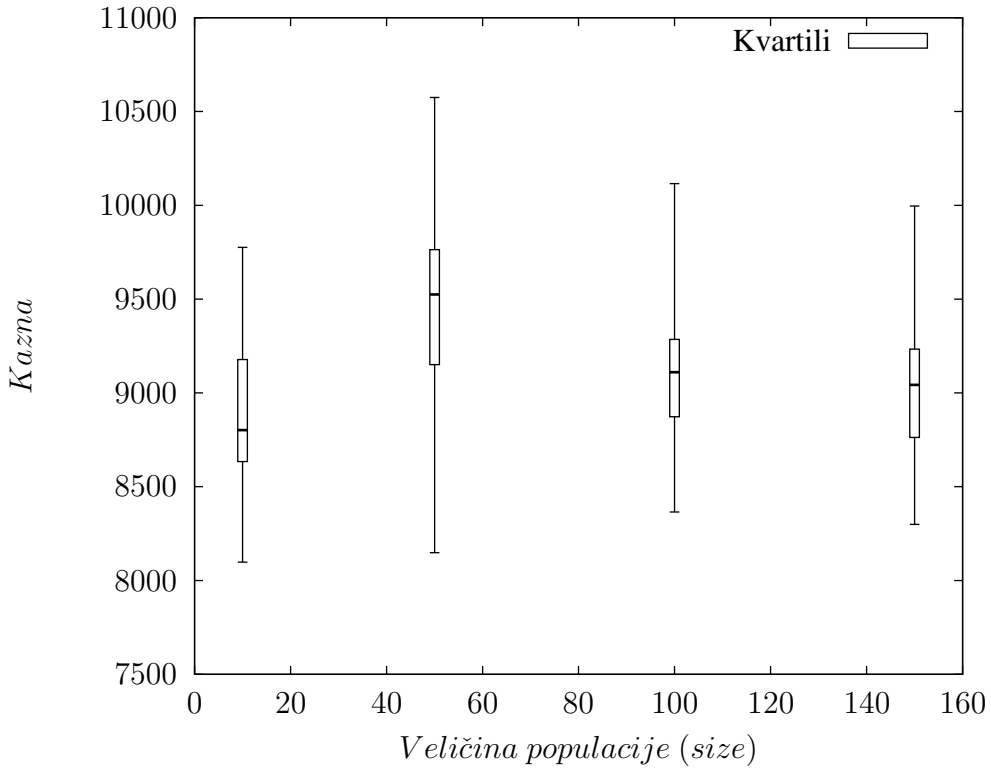


Slika 5.1: Generacijski GA. Ovisnost kazne o vjerojatnosti mutacije

Veličine populacije koje su ispitane ne utječu previše na kvalitetu rješenja osim kod Harmonijske pretrage (Graf 5.7). Također je vjerojatnost ugađanja komponente kod Harmonijske pretrage p_{pa} bolja što je manja za problem rasporeda izrade međuispita (Graf 5.6).

vjerojatnost mutacije	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
0.0007	9460	10391	10785	11250	12032
0.001	8608	9397	9633	9940	10942
0.005	8148	9150	9524	9763	10574
0.01	8325	8936	9200	9510	10424
0.015	8227	8762	8896	9258	9898
0.02	9227	9920	10341	10617	11111

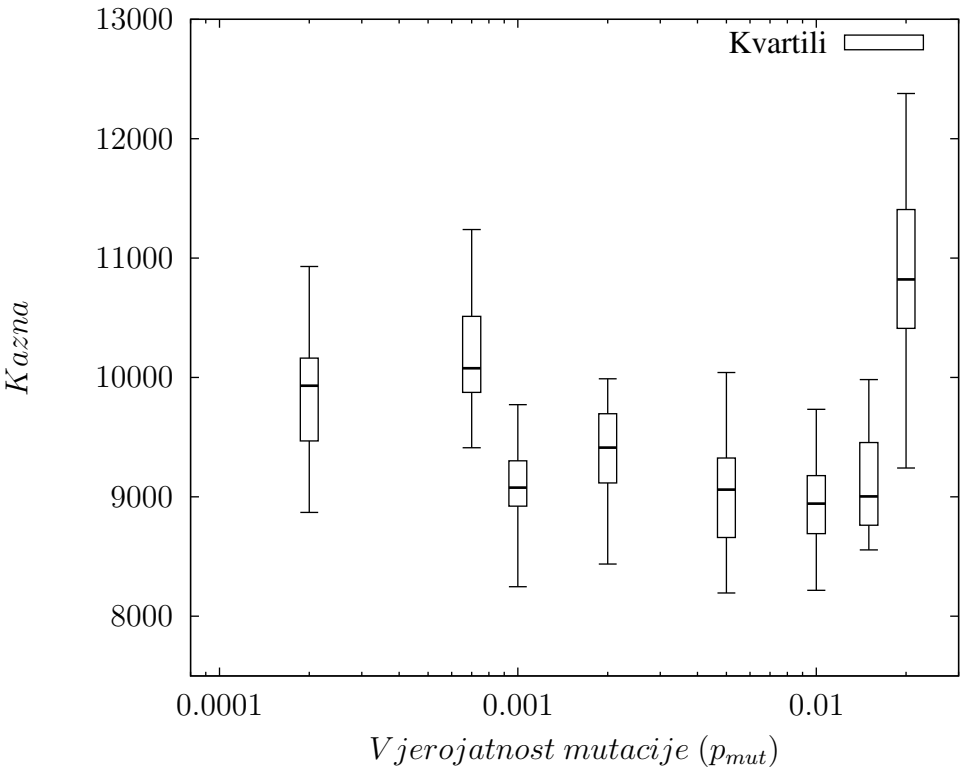
Tablica 5.5: Generacijski GA. Ovisnost kazne o vjerojatnosti mutacije.



Slika 5.2: Generacijski GA. Ovisnost kazne o veličini populacije.

veliina populacije	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
10	8097	8633	8801	9178	9775
50	8148	9150	9524	9763	10574
100	8365	8872	9109	9285	10115
150	8299	8762	9042	9233	9996

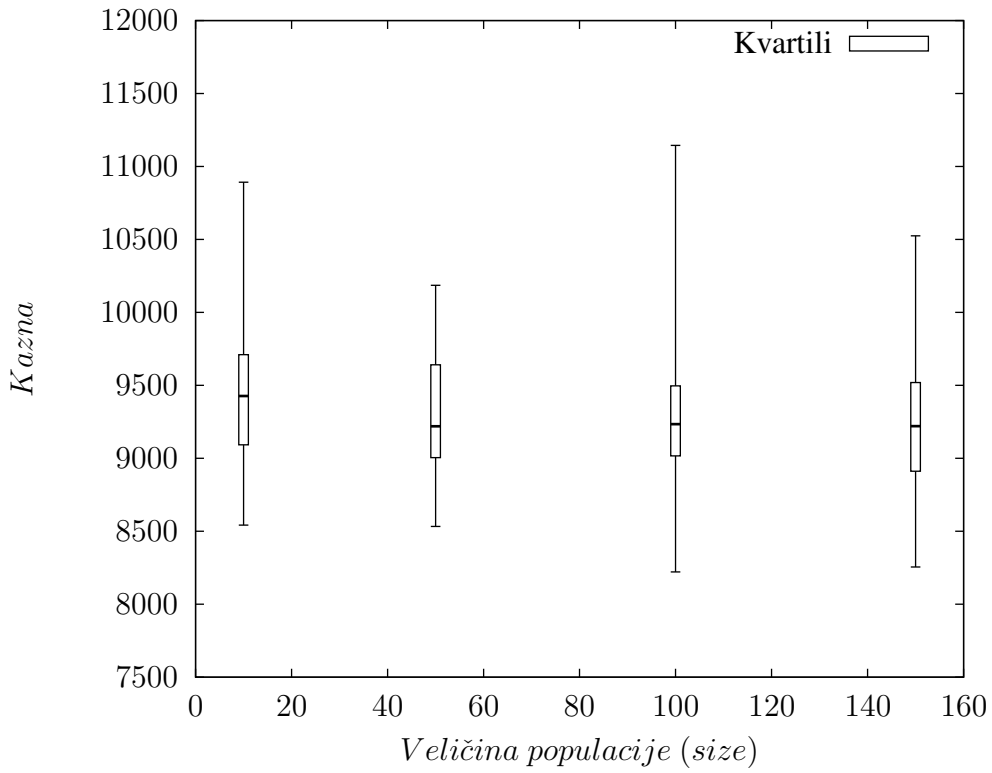
Tablica 5.6: Generacijski GA. Ovisnost kazne o veličini populacije.



Slika 5.3: Eliminacijski GA. Ovisnost kazne o vjerojatnosti mutacije.

vjerojatnost mutacije	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
0.0002	8869	9468	9930	10162	10930
0.0007	9411	9875	10076	10512	11239
0.001	8247	8921	9078	9301	9772
0.005	8194	8659	9060	9325	10041
0.01	8216	8692	8943	9178	9733
0.015	8555	8762	9003	9455	9982
0.02	9241	10411	10820	11406	12379
0.002	8437	9116	9413	9696	9989

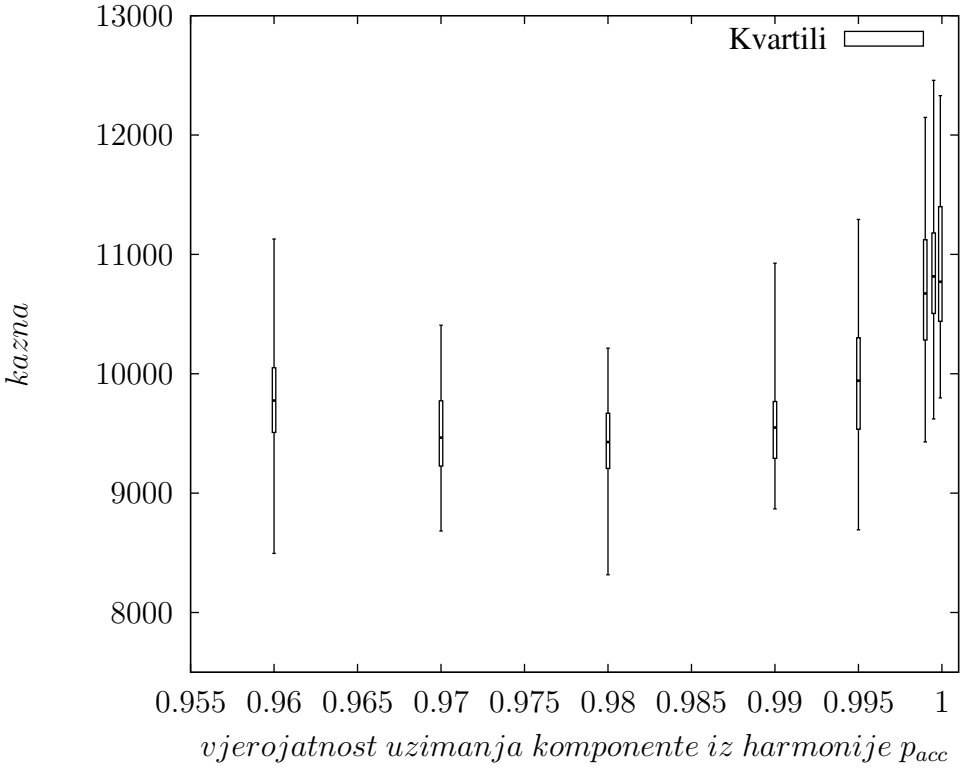
Tablica 5.7: Eliminacijski GA. Ovisnost kazne o vjerojatnosti mutacije.



Slika 5.4: Eliminacijski GA. Ovisnost kazne o veličini populacije.

veliina populacije	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
10	8541	9092	9427	9710	10892
50	8532	9004	9218	9640	10185
100	8220	9016	9234	9495	11144
150	8255	8911	9219	9519	10525

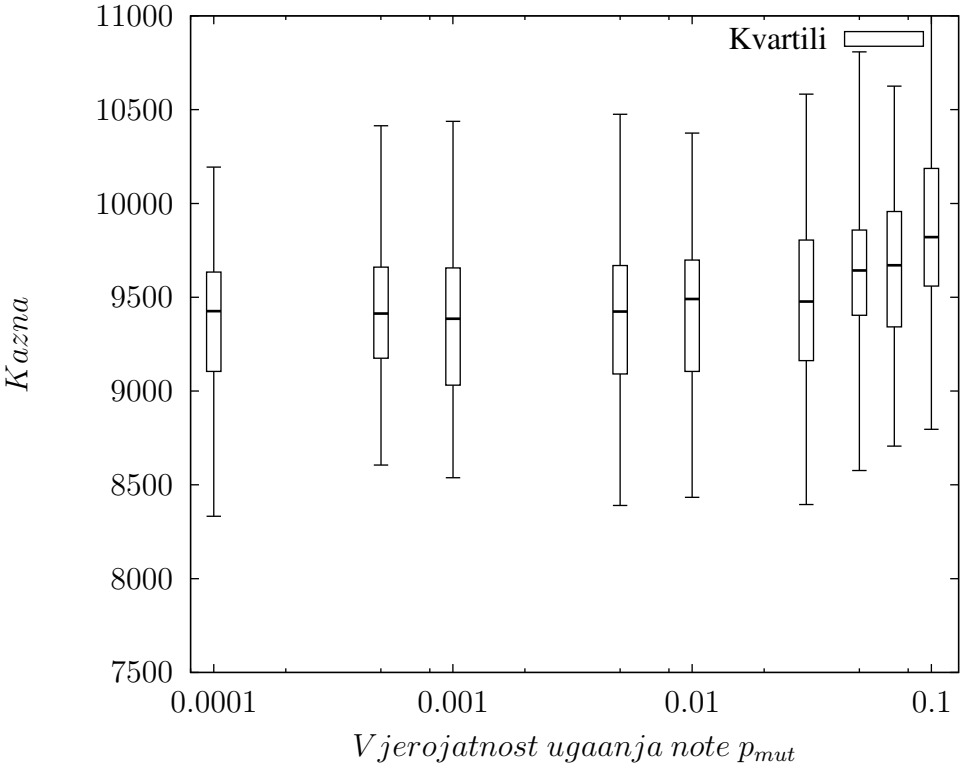
Tablica 5.8: Eliminacijski GA. Ovisnost kazne o veličini populacije.



Slika 5.5: Algoritam harmonijske pretrage. Ovisnost kazne o vjerojatnosti uzimanja komponente iz harmonijske memorije

p_{acc}	min	1. $kvartil$	$medijan$	2. $kvartil$	max
0.9999	9798	10439	10771	11399	12330
0.9995	9621	10505	10815	11179	12458
0.999	9429	10283	10672	11123	12147
0.995	8693	9535	9942	10302	11292
0.99	8868	9291	9550	9768	10926
0.98	8316	9207	9428	9669	10215
0.97	8683	9226	9465	9773	10406
0.96	8495	9508	9776	10051	11129

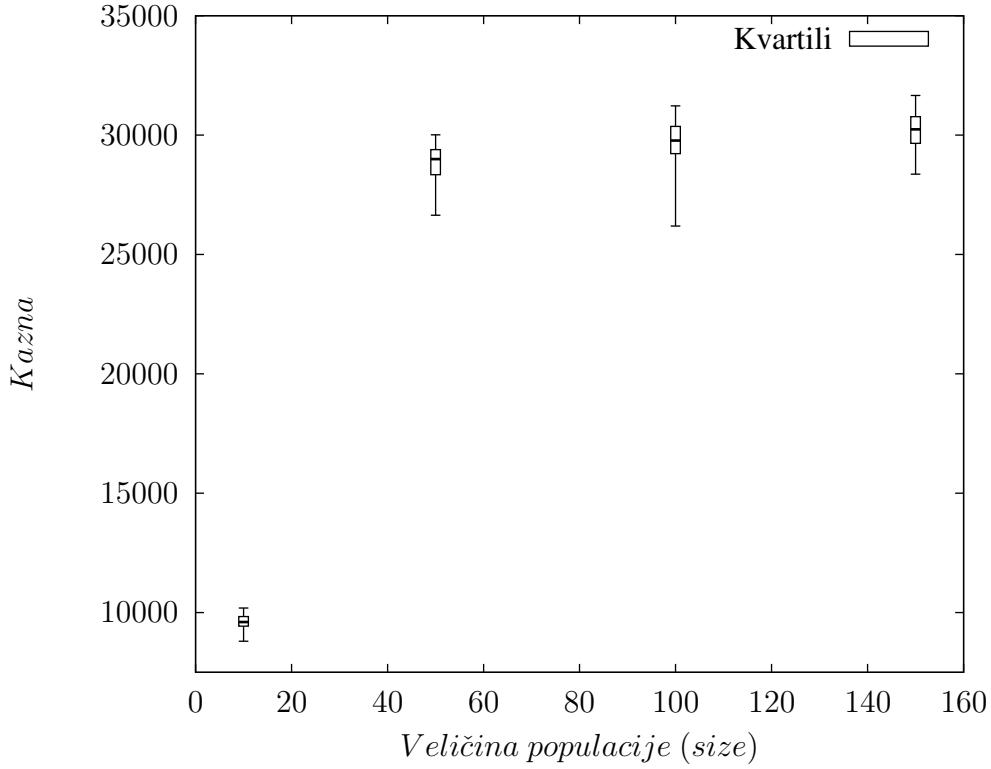
Tablica 5.9: Algoritam harmonijske pretrage. Ovisnost kazne o vjerojatnosti uzimanja komponente iz harmonijske memorije



Slika 5.6: Algoritam harmonijske pretrage. Ovisnost kazne o vjerojatnosti ugađanja komponente

p_{pa}	min	$1.kvartil$	$medijan$	$2.kvartil$	max
0.0001	8332	9104	9425	9634	10193
0.0005	8605	9174	9413	9660	10414
0.001	8538	9031	9385	9655	10437
0.005	8389	9091	9423	9669	10475
0.01	8433	9104	9490	9698	10375
0.03	8395	9161	9477	9804	10582
0.05	8576	9403	9643	9858	10808
0.07	8706	9341	9670	9956	10625
0.1	8796	9559	9820	10186	11296

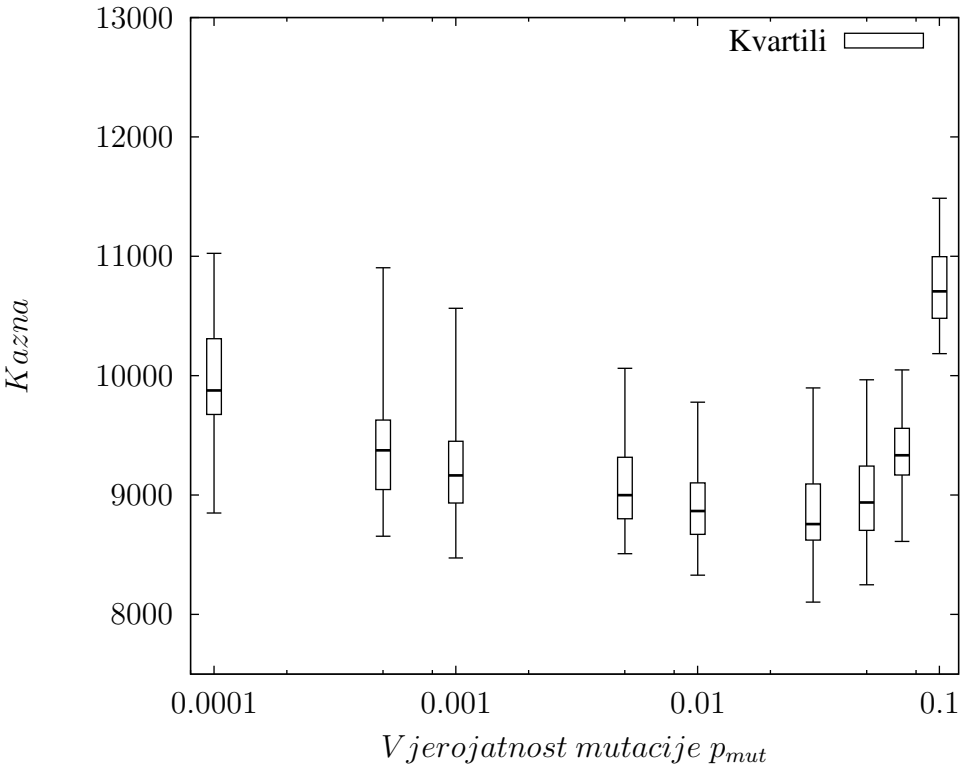
Tablica 5.10: Algoritam harmonijske pretrage. Ovisnost kazne o vjerojatnosti ugađanja komponente



Slika 5.7: Algoritam harmonijske pretrage. Ovisnost kazne o veličini harmonijske memorije.

veliinamemorije	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
10	8801	9430	9607	9836	10195
50	26645	28340	28992	29389	30010
100	26192	29223	29770	30365	31223
150	28366	29653	30238	30773	31656

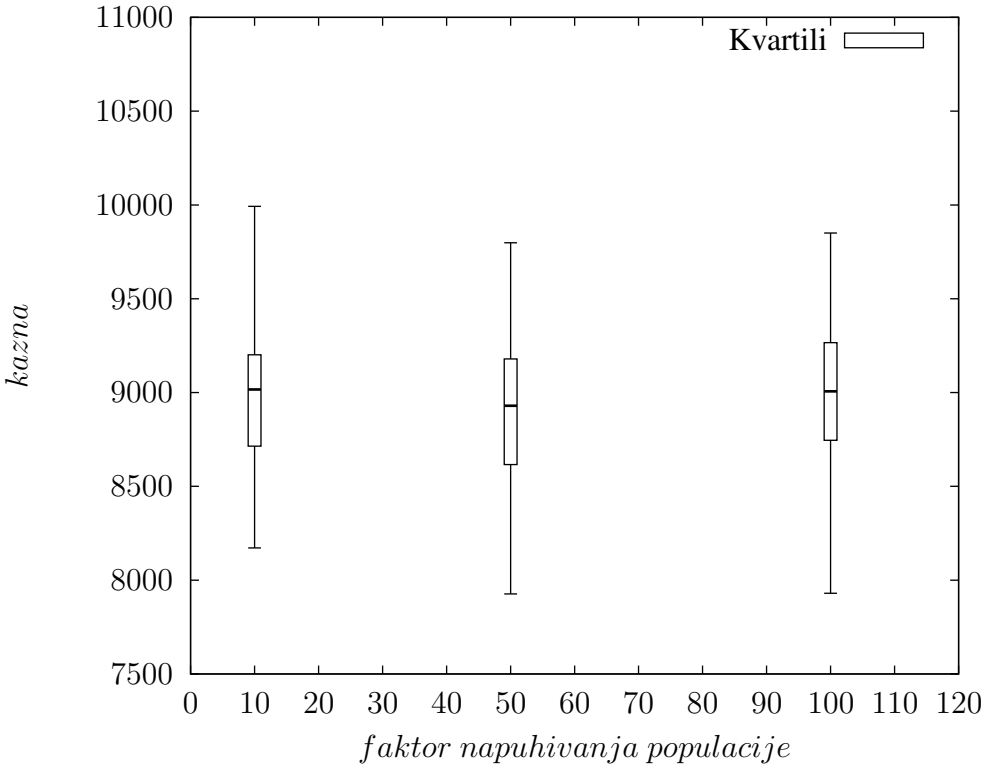
Tablica 5.11: Algoritam harmonijske pretrage. Ovisnost kazne o veličini harmonijske memorije.



Slika 5.8: Jednostavni imunološki algoritam. Ovisnost kazne o vjerojatnosti mutacije

p_{mut}	min	1. $kvartil$	$medijan$	2. $kvartil$	max
0.0001	8849	9675	9876	10310	11024
0.0005	8655	9045	9374	9628	10904
0.001	8473	8933	9164	9450	10564
0.005	8508	8801	8999	9316	10062
0.01	8328	8671	8866	9102	9778
0.03	8102	8622	8756	9092	9897
0.05	8248	8704	8938	9243	9965
0.07	8611	9168	9333	9559	10048
0.1	10184	10480	10706	10996	11486

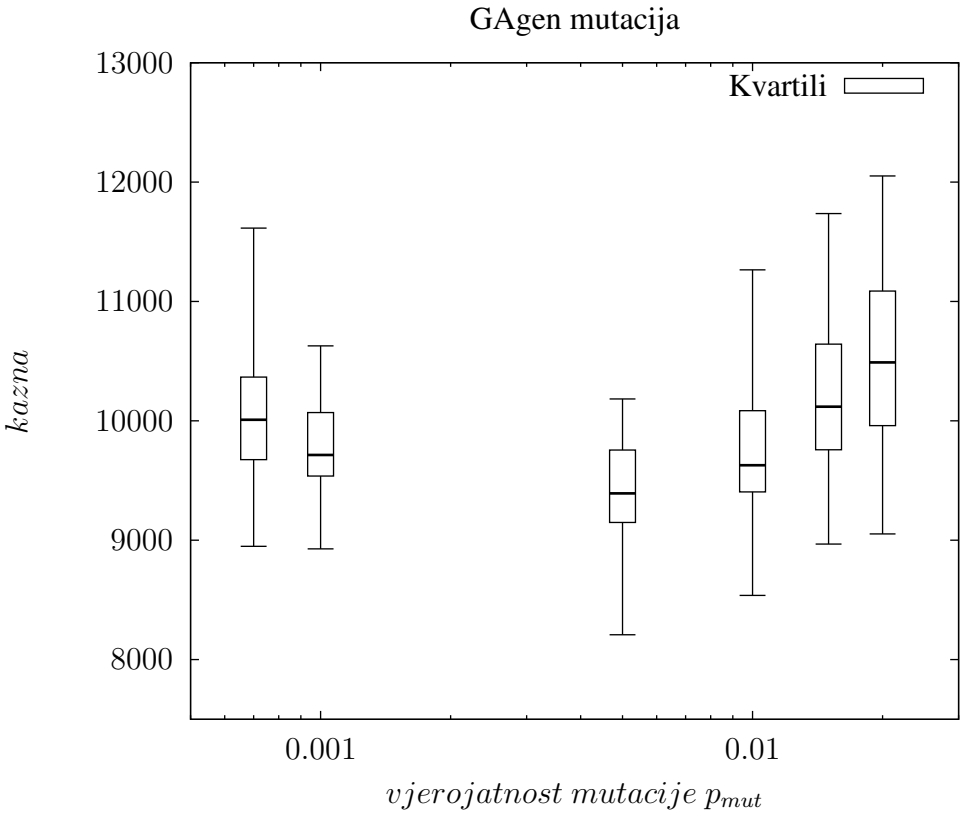
Tablica 5.12: Jednostavni imunološki algoritam. Ovisnost kazne o vjerojatnosti mutacije



Slika 5.9: Jednostavni imunološki algoritam. Ovisnost kazne o faktoru rasta populacije

	min	$1.kvartil$	$medijan$	$2.kvartil$	max
10	8172	8714	9016	9201	9992
50	7926	8616	8929	9179	9798
100	7930	8745	9006	9266	9850

Tablica 5.13: Jednostavni imunološki algoritam. Ovisnost kazne o faktoru rasta populacije



Slika 5.10: Generacijski GA. Ovisnost kazne o vjerojatnosti mutacije p_{mut}. Uključena lokalna pretraga.

5.2. Ponašanje algoritama ovisno o lokalnoj pretrazi i rastu mutacije uslijed stagnacije

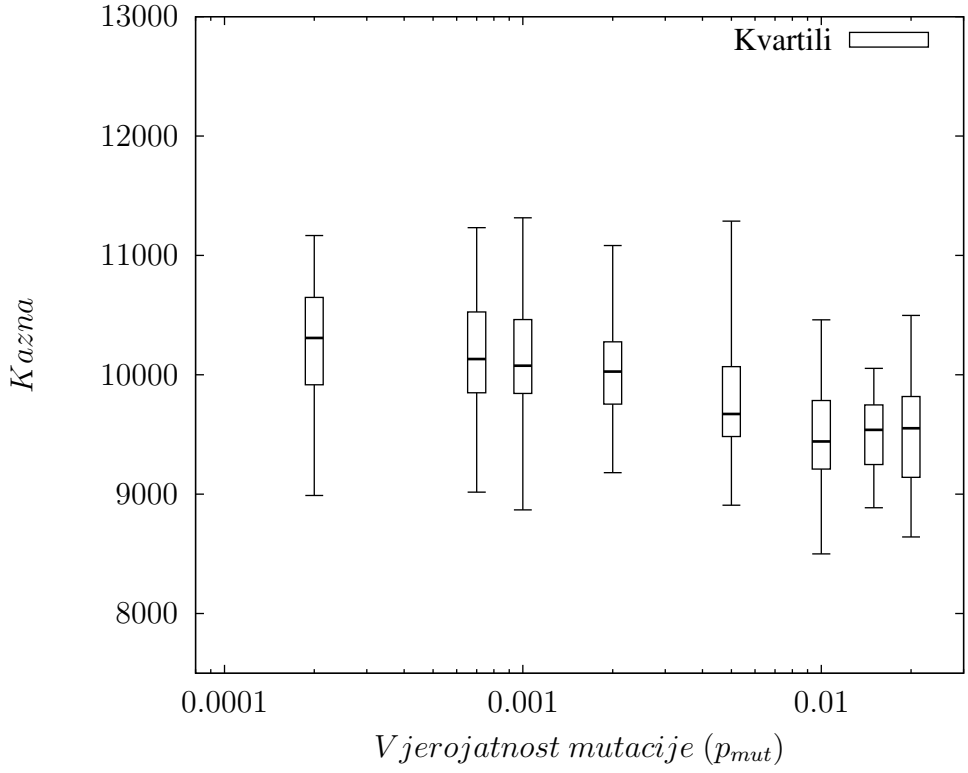
Pokusi su pokazali da lokalna pretraga utječe na kvalitetu dobivenih rješenja. Rješenja dobivena simbiozom Harmonijske pretrage i lokalne pretrage su bolja nego rješenja dobivena samo Harmonijskom pretragom (Tablice: 5.9 i 5.16).

Povćavanje mutacije nakon stagnacije utječe negativno na kvalitetu pronađenih rješenja (Grafovi: 5.14, 5.15, itd.).

Uvjetno uključivanje lokalne pretrage utječe pozitivno na kvalitetu rješenja eliminacijskog genetskog algoritma (Tablice: 5.27 i 5.23). Moguće objašnjenje ove pojave je to da ovakav režim rada lokalne pretrage ne dozvoljava lokalnoj pretrazi da odvede cijelu populaciju u neki od lokalnih optimuma. Također ovaj režim rada lokalne pretrage veoma loše utječe na kvalitetu rješenja dobivenih Harmonijskom pretragom (Tablica: 5.28).

p_{mut}	min	1. $kvartil$	$medijan$	2. $kvartil$	max
0.0007	8948	9675	10009	10366	11615
0.001	8927	9537	9714	10070	10628
0.005	8208	9148	9392	9755	10183
0.01	8537	9405	9628	10084	11264
0.015	8967	9757	10118	10642	11736
0.02	9053	9960	10489	11087	12051

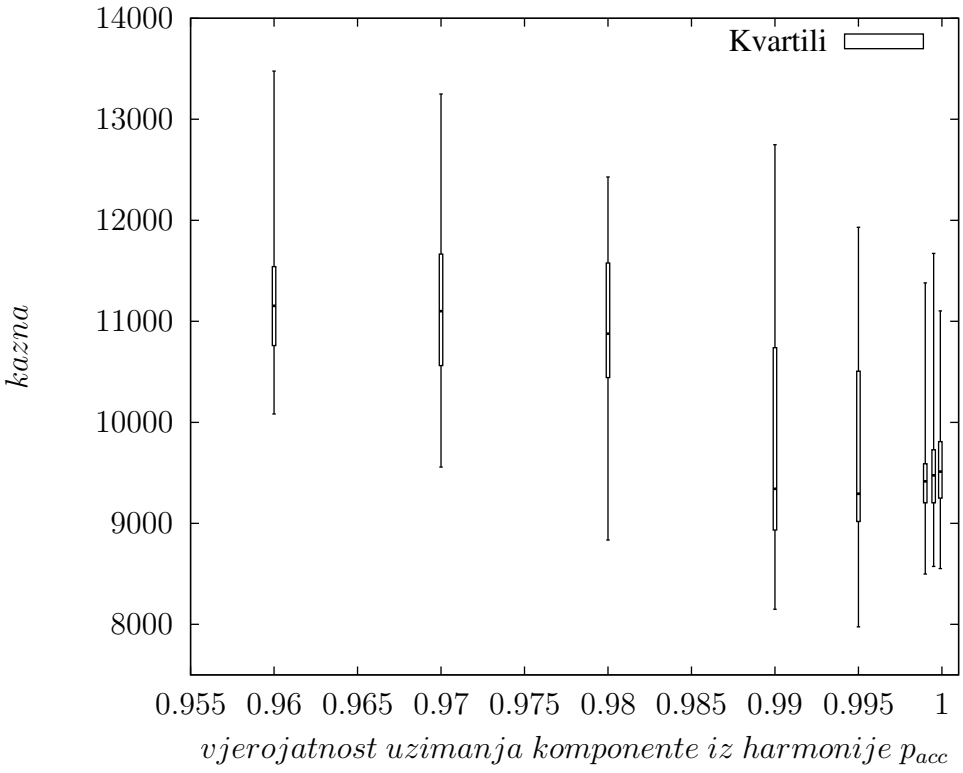
Tablica 5.14: Generacijski GA. Ovisnost kazne o vjerojatnosti mutacije p_{mut}. Uključena lokalna pretraga.



Slika 5.11: Eliminacijski GA. Ovisnost kazne o vjerojatnosti mutacije p_{mut}. Uključena lokalna pretraga.

p_{mut}	min	1. $kvartil$	$medijan$	2. $kvartil$	max
0.002	9180	9755	10027	10275	11082
0.0002	8989	9916	10308	10647	11166
0.0007	9016	9848	10132	10526	11232
0.001	8868	9844	10076	10462	11315
0.005	8908	9483	9671	10068	11287
0.01	8500	9210	9441	9784	10460
0.015	8886	9247	9538	9747	10053
0.02	8642	9141	9552	9818	10497

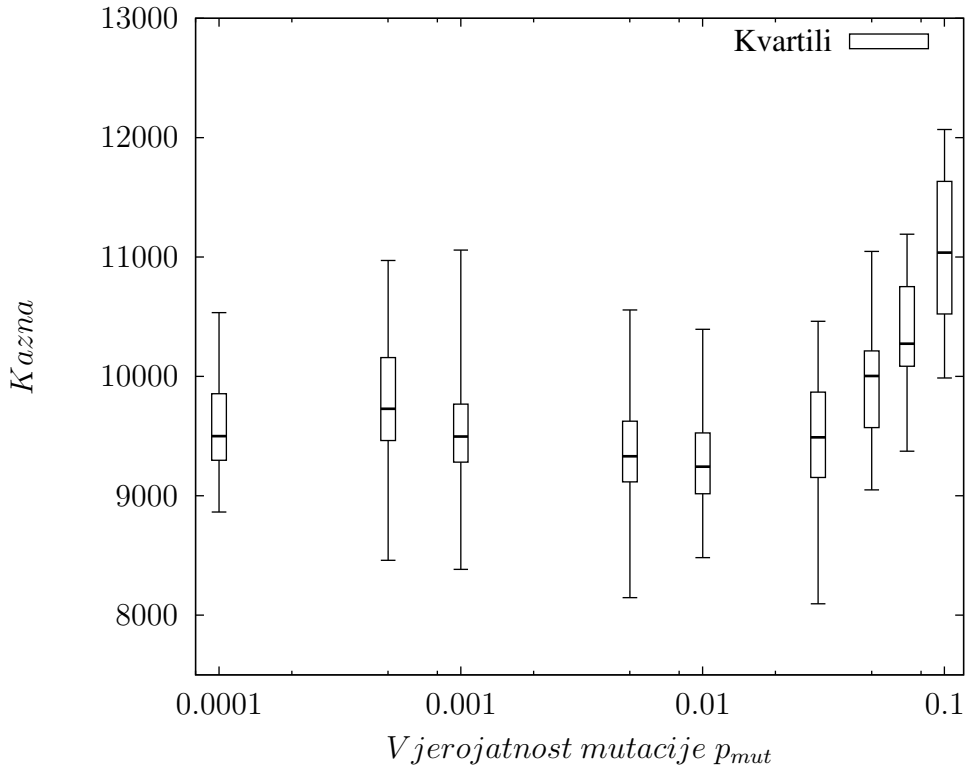
Tablica 5.15: Eliminacijski GA. Ovisnost kazne o vjerojatnosti mutacije p_{mut}. Uključena lokalna pretraga.



Slika 5.12: Algoritam harmonijske pretrage. Ovisnost kazne o vjerojatnosti uzimanja komponente iz memorije p_{acc}. Uključena lokalna pretraga

p_{acc}	min	1. $kvartil$	$medijan$	2. $kvartil$	max
0.9999	8551	9249	9512	9809	11103
0.9995	8574	9203	9474	9728	11671
0.999	8498	9203	9416	9590	11381
0.995	7975	9018	9293	10506	11931
0.99	8149	8933	9341	10739	12748
0.98	8835	10442	10877	11576	12429
0.97	9557	10562	11100	11665	13250
0.96	10082	10759	11152	11541	13477

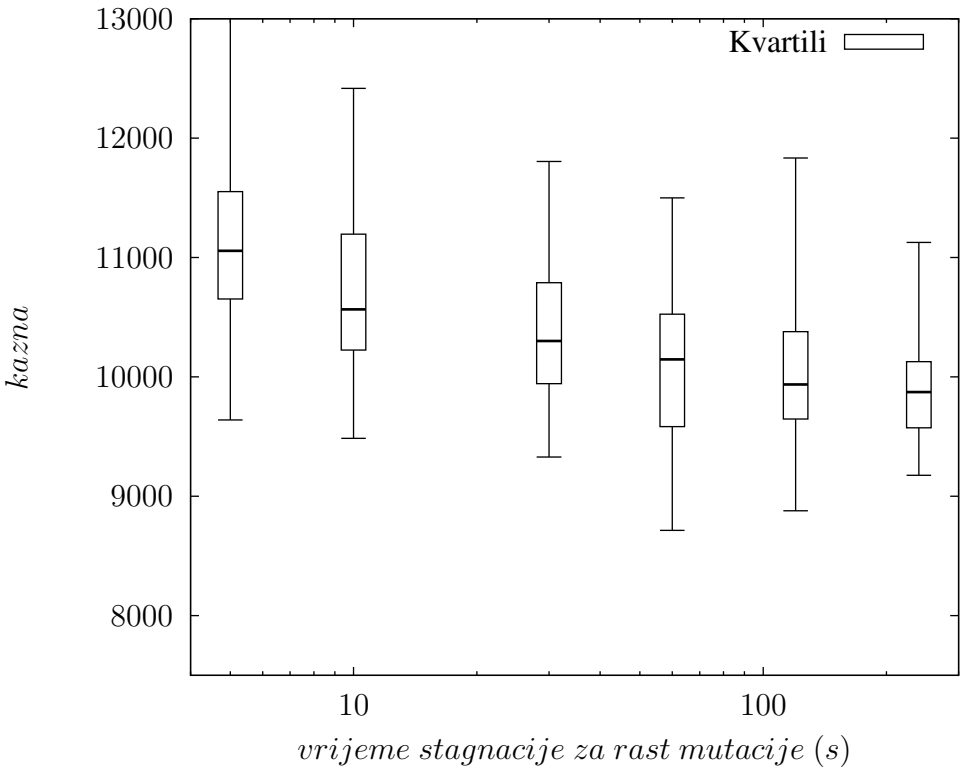
Tablica 5.16: Algoritam harmonijske pretrage. Ovisnost kazne o vjerojatnosti uzimanja komponente iz memorije p_{acc}. Uključena lokalna pretraga.



Slika 5.13: Jednostavni imunološki algoritam. Ovisnost kazne o vjerojatnosti mutacije p_{mut}. Uključena lokalna pretraga.

p_{mut}	min	1. $kvartil$	$medijan$	2. $kvartil$	max
0.0001	8864	9297	9500	9854	10533
0.0005	8459	9462	9729	10157	10970
0.001	8383	9282	9496	9767	11058
0.005	8147	9116	9331	9624	10556
0.01	8481	9017	9243	9527	10394
0.03	8094	9153	9489	9868	10461
0.05	9049	9571	10003	10213	11047
0.07	9373	10085	10273	10752	11190
0.1	9987	10523	11037	11633	12067

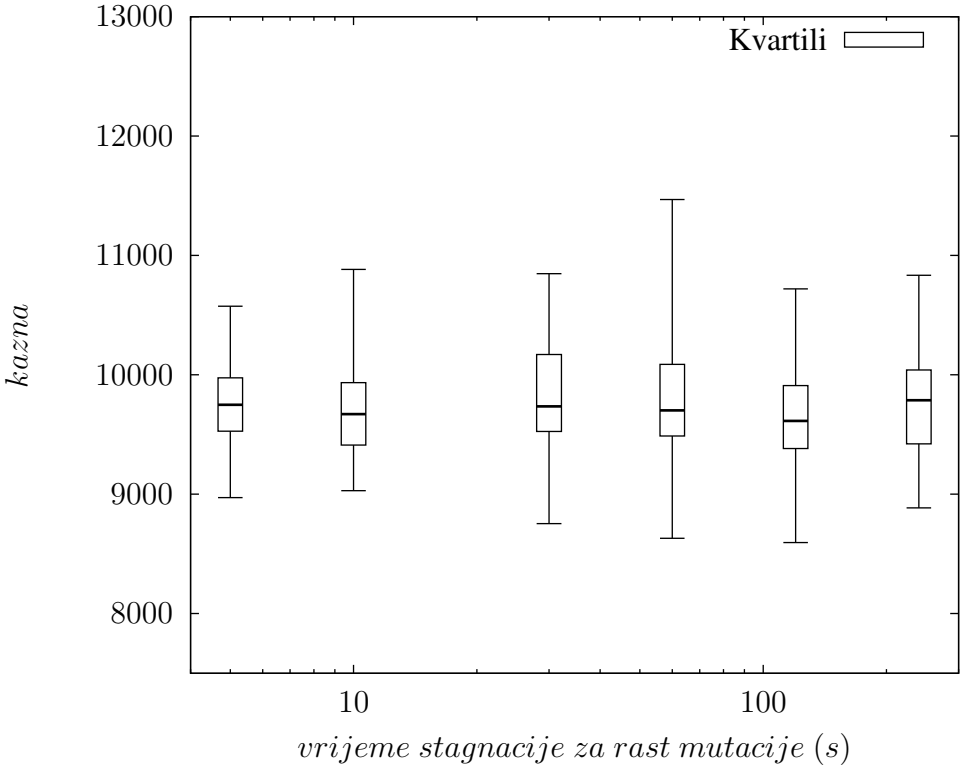
Tablica 5.17: Jednostavni imunološki algoritam. Ovisnost kazne o vjerojatnosti mutacije p_{mut}. Uključena lokalna pretraga.



Slika 5.14: Generacijski genetski algoritam. Ovisnost kazne o vremenu nakon kojeg se vrši rast vjerojatnosti mutacije p_{mut}. Rast mutacije je 10x. Uključena lokalna pretraga.

$vrijeme\ (s)$	min	1. $kvartil$	$medijan$	2. $kvartil$	max
5	9639	10652	11056	11552	13175
10	9484	10225	10566	11195	12416
30	9329	9943	10301	10789	11804
60	8714	9583	10146	10525	11499
120	8879	9647	9936	10378	11833
240	9176	9573	9873	10127	11126

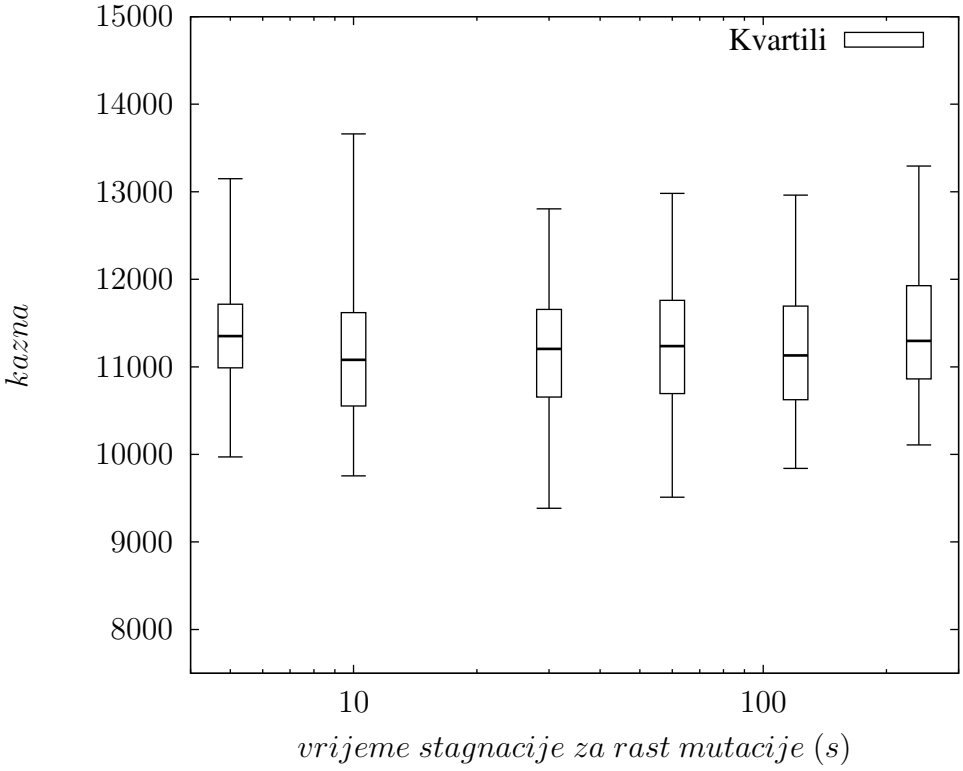
Tablica 5.18: Generacijski genetski algoritam. Ovisnost kazne o vremenu nakon kojeg se vrši rast vjerojatnosti mutacije p_{mut}. Rast mutacije je 10x. Uključena lokalna pretraga.



Slika 5.15: Eliminacijski genetski algoritam. Ovisnost kazne o vremenu nakon kojeg se vrši rast vjerojatnosti mutacije p_{mut}. Rast mutacije je 10x. Uključena lokalna pretraga.

$vrijeme\ (s)$	min	1. $kvartil$	$medijan$	2. $kvartil$	max
5	8971	9527	9748	9974	10574
10	9029	9411	9670	9934	10882
30	8753	9525	9735	10169	10847
60	8631	9486	9701	10087	11468
120	8594	9382	9613	9909	10719
240	8884	9421	9787	10040	10833

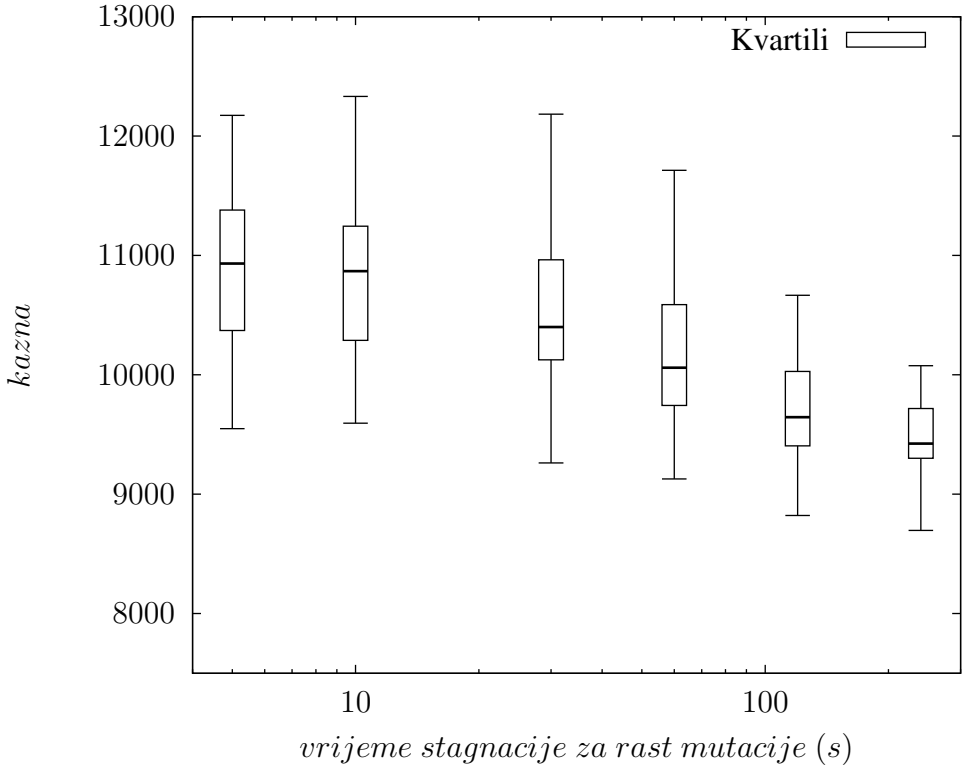
Tablica 5.19: Eliminacijski genetski algoritam. Ovisnost kazne o vremenu nakon kojeg se vrši rast vjerojatnosti mutacije p_{mut}. Rast mutacije je 10x. Uključena lokalna pretraga.



Slika 5.16: Hamonijska pretraga. Ovisnost kazne o vremenu nakon kojeg se vrši rast vjerojatnosti $1 - p_{acc}$. Rast je 10x. Uključena lokalna pretraga.

$vrijeme\ (s)$	min	1. $kvartil$	$medijan$	2. $kvartil$	max
5	9970	10988	11351	11715	13149
10	9754	10553	11080	11619	13661
30	9384	10654	11205	11655	12804
60	9510	10694	11236	11760	12981
120	9840	10625	11130	11694	12961
240	10108	10862	11296	11927	13293

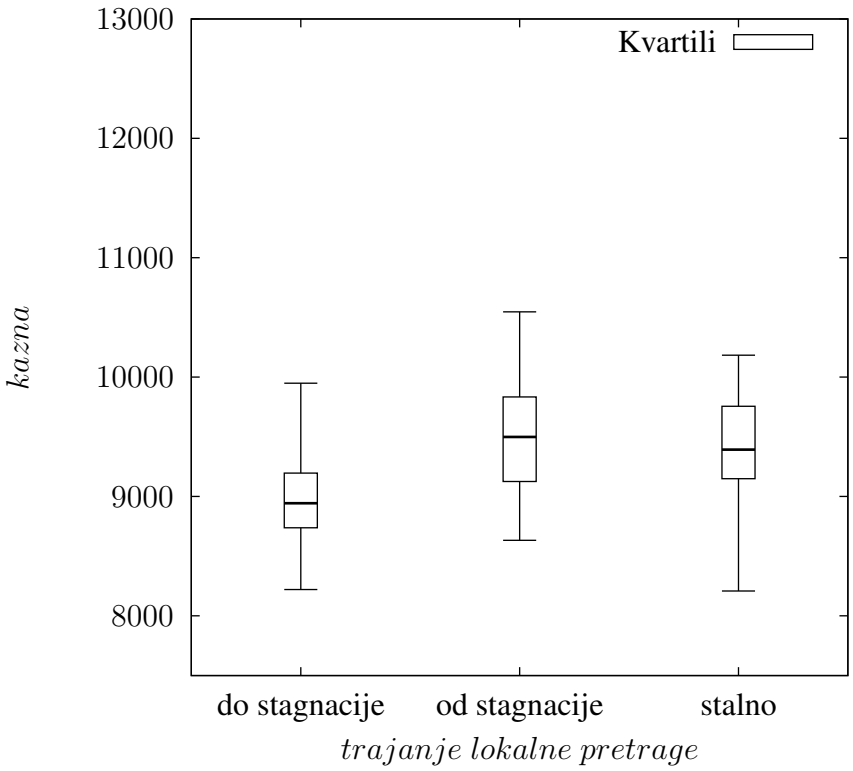
Tablica 5.20: Hamonijska pretraga. Ovisnost kazne o vremenu nakon kojeg se vrši rast vjerojatnosti $1 - p_{acc}$. Rast je 10x. Uključena lokalna pretraga.



Slika 5.17: Jednostavni imunološki algoritam. Ovisnost kazne o vremenu nakon kojeg se vrši rast vjerojatnosti mutacije p_{mut}. Rast je 10x. Uključena lokalna pretraga.

$vrijeme\ (s)$	min	1. $kvartil$	$medijan$	2. $kvartil$	max
5	9548	10371	10932	11379	12173
10	9595	10288	10868	11244	12331
30	9262	10124	10400	10962	12183
60	9128	9743	10059	10587	11712
120	8821	9404	9644	10028	10665
240	8696	9300	9424	9718	10076

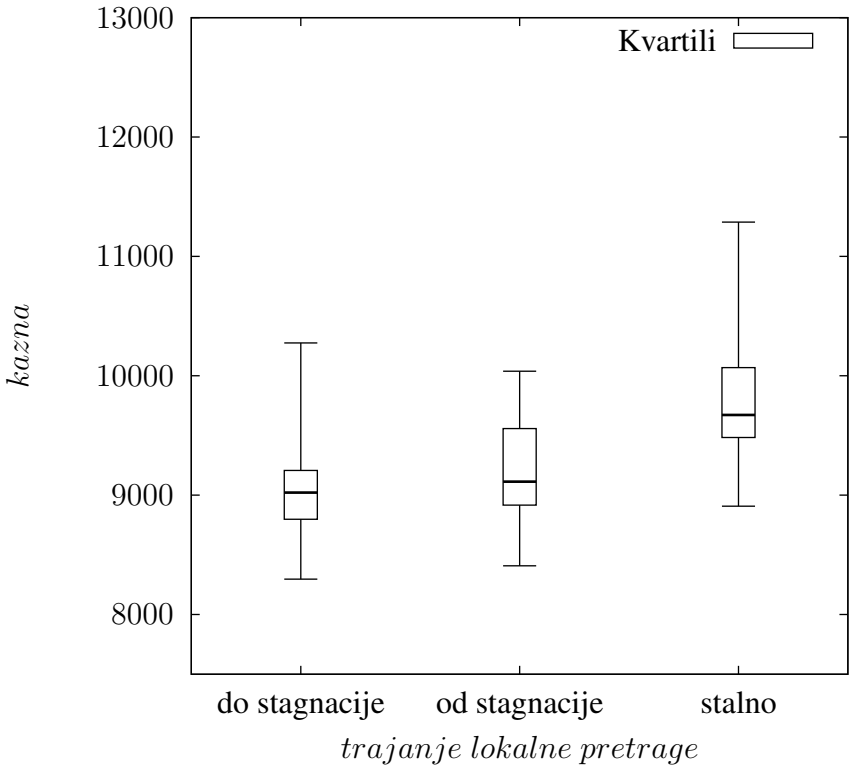
Tablica 5.21: Jednostavni imunološki algoritam. Ovisnost kazne o vremenu nakon kojeg se vrši rast vjerojatnosti mutacije p_{mut}. Rast je 10x. Uključena lokalna pretraga.



Slika 5.18: Generacijski GA. Ovisnost kazne o uključivanju/isključivanju lokalne pretrage nakon što se pojavila stagnacija.

trajanje lok. pretrage	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
do stag.	8220	8737	8943	9195	9948
od stag.	8633	9125	9498	9833	10546
stalno	8208	9148	9392	9755	10183

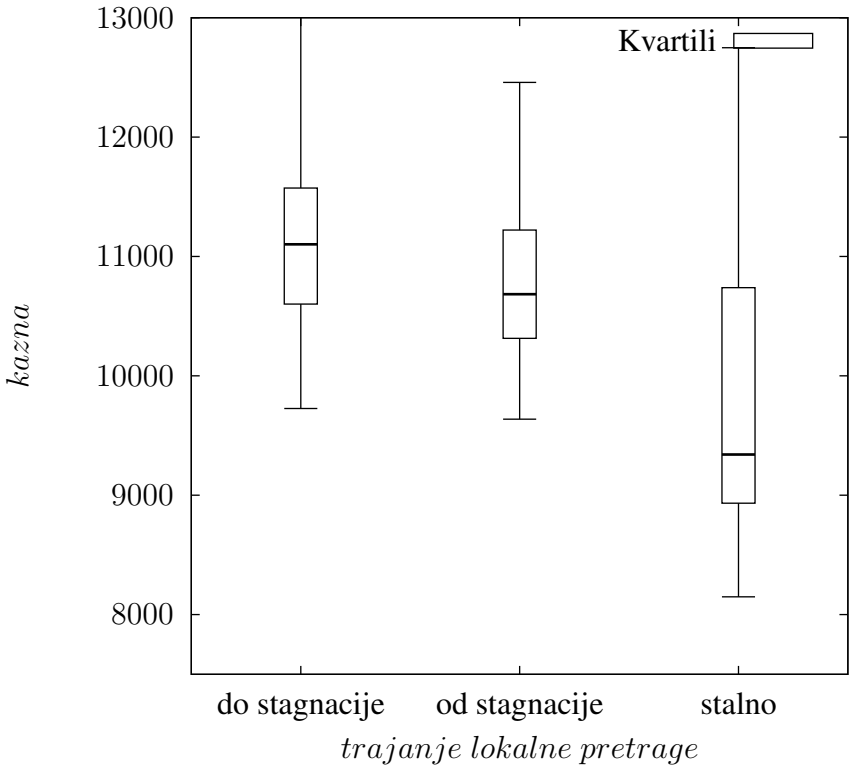
Tablica 5.22: Generacijski GA. Ovisnost kazne o uključivanju/isključivanju lokalne pretrage nakon što se pojavila stagnacija.



Slika 5.19: Eliminacijski GA. Ovisnost kazne o uključivanju/isključivanju lokalne pretrage nakon što se pojavila stagnacija.

trajanje lok. pretrage	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
do stag.	8297	8798	9022	9207	10275
od stag	8408	8916	9113	9557	10038
stalno	8908	9483	9671	10068	11287

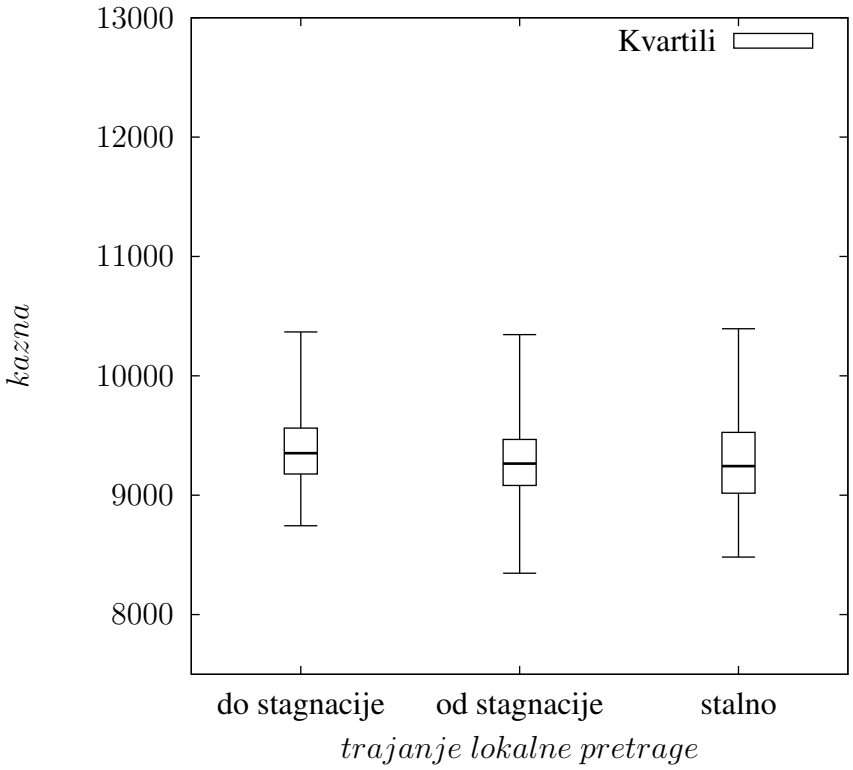
Tablica 5.23: Eliminacijski GA. Ovisnost kazne o uključivanju/isključivanju lokalne pretrage nakon što se pojavila stagnacija.



Slika 5.20: Algoritam harmonijske pretrage. Ovisnost kazne o uključivanju/isključivanju lokalne pretrage nakon što se pojavila stagnacija.

$trajanje\ lok.\ pretrage$	min	1. $kvartil$	$medijan$	2. $kvartil$	max
$do\ stag.$	9727	10601	11101	11573	13168
$od\ stag$	9636	10314	10683	11220	12458
$stalno$	8149	8933	9341	10739	12748

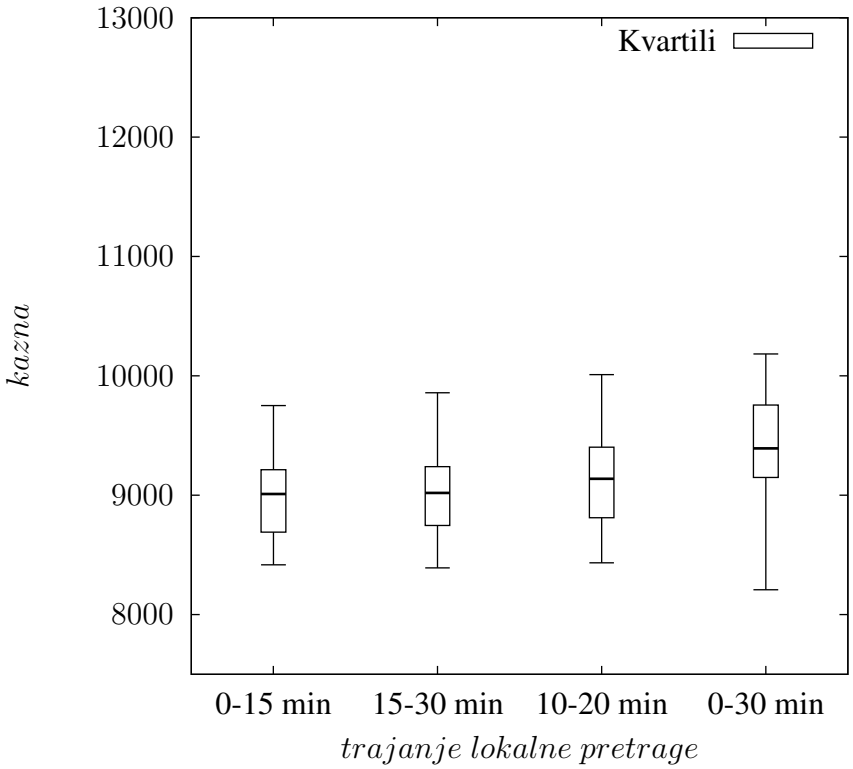
Tablica 5.24: Algoritam harmonijske pretrage. Ovisnost kazne o uključivanju/isključivanju lokalne pretrage nakon što se pojavila stagnacija.



Slika 5.21: Jednostavni imunološki algoritam. Ovisnost kazne o uključivanju/isključivanju lokalne pretrage nakon što se pojavila stagnacija.

trajanje lok. pretrage	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
do stag.	8744	9177	9351	9562	10367
od stag	8346	9081	9264	9467	10345
stalno	8481	9017	9243	9527	10394

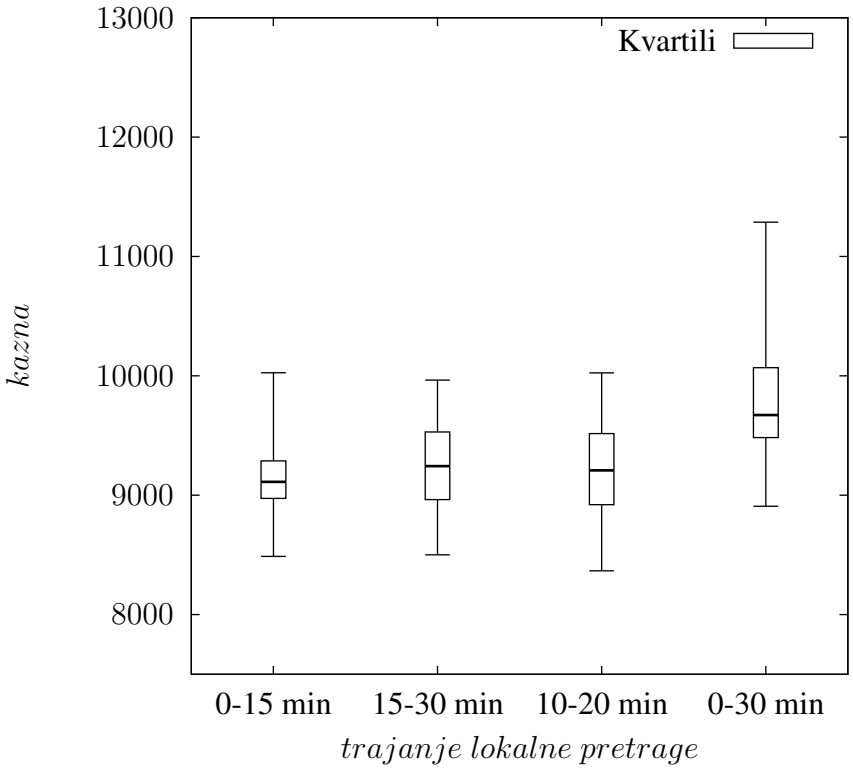
Tablica 5.25: Jednostavni imunološki algoritam. Ovisnost kazne o uključivanju/isključivanju lokalne pretrage nakon što se pojavila stagnacija.



Slika 5.22: Generacijski GA. Ovisnost kazne o trajanju lokalne pretrage.

trajanje lok. pretrage	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
$0 - 15\ min$	8417	8691	9010	9213	9750
$15 - 30\ min$	8391	8746	9019	9239	9858
$10 - 20\ min$	8434	8811	9137	9402	10009
stalno	8208	9148	9392	9755	10183

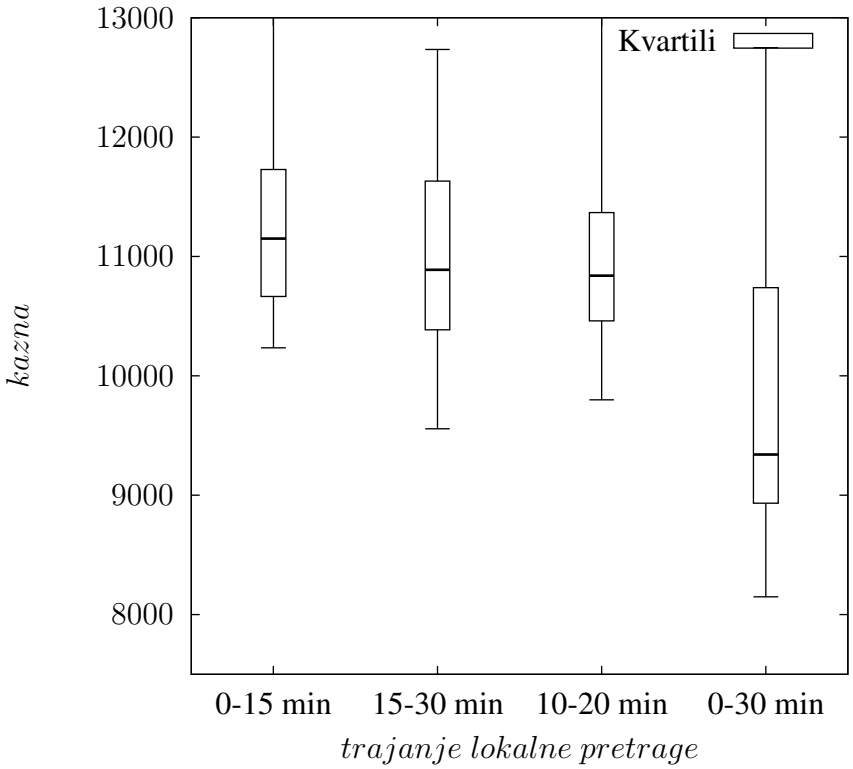
Tablica 5.26: Generacijski GA. Ovisnost kazne o trajanju lokalne pretrage.



Slika 5.23: Eliminacijski GA. Ovisnost kazne o trajanju lokalne pretrage.

trajanje lok. pretrage	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
$0 - 15\ min$	8487	8973	9112	9288	10026
$15 - 30\ min$	8500	8963	9244	9530	9964
$10 - 20\ min$	8366	8920	9208	9516	10025
stalno	8908	9483	9671	10068	11287

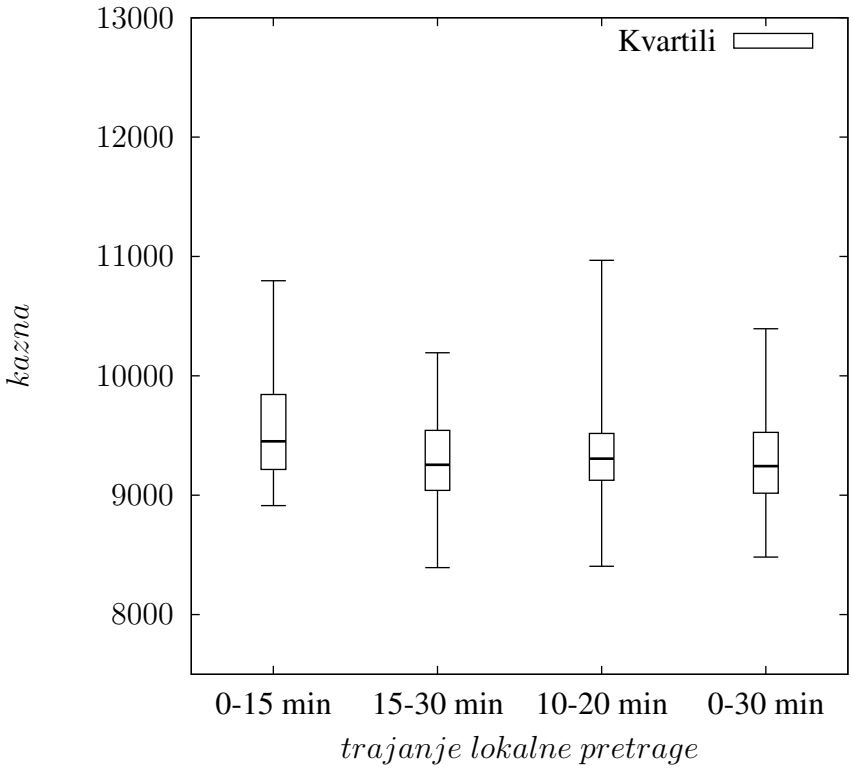
Tablica 5.27: Eliminacijski GA. Ovisnost kazne o trajanju lokalne pretrage.



Slika 5.24: Algoritam harmonijske pretrage. Ovisnost kazne o trajanju lokalne pretrage.

$trajanje\ lok.\ pretrage$	min	1. $kvartil$	$medijan$	2. $kvartil$	max
$0-15\ min$	10235	10665	11149	11728	13314
$15-30\ min$	9556	10385	10888	11630	12733
$10-20\ min$	9799	10460	10838	11367	13060
$stalno$	8149	8933	9341	10739	12748

Tablica 5.28: Algoritam harmonijske pretrage. Ovisnost kazne o trajanju lokalne pretrage.



Slika 5.25: Jednostavni imunološki algoritam. Ovisnost kazne o trajanju lokalne pretrage.

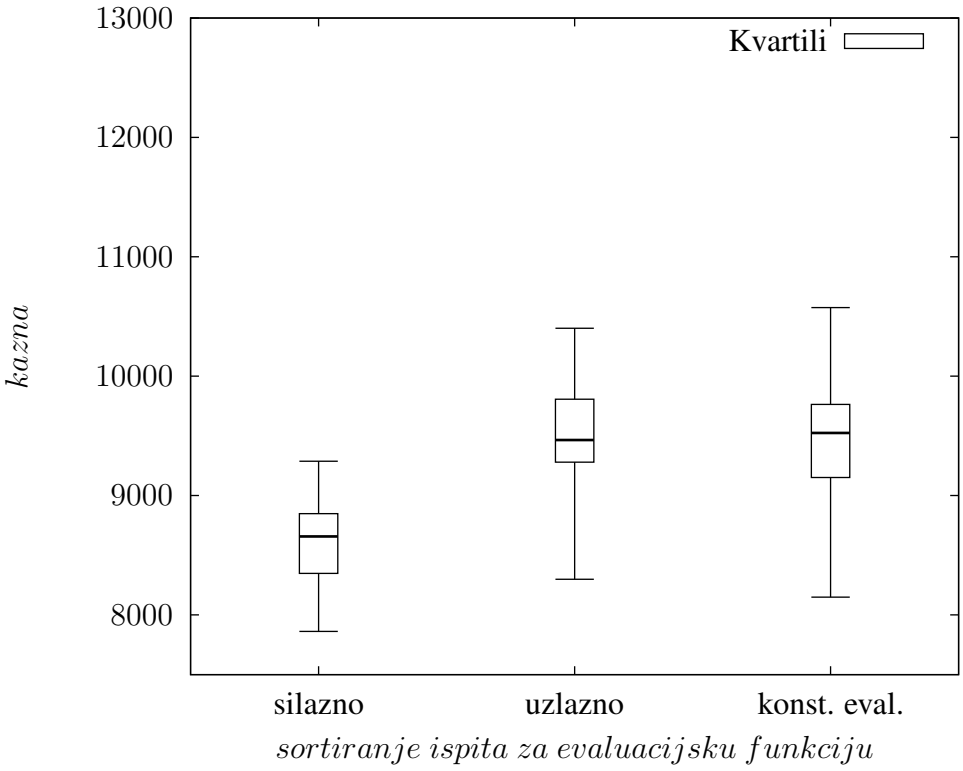
trajanje lok. pretrage	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
$0 - 15\ min$	8912	9216	9451	9843	10797
$15 - 30\ min$	8394	9040	9255	9543	10194
$10 - 20\ min$	8404	9126	9306	9517	10967
stalno	8481	9017	9243	9527	10394

Tablica 5.29: Jednostavni imunološki algoritam. Ovisnost kazne o trajanju lokalne pretrage.

5.3. Ponašanje algoritama ovisno o dinamičkoj promjeni evaluacijske funkcije

Zadnji odlomak o pokusima se bavi dinamičkom funkcijom evaluacije. Dinamička evaluacijska funkcija radi tako da u određenom vremenskom trenutku u obzir uzima samo dio međuispita pri računanju kazne za raspored. Koje međuispite će uzeti u obzir ovisi o trenutku u kojem računa kazna i ovisi o zahtjevnosti međuispita. Zahtjevnost ispita je veća što međuispit više pridonosi kazni rasporeda. Lista međuispita se sortira

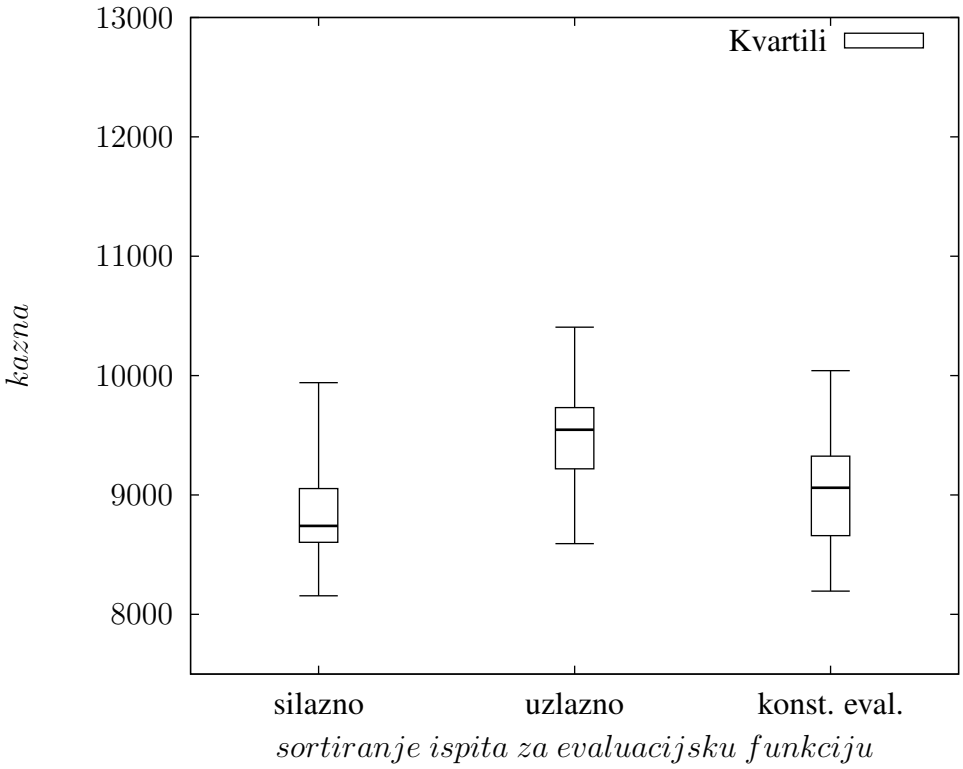
ovisno o zahtjevnosti uzlazno, odnosno silazno. Ako je prošlo manje od 10 minuta od početka pretrage algoritma, tada se u obzir uzima samo prva trećina međuispita iz sortirane liste. Između 10. minute i 20. minute se računa kazna za prve dvije trećine međuispita, a od 20. minute nadalje se računa ukupna kazna rasporeda. Pokusi su pokazali da na ovaj tip evaluacijske funkcije iznimno povoljno reagiraju imunološki i generacijski genetski algoritam (Tablice: 5.33 i 5.30), dok Harmonijska pretraga vrlo loše radi s ovakvim tipom evaluacije (Tablica 5.32).



Slika 5.26: Generacijski GA. Ovisnost kazne o ponašanju dinamičke evaluacijske funkcije. Međuispiti se sortiraju po svom utjecaju na kaznu uzlazno ili silazno. Sortirani ispiti se postupno uključuju u računanje kazne. Prva trećina sortiranih se uključuje odmah, zatim se druga trećina uključuje nakon 10 min. Nakon 20 min se uključuje zadnja trećina sortiranih ispita. Zadnji stupac koristi nepromjenjivo računanje kazne sa svim međuispitima.

sortiranje ispita	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
silazno	7862	8347	8657	8848	9287
uzlazno	8298	9279	9464	9806	10401
konst. eval.	8148	9150	9524	9763	10574

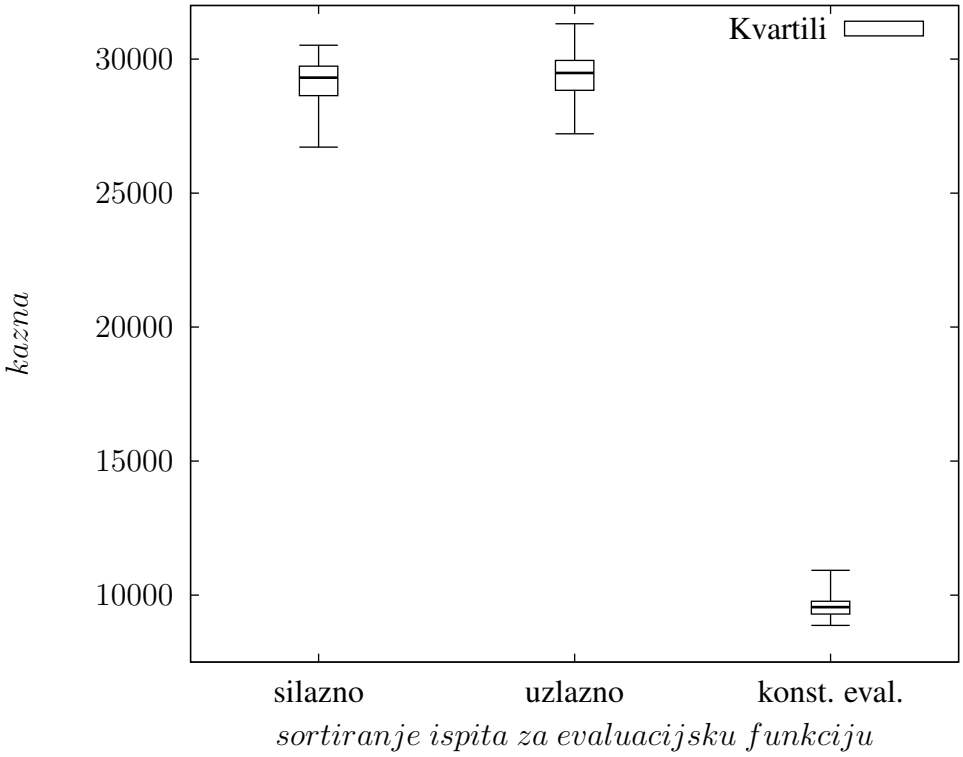
Tablica 5.30: Generacijski GA. Ovisnost kazne o ponašanju dinamičke evaluacijske funkcije. Međuispiti se sortiraju po svom utjecaju na kaznu uzlazno ili silazno. Sortirani ispiti se postupno uključuju u računanje kazne. Prva trećina sortiranih se uključuje odmah, zatim se druga trećina uključuje nakon 10 min. Nakon 20 min se uključuje zadnja trećina sortiranih ispita. Zadnji stupac koristi nepromjenjivo računanje kazne sa svim međuispitima.



Slika 5.27: Eliminacijski GA. Ovisnost kazne o ponašanju dinamičke evaluacijske funkcije. Međuispiti se sortiraju po svom utjecaju na kaznu uzlazno ili silazno. Sortirani ispiti se postupno uključuju u računanje kazne. Prva trećina sortiranih se uključuje odmah, zatim se druga trećina uključuje nakon 10 min. Nakon 20 min se uključuje zadnja trećina sortiranih ispita. Zadnji stupac koristi nepromjenjivo računanje kazne sa svim međuispitima.

sortiranje ispita	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
silazno	8155	8603	8741	9053	9940
uzlazno	8592	9219	9546	9732	10405
konst. eval.	8194	8659	9060	9325	10041

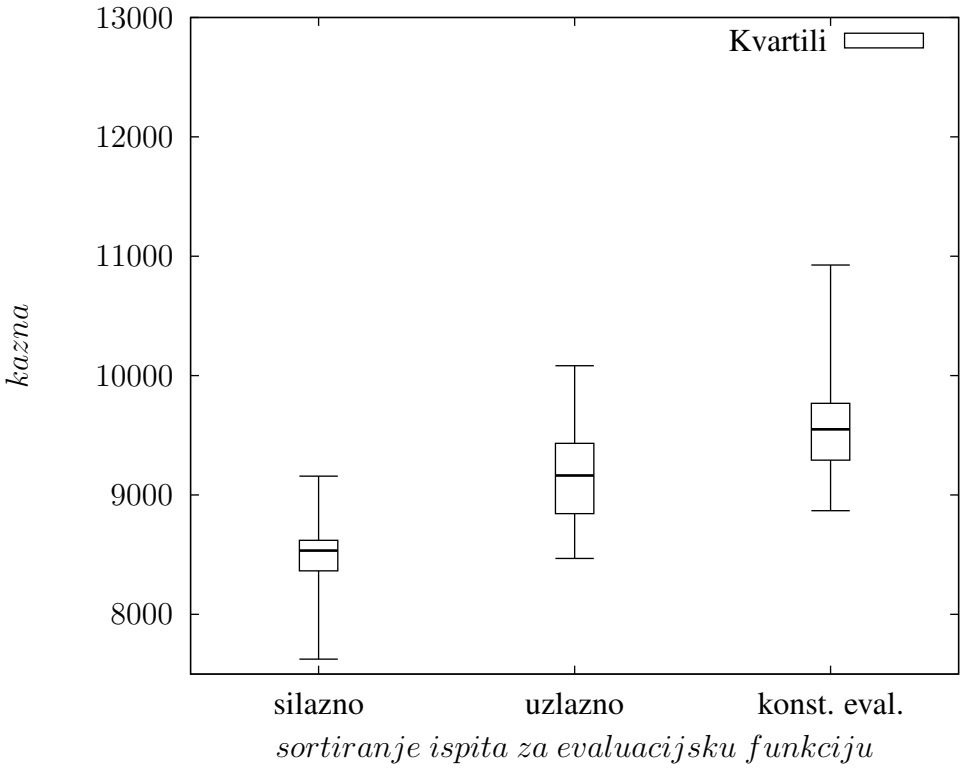
Tablica 5.31: Eliminacijski GA. Ovisnost kazne o ponašanju dinamičke evaluacijske funkcije. Međuispiti se sortiraju po svom utjecaju na kaznu uzlazno ili silazno. Sortirani ispiti se postupno uključuju u računanje kazne. Prva trećina sortiranih se uključuje odmah, zatim se druga trećina uključuje nakon 10 min. Nakon 20 min se uključuje zadnja trećina sortiranih ispita. Zadnji stupac koristi nepromjenjivo računanje kazne sa svim međuispitima.



Slika 5.28: Algoritam harmonijske pretrage. Ovisnost kazne o ponašanju dinamičke evaluacijske funkcije. Međuispiti se sortiraju po svom utjecaju na kaznu uzlazno ili silazno. Sortirani ispiti se postupno uključuju u računanje kazne. Prva trećina sortiranih se uključuje odmah, zatim se druga trećina uključuje nakon 10 min. Nakon 20 min se uključuje zadnja trećina sortiranih ispita. Zadnji stupac koristi nepromjenjivo računanje kazne sa svim međuispitima.

sortiranje ispita	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
silazno	26712	28633	29307	29734	30514
uzlazno	27210	28834	29482	29950	31316
konst. eval	8868	9291	9550	9768	10926

Tablica 5.32: Algoritam harmonijske pretrage. Ovisnost kazne o ponašanju dinamičke evaluacijske funkcije. Međuispiti se sortiraju po svom utjecaju na kaznu uzlazno ili silazno. Sortirani ispiti se postupno uključuju u računanje kazne. Prva trećina sortiranih se uključuje odmah, zatim se druga trećina uključuje nakon 10 min. Nakon 20 min se uključuje zadnja trećina sortiranih ispita. Zadnji stupac koristi nepromjenjivo računanje kazne sa svim međuispitima.



Slika 5.29: Jednostavni imunološki algoritam. Ovisnost kazne o ponašanju dinamičke evaluacijske funkcije. Međuispiti se sortiraju po svom utjecaju na kaznu uzlazno ili silazno. Sortirani ispiti se postupno uključuju u računanje kazne. Prva trećina sortiranih se uključuje odmah, zatim se druga trećina uključuje nakon 10 min. Nakon 20 min se uključuje zadnja trećina sortiranih ispita. Zadnji stupac koristi nepromjenjivo računanje kazne sa svim međuispitima.

sortiranje ispita	*min*	1. *kvartil*	*medijan*	2. *kvartil*	*max*
silazno	7625	8364	8533	8620	9157
uzlazno	8468	8844	9163	9433	10083
konst. eval.	8868	9291	9550	9768	10926

Tablica 5.33: Jednostavni imunološki algoritam. Ovisnost kazne o ponašanju dinamičke evaluacijske funkcije. Međuispiti se sortiraju po svom utjecaju na kaznu uzlazno ili silazno. Sortirani ispiti se postupno uključuju u računanje kazne. Prva trećina sortiranih se uključuje odmah, zatim se druga trećina uključuje nakon 10 min. Nakon 20 min se uključuje zadnja trećina sortiranih ispita. Zadnji stupac koristi nepromjenjivo računanje kazne sa svim međuispitima.

6. Zaključak

U ovom radu je implementirana nekolko algoritama s područja evolucijskog računanja te dva algoritma iz skupine lokalnih pretraga. Eksperimenti su provedeni za parametre populacijskih algoritama, ponašanje nekoliko memetičkih varijanti tih algoritma te ponašanje dinamičke evaluacijske funkcije. 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: eliminacijski i generacijski genetski algoritam, jednostavni imunološki algoritam i algoritam harmonijske pretrage. Rezultati su pokazali da parametar vjerojatnosti istraživačkih operatora (mutacija ili sličan operator) utječe na kvalitetu svih algoritama koji koriste taj operator. Algoritmi su relativno otporni na veličinu populacije. Obećavajuće rezultate je dala uporaba algoritama lokalne pretrage zajedno s populacijskim algoritmima u nekoliko oblika simbioze. Dinamička funkcija evaluacije koja je implementirana u provedenim pokusima je također dala obećavajuće rezultate ka unaprijeđenju dobivenih rješenja što ohrabruje istraživače u daljnjim pokušajima da istraže slične pristupes.

LITERATURA

M. Al-Betar, A. Khader, i T. Gani. A harmony search algorithm for university course timetabling. U *In 7th Intl. Conf. on the Practice and Theory of Automated Timetabling*. Springer Netherlands, 2008.

H. Arntzen i A. Løkketangen. A local search heuristic for a university timetabling problem. *nine*, 1(T2):T45.

Hugues Bersini i Francisco Varela. Hints for adaptive problem solving gleaned from immune networks. U *Parallel Problem Solving from Nature*, svezak 496, stranice 343–354. Springer Berlin / Heidelberg, 1991.

E. Burke, J. Newall, i R. Weare. A memetic algorithm for university exam timetabling. *Practice and Theory of Automated Timetabling*, stranice 241–250, 1996.

Thomas H. Cormen, Charles E. Leiserson, Ronald R. Rivest, i Cliff Stein. *Introduction to Algorithms*. McGraw-Hill, 1990.

V. Cutello i G. Nicosia. An immunological approach to combinatorial optimization problems. *Advances in Artificial Intelligence—IBERAMIA 2002*, stranice 361–370, 2002a.

V. Cutello i G. Nicosia. Multiple learning using immune algorithms. U *Proceedings of 4th International Conference on Recent Advances in Soft Computing, RASC*, stranice 102–107, 2002b.

V. Cutello i G. Nicosia. The clonal selection principle for in silico and in vitro computing. *Recent developments in biologically inspired computing*, stranice 104–146, 2004.

Charles Darwin. *On the Origin of Species by Natural Selection*. Murray, 1859.

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

D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.

M. Dorigo i T. Stützle. *Ant colony optimization*. the MIT Press, 2004.

J. Doyne Farmer, Norman H. Packard, i Alan S. Perelson. The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, 22(1-3):187 – 204, 1986. ISSN 0167-2789. Proceedings of the Fifth Annual International Conference.

Green Fourie, Mills. Harmony filter: A robust visual tracking system using the improved harmony search algorithm. *Image and Vision Computing*, 28(12):1702 – 1716, 2010. ISSN 0262-8856.

Michael R. Garey i David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman Co., 1979.

Zong Geem. Optimal cost design of water distribution networks using harmony search. *Engineering Optimization*, 2006.

Zong Geem. Harmony search algorithm for solving sudoku. U *Knowledge-Based Intelligent Information and Engineering Systems*, svezak 4692, stranice 371–378. Springer Berlin / Heidelberg, 2007.

Zong Geem, Chung-Li Tseng, i Yongjin Park. Harmony search for generalized orienteering problem: Best touring in china. U *Advances in Natural Computation*, svezak 3612, stranice 439–439. Springer Berlin / Heidelberg, 2005.

Z.W. Geem, J.H. Kim, i GV Loganathan. A new heuristic optimization algorithm: harmony search. *Simulation*, 76(2):60, 2001.

S. Goss, S. Aron, J.L. Deneubourg, i J.M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

E.G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009.

Yongjin Park Zong Woo Geem, Kang Seok Lee. Application of harmony search to vehicle routing. *American Journal of Applied Sciences*, 2005.

Utjecaj parametara algoritama evolucijskog računanja na kvalitetu rješenja za problem rasporeda međuispita

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 korišteno je pet algoritma evolucijskog računarstva: generacijski i eliminacijski genetski algoritam, jednostavni imunološki algoritam, algoritam harmonijske pretrage i algoritam mravlje kolonije. 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. Također je ispitano ponašanje simbioze populacijskih algoritama i algoritama lokalne pretrage i način na koji lokalna pretraga utječe na kvalitetu rješenja. Rezultati su pokazali da su algoritmi vrlo osjetljivi na promjenu nekih od radnih parametara. Također je kod nekih algoritama pokazan napredak u pronalaženju rješenja ako lokalna pretraga radi povremeno. Obećavajuće rezultate je dala i dinamička evaluacijska funkcija.

Ključne riječi: evolucijsko računanje, problem rasporeda ispita, genetski algoritam, harmonijska pretraga, imunološki sustav, dinamička evaluacijska funkcija

Influence of parameters of evolutionary computing algorithms on the quality of found solutions of university exam timetabling problem

Abstract

University exam timetabling problem is a NP-complete problem wich can be often found in educational institutions. Lots of heuristics based algorithms that can solve this problem can be found. Algorithms based on evolutionary computing can be used to found good solutions for many of the NP-complete problems. In order to solve exam timetabling problem in our institution we implemented five evolutionary computing algorithms: generation and steady-state genetic algorithm; harmony search algorithm; simple immune algorithm and MIN-MAX Ant System. Problem is formaly defined and implementations of algorithms are described. Later sections contain test results that show quality of solutions depending on working parameters. Also, combination of evolutionary computation algorithms and local search algorithms is tested and a way in wich local search influences solution quality. Results showed that algorithms can be very sensitive to values of some working parameters. Also, some of evolutionary algorithms usualy find better solutions if local search is working alongside. Time dependent evaluation function gave promising results in attemps to find better solutions.

Keywords: evolutionary computing, university exam timetabling problem, genetic algorithm, harmony search, simple immune algorithm, time dependent evaluation function