SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1840

Rješavanje problema raspoređivanja u visokoškolskim ustanovama evolucijskim algoritmom

Tomislav Herman

Zagreb, lipanj 2010.

Zahvaljujem mentoru, prof. dr. sc. Domagoju Jakoboviću na podršci i pruženim idejama te asistentu mr. sc. Marku Čupiću na puno uloženog vremena i velikoj pomoći pomoći bez koje projekt ne bi zaživio. Zahvaljujem članovima jAgenda tima: Zlatku Bratkoviću, Vatroslavu-Dini Matijašu, Goranu Molnaru i Vjeri Omrčen na ugodnoj suradnji, konstruktivnim raspravama i idejama.

SADRŽAJ

POPIS SLIKA

1. Uvod

U današnje vrijeme ljudi se u gotovo svim granama djelatnosti susreću s nekim oblikom problema raspoređivanja aktivnosti: od raspoređivanja predavanja i ispita u obrazovnim institucijama do raspoređivanja voznog reda u prometu. Problem raspoređivanja spada u skupinu *NP-potpunih* problema[1], koji imaju svojstvo izuzetno visoke vremenske složenosti. Tradicionalan pristup rješavanju takvih problema, korištenjem npr. ručnog raspoređivanja, deterministčkih algoritama ili iscrpne pretrage, nije preporučljiv jer može zahtijevati nerazumno mnogo vremena za pronalazak rješenja. Pri rješavanju takvih problema javlja se potreba za korištenjem heurističkih metoda. U heurističke metode ubrajaju se algoritmi koji ne jamče pronalazak najboljeg rješenja, ali su sposobni naći rješenja dovoljno dobra za korištenje, i to tijekom prihvatljivo kratkih vremena izvođenja.

Ovaj rad nastao je na temelju projekta *jAgenda* pokrenutog na Fakultetu elektrotehnike i računarstva u Zagrebu u jesen 2007. godine u sklopu seminara za predmet *Strojno učenje*. Uvođenjem Bolonjskog procesa u nastavu promijenio se način polaganja kolegija: studenti mogu birati predmete sa različitih godina, a polaganje međuispita i laboratorijskih vježbi počelo se odvijati tijekom kratkog vremenskog intervala (1–2 tjedna). Ove promjene utjecale su na složenost izrade rasporeda studentskih obaveza, jer više nije bilo moguće formiranje velikih grupa studenata s jednakim obvezama (eksperimentalno je potvrđeno da je u akademskoj godini 2007/2008 postojalo oko 1800 grupa na 3000 ljudi). Prije promjena nastavnog plana i programa, raspored predavanja bio je generiran automatski početkom godine, dok su se rasporedi laboratorijskih vježbi izrađivali decentralizirano —- svaki je zavod neovisno od drugih izrađivao raspored sati za vježbe koje je održavalo osoblje tog zavoda. Problem decentralizirane izrade bio je u tome što je za mnoge studente postojao velik broj potencijalnih vremenskih preklapanja sa obavezama na drugim zavodima. Cilj projekta *jAgenda* bio je

[1] *NP* problemi u teoriji računarstva su problemi koji su rješivi nedeterminističkim strojem u vremenu koje polinomno ovisi o veličini problema. Problem je *NP-potpun* ako se svaki NP problem može u polinomnom vremenu svesti na njega.

izraditi raspored za laboratorijske vježbe centralizirano, vodeći računa o svim obavezama studenata, te o raspoloživosti dijeljenih resursa (prostorije, oprema, asistenti).

Aplikacija za izradu rasporeda laboratorijskih vježbi temeljila se na dvjema metodama evolucijskog računarstva: *genetskom algoritmu* i *optimizaciji kolonijom mrava*. Oba pristupa problemu razvijana su istodobno, te su se oba pokazala uspješnim u generiranju rasporeda. Razvijeni sustav primjenjuje se na fakultetu od ljetnog semestra akademske godine 2007/2008 uz povremene modifikacije i nadogradnje. Razvijena je i web aplikacija koja prije izrade laboratorijskih vježbi omogućava definiranje zahtjeva od strane nastavnog osoblja. Nakon što se prikupe svi zahtjevi, pokreće se automatizirana izrada, a generirani raspored se objavljuje kroz *Ferko*, sustav za upravljanje kolegijima.

Ovaj rad je usmjerem prvoj od prethodno navedenih metoda, rješavanju problema raspoređivanja primjenom *genetskog algoritma*. U nastavku rada u poglavlju 2. donosi se uvod u *evolucijsko računarstvo* i njegove metode, s posebnim naglaskom na *evolucijske algoritme* i *genetski algoritam*. Opis problema raspoređivanja dan je u poglavlju 3., uz definiciju problema raspoređivanja laboratorijskih vježbi na Fakultetu elektrotehnike i računarstva. U poglavljima 4. i 5. opisuje se implementirani sustav za izradu rasporeda laboratorijskih vježbi temeljen na genetskom algoritmu. Poglavlje 6. pruža prikaz rezultata ispitivanja učinkovitosti sustava u ovisnosti o različitim vrijednostima parametara. Na kraju, u poglavlju 7. iznosi se zaključak.

2. Evolucijsko računarstvo

Evolucijsko računarstvo je grana umjetne inteligencije koja je usmjerena rješavanju problema iz domene kombinatorne optimizacije. Tehnikama evolucijskog računarstva prostor rješenja zadanog problema pretražuje se nasumično i heuristički usmjereno. Heuristike koje se pritom koriste najčešće su inspirirane procesima u prirodi: evolucijom, procesima u velikim skupinama životinja, procesima u organizmu, procesima na mikroskopskoj razini u materijalima, itd. Tehnike evolucijskog računarstva dijele se u dvije velike skupine: *Evolucijski algoritmi* i *Inteligencija roja*. Ovaj rad usmjeren je prvoj skupini, preciznije, tehnikama iz te skupine koje se nazivaju *Genetski algoritmi*, pa će stoga ta skupina biti detaljnije obrađena.

2.1. Evolucijski algoritmi

Evolucijski algoritmi zasnivaju se na ideji evolucije vrsta u prirodi. U prirodi postoji populacija jedinki koja je pod pritiskom okoline. Neke od tih jedinki bolje su prilagođene okolini, što im daje veću vjerojatnost preživljavanja. Jedinke koje prežive u populaciji stvorit će potomke kojima će prenijeti dio svojih dobrih svojstava. Iz generacije u generaciju jedinke u populaciji postaju u sve većoj mjeri prilagođene okolini i vrsta se s vremenom približava optimalnom obliku. Na proces evolucije djeluju dva osnovna procesa: *nasljeđivanje* i *mutacija*. Nasljeđivanje djeluje tako da dobar genetski materijal ostaje u populaciji i usmjerava evoluciju prema što većoj prilagođenosti jedinki okolini. Mutacija pak djeluje tako da u populaciju unosi novi genetski materijal. Oba navedena procesa su ključna u procesu evolucije. Odsutnost mutacije može uzrokovati nestanak različitosti u populaciji, a time i nestanak mogućnosti napretka, dok odsutnost svojstva nasljeđivanja evoluciju pretvara u potpuno slučajni i neusmjereni proces.

Evolucijskim algoritmima se po uzoru na prirodnu odvija umjetna evolucija u računalu. Primjenjuju se kod problema kod kojih postoji velik prostor rješenja čije bi iscrpno pretraživanje ili primjena konvencionalnih algoritama zahtijevalo nerazumno

mnogo vremena. Prilikom implementacije evolucijskog algoritma za rješavanje takvog problema primjenjuje se analogija sa evolucijom u prirodi kao što je navedeno u tablici 2.1.

Tablica 2.1: Analogija evolucije u prirodi i evolucijskih algoritama

Prirodna evolucija	$\leftrightarrow$	Evolucijski algoritmi
Jedinka	$\leftrightarrow$	Moguće rješenje
Okolina	$\leftrightarrow$	Optimizacijski problem
Prilagođenost okolini	$\leftrightarrow$	Funkcija dobrote

Evolucijski algoritmi se razlikuju od ostalih metoda pretraživanja prostora rješenja po tome što se pretraga ne vrši iz jedne početne točke u prostoru rješenja, nego iz više njih istovremeno. Skup točaka iz kojih počinje pretraga naziva se početna populacija. Veličina početne populacije ovisi o problemu koji se rješava i o njoj može ovisiti učinkovitost pretrage. Tijekom evolucije veličina populacije ostaje nepromijenjena: jedinke u populaciji koje zbog male vrijednosti dobrote izumru zamijenit će nove jedinke nastale primjenom genetskih operatora rekombinacije nad preživjelim jedinkama. Također, preživjele jedinke u populaciju mogu se promijeniti kao posljedica primjene operatora mutacije. Proces evolucije je iterativan i u svakoj iteraciji primjenjuju se koraci kako je prikazano u algoritmu 1:

Algoritam 1 Evolucijski algoritam

$t = 0$
$P(0) = generirajPocetnuPopulaciju()$
$evaluiraj(P(0))$
repeat
  $t = t + 1$
  $P'(t) = izaberiRoditeljeIz(P(t))$
  $rekombiniraj(P'(t))$
  $mutiraj(P'(t))$
  $evaluiraj(P'(t))$
  $P(t) = prezivjeliIz(P'(t) \cup P(t))$
until zadovoljen uvjet zavrsetka

2.2. Implementacija evolucijskog algoritma

Evolucijski algoritam ima brojne sastavne dijelove, operatore i procedure koji moraju biti definirani kako bi se navedeni evolucijski algoritam mogao implementirati [4]. Najvažnije komponente su:

- Reprezentacija rješenja
- Evaluacijska funkcija
- Populacija
- Mehanizam odabira roditelja
- Operatori varijacije: rekombinacija i mutacija
- Mehanizam odabira preživjelih

Da bi se evolucijski algoritam mogao izvoditi potrebno je osim prethodnih elemenata definirati još i postupak inicijalizacije i uvjet završetka izvođenja.

2.2.1. Reprezentacija rješenja

Prije implementacije evolucijskog algoritma potrebno je dobro poznavati problem, te ga precizno definirati. Prvi korak implementacije je definiranje preslikavanja iz domene originalnog problema u domenu u kojoj djeluje evolucija. Elementi iz domene problema koji predstavljaju moguće rješenje nazivaju se *fenotipovi*, dok se njihovi ekvivalenti, elementi iz domene evolucijskog algoritma nazivaju *genotipovi*. Postupak preslikavanja fenotipa u ekvivalentni genotip naziva se *kodiranje*, dok se obrnut postupak naziva *dekodiranje*. Česta reprezentacija fenotipa u prostoru genotipa je binarni kod, npr. cijeli brojevi iz domene problema kodiraju se u binarni zapis u domeni genotipa. Prostor fenotipa se može bitno razlikovati od prostora genotipa. Pretraživanje prostora rješenja uvijek se odvija u prostoru genotipa, dok se konačno rješenje nakon završetka izvođenja evolucijskog algoritma dobiva dekodiranjem najboljeg genotipa iz populacije. Reprezentacija rješenja, osim preslikavanja iz domene problema u domenu evolucijskog algoritma, znači i definiranje podatkovne strukture genotipa.

2.2.2. Evaluacijska funkcija

Evaluacijska funkcija u evolucijskom algoritmu definira zahtjeve kojima se rješenja trebaju prilagoditi. Ona je jedini kriterij selekcije preživjelih jedinki i uloga joj je usmjeravanje evolucije ka boljim rješenjima. Evaluacijska funkcija svakom prijedlogu

rješenja dodjeljuje numeričku vrijednost koja govori koliko je to rješenje zapravo dobro u domeni problema. Evaluacija rješenja se odvija u prostoru fenotipa, pa je zbog toga prethodno potrebno provesti dekodiranje genotipa. U terminologiji evolucijskog računarstva evaluacijska funkcija se često naziva *funkcija dobrote*, ili *fitness*.

2.2.3. Populacija

Populacija je skup genotipa koji se tijekom evolucije mijenja pod utjecajem operatora. Na razini populacije djeluju operatori selekcije, dok operatori varijacije djeluju na razini jedinke. Selekcijski operatori uzimaju u obzir cijelu populaciju i djeluju u odnosu na njezino trenutno stanje. Primjerice, vjerojatnost preživljavanja jedinke ne ovisi o apsolutnoj vrijednosti njezine dobrote, nego o vrijednosti dobrote u odnosu na ostale jedinke iz populacije. Parametar koji opisuje populaciju je njezina veličina, koja je u većini implementacija konstantna tijekom cijelog evolucijskog procesa. U nekim složenijim implementacijama populacija ima prostornu strukturu temeljenu na mjeri udaljenosti između jedinki ili relaciji susjedstva. Raznovrsnost populacije je mjera kojom se izražava broj različitih jedinki u populaciji. Postoji više mjera raznovrsnosti: broj različitih genotipa, broj različitih fenotipa, broj različitih vrijednosti dobrote ili entropija u nekim složenijim implementacijama. Treba imati na umu da se jedna vrijednost funkcije dobrote može se pojaviti kod više fenotipova i da jedan fenotip može biti kodiran sa više genotipova, dok obrnutom smjeru to ne vrijedi.

2.2.4. Mehanizam odabira roditelja

Uloga mehanizma odabira roditelja je razlikovanje jedinki u populaciji na temelju njihove kvalitete, te da kvalitetnijim jedinkama da veću vjerojatnost da postanu roditelji sljedećoj generaciji. Na jedinke koje su odabrane kao roditelji primjenjuju se varijacijski operatori kojima se stvaraju potomci. Uz mehanizam odabira preživjelih jedinki, odabir roditelja odgovoran je za povećanje kvalitete populacije. Mehanizam odabira roditelja je stohastički: kvalitetnije jedinke iz populacije imaju veću vjerojatnost da postanu roditelji od lošijih jedinki. Lošije jedinke moraju također imati mogućnost da postanu roditelji, inače bi pretraga postala pohlepna i zapela u lokalnom optimumu.

2.2.5. Operatori varijacije: rekombinacija i mutacija

Uloga operatora varijacije je stvaranje novih jedinki od postojećih u populaciji. Postoje dva operatora varijacije koji se razlikuju po načinu djelovanja i po mjesnosti.

Unarni operator varijacije naziva se *mutacija*. Primjenjuje se na jednu jedinku i stvara novu jedinku koja se naziva *mutant* ili *potomak*. Operator mutacije djeluje na genotip jedinke tako da nasumično odabire njegove dijelove te ih mijenja. Važno je napomenuti da se jedino operatorima varijacije ostvaruju pomaci u prostoru pretraživanja. Mutacija ovdje ima ulogu da jamči povezanost prostora rješenja, tj. da se iz svake točke prostora može djelovanjem operatora mutacije doći do bilo koje druge točke. To se postiže davanjem pozitivne vjerojatnosti mutacije svakom genu unutar kromosoma. Ovo svojstvo je važno jer pronalazak globalnog optimima (uz neograničeno vrijeme na raspolaganju) oslanja se na dohvatljivost svakog genotipa primjenom isključivo operatora varijacije.

Binarni operator varijacije naziva se *rekombinacija* ili *križanje*. Križanje uzima genetski materijal dvije ili više jedinki i stvara potomka koji sadrži dio informacija od svakog roditelja. Postupak je nedeterministički: nasumično se bira koji dio informacija će potomak naslijediti od kojeg roditelja i način na koji će se ti dijelovi kombinirati. Priliku za križanje u većoj mjeri dobiju kvalitetnije jedinke - to su jedinke koje posjeduju jednim dijelom kvalitan genetski materijal. Križanjem dviju kvalitetnih jedinki može se dobiti potomak koji će od svakog roditelja naslijediti njegova dobra svojstva te njihovom kombinacijom postati kvalitetniji od roditelja.

Implementacija operatora varijacije ovisi o reprezentaciji kromosoma. Kod binarne reprezentacije kromosoma geni su bitovi, dok u složenijim reprezentacijama geni mogu biti cijeli brojevi ili još složenije strukture podataka.

2.2.6. Mehanizam odabira preživjelih jedinki

Križanjem nastaju nove jedinke. Budući da je veličina populacije stalna, potrebno je iz populacije izbaciti broj jedinki jednak broju stvorenih potomaka. Mehanizam odabira preživjelih jedinki oslanja se, kao i odabir roditelja, na kvalitetu jedinki - u većoj mjeri preživjet će jedinke koje su kvalitetnije. Postoji više strategija selekcije jedinki, a najčešće su:

- Jednostavna selekcija stvara novu generaciju tako da iz stare bira jednu po jednu jedinku koja će se pojaviti u novoj generaciji. Svaka jedinka ima vjerojatnost odabira proporcionalnu svojoj dobroti. Može se dogoditi da jedna jedinka bude više puta izabrana i zbog toga više njezinih kopija prisutno u sljedećoj generaciji.
- Turnirska selekcija uzima k slučajnih jedinki iz populacije, odabire najbolju od njih i smješta ju u novu generaciju. Nakon popunjavanaja nove generacije

primjenjuju se operatori varijacije.

– Eliminacijska selekcija odabire jednu po jednu jedinku koju će izbaciti iz populacije, a nju nadomješta križanjem preživjelih. Jedinke koje će se izbaciti biraju se tako da veću vjerojatnost imaju one sa manjom dobrotom, tj. jedinke sa većom kaznom. Prednost ove strategije je u tome što nije potrebno čuvati dvije uzastopne generacije u istom koraku algoritma.

2.2.7. Inicijalizacija

Inicijalizacija je postupak stvaranja početne populacije jedinki. U većini strategija postpak inicijalizacije je identičan - slučajno se generiraju genotipovi zadanog broja jedinki. Neke strategije pokušavaju već kod inicijalizacije generirati kvalitetnu populaciju korištenjem dodatnih metoda pretraživanja ili uvođenjem kvalitetnih jedinki koje su rezultat prethodno izvođenog evolucijskog algoritma ili neke druge heurističke metode. Optimiranje početne populacije iziskuje dodatno utrošeno procesorsko vrijeme, pa treba uzeti u obzir isplativost optimiranja.

2.2.8. Uvjet završetka

Uvjetom završetka definiraju se ciljevi koje evolucija mora postići. Ako je unaprijed poznata optimalna vrijednost funkcije dobrote, tada uvijet završetka postaje približavanje populacije toj vrijednosti, sa nekom minimalnom udaljenošću $\varepsilon > 0$. No često optimalna vrijednost nije unaprijed poznata, ili evolucijski algoritam ne može dostići tu vrijednost u razumnom vremenu. Zbog toga se definiraju dodatni uvijeti koji, uz uvijet dostizanja optimalne vrijednosti dobrote, određuju trenutak završetka izvođenja. Dodatni uvijeti koji se koriste su:

– maksimalno vrijeme izvođenja

– maksimalan broj iteracija, generacija ili izračunavanja vrijednosti funkcije dobrote

– u zadanom vremenu napredak populacije je manji od nekog praga

– raznolikost populacije pala je ispod zadanog praga

Najčešće se za uvijet završetka uzima dostizanje optimalne vrijednosti funkcije dobrote u kombinaciji sa jednim od prethodno navedenih uvijeta.

2.3. Podjela Evolucijskih algoritama

Evolucijski algoritmi po načinu djelovanja i reprezentaciji jedinke dijele u više podklasa:

- Genetski algoritam
- Genetsko programiranje
- Evolucijska strategija
- Evolucijsko programiranje

2.3.1. Genetski algoritam

Genetski algoritam koristi podatkovnu strukturu (niz bitova ili složenijih podataka) kao zapis jedne jedinke. Takva podatkovna struktura naziva se kromosom i u potpunosti predstavlja jedno rješenje. U svakoj iteraciji evolucije primjenjuje se selekcija preživjelih jedinki kao i selekcija jedinki koje će postati roditelji. Genetski operatori djeluju nad kromosomima: operator mutacije mijenja vrijednost nasumičnog dijela podatkovne strukture, dok operator križanja uzima dijelove podatkovnih struktura iz roditelja te njima gradi potomka. U većini slučajeva se koristi konstantna veličina zapisa kromosoma, što olakšava implementaciju genetskih operatora. Prije evaluacije jedinke potrebno je prevesti podatkovnu strukturu iz domene kromosoma u domenu problema, što ovisno o načinu kodiranja može biti vremenski zahtjevno.

2.3.2. Genetsko programiranje

Genetsko programiranje razlikuje se od genetskog algoritma po tome što svaka jedinka predstavlja program koji će stvoriti rješenje problema, dok je jedinka kod genetskog algoritma predstavlja gotovo rješenje. Jedinka se uobičajeno predstavlja kao stablasta struktura. Svaki nezavršni čvor unutar stabla predstavlja neki operator (aritmetički, logički ili funkcijski), a završni čvorovi sadrži operande. Nad stablima se primjenjuju operatori mutacije i rekombinacije. Rekombinacija na mjesto čvora jedne jedinke stavlja odgovarajući čvor iz druge jedinke, pritom zamjena čvora znači zamjena cijelog podstabla koje pripada tom čvoru. Mutacija mijenja nasumični čvor promjenom operatora u čvoru, ili promjenom čvoru pripadajućeg podstabla. Jedinka se evaluira na temelju kvalitete rješenja kojeg daje njezin program.

2.3.3. Evolucijska strategija

Evolucijska strategija je po načinu djelovanja slična je genetskom algoritmu po korištenju operatora mutacije i rekombinacije. Razlikuje se po tome što je reprezentacija jedinke bliska domeni problema — kromosom je sastavljen od niza realnih brojeva koji predstavljaju argumente neke višedimenzionalne funkcije koja se optimira, a evaluacija jedinke ne zahtjeva dekodiranje genotipa. Naglasak je na primjeni operatora mutacije kojim se stvaraju potomci, tako da se svakom elementu niza dodaje slučajan broj po Gaussovoj distribiciji sa srednjom vrijednošću $\xi = 0$ i promjenjivom standardnom devijacijom σ. Nakon što se stvori dovoljan broj potomaka slijedi proces selekcije. Koristi se (μ, λ) selekcija, gdje je μ broj jedinki u populaciji, a λ broj potomaka koji se generiraju u jednoj iteraciji, i vrijedi $\lambda > \mu$. Nakon što se stvori λ potomaka, među njima se bira μ najboljih koji će postati roditelji sljedeće generacije. Rekombinacija se koristi u manjoj mjeri, a selekcija roditelja koji će sudjelovati u tom procesu ne temelji se na dobroti, nego je potpuno slučajna. Novost koju uvodi evolucijska strategija je samoprilagodljivi operator mutacije. Parametar mutacije σ (korak) koji definira maksimalan pomak jedinke mijenja se zajedno sa jedinkom kroz proces evolucije. Svaka jedinka osim niza realnih brojeva sadrži korak mutacije. Prilagodljiva mutacija omogućuje jedinkama da se prilagode fazi evolucije, ili trenutnoj okolini. U početku evolucije potrebne su jače mutacije kako bi se pretražilo što više regija prostora, a kasnije, dok je populacija blizu optimuma, slabe mutacije kojima će se pretraživati okolina u manjim koracima.

2.3.4. Evolucijsko programiranje

Evolucijsko programiranje u počecima je imalo primjenu na optimizinaciji konačnih automata za predviđanje podataka i njihovu klasifikaciju. U osnovi je slično evolucijskoj strategiji, a razlikuje se po tome što ne koristi operator rekombinacije. Konceptualno, na svaku jedinku u populaciji gleda se kao na zasebnu vrstu, te je iz tog razloga operator rekombinacije izbačen iz upotrebe. Mutacija je kao i u evolucijskoj strategiji prilagodljiva, a parametri koji definiraju mutaciju dio su genotipa svake jedinke. Koristi se $(\mu + \mu)$ selekcija preživjelih jedinki, gdje je μ veličina populacije. Svaka jedinka iz trenutne generacije stvara potomka primjenom operatora mutacije, a nakon toga se turnirskom selekcijom od $\mu + \mu$ izabire μ jedinki koje će postati roditelji sljedećoj generaciji. Zbog turnirskog odabira i loše jedinke imaju vjerojatnost da budu roditelji u sljedećoj generaciji, za razliku od evolucijske strategije, gdje se preživjele jedinke biraju deterministički. Primjena evolucijskog programiranja nije usko vezana

za jedan problem pa se zbog toga operator mutacije implementira ovisno o problemu.

2.4. Genetski algoritam

Genetski algoritam formalno je definirao John Holland 1970-ih godina na sveučilištu u Michiganu. Povećavanje računalne snage u odnosu na cijenu kroz zadnjih četrdeset godina učinili su genetski algoritam sve popularnijom metodom optimizacije raznih problema. U praksi, pokazali su se vrlo uspješnom tehnikom na diskretnim i kontinuiranim kombinatornim problemima. Za razliku od metoda koje koriste gradijent kao informaciju za pretragu, genetski algoritmi su otporniji na zapinjanje u lokalnom optimumu, ali su zato računski zahtjevniji.

U domeni problema prostor rješenja može biti diskretan ili kontinuiran. Primjer za problem sa diskretnim prostorom rješenja je problem trgovačkog putnika. U tom problemu postoji određen broj gradova i definirana udaljenost za svaki par različitih gradova. Zadatak je obići sve gradove tako da se prođe najmanja udaljenost. Prostor rješenja ovog problema je diskretan konačan skup: skup permutacija gradova. Predstavnik problema s kontinuiranim prostorom rješenja je minimizacija realne funkcije jedne ili više varijabli. Budući da se genetski algoritam izvodi na računalu, prostor rješenja u toj domeni može biti jedino diskretan. Zbog toga je potrebno probleme sa kontinuiranim prostorom prilagoditi, tako da genetski algoritam pretražuje konačan i diskretan prostor. Prilagodba uključuje definianje intervala u domeni nad kojim će se odvijati pretraga, te uzorkovanje intervala na zadani broj uzoraka. Prilikom uzorkovanja intervala treba uzeti u obzir da će veći broj uzoraka povećati preciznost pretrage, ali će zato povećati njezino trajanje.

Najjednostavnija reprezentacija rješenja kod genetskog algoritma je kromosom zapisan kao niz bitova. Pritom prostor rješenja sadrži upravo onoliko elemenata koliko je različitih vrijednosti moguće zapisati sa zadanim brojem bitova. Prednosti takve reprezentacije su jednostavnost i univerzalnost genetskih operatora koji će djelovati nad kromosomom: križanjem se zamjenjuju dijelovi binarnog zapisa, dok se mutacijom mijenja binarni zapis na nasumičnoj poziciji. No postoje problemi kod kojih nije jednostavno provesti kodiranje rješenja u niz bitova. Primjer takvog problema je problem rasporeda sati koji ima diskretan prostor rješenja, a osim toga različite komponente unutar jednog rješenja mogu biti međusobno zavisne. Reprezentacija nizom bitova ima za cijenu skup postupak kodiranja i dekodiranja rješenja pri evaluaciji funkcije dobrote. Zbog toga postoje dva pristupa implementaciji genetskog algoritma s obzirom na reprezentaciju rješenja:

1. Problem prilagoditi genetskom algoritmu tako da se svako rješenje može prikazati nizom bitova. Genetski operatori su univerzalni i brzo se izvode. Prije evaluacije rješenja potrebno je dekodirati kromosom da se dobije rješenje iz domene problema.

2. Genetski algoritam prilagoditi problemu tako da svaka jedinka bude predstavljena kao rješenje iz domene problema. Potrebno je definirati genetske operatore koji će biti specifični za ovakvu reprezentaciju rješenja. Izvođenje genetskih operatora je znatno sporije, a moguće je i da prilikom izvođenja genetski operatori moraju paziti na razna ograničenja prostora rješenja. Zbog intuitivnog zapisa jedinke evaluacija dobrote postaje jednostavnija za implementaciju i izvođenje.

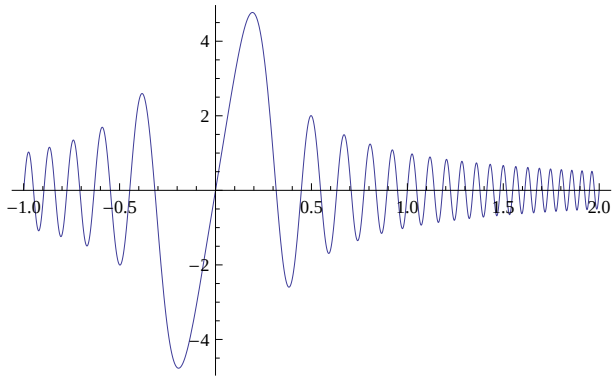
2.5. Primjer jednostavnog genetskog algoritma na problemu minimiziranja funkcije

Jednostavan genetski algoritam bit će prikazan na primjeru problema minimizacije analitičke funkcije zadane izrazom 2.1.

$$f(x) = \frac{sin(10\pi x^2)}{x} \tag{2.1}$$

Traži se minimum zadane funkcije na intervalu $[-1, 2]$ iz skupa realnih brojeva. Graf funkcije na tom intervalu prikazan je na slici 2.1. Funkcija dobrote koju genetski algoritam koristi za evaluaciju jedinki povezana je sa zadanom funkcijom: jedinka je to bolja što je manja odgovarajuća vrijednost zadane funkcije za tu jedinku. Može se primijetiti da funkcija ima brojne lokalne minimume, te samo jedan globalni. Od genetskog algoritma se očekuje da ne zapne u nekom od lokalnih minimuma te da se što više približi globalnom.

Genetski algoritam u ovom primjeru koristi binarni zapis kromosoma duljine 16 (slika 2.2). Zadanom duljinom kromosoma može se prikazati 2^{16}, točnije 65536 različitih genotipova. Preslikavanje iz domene genotipova u domenu fenotipova bit će izvedeno tako da najmanja vrijednost genotipa (svi bitovi u nuli) odgovara fenotipu sa najmanjom numeričkom vrijednošću -1, a najveća vrijednost genotipa (svi bitovi su jedan) fenotipu s najvećom vrijednošću 2. Svi ostali genotipovi preslikavaju se tako da su ravnomjerno raspoređeni između vrijednosti -1 i 2. Svaki genotip može se predstaviti kao pozitivan cijeli broj b po formuli 2.2



Slika 2.1: Optimizacijska funkcija

0	1	0	1	1	1	0	0	1	1	0	0	0	1	0	1

Slika 2.2: Genotip jedne jedinke

$$b = \sum_{i=0}^{n-1} B_i 2^i \tag{2.2}$$

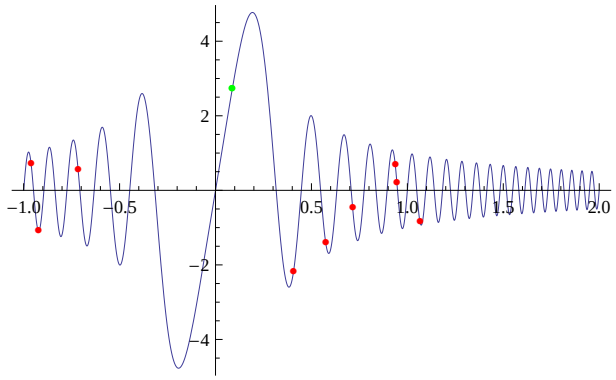
Gdje je B_i vrijednost i-tog bita u zapisu genotipa. Iz poznate brojčane vrijednosti b genotipa može se dekodirati odgovarajuća vrijednost genotipa, točnije, vrijednost argumenta minimizirajuće funkcije, po formuli 2.3. Fenotip se kodira u odgovarajući fenotip po formuli 2.4. Oznake dg i gg označavaju vrijednosti donje i gornje granice intervala fenotipova, n duljinu binarnog zapisa genotipa, a b i x odgovarajuće vrijednosti genotipa i fenotipa kod preslikavanja.

$$x = dg + \frac{b}{2^n - 1}(gg - dg) \tag{2.3}$$

$$b = \frac{x - dg}{gg - dg}(2^n - 1) \tag{2.4}$$

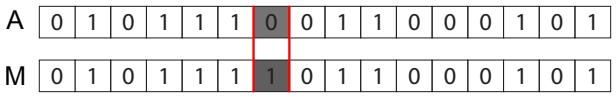
Početna populacija se inicijalizira tako da se nasumično generira zadani broj genotipova, u ovom primjeru 10. Na slici 2.3 prikazana je početna populacija u domeni problema. Zelenom točkom prikazana je jedinka sa slike 2.2 čiji je postupak dekodiranja i evaluacije naveden u nastavku:

$$b = 23749$$
$$x = dg + \frac{b}{2^n - 1}(gg - dg) = -1 + \frac{23749}{65535} \cdot 3 = 0.0871595$$
$$f(x) = \frac{sin(10\pi x^2)}{x} = \frac{sin(10\pi \cdot 0.0871595^2)}{0.0871595} = 2.71228$$



Slika 2.3: Početna populacija jedinki

Operator mutacije djeluje na slučajno odabrani element binarnog niza tako da mu promijeni vrijednost. Svaki bit u genotipu ima neku, relativno malu, vjerojatnost da će mutirati. Na slici 2.4 prikazano je djelovanje operatora mutacije na jedan bit genotipa.

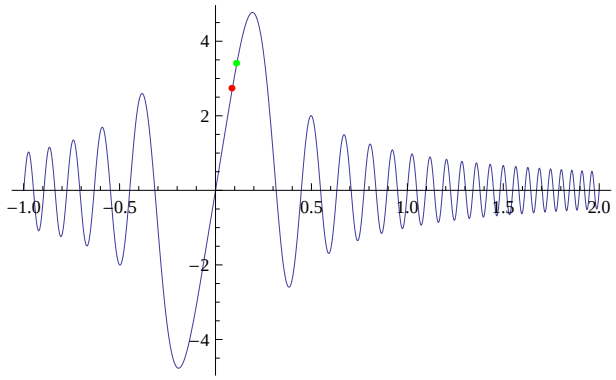


Slika 2.4: Mutacija jednog bita jedinke

Promjenom genotipa jedinke, mijenja se i odgovarajući fenotip te vrijednost funkcije. Rezultat mutacije u domeni fenotipa prikazan je na slici 2.5. Originalna jedinka označena je crvenom, a mutirana zelenom točkom. U ovom slučaju rezultat mutacije je lošija jedinka: mutacija je pomaknula fenotip u desno za relativno mali interval, što je rezultiralo povećanjem vrijednosti funkcije. Rezultat mutacije može biti, naravno, i bolja jedinka. Treba primijetiti da jačina mutacije ovisi o poziciji bita koji se mijenja: promjena bita najveće težine pomaknut će fenotip jedinke za duljinu pola intervala domene, dok će ga promjena najnižeg bita pomaknuti na najbližu susjednu vrijednost.

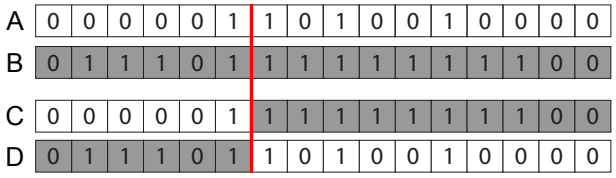
Osim jednostavne, postoji i mutacija koja djeluje na temelju vjerojatnosti na cijelom genotipu. Ako se genotip odabere za mutaciju, tada ona djeluje na podskup njegovih bitova. Djelovanje mutacije može biti invertirajuće, gdje se svakom odabranom bitu promijeni vrijednost, ili miješajuće, gdje se nasumično zamijene pozicije odabranih bitova.

Operator križanja kombinira dijelove genotipa roditelja te njima stvara potomka. Jednostavno križanje dva roditelja prethodno navedene populacije prikazan je na slici 2.6. Mjesto u genotipu označeno vertikalnom crtom naziva se točka križanja. Genetski materijal prije točke križanja potomak dobiva od jednog roditelja, a ostatak od drugog

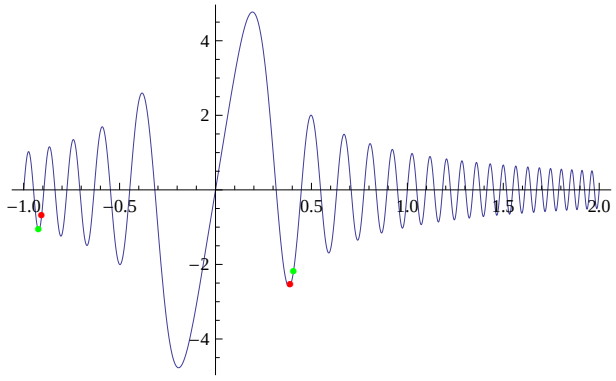


Slika 2.5: Rezultat mutacije u domeni fenotipa

roditelja. Križanjem nastaje jedinka koja se razlikuje od oba roditelja, a od svakog nosi dio gena. U domeni fenotipa potomak će također dobiti novu vrijednost što je ilustrirano slikom 2.7. Na slici su crvenim točkama označeni fenotipovi roditelja, a zelenom fenotipovi potomaka.



Slika 2.6: Djelovanje operatora križanja na genotip



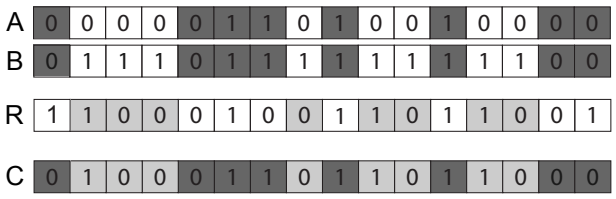
Slika 2.7: Rezultat križanja u domeni problema

Križanje se može odvijati i u više točaka gdje se između svake dvije susjedne točke bira roditelj od kojeg će biti preuzet genetski materijal. Najveći broj točaka križanja je $n - 1$, gdje je n duljina genotipa. U tom slučaju križanje postaje uniformno: ako su

odgovarajući bitovi(geni) kod roditelja jednaki, tada će i potomak naslijediti tu vrijednost, a ako nisu jednaki tada dijete na tom mjestu dobiva slučajnu vrijednost. Uniformno križanje može se izraziti korištenjem jednostavnih logičkih operatora(2.5) te se zbog toga može vrlo brzo izvoditi na računalu.

$$C = AB + R(A \oplus B) \tag{2.5}$$

A i B su binarni zapisi roditelja, dok je R slučajno generiran binarni niz. Ako za poziciju u nizu i vrijedi $A_i = B_i$, tada za dijete vrijedi $C_i = A_i = B_i$, inače $C_i = R_i$. Primjer uniformnog križanja ilustriran je na slici 2.8.



Slika 2.8: Djelovanje uniformnog križanja na genotip

3. Problem raspoređivanja

Problem raspoređivanja je problem kombinatorne optimizacije [3] koji se sastoji od četiri konačna skupa: (1) skupa sastanaka, (2) skupa resursa poput soba i nastavnog osoblja, (3) skupa vremenskih odsječaka u kojima je raspoređivanje moguće i (4) skupa ograničenja. Zadatak je svakom danom događaju pridružiti resurse i vremenske odsječke, a pritom u što većoj mjeri zadovoljiti ograničenja. Skup ograničenja obično se dijeli na "čvrsta ograničenja" i "meka ograničenja". Čvrsta ograničenja nužno moraju biti zadovoljena u svakom primjerku izvedivog rasporeda. Kršenje čvrstih ograničenja raspored čini neprikladnim za uporabu (neizvedivim). Ispunjavanje mekih ograničenja osigurava veću kvalitetu rasporeda, ali ona ne moraju biti ispunjena i njihovo kršenje ne čini raspored neizvedivim.

3.1. Problem sveučilišnog rasporeda

Problem sveučilišnog rasporeda (engl. University Course Timetabling Problem, skraćeno UCTP) definiran je kao uređena četvorka:

$$UCTP = (M, R, T, C),$$

gdje je:

- M skup nastavnih aktivnosti koje je potrebno rasporediti
- R skup resursa koje je potrebno rasporediti i dodijeliti nastavnim aktivnostima
- T skup termina u kojima je raspoređivanje moguće
- C skup mogućih ograničenja nastavnih aktivnosti

Radi ilustracije, u skupu M mogu se nalaziti predmeti, pojedine laboratorijske vježbe i sl, u skupu R prostorije fakulteta, u skupu T vremenski intervali tijekom dana, a u skupu C specifična pravila koja moraju vrijediti za pojedine događaje u rasporedu sati. Kao podrazrede ovog problema moguće je denirati različite probleme raspore-

đivanja, poput problema rasporeda predavanja, problema rasporeda auditornih vježbi, problema rarasporeda laboratorijskih vjezbi itd.

Budući da UCTP obuhvaća vrlo širok skup problema, ostvarenje konkretnog sustava za rješavanje cijele ove klase je teško zamislivo te su sustavi za automatiziranu izradu rasporeda u praksi redovito usredotočeni na neki od njegovih precizno definiranih potproblema. Detaljan pregled trendova u rješavanju problema raspoređivanja može se naći u [2], [5], [6], [7] i [8]. I u ovom diplomskom radu korišten je navedeni princip. Budući da prethodna definicija problema ne zadovoljava zahtjeve izrade rasporeda na našoj ustanovi, opis problema proširen je dodatnim elementima koji omogućuju primjenu na specifičan tip problema kakav se pojavljuje u praksi na FER-u.

3.2. Problem rasporeda laboratorijskih vježbi

Problem raspoređivanja laboratorijskih vježbi (LETP) definiran je kao šestorka [1]:

$$LETP = (T, L, R, E, S, C)\,,$$

Gdje je T skup *vremenskih kvanata* u kojima je raspored moguć, L skup ograničenih pomagala u ustanovi, *R* skup prostorija, E skup događaja za koje treba izraditi raspored, S skup studenata koje treba rasporediti i C skup ograničenja.

- Skup vremenskih kvanata u kojima je moguće održavati vježbe označen je sa T. Pretpostavka je da se trajanje svake vježbe može izraziti kao višekratnik konstantnog vremenskog intervala, *vremenskog kvanata*, označenog t_q. *Vremenski odsječak* je definiran kao niz jednog ili više uzastopnih vremenskih kvanata u rasporedu. Manja duljina vremenskog kvanta pruža veću zrnatost vremenskih odsječaka, dok se druge strane znatno povećava prostor pretraživanja. Stoga pri odabiru duljine vremenskog kvanta treba birati tako da se dobije bolja zrnatost, a da posljedično prostor rješenja ne postane prevelik. U praktičnoj primjeni odabrana je duljina vremenskog kvanta od 15 minuta, a vremenski odsječci mogu biti smješteni između 8:00 i 20:00.
- Skup ograničenih pomagala raspoloživih za laboratorijske vježbe označen je sa L. Svako od raspoloživih pomagala može se koristiti na raznim laboratorijskim vježbama, a njihov broj je ograničen. Na primjer, na laboratorijskoj vježbi može se koristiti komercijalni programski paket, ali fakultet može posjedovati samo ograničen broj licenci. To utječe na istovremeno izvođenje vježbi za koje su potrebna ista pomagala. Za svako pomagalo $l \in L$, definirana je ko-

ličina $kolicina_l$ kao broj radnih mjesta na kojima se može istovremeno koristiti pomagalo.

– Svakoj prostoriji $r \in R$ pridružen je uređeni par svojstava: $(kapacitet_r, T_r)$, $kapacitet_r \in \mathbb{N}$, $T_r \subseteq T$, definiran na sljedeći način:
  - *Radno mjesto* definirano je kao nedjeljivi resurs prostorije koji ovisi o vrsti prostorije, kao što su sjedala u normalnim prostorijama, računala u računalnim laboratorijima, itd. Za svaku prostoriju $r \in R$, definiran je broj radnih mjesta, $kapacitet_r \in \mathbb{N}$.
  - Prostorije koje se koriste za laboratorijske vježbe mogu biti korištene i za druge svrhe, kao što su ispiti i predavanja. Zbog toga, za svaku prostoriju $r \in R$ definiran je skup vremenskih kvanata, $T_r \subseteq T$, u kojima je prostorija raspoloživa.

– Događaj $e \in E$ definiran je kao jedna laboratorijska vježba za neki predmet. Jedan događaj može biti raspoređen u jednoj ili u više *instanci događaja* u različitim vremenskim odsječcima. Na primjer, jedna instanca događaja može se održavati od 8:00 do 9:00 u ponedjeljak (za jednu grupu studenata) i druga od 10:00 do 11:00 u utorak (za drugu grupu studenata). Svaki događaj ima skup svojstava kako slijedi:
  - Svaki događaj e ima trajanje, $trajanje_e \in \mathbb{N}$, definirano kao višekratnik vremenskog kvanta. Na primjer "Umjetna inteligencija, vježba 1" može imati trajanje $trajanje_{UI1} = 4$ vremenska kvanta, ili 60 minuta.
  - Vremenski raspon događaja, $raspon_e \in \mathbb{N}$, može se definirati da bi se osiguralo da sve instance događaja budu smještene unutar zadanog vremenskog raspona. Definiran je kao maksimalna vremenska razlika između završetka zadnje instance i početka prve instance događaja. Promotrimo kao primjer vježbu koja uključuje kratku provjeru znanja na njenom završetku. Da bi se spriječilo širenje informacija o testu, vremenski raspon vježbe može se ograničiti tako da svi studenti budu prisutni na vježbi u isto vrijeme.
  - Događaji se mogu održavati u različitim vrstama prostorija, od računalnih do specijaliziranih elektroničkih ili električnih laboratorija. Zbog toga je za svaki događaj e definiran skup prostorija, $R_e \subseteq R$, u kojima se događaj može održati.
  - Za svaki događaj e, definiran je skup prikladnih vremenskih kvanata, $T_e \subseteq T$. Na primjer, osoblje nekog predmeta iz organizacijskih raz-

loga može zahtijevati da se vježbe mogu održavati isključivo u srijedu i petak.

- Za održavanje nekih događaja može se koristiti jedno ili više ograničenih pomagala. Za svaki događaj e definiran je skup $L_e \subseteq L$ pomagala koja se koriste.
- Obično je za održavanje vježbe potrebno osoblje koje pomaže studentima obaviti vježbu ili nadzire studente tokom testova. Broj osoba ovisi o događaju i o prostoriji u kojoj se događaj održava (u većim prostorijama često postoji potreba za više osoba). Za događaj e i za prostoriju $r \in R_e$ u kojoj se događaj održava definiran je broj osoblja $osoblje_{e,r}$. Za $r \notin R_e$ broj nije definiran. Osoblje se može promatrati kao ograničen resurs za održavanje događaja. Zbog toga je se za događaj e definira $osoblje_e$ kao ukupan raspoloživ broj osoblja za događaj.
- Za svaki događaj e može se definirati maksimalan broj prostorija, $prostorije_e \in \mathbb{N}$, koji može biti zauzet u isto vrijeme.
- Neki parovi događaja zahtijevaju relaciju uređaja između sebe. Taj zahtjev može se javiti kod predmeta čije vježbe zavise jedna o drugoj. Na primjer događaji "UI vježba 1" i "UI vježba 2" moraju biti raspoređeni unutar istog tjedna, ali mora se osigurati da se drugi događaj održi nakon prvog događaja. Također, studenti trebaju imati barem jedan dan između tih vježbi kako bi se stigli pripremiti. Relacija $\succ_d$ definira se za par događaja, $\succ_d$: $E \times E$. Događaji e_1 i e_2 su u relaciji $e_1 \succ_d e_2$ ako i samo ako je svaka instanca događaja e_2 raspoređena barem d dana nakon zadnje instance događaj e_1.
- Za svaki događaj definiran je broj studenata po radnom mjestu, $sprm_e \in \mathbb{N}$. Na nekim predmetima, na primjer, svaki student obavlja vježbu sam na svom radnom mjestu, dok na drugim vježbama studenti mogu raditi u grupama.

– Skup S je skup studenata koje treba rasporediti. Svaki student $s \in S$ ima sljedeći skup svojstava:

- Budući da studenti imaju razne obaveze na fakultetu(pohađanje predavanja, ispiti), pretpostavlja se da postoje termini u kojima student nije raspoloživ za obavljanje laboratorijskih vježbi. Zbog toga, za svakog studenta s definira se skup vremenskih kvanata $T_s \subseteq T$kada je raspoloživ.

- Ovisno o predmetima koje student odabere, on mora biti prisutan na skupu događaja $E_s \subseteq E$.

– Zahtjevi predmeta predstavljeni su skupom ograničenja C. Skup ograničenja podijeljen je u skup tvrdih ograničenja C_h, koja je nužno zadovoljiti, i skup mekih ograničenja C_s, koja zahtijevaju lakšu ručnu intervenciju ukoliko nisu zadovoljenja. Tvrda ograničenja definirana su na sljedeći način:
  - Jedna prostorija u svakom trenutku može biti zauzeta sa najviše jednim, događajem.
  - Da bi se događaju dodijelila prostorija u nekom vremenskom terminu, ona u tom terminu mora biti raspoloživa.
  - Događaj se može održavati samo u prostorijama $r \subseteq R_e$ koje su prikladne za taj događaj.
  - Prostorija r, kada se u njoj održava događaj e, može primiti najviše $kapacitet_r \cdot sprm_e$ studenata.
  - Događaj e se može održati samo u vremenu $t \subseteq T_e$ koje je prikladno za održavanje događaja.
  - Kad se instanci događaja e dodijeli termin, tada termin zauzima $trajanje_e$ slijednih vremenskih kvanata unutar istog dana.
  - Sve instance događaja e moraju biti smještene unutar $raspon_e$ vremenskih kvanata.
  - Relacija $\succ_d$ mora biti zadovoljena između svih parova događaja za koje je definirana.
  - Pomagalo $l \in L$ može biti korišteno istovremeno na najviše $kolicina_l$ radnih mjesta. Time je ograničen ukupan broj studenata koji mogu istovremeno biti na vježbi koja koristi pomagalo. Isto tako dva različita događaja koja koriste isto pomagalo mogu utjecati jedan na drugog.
  - Kada se događaj održava istovremeno u različitim prostorijama, mora postojati dovoljan broj nastavnog osoblja za sve prostorije.
  - Događaj e može se održavati u najviše $prostorije_e$ prostorija istovremeno.
  - Studenti moraju biti prisutni na svim događajima u koje su uključeni.

– Skup mekih ograničenja C_s zadrži dva elementa:
  - Student može biti prisutan na događaju jedino ako je u vrijeme održavanja događaja slobodan od ostalih obaveza.

- Student može istovremeno biti prisutan na najviše jednom događaju.

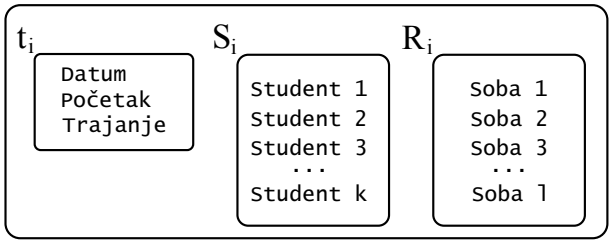
4. Rješavanje problema raspoređivanja genetskim algoritmom

4.1. Reprezentacija rješenja

Reprezentacija rješenja $LETP$ problema proizlazi iz njegove definicije: rješenje (raspored) mora imati raspoređen svaki definirani događaj. Za događaj $e \in E$ kažemo da je raspoređen ako je svakom studentu $s \in S_e$ dodijeljen termin, gdje je $S_e \subseteq S$ skup studenata pridruženih tom događaju. Ovisno o broju studenata, raspoloživosti prostorija i njihovoj veličini svaki događaj može biti raspoređen u jednoj ili više *instanci događaja*. Instanca događaja i definirana je kao uređena trojka:

$$i = (t_i, R_i, S_i)$$

gdje su $t_i \in T_e$ vremenski interval i $R_i \subseteq R_e$ skup prostorija u kojima se vježba održava, a $S_i \subseteq S_e$ skup studenata koji su dodijeljeni toj instanci (slika 4.1).

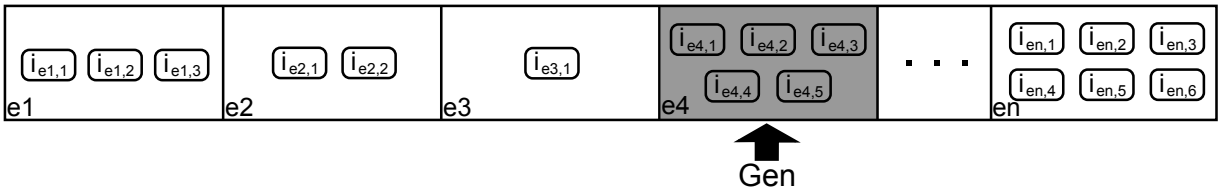


Slika 4.1: Raspoređena instanca događaja

Svakom raspoređenom događaju e dodijeljen je skup od m instanci $I_e = \{i_{e,1}, i_{e,2}, ..i_{e,m}\}$. Broj instanci dodijeljen svakom događaju mora biti dovoljan da se njima pokriju svi studenti, a da se istovremeno zadovolje čvrsta ograničenja.

Pridruživanjem skupa instanci svakom događaju iz rasporeda gradi se jedan prijedlog rješenja (jedinka). Budući da je broj događaja unutar rasporeda nepromjenjiv,

skup instanci dodijeljenih jednom dodađaju predstavlja jedan gen — osnovnu jedinicu nasljeđivanja u evolucijskom procesu (slika 4.2).



Slika 4.2: Genotip jednog mogućeg rješenja

Prostor rješenja čine sva moguća pridruživanja skupova instanci događaja svim događajima, tako da su zadovoljena čvrsta ograničenja. Budući da je prostor rješenja ograničen čvrstim ograničenjima, implementacija operatora genetskih operatora mora osigurati da ih njihov rezultat zadovoljava. Kako bi se ubrzala provjera čvrstih ograničenja svaka jedinka sadrži dodatnu strukturu koja za svaku prostoriju čuva vremenske termine u kojima je raspoloživa. Prije zauzimanja neke od prihvatljivih prostorija za događaj provjerava se njena raspoloživost, a nakon zauzimanja ažurira se njena zauzetost. Sličnu potpornu strukuturu jedinka ima i za dijeljene resurse među događajima.

4.2. Stvaranje početne populacije

Postupak stvaranja početne populacije nasumično stvara zadani broj jedinki tako da svaka stvorena jedinka zadovoljava čvrsta ograničenja. Jedinka se stvara u dva koraka. U prvom koraku dodjeljuju se instance događaja: za svaki događaj rezervira se dovoljan broj instanci (skup dvorana u nekom terminu) za sve studente koji su prisutni na događaju. U drugom koraku se u raspored smještaju studenti: svakom studentu iz događaja dodjeljuje se nasumično jedna od dodijeljenih instanci. Postupak dodjele instanci za događaje prikazan je algoritmom 2.

Algoritam dodjele instanci rješenju ne jamči da će uspjeti. Pri dodjeli instanci za događaj ne mora uvijek biti moguće zauzeti dovoljan broj dvorana za sve studente, a da pri tom ostanu zadovoljena čvrsta ograničenja. Najčešće je uzrok tome izbor instanci za prethodno raspoređene događaje u rasporedu: niz slučajnih odabira termina i dvorana za prethodne događaje je takav da kod dodjele dvorana i termina za događaj nakon tog ne postoji izbor. Problem se može riješiti tako da se kod dodjele prethodnih događaja odvije drugačiji niz odabira termina i dvorana. Npr. nekom događaju može se dodijeliti jedna od više prihvatljivih prostorija, a da se nakon tog za drugi događaj,

Algoritam 2 Dodjela instanci događaja

for svaki događaj $e \in E$ **do**
  D_e = skup odgovarajučih dana za događaj e;
  N_e = broj studenata koji pohađaju događaj e;
  while (D_e nije prazan) AND ($N_e > 0$) **do**
    d = odaberi i ukloni slučajni dan iz D_e;
    $T_{e,d}$ = skup valjanih vremenskih termina za događaj e u danu d;
    while ($T_{e,d}$ nije prazan) AND ($N_e > 0$) **do**
      odaberi slučajni vremenski termin t i makni ga iz $T_{e,d}$;
      stvori instancu događaja $i = (t, R_i, S_i)$ sa $R_i = \emptyset$ i $S_i = \emptyset$;
      for svaku prikladnu prostoriju $r \in R_e$ **do**
        if (r raspoloživa u terminu t) AND (r zadovoljava čvrsta ograničenja) **then**
          zauzmi prostoriju r i dodaj ju u R_i;
          $N_e = N_e - kapacitet_r \cdot sprm_e$;
        end if
      end for
      pridruži instancu događaja i događaju e;
    end while
  end while
  if $N_e > 0$ **then**
    return dodjela neuspješna;
  end if
end for

kojem je upravo ta prostorija jedina prihvatljiva, ona ne može rezervirati ni u jednom terminu. Odabirom neke druge dvorane za prvi događaj oslobađa se tražena dvorana za drugi događaj, te se može uspješno rezervirati. Zbog toga se algoritam dodjele instanci ponavlja ispočetka sve dok slučajan niz odabira ne postane takav da se svi događaju uspješno rasporede. Neuspjeh dodjele instanci može ukazivati i na loše definirane zahtjeve: čvrsta ograničenja mogu biti postavljena tako da ne postoji rješenje koje će ih zadovoljiti.

Nakon dodjele instanci raspoređuju se studenti na način kako je u opisano u algoritmu 3.

Algoritam 3 Raspoređivanje studenata po dodijeljenim instancama

for svaki događaj $e \in E$ **do**
  S_e = skup studenata koji pohađaju događaj e;
  for svaku instancu događaja $i \in I(e)$ **do**
    N_i = kapacitet instance i;
    S_i = odaberi i ukloni slučajnih N_i studenata iz S_e;
    $i.S_i = S_i$;
  end for
end for

Za razliku od prvog koraka, raspoređivanje studenata po instancama je uvijek uspješno jer prvim korakom(ako uspješno završi) osigurava se dovoljan broj mjesta za sve studente. Nakon raspoređivanja studenata najčešće postoje preklapanja između njihovih obaveza, no budući da preklapanje obaveza spada u kategoriju mekih ograničenja, o tome se u ovoj fazi ne vodi računa.

4.3. Operator križanja

Operator križanja u osnovi djeluje po uzoru na jednostavan operator križanja opisan u odjeljku 2.5.. Križanjem dviju jedinki nastaje potomak tako da od svakog roditelja dobije dio gena. Kako je gen ovdje predstavljen skupom instanci dodijeljenim jednom događaju (slika 4.2), operator križanja djeluje tako da potomak nasljeđuje raspoređen cijeli događaj od prvog ili drugog roditelja. Križanje je uniformno, budući da se za svaki događaj nasumično bira od kojeg će roditelja biti naslijeđen.

Bez obzira na to što svaki od roditelja zadovoljava čvrsta ograničenja, potomak dobiven kombinacijom njihovih gena ih ne mora nužno zadovoljavati. Zbog toga je

operator križanja modificiran kako bi se osiguralo svojstvo zatvorenosti s obzirom na čvrsta ograničenja. Za modificirani operator križanja definira se sljedeće:

- Definira se $I(sol, e)$ kao skup instanci dodijeljen događaju $e \in E$ u rješenju $sol \in \mathcal{SS}$, gdje je $\mathcal{SS}$ skup svih mogućih rješenja.
- Relacija konflikta $\otimes$ definirana je nad kartezijevim produktom $\mathcal{I} \times \mathcal{SS}$, gdje je $\mathcal{I}$ skup svih mogućih instanci događaja. Skup instanci I i rješenje sol su u relaciji $I \otimes sol$ akko dodavanje instance I u rješenje sol uzrokuje kršenje barem jednog od čvrstih ograničenja.

Postupak križanja opisan je algoritmom 4. Algoritam gradi novu jedinku tako da izabere nasumično gene koje će potomak naslijediti od prvog roditelja. Preostali geni koji nisu u relaciju konflikta sa potomkom uzimaju se od drugog roditelja. Geni drugog roditelja koji jesu u relaciji konflikta sa potomkom stvaraju se iznova postupkom koji se koristi za stvaranje jedinke u početnoj populaciji (odjeljak 4.2.).

Algoritam križanja može završiti neuspješno prilikom ponovne dodjele konfliktnih gena iz istih razloga kao i kreiranje jedinke u početnoj populaciji. Neuspješno križanje se ponavlja nad istim roditeljima sa maksimalnim brojem pokušaja. Ako nakon zadanog maksimalnog broja pokušaja križanje ne uspije, genetski algoritam se nastavlja kao da križanja nije ni bilo.

Algoritam 4 Križanje($Jedinka\ a$, $Jedinka\ b$)

$Jedinka\ dijete$ = stvori neraspoređenu jedinku;
E_1 = odaberi slučajni podskup iz E;
for svaki događaj e iz E_1 **do**
  pridruži $I(a, e)$ jedinci $dijete$ tako da vrijedi $I(dijete, e) = I(a, e)$;
end for
$E_2 = E - E_1$;
$E_n = \emptyset$;
for svaki događaj e iz E_2 **do**
  if nije $I(b, e) \otimes dijete$ **then**
    pridruži $I(b, e)$ jedinci $dijete$ tako da vrijedi $I(dijete, e) = I(b, e)$;
  else
    dodaj e u E_n;
  end if
end for
for svaki događaj e iz E_n **do**
  $I(dijete, e)$ = dodijeli nasumično novi skup instanci;
  if $I(dijete, e)$ neuspješno dodijeljen **then**
    return križanje neuspješno;
  end if
end for
return $dijete$;

4.4. Operator mutacije

Operator mutacije djeluje na temelju vjerojatnosti za mutaciju cijele jedinke. Ako je jedinka odabrana za mutaciju tada se mutira zadani broj njezinih gena. Broj gena koji će mutirati je parametar algoritma koji se može mijenjati tijekom evolucijskog procesa. Geni odabrani za mutaciju uklanjaju se iz jedinke, te se iznova nasumično dodjeljuju. Postupak mutacije opisan je algoritmom 5.

Algoritam 5 Mutacija($Jedinka\ a$, $int\ brojGena$)

E_m = odberi slučajno $brojGena$ događaja iz E;
for svaki događaj e iz E_m **do**
  odbaci $I(a, e)$;
  $I(a, e)$ = dodijeli nasumično novi skup instanci;
  if $I(a, e)$ neuspješno dodijeljen **then**
    return mutacija neuspješna;
  end if
end for

Algoritam mutacije može također završiti neuspješno, pa se ponavlja dok ne uspije ili dok je broj neuspjelih pokušaja ispod definiranog praga. Ako nakon maksimalnog broja pokušaja algoritam ne usije mutirati jedinku, mutacija se ignorira.

4.5. Evaluacija rješenja

Sve jedinke u populaciji moraju zadovoljavati čvrsta ograničenja, što je osigurano specifičnim operatorima mutacije, križanja i algoritmom stvaranje početnih jedinki. Meka ograničenja ne moraju biti zadovoljenja, no o njima ovisi kvaliteta rješenja: rješenje je kvalitetnije što su meka ograničenja u većoj mjeri zadovoljena. Zbog toga se za svaku jedinku definira *kazna*, mjera kojom se opisuje zadovoljenost mekih ograničenja.

Budući da meka ograničenja zabranjuju preklapanje studentskih obaveza (odjeljak 3.2.), kazna se izražava u količini preklapanja studentskih obaveza. Student ima obavezu u nekom vremenskom intervalu ako je u to vrijeme prisutan na na nekom vanjskom događaju (predavanje, ili ispit) ili ako mu je u tom vremenu algoritmom dodijeljen termin neke laboratorijske vježbe. Pretpostavlja se da vanjske obaveze studenta nemaju preklapanja. Kazna za jednog studenta definirana je sljedećom formulom:

$$K(s) = \sum_{t \in T} \frac{n(t)(n(t)-1)}{2} \tag{4.1}$$

gdje je T skup svih vremenskih kvanata, a $n(t)$ broj obaveza studenta u kvantu t. Svaki vremenski kvant koji nema preklapanja (kvant sa najviše jednom obavezom) ne pridonosti kazni, kvant sa dvije obaveze pridonosi kazni sa vrijedošću 1, a kvanti sa više od dvije obaveze pridonose kazni sve većim vrijednostima. Ukupna kazna jedinke dobija se zbrajanjem kazni za sve studente unutar rasporeda:

$$K = \sum_{s \in S} K(s) \tag{4.2}$$

Evaluacija kazne je vremenski zahtjevan postupak, pa se zbog ubrzavanja algoritma svaka jedinka proširila dodatnom potpornom strukurom. Struktura za svakog studenta pamti sve njegove obaveze, sva preklapanja obaveza i iznos kazne. Struktura se popunjava prilikom stvaranja jedinke, a nakon toga se evaluira i čuva kazna za svakog studenta. Prilikom modifikacije jedinke potporna struktura se mijenja samo za studente na koje je modifikacija utjecala: analizira se da li je promjena obaveze studenta uzrokovala nastanak ili nestanak preklapanja, te se ažurira vrijednost kazne.

Budući da je genetski algoritam vođen jedino evaluacijom jedinke, može se zaključiti da se ovakvom implementacijom genetskog algoritma nastoji pronaći raspored u kojem je broj preklapanja minimalan. Može se reći da je jedinka bez preklapanja obaveza globalni optimum jer ima minimalnu moguću kaznu $K = 0$. Za instancu problema nije moguće znati unaprijed postoji li globalni optimum, zato se ni taj uvijet ne postavlja kao jedini uvijet završetka algoritma. U nekim instancama problema može

postojati više globalnih optimuma, no kako je broj preklapanja jedina mjera kvalitete rješenja svi se gledaju kao jednako kvalitetna rješenja. Uvođenjem dodatnih elemenata u evaluaciju jedinke mogla bi se preciznije razlikovati rješenja bez preklapanja, no u praksi se pokazalo da se i takva rješenja teško dobivaju, pa nije bilo potrebe bilo kakvim proširivanjem.

5. Prilagodba genetskog algoritma

Osnovni genetski algoritam koji je opisam u prethodnom poglavlju susreo se sa mnogim poteškoćama u radu, zbog čeg se javila potreba za dodatnim prilagodbama. Trajanje izvođenja znatno je smanjeno uvođenjem potpornih struktura za evidenciju zauzetosti studenata, dvorana i resursa. Osim tih učinjene su dodatne prilagodbe s ciljem poboljšanja učinkovitosti algoritma opisane u ovom poglavlju.

5.1. Detekcija stagnacije

Detekcijom stagnacije pokušava se riješiti problem zapinjanja populacije u nekom lokalnom minimumu. Tipičan razvoj događaja nakon pokretanja algoritma je sljedeći:

1. Početna populacija stvorena je sa relativno visokom kaznom (5000-10000).
2. U početku kazna pada velikom brzinom.
3. Kazna se nakon nekog vremena prestane smanjivati, ili se smanjuje za mali iznos u puno vremena.

U trećoj fazi populacija je ušla u stagnaciju i obično je kazna u toj fazi nezadovoljavajuća. Kako bi se spriječilo zapinjanje u lokalnom optimumu, algoritam je priladođen da nakon ulaska populacije u stagnaciju pojača djelovanje operatora mutacije. Definiran je parametar *pragStagnacije* koji služi kao detekciju stagnacije. Za populaciju se kaže da je ušla u stagnaciju ako nakon *pragStagnacije* iteracija nije stvorena jedinka koja je bolja od svih dosadašnjih jedinki. Ako je populacija u stagnaciji ona izlazi iz tog stanja u trenutku kad se stvori jedinka koja je bolja od svih dosadašnjih jedinki.

Sustav za detekciju i razrješavanje stagnacije prati stanje populacije, te pojavom stagnacije pojačava mutaciju. Kako je upisano u odjeljku 4.4., broj gena na koje će mutacija utjecati definiran je parametrom. Tim parametrom upravlja upravo ovaj sustav, i u tom kontekstu naziva se *razinaMutacije*. U početku se taj parametar postavlja

na najmanju moguću vrijednost: *razinaMutacije = 1*, a pojavom stagnacije se inkrementira. Kad sustav detektira pojavu stagnacije u populaciji odvija se sljedeći niz akcija:

$$razinaMutacije = min(maxRazinaMutacije, razinaMutacije + 1)$$
$$pragStagnacije = pragStagnacije \cdot 1.5$$

Kad populacija uđe u stanje stagnacije tada se prag detekcije sljedeće stagnacije množi konstantnim faktorom *1.5*. To znači da će za sljedeću reakciju sustava, u slučaju da populacija ostane u stagnaciji, trebati više iteracija. Važno je primijetiti da postoji gornja granica za parametar *razinaMutacije*: broj gena na koji mutacija može djelovati ograničen je brojem događaja koji se raspoređuju. Parametar *maxRazinaMutacije* definiran je brojem događaja u rasporedu i razina mutacije ne može biti veća od njega. U trenutku kad populacija izađe iz stanja stagnacije odvija se sljedeći niz akcija:

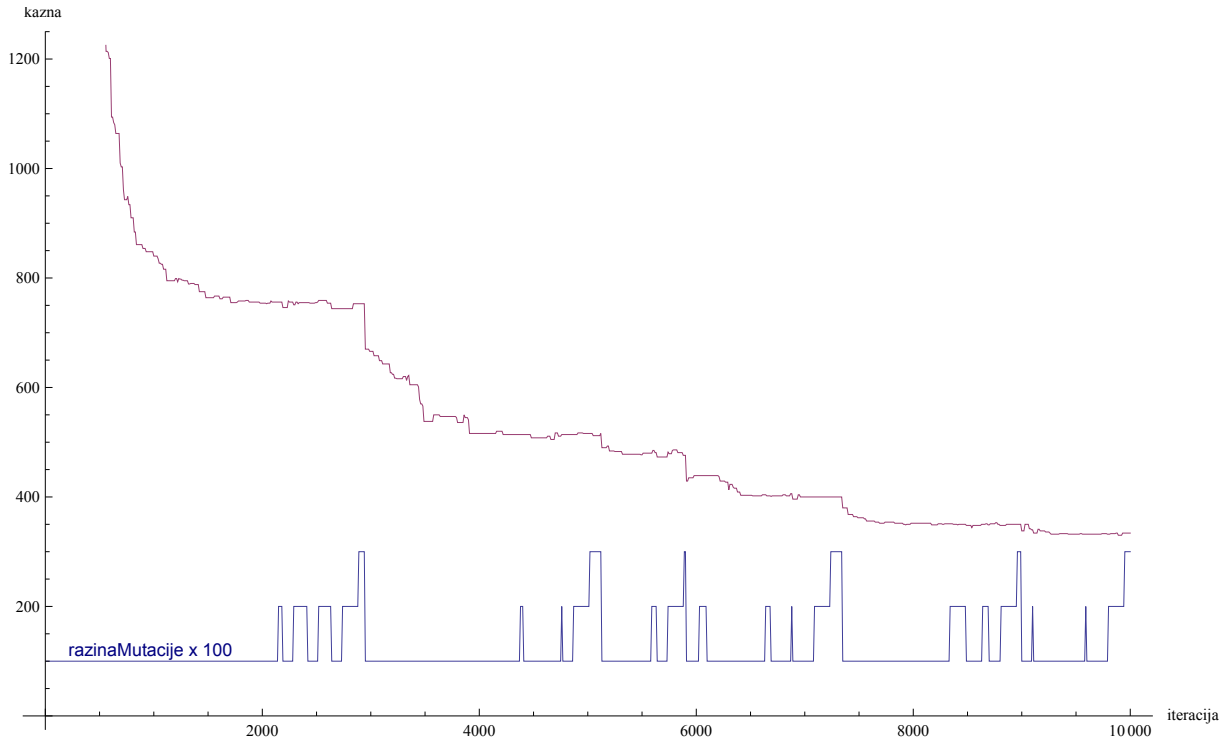
$$razinaMutacije = 1$$
$$pragStagnacije = initPragStagnacije$$

Nakon izlaska iz stagnacije mutacija se smanjuje na najmanju razinu, te se prag stagnacije postavlja na podrazumijevanu vrijednost *initPragStagnacije*, koja je u primjeni iznosila *initPragStagnacije = 100*.

U praksi se pokazalo da je djelovanje sustava za razrješavanje stagnacije uspješno u prvim pojavama stagnacije. Nakon više uzastopnih povećavanja razine mutacije populacija više ne pokazuje napredak, ali kvaliteta rješenja je bolja u odnosu na prvu pojavu. Primjer kazne u populaciji uz promjenu mutacije ilustrirano je grafom na slici 5.1.

5.2. Lokalna pretraga

Operatori mutacije i križanja djeluju prvenstveno na raspored prostorija po terminima, a genetski algoritam nastoji pronaći optimalan raspored istih, tj. takav raspored prostorija i termina za svaki događaj da se studenti mogu razmjestiti po njima sa što manjim brojem konflikata. Problem razmiještanja studenata u okviru postojćeg rasporeda termina i dvorana je također problem kombinatorne optimizacije koji bi se mogao rješavati zasebnim genetskim algoritmom. Djelovanje dva ugniježđena genetska algoritma (u svakoj iteraciji vanjskog algoritma izvršava se još jedan) imalo bi iznimno veliku vremensku složenost, pa se zbog toga optimalan razmještaj studenata po terminima nastoji postići jednostavnim heurističkim postupkom koji u se u okviru ovog rada naziva *Lokalna pretraga*.



Slika 5.1: Kazna u populaciji i razina mutacije u ovisnosti o broju iteracija

Lokalna pretraga djeluje kao burza grupa: svakom studentu sa preklapanjem obaveza nastoji se za jednu od preklapajućih obaveza pronaći drugi termin održavanja, ako za tu obavezu postoji više različitih termina. Nakon odabira drugog termina student se briše iz trenutnog i dodaje u novi, uz ažuriranje strukture sa obavezama studenta (odjeljak 4.5.). Ako se dodavanjem studenta u novi termin prekoračio njegov kapacitet, bira se student iz drugog termina koji će se preseliti u termin prvog studenta. Postupak lokalne pretrage opisan je algoritmom 6.

Postoji više implementacija lokalnih pretraga s obzirom na način odabira novog termina studentu koji ima preklapanje, i način odabira studenta koji će doći na njegovo mjesto ako je potrebno (u algoritmu 6 označeno zvjezdicama). Vrste lokalnih pretraga i njihova svojstva prikazane su su tablici 5.1.

Tablica 5.1: Vrste lokalnih pretraga

Vrsta	Odabir novog termina	Odabir drugog studenta
Tip 0	Slučajno	Slučajno
Tip 1	Student nema preklapanje	Slučajno
Tip 2	Student nema preklapanje	Nema preklapanje u prvom terminu

Algoritam 6 Lokalna pretraga

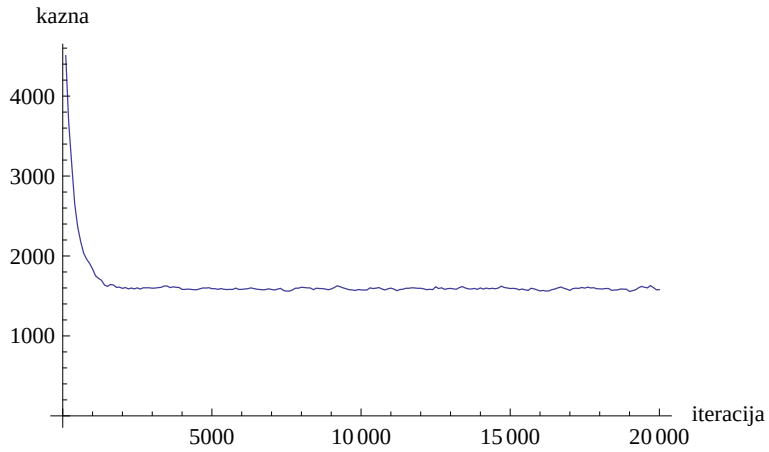
s = odaberi slučajnog studenta sa konfliktom;
e = izaberi slučajno jednu od konfliktnih obaveza studenta s;
i = instanca iz $I(e)$ u kojoj je smješten student s;
$I_{e,2} = I(e) - i$;
if $I_{e,2} = \emptyset$ **then**
  return ;
end if
i_2 = odaberi odgovarajuću instancu iz $I_{e,2}$ *;
ukloni studenta s iz i;
dodaj studenta s u i_2;
if prekoračen kapacitet u i_2 **then**
  s_2 = odaberi odgovarajučeg studenta iz i_2 *;
  ukloni studenta s_2 iz i_2;
  dodaj studenta s_2 u i;
end if

Postoje dvije strategije primjene lokalne pretrage. U prvoj strategiji lokalna pretraga djeluje na jedinku u zadanom broju iteracija, dok u svakoj iteraciji pokušava riješiti preklapanje jednog studenta. U drugoj strategiji lokalna pretraga djeluje na jednu jedinku iterativno sve dok se ne dogodi stagnacija u opadanju kazne. U praksi je češće korištena druga strategija koja je primjenom na jedinke iz početne populacije smanjila kaznu za red veličine. Slika 5.2 prikazuje djelovanje lokalne pretrage na jedinku iz početne populacije.

Lokalna pretraga koristi se na svakoj jedinci koja je stvorena u početnoj populaciji i na svakom rezultatu operatora mutacije i križanja. Osim toga u svakoj iteraciji postoji vjerojatnost primjene lokalne pretrage za svaku jedinku iz populacije.

5.3. Nadzirano stvaranje početne populacije

Stvaranje početne populacije oslanja se na pretpostavku da će se nasumučnim odabirom uspješno rezervirati prostorije i termini za sve studente i da takav raspored zadovoljava čvrsta ograničenja. U odjeljku 4.2. dani su razlozi za neuspješno stvaranje jedinke. Mjera za uspješnost stvaranja početne populacije izražava se odnosom veličine populacije i ukupnim brojem pokušaja stvaranja jedinki. U praktičnoj primjeni uspješnost stvaranja početne populacije kretala se između 0% i 90%. Kod nekih pri-



Slika 5.2: Kazna jedinke na koju djeluje samo lokalna pretraga

mjeraka problema uspješnost je bila izrazito mala, pa je zbog toga postupak stvaranja početne populacije trošio puno vremena (par sati).

Analizom ulaznih podataka utvrđen je mogući uzrok niskoj uspješnosti stvaranja početne populacije. Postoje događaji sa velikom količinom studenata, uskim vremenskim ograničenjima i malim izborom prostorija. Takvi događaji imaju nizak stupanj slobode u odabiru prostorija i termina, dok ostali događaji sa manjim brojem studenata i većim izborom termina i prostorija imaju puno veći stupanj slobode. Budući da je redoslijed događaja kojima će se rezervirati prostorije i termini nasumičan, u velikom broju slučaja događaji sa visokim stupnjem slobode došli su na red prije događaja sa niskim stupnjem. U trenutku dodjele prostorija i termina događaju sa niskim stupnjem slobode više nije bilo dovoljno raspoloživih dvorana u uskim vremenskim oraničenjima da se smjeste svi studenti. Rješenje tog problema je da događaji sa niskim stupnjem slobode dobiju veći prioritet, tj. da se prvo dodijele termini i prostorije takvim događajima.

Postupak stvaranja početne populacije modificiran je na sljedeći način: svakom događaju dodijeljena je težina o kojoj ovisi prioritet dodjele prostorija i termina. Svaki događaj ima vjerojatnost da će biti izabran za dodjelu koja je jednaka omjeru njegove težine i težine svih ostalih neraspoređenih događaja. Ako se u postupku dodjele za neki događaj ne uspije rezervirati dovoljno termina, tada se tom događaju težina poveća za konstantan iznos i postupak stvaranja jedinke kreće ispočetka. Na taj način "problematični" događaji nakon nekog vremena dobiju na težini i postoji velika vjerojatnost da će kod dodjele baš oni doći prvi na red.

Prilagodbom procesa stvaranja početne populacije povećala se uspješnost njezinog stvaranja. U tablici 5.2 dana je usporedba uspješnosti stvaranja početne populacije bez

nadzora i sa nadzorom na stvarnim primjerima (*CX*).

Tablica 5.2: Usporedba nadziranog i nenadrizanog stvaranja početne populacije

	C4	C6	C7	C8	C10	C11	C12
Bez nadzora	7.55%	6.80%	8.04%	1.28%	19.27%	7.56%	0.09%
Sa nadzorom	12.62%	11.21%	26.80%	6.15%	24.22%	19.52%	1.17%

6. Ispitivanje učinkovitosti algoritma

Implementirani sustav ima brojne promjenjive parametre čije vrijednosti utječu na učinkovitost sustava. Važniji parametri navedeni su u nastavku:

- Veličina populacije
- Veličina turnira
- Vjerojatnost mutacije jedinke
- Razina mutacije
- Vrsta lokalne pretrage
- Broj ponavljanja lokalne pretrage

Tipične vrijednosti parametara koje su korištene u stvarnim primjerima navedeni su u tablici 6.1

Tablica 6.1: Vrijednosti parametara algoritma na stvarnim primjerima

Parametar	Vrijednost
Veličina populacije	30
Veličina turnira	3
Vjerojatnost mutacije jedinke	0.3
Razina mutacije	Prilagodljiva (1-10)
Vrsta lokalne pretrage	Tip 1
Broj ponavljanja lokalne pretrage	Uvjetovano stagnacijom

Razina mutacije mijenja se tijekom evolucije pod utjecajem detekcije stagnacije kako je opisano u odjeljku 5.1.. Broj ponavljanja lokalne pretrage nije unaprijed zadan, već se ona ponavlja sve dok smanjivanje kazne jedinke ne dođe u stanje stagnacije.

Za potrebe ovog rada izvršeno je ispitivanje sustava ovisno o tri parametra:

- Vjerojatnost mutacije
- Vrsta lokalne pretrage

– Udio korištenja pojedine vrste lokalne pretrage u kombinaciji više njih

Svi ostali parametri imaju vrijednost kako je navedeno u tablici 6.1. Ispitivanje se odnosi na ovisnost trajanja izvođenja algoritma[1] o vrijednosti parametra, te krajnu kaznu u populaciji. Ispitivanje je izvršeno na sedam stvarnih primjera problema[2]. Za svaku vrijednost parametra algoritam se pokreće deset puta, gdje svako pokretanje traje 100000 iteracija. Prikaz trajanja izvođenja u sljedećim odjeljcima odnosi se na srednju vrijednost u deset pokretanja izraženu u sekundama, dok se kazna prikazuje kao minimalna, maksimalna i srednja u istom broju pokretanja.

6.1. Ispitivanje učinkovitosti algoritma ovisno o vjerojatnosti mutacije

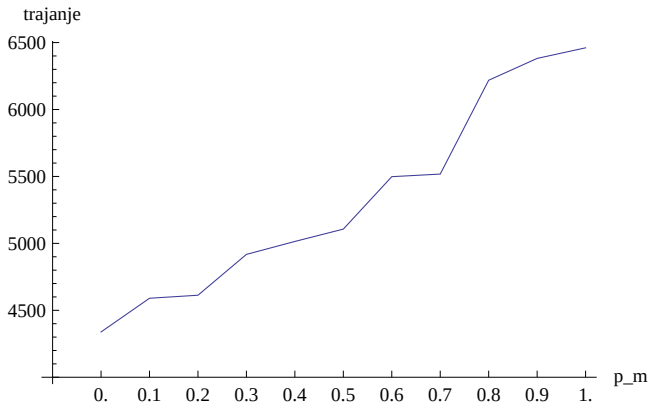
Vjerojatnost mutacije kretala u intervalu $[0.0, 1.0]$ sa korakom 0.1. Trajanja izvođenja 100000 iteracija za različite vrijednosti parametara i različite primjere problema prikazano je tablicom 6.2, a grafički prikaz na primjeru problema *C8* nalazi se na slici 6.1.

Tablica 6.2: Ovisnost trajanja izvođenja o vjerojatnosti mutacije

	C4	C6	C7	C8	C10	C11	C12
0.0	4002	4169	4666	4338	4241	3710	5322
0.1	4044	4576	5108	4590	4441	3866	6553
0.2	4120	4705	5133	4612	4541	3876	6869
0.3	4198	5011	5801	4917	4514	3968	7122
0.4	4221	5278	5726	5014	4589	4172	7857
0.5	4296	5844	6010	5106	4612	4180	7977
0.6	4339	6028	6184	5498	4728	4064	8396
0.7	4481	6374	6466	5517	4914	4156	8848
0.8	4526	6640	6531	6218	5024	4174	9229
0.9	4557	6728	6702	6381	5102	4148	9451
1.0	4580	6879	6808	6460	5000	4161	9629

[1]Sva ispitivanja izvodila su se na identičnima računalima: Intel Pentium 4 2.6GHz sa Hyper-Threading tehnologijom, 496MB RAM, Windows XP.

[2]Primjeri problema su zahtjevi za izradu rasporeda za sedam različitih ciklusa iz akademske godine 2008/2009. U tablicama su označene kao *C4*, *C6*, *C7*, *C8*, *C10*, *C11*, *C12*.



Slika 6.1: Ovisnost trajanja izvođenja o vjerojatnosti mutacije za primjer problema *C8*

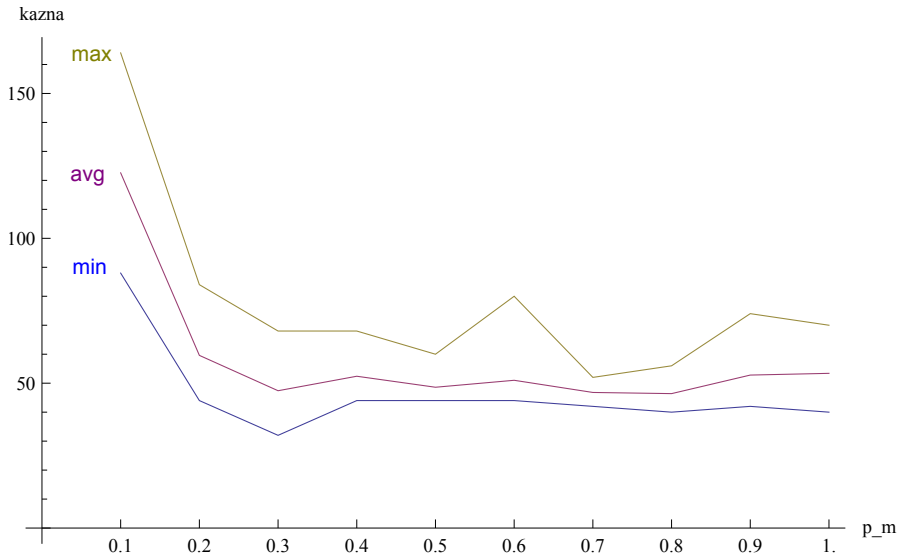
U svim primjerima problema može se primijetiti da vrijeme izvođenja linearno raste s povećavanjem vjerojatnost mutacije. Takvo ponašanje je i očekivano, budući da izvršavanje algoritma mutacije iziskuje dodatno vrijeme.

U tablici 6.3 prikazana je ovisnost minimalne, srednje i maksimalne kazne o vjerojatnosti mutacije. Za svako pokretanje uzima se minimalna kazna iz cijele populacije, te se najmanja od tih vrijednosti izražava kao minimalna, najveća od tih vrijednosti kao maksimalna, i prosjek svih vrijedosti kao srednja vrijednost u tablici. Grafički prikazi za tri primjera problema (*C4*, *C6* i *C8*) nalaze se na slikama 6.2, 6.3 i 6.4.

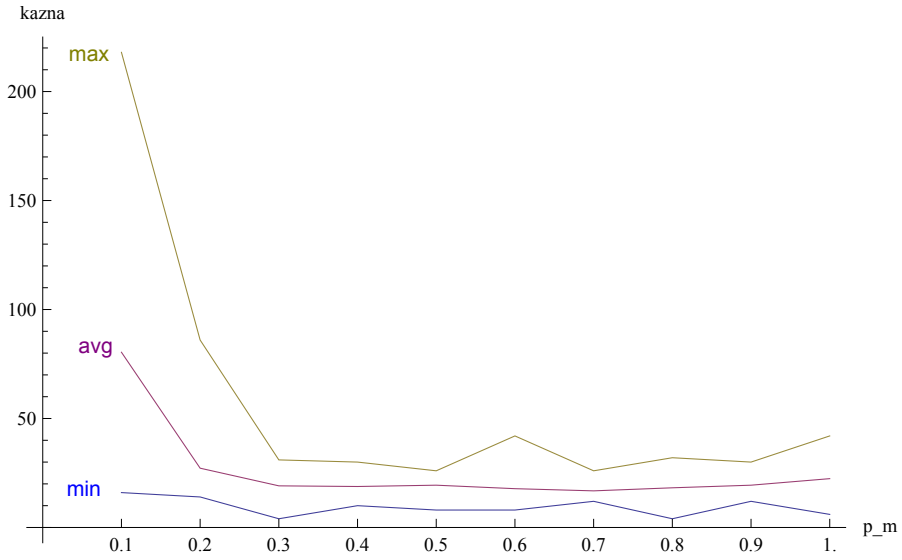
Tablica 6.3: Ovisnost kazne o vjerojatnosti mutacije

		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
C4	min	729,0	88,0	44,0	32,0	44,0	44,0	44,0	42,0	40,0	42,0	40,0
	avg	879,0	122,6	59,6	47,4	52,4	48,6	51,0	46,8	46,4	52,8	53,4
	max	1070,0	164,0	84,0	68,0	68,0	60,0	80,0	52,0	56,0	74,0	70,0
C6	min	315,0	16,0	14,0	4,0	10,0	8,0	8,0	12,0	4,0	12,0	6,0
	avg	504,6	80,4	27,2	19,1	18,8	19,4	17,8	16,8	18,2	19,4	22,4
	max	781,0	218,0	86,0	31,0	30,0	26,0	42,0	26,0	32,0	30,0	42,0
C7	min	389,0	14,0	4,0	0,0	2,0	0,0	0,0	0,0	0,0	0,0	0,0
	avg	545,0	24,0	12,8	6,2	8,0	8,0	7,0	4,8	9,0	5,0	7,0
	max	716,0	42,0	28,0	16,0	20,0	26,0	18,0	14,0	24,0	18,0	16,0
C8	min	647,0	46,0	36,0	36,0	46,0	30,0	37,0	36,0	40,0	46,0	44,0
	avg	1079,8	87,2	62,6	52,6	61,2	53,9	59,7	48,0	53,4	61,3	62,3
	max	1605,0	148,0	104,0	68,0	78,0	93,0	84,0	68,0	7,0	84,0	96,0
C10	min	344,0	16,0	6,0	4,0	4,0	4,0	4,0	4,0	4,0	4,0	4,0
	avg	526,4	35,2	12,8	5,2	5,4	6,4	6,2	5,4	5,6	5,2	8,2
	max	738,0	56,0	22,0	8,0	8,0	8,0	8,0	8,0	8,0	8,0	18,0
C11	min	464,0	54,0	16,0	16,0	12,0	12,0	16,0	12,0	16,0	20,0	18,0
	avg	658,6	106,2	35,2	18,6	19,0	17,8	25,0	19,0	21,6	26,2	23,6
	max	836,0	218,0	76,0	22,0	30,0	24,0	32,0	32,0	28,0	48,0	30,0
C12	min	415,0	74,0	32,0	28,0	36,0	12,0	22,0	26,0	32,0	20,0	24,0
	avg	651,7	103,8	65,0	43,6	51,0	47,2	48,4	48,3	52,2	39,8	51,0
	max	868,0	154,0	98,0	74,0	66,0	76,0	72,0	73,0	82,0	58,0	74,0

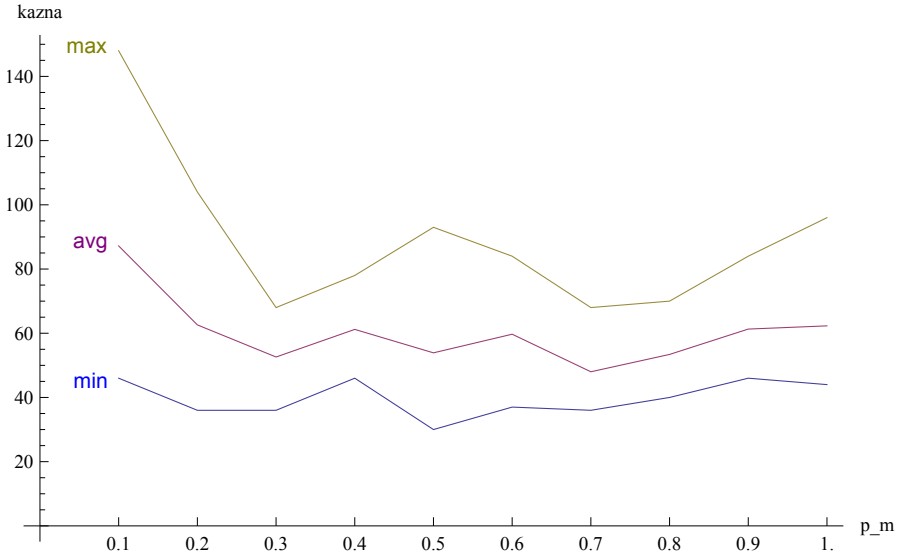
Iz tablice se može primijetiti da je kvaliteta rješenja znatno lošija ako je vjerojatnost mutacije jednaka nuli u odnosu na ostale vrijednosti. Zbog toga su rezultati za nultu vjerojatnost isključeni iz grafičkog prikaza kako bi se ostale vrijednosti bolje razlikovale. Od ostalih vjerojatnosti ističe se vrijednost 0.1 po lošijoj kvaliteti, a iza nje vrijednost 0.2. Za ostale vrijednosti kvaliteta rješenja je podjednako dobra. Budući



Slika 6.2: Ovisnost kazne o vjerojatnosti mutacije za primjer *C4*



Slika 6.3: Ovisnost kazne o vjerojatnosti mutacije za primjer *C6*



Slika 6.4: Ovisnost kazne o vjerojatnosti mutacije za primjer *C8*

da trajanje izvođenja raste sa porastom vjerojatnosti mutacije, nema ju smisla postavljati na visoke vrijednosti. Iz dobivenih rezultata može se zaključiti da je optimalna vrijednost vjerojatnost jednaka 0.3: kvaliteta rješenja je zadovoljavajuća, a vrijeme izvođenja je kraće.

6.2. Ispitivanje učinkovitosti algoritma ovisno o vrsti lokalne pretrage

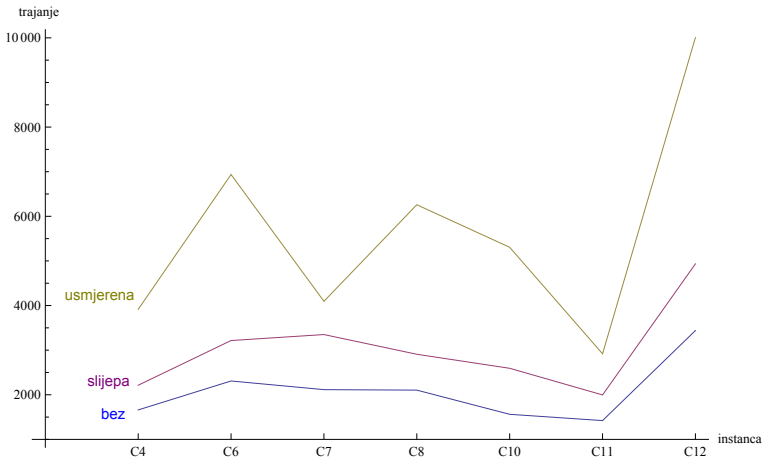
Izvršeno je ispitivanje učinkovitosti u ovisnosti o vrsti lokalne pretrage. U odjeljku 5.2. opisane su različite verzije operatora lokalne pretrage. Testiranje uključuje tri različite postavke sustava:

1. Lokalna pretraga se ne koristi.
2. Koristi se slijepa lokalna pretraga (Tip 0): student sa konfliktom pokušava se preseliti u bilo koji termin bez obzira na mogući konflikt u tom terminu.
3. Koristi se usmjerena lokalna pretraga (Tip 2): za sudenta sa konfliktom se pokušavaju pronaći svi studenti iz drugih termina čiji termin odgovara konfliktnom studentu i da njima odgovara termin konfliktnog studenta.

U tablici 6.4 prikazana je ovisnost trajanja izvođenja u ovisnosti o vrsti lokalne pretrage za svaki od sedam primjera problema.

Tablica 6.4: Ovisnost trajanja izvođenja o vrsti lokalne pretrage

	C4	C6	C7	C8	C10	C11	C12
Bez	1660	2309	2115	2104	1562	1421	3439
Slijepa	2213	3215	3349	2907	2593	1996	4934
Usmjerena	3914	6936	4095	6258	5305	2918	10006



Slika 6.5: Ovisnost trajanja izvođenja o vrsti lokalne pretrage

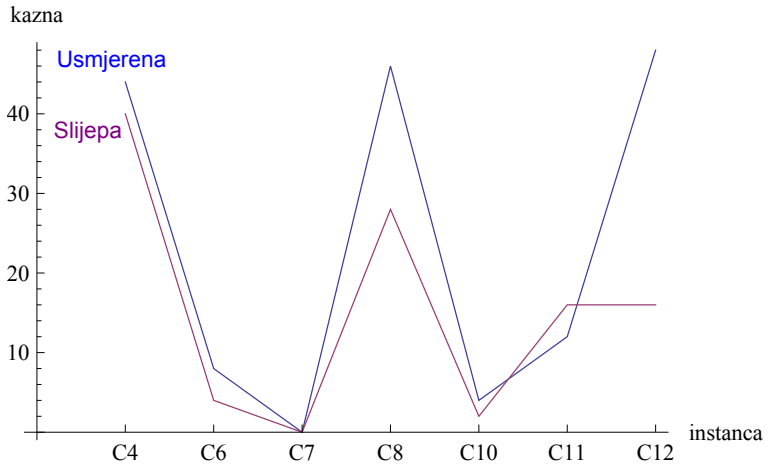
Vidljivo je da je za usmjerenu lokalnu pretragu potrebno najviše vremena, dok je za rad sustava bez lokalne pretrage potrebno najmanje vremena. Takvo ponašanje je očekivano zbog toga što usmjerena lokalna pretraga troši vrijeme na traženje odgovarajućih zamjena, dok slijepa pretraga radi zamjene bez ikakve provjere.

Ovisnost minimalne, srednje i maksimalne kazne o vrsti lokalne pretrage prikazana je na tablici 6.5, dok je grafički prikaz minimalne kazne vidljiv na slici 6.6.

Tablica 6.5: Ovisnost kazne o vrsti lokalne pretrage

		Bez	Slijepa	Usmjerena
C4	min	440,0	44,0	40,0
	avg	495,5	49,4	49,2
	max	550,0	62,0	82,0
C6	min	1414,0	8,0	4,0
	avg	1515,2	21,0	10,4
	max	1659,0	38,0	22,0
C7	min	1290,0	0,0	0,0
	avg	1524,0	15,2	8,0
	max	1640,0	32,0	22,0
C8	min	874,0	46,0	28,0
	avg	988,8	80,0	66,2
	max	1077,0	126,0	138,0
C10	min	964,0	4,00	2,0
	avg	1124,7	5,00	17,4
	max	1252,0	8,00	80,0
C11	min	312,0	12,0	16,0
	avg	372,4	19,6	19,4
	max	418,0	36,0	24,0
C12	min	2005,0	48,0	16,0
	avg	2153,2	59,4	40,8
	max	2237,0	74,0	148,0

Slika ovisnosti minimalne kazne o vrsti lokalne pretrage isključuje rezultate dobivene bez korištenja lokalne pretrage kako bi se bolje uočila razlika između slijepe i usmjerene pretrage. Treba samo napomenuti da su rješenja dobivena bez lokalne pretrage do deset puta lošija (deset puta veća kazna) od rješenja dobivenih korištenjem



Slika 6.6: Ovisnost minimalne kazne o vrsti lokalne pretrage

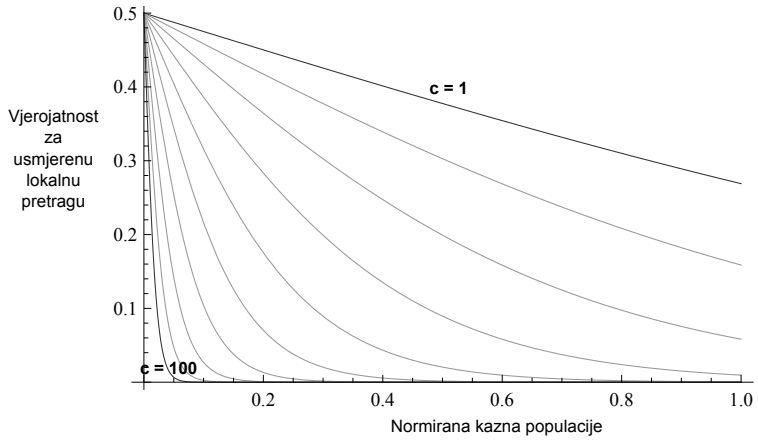
lokalne pretrage. Vidljivo je da je usmjerena pretraga u manjoj mjeri bolja od slijepe, no budući da troši puno više vremena, upitna je njezina isplativost.

6.3. Ispitivanje učinkovitosti algoritma ovisno o udjelu usmjerene lokalne pretrage u kombinaciji sa slijepom

U ovom ispitivanju pretpostavljeno se upotrebljava slijepa lokalna pretraga. U svakom trenutku definirana je vjerojatnost korištenja usmjerene lokalne pretraga umjesto slijepe. Ideja je da se u početku u većoj mjeri koristi slijepa lokalna pretraga koja je po djelovanju slabija od usmjerene. Povećavanjem kvalitete populacije usmjerena lokalna pretraga postaje izraženija. Usmjerena lokalna pretraga bi trebala detaljnije pretražiti okolni prostor jedinke u potrazi za boljim rješenjem. Vjerojatnost primjene usmjerene lokalne pretrage opisana je sljedećim izrazom:

$$p(K) = \frac{1}{1 + e^{cK}} \tag{6.1}$$

gdje je K trenutna srednja vrijednost kazne populacije normirana na prosječnu kaznu populacije u trenutku njezinog stvaranja[3], a c parametar koji definira strminu funkcije vjerojatnosti. Kod ispitivanja parametar c se mijenja od vrijednosti 100 do vrijednosti 1 po logaritamskoj skali. Funkcija vjerojatnosti za različite vrijednosti parametra c prikazana je na slici 6.7. Uočljivo je da je kod visoke vrijednosti parametra $c = 100$ vjerojatnost zanemariva sve do trenutka kad kazna postane vrlo mala, dok je za vrijednost $c = 1$ vjerojatnost relativno visoka za sve vrijednosti kazne.



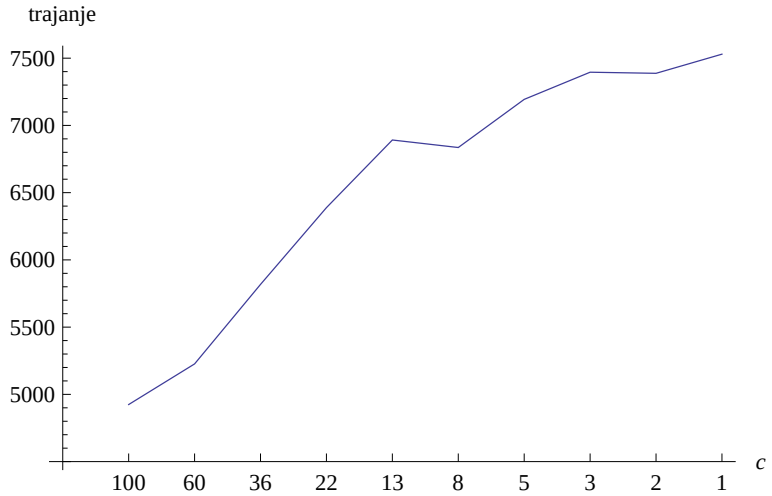
Slika 6.7: Funkcija vjerojatnosti usmjerene lokalne pretrage

[3]Normirana vrijednost 1 znači da je trenutna prosječna kazna populacije jednaka početnoj prosječnoj kazni.

Ovisnost trajanja izvođenja o vrijednosti parametra c dana je u tablici 6.6, dok je grafički prikaz za primjer problema *C12* dan na slici 6.8.

Tablica 6.6: Ovisnost trajanja izvođenja o vrijednosti parametra c

	C4	C6	C7	C8	C10	C11	C12
100	2260	3433	3378	2992	2806	2048	4924
60	2295	3568	3572	3083	2831	2073	5226
36	2444	4084	3941	3273	3175	2168	5817
22	2596	4321	4738	3620	3451	2246	6388
13	2754	4716	4922	3918	3677	2313	6891
8	2869	4864	5078	4151	3778	2368	6835
5	2944	5032	5202	4313	3898	2428	7193
3	3018	5050	5383	4362	3855	2435	7395
2	3022	5168	5487	4469	3926	2459	7387
1	3004	5112	5557	4520	3942	2466	7529



Slika 6.8: Ovisnosti trajanja izvođenja o vrijednosti parametra c za primjer *C12*

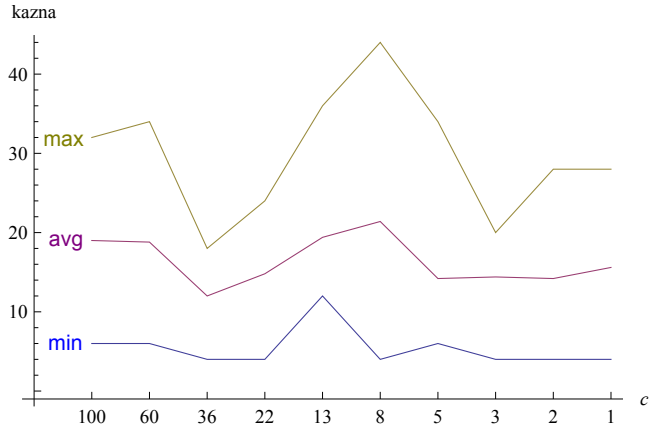
Grafički prikaz vremenske ovisnosti dan je na slici 6.5. Trajanje raste sa padom parametra c, a to se može objasniti time što se udio usmjerene pretrage povećava kako se c smanjuje (slika 6.7), a usmjerena lokalna pretraga je vremenski zahtjevnija.

Minimalna, srednja i maksimalna završna kazna populacije u ovisnosti o parametru c prikazana je tablicom 6.7, dok je grafički prikaz za tri primjera problema (*C6*, *C7* i *C11*) dan na slikama 6.9, 6.10 i 6.11.

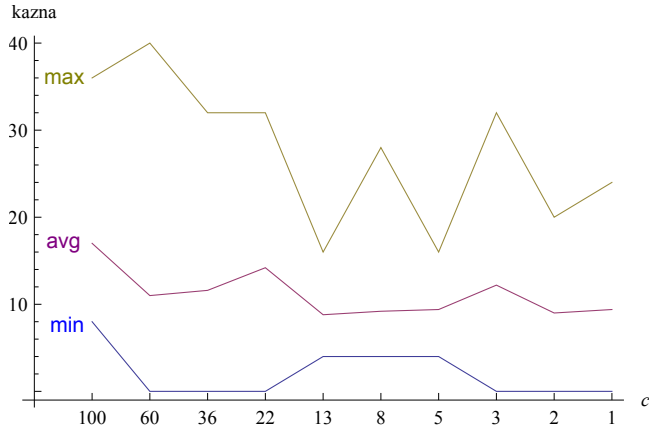
Tablica 6.7: Ovisnost kazne o parametru c

		100	60	36	22	13	8	5	3	2	1
C4	min	44,0	42,0	40,0	40,0	40,0	40,0	40,0	40,0	44,0	40,0
	avg	51,2	49,4	44,6	49,2	51,0	47,0	48,6	48,2	47,0	47,8
	max	68,0	68,0	54,0	74,0	74,0	60,0	60,0	62,0	52,0	60,0
C6	min	6,0	6,0	4,0	4,0	12,0	4,0	6,0	4,0	4,0	4,0
	avg	19,0	18,8	12,0	14,8	19,4	21,4	14,2	14,4	14,2	15,6
	max	32,0	34,0	18,0	24,0	36,0	44,0	34,0	20,0	28,0	28,0
C7	min	8,0	0,0	0,0	0,0	4,0	4,0	4,0	0,0	0,0	0,0
	avg	17,0	11,0	11,6	14,2	8,8	9,2	9,4	12,2	9,0	9,4
	max	36,0	40,0	32,0	32,0	16,0	28,0	16,0	32,0	20,0	24,0
C8	min	44,0	40,0	46,0	46,0	32,0	50,0	40,0	32,0	26,0	42,0
	avg	66,4	77,1	74,5	63,4	59,2	72,1	71,7	69,0	58,4	75,2
	max	97,0	164,0	116,0	92,0	86,0	106,0	145,0	172,0	118,0	218,0
C10	min	2,0	0,0	0,0	0,0	0,0	2,0	2,0	2,0	0,0	4,0
	avg	4,4	7,2	4,2	6,0	16,2	19,8	5,8	29,4	7,4	5,6
	max	6,0	20,0	6,0	24,0	130,0	154,0	14,0	168,0	26,0	12,0
C11	min	16,0	12,0	12,0	16,0	16,0	16,0	12,0	12,0	12,0	12,0
	avg	20,6	20,0	16,8	22,6	21,0	19,4	18,0	18,6	19,2	19,6
	max	34,0	28,0	20,0	34,0	28,0	30,0	28,0	28,0	30,0	28,0
C12	min	32,0	36,0	36,0	18,0	14,0	20,0	22,0	20,0	28,0	20,0
	avg	65,6	59,0	50,2	39,8	30,0	92,6	63,0	72,7	76,1	38,8
	max	94,0	74,0	72,0	94,0	56,0	246,0	222,0	250,0	249,0	80,0

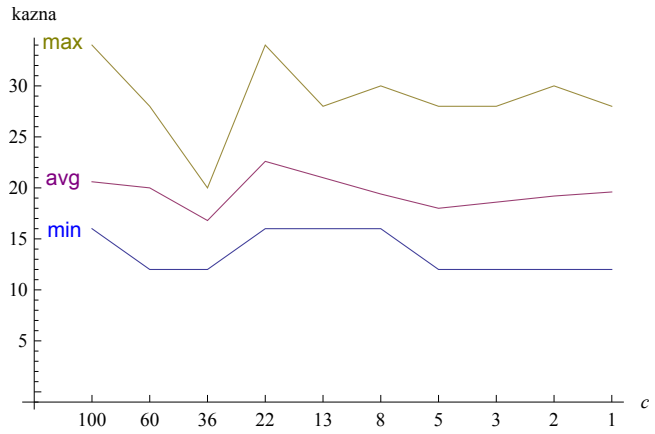
Na zadnjim primjerima ne može se uočiti trent rasta, pa se može zaključiti da su rezultati neosjetljivi na parametar c.



Slika 6.9: Ovisnost kazne o parametru *c* za primjer *C6*



Slika 6.10: Ovisnost kazne o parametru *c* za primjer *C7*



Slika 6.11: Ovisnost kazne o parametru *c* za primjer *C11*

7. Zaključak

Po uzoru na *problem sveučilišnog rasporeda(UTCP)* formalno je definiran problem *rasporeda laboratorijskih vježbi (LETP)* na Fakultetu elektrotehnike i računarstva. *LETP* problem odlikuje se vrlo specifičnim zahtjevima koji mogu biti postavljeni od osoblja fakulteta. Na temelju definicije *LETP* problema implementiran je sustav temeljen na *genetskom algoritmu* za njegovo rješavanje: izgrađena je struktura jedinke, i definirani su genetski operatori specifični za tu strukturu.

Implementirani sustav osim genetskog algoritma sadrži dodatne mehanizme bez kojih bi dobivena rješenja imala znatno lošiju kvalitetu. Ispitivanjem sustava pokazalo se da rješenja dobivena bez korištenja lokalne pretrage imaju do deset puta više preklapanja u odnosu na rješenja dobivena korištenjem lokalne pretrage, što lokalnu pretragu čini najvažnijom nadogradnjom sustava. Dodavanjem lokalne pretrage koja se intenzivno koristi genetski algoritam je proširen, te kao takav spada u klasu *hibridnog genetskog algoritma*. Bez obzira što lokalna pretraga u velikoj mjeri poboljšava učinkovitost sustava, ako se za pronalazak rješenja koristi jedino ona, bez genetskih operatora, rješenja koja bi se dobila bila bi opet loša. Operator mutacije je vrlo bitan u pretraživanju prostora stanja. Ispitivanjima se pokazalo da je s postavkama koje su isključivale operator mutacije sustav davao rješenja koja su znatno lošija u odnosu na rješenja koja su dobivena primjenom mutacije. Mehanizam detekcije stagnacije, koji kontrolira jačinu mutacije, dodatno povećava kvalitetu rješenja jer se njime spriječava da populacija prerano zapne u lokalnom optimumu.

Implementirani sustav temeljen na genetskom algoritmu pokazao se uspješan u rješavanju problema raspoređivanja laboratorijskih vježbi na Fakultetu elektrotehnike i računarstva i uspješno se primjenjuje od ljetnog semestra akademske godine 2007/2008. U dobivenim rješenjima obično ostaje manji broj preklapanja studentskih obaveza koja se rješavaju ručnom intervencijom. Time je pokazano je da su evolucijski algoritmi primjenjivi i na probleme sa vrlo specifičnim zahtjevima, kao što je problem na ovoj instituciji.

Mogući nastavak razvoja mogao bi krenuti u smijeru optimizacije raspodjele stu-

denata po terminima nakon što se primjenom postojećeg algoritma dobije rješenje sa zaostalim preklapanjima. Osim toga način evaluacije rješenja mogao bi se proširiti tako da uključuje dodatne elemente, kao što su npr. dnevna opterećenost studenata i nastavnog osoblja, optimalno iskorištavanje kapaciteta prostorija i svi drugi elementi koji mogu doprinjeti kvalitetnijem obavljanju studentskih obaveza.

Rješavanje problema raspoređivanja u visokoškolskim ustanovama evolucijskim algoritmom

Sažetak

Problem raspoređivanja u visokoškolskim ustanovama često se odlikuje izuzetno velikom vremenskom složenošću. Evolucijski algoritmi se upravo na takvoj vrsti problema pokazuju uspješnima zbog toga što pronalaze rješenje prihvatljive kvalitete u razumnom vremenu. U radu je dan kratak pregled evolucijskih algoritama, s posebnim naglaskom na genetski algoritam. Dana je definicija problema raspoređivanja laboratorijskih vježbi na Fakultetu elektrotehnike i računarstva te je opisan sustav za rješavanje tog problema temeljen na genetskom algoritmu. Genetski algoritam je potpomognut dodatnim mehanizmima kako bi se povećala kvaliteta krajnjeg rješenja. Na kraju se iznose rezultati ispitivanja učinkovitosti sustava ovisno o vrijednosti odabranih parametara sustava.

Ključne riječi: genetski algoritam, raspoređivanje, laboratorijske vježbe, lokalna pretraga, mutacija

University course timetabling using evolutionary algorithm

Abstract

University course timetabling problem instances are often characterised by a very high time complexity. Due to their ability to find solutions of acceptable quality within a reasonable amount of time, evolutionary algorithms have proven to be successful when applied to such kind of problems. This paper provides a brief overview of evolutionary algorithms, with a particular emphasis on genetic algorithm. The definition of the laboratory exercices scheduling problem at the Faculty of Electrical Engineering and Computer Science, University of Zagreb is given, and a genetic algorithm based solver is described. The genetic algorithm is supported by additional mechanisms to enhance the quality of the final solution. Finally the results of testing the effectiveness of the system depending on the values of selected parameters of the system are shown.

Keywords: genetic algorithm, timetabling, laboratory exercices, local search, mutation

LITERATURA

[1] Z. Bratković, T. Herman, V. Omrčen, M. Čupić, i D. Jakobović. University Course Timetabling with Genetic Algorithm: A Laboratory Excercises Case Study. *Evolutionary Computation in Combinatorial Optimization*, stranice 240–251.

[2] E.K. Burke i S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 127(2):266–280, July 2002. URL `http://ideas.repec.org/a/eee/ejores/v140y2002i2p266-280.html`.

[3] Tim B. Cooper i Jeffrey H. Kingston. The complexity of timetable construction problems. U *Proc. of the 1st Int. Conference on the Practice and Theory of Automated Timetabling (ICPTAT '95)*, stranice 511–522, 1995. URL `citeseer.ist.psu.edu/232089.html`.

[4] A.E. Eiben i J.E. Smith. *Introduction to evolutionary computing*. Springer Verlag, 2003.

[5] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 2007. doi: 10.1007/s00291-007-0097-0.

[6] B. McCollum. University timetabling: Bridging the gap between research and practice. U H Rudova E Burke, urednik, *PATAT 2006 — Proc. 6th Int. Conf. on the Practice And Theory of Automated Timetabling*, stranice 15 – 35. Masaryk University, 2006. URL `citeseer.ist.psu.edu/mccollum06university.html`.

[7] R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot, i S.Y. Lee. A Survey of Search Methodologies and Automated Approaches for Examination Timetabling. Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham, 2006.

[8] Andrea Schaerf. A survey of automated timetabling. U *115*, stranica 33. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 30 1995. URL citeseer.ist.psu.edu/schaerf95survey.html.