SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Marko Čupić

RASPOREĐIVANJE NASTAVNIH AKTIVNOSTI EVOLUCIJSKIM RAČUNANJEM

DOKTORSKI RAD

Zagreb, 2011.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Marko Čupić

RASPOREĐIVANJE NASTAVNIH AKTIVNOSTI EVOLUCIJSKIM RAČUNANJEM

DOKTORSKI RAD

Zagreb, 2011.

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Marko Čupić

SCHEDULING SCHOOL ACTIVITIES USING EVOLUTIONARY COMPUTATION

DOCTORAL THESIS

Zagreb, 2011.

Doktorski rad je izrađen na Sveučilištu u Zagrebu,

Fakultetu elektrotehnike i računarstva,

Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave

Mentor:

prof. dr. sc. Marin Golub

Doktorski rad ima: 285 stranica.

Doktorski rad br.: _____

Doktorski rad ocijenilo je povjerenstvo u sastavu:

1. Akademik dr. sc. Leo Budin, redoviti profesor emeritus
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

2. Dr. sc. Vedran Mornar, redoviti profesor
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

3. Dr. sc. Nedeljko Štefanić, izvanredni profesor
   Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje

Doktorski rad obranjen je pred povjerenstvom u sastavu:

1. Akademik dr. sc. Leo Budin, redoviti profesor emeritus
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

2. Dr. sc. Vedran Mornar, redoviti profesor
   Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

3. Dr. sc. Nedeljko Štefanić, izvanredni profesor
   Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje

Datum obrane disertacije: 13. srpnja 2011. godine.

Sažetak

Raspoređivanje nastavnih aktivnosti evolucijskim računanjem

Problemi izrade rasporeda nastavnih aktivnosti sveprisutni su i izrazito važni na svim obrazovnim institucijama. Prema računalnoj složenosti ovi problemi uobičajeno pripadaju u razred $\mathcal{NP}$-teških problema. U znanstvenoj literaturi obrađuje se nekoliko pojednostavljenih verzija problema; međutim, u praksi je broj problema koje je potrebno riješiti znatno veći. Stoga je u okviru ovog rada dan pregled većeg skupa problema raspoređivanja nastavnih aktivnosti. Za svaki od problema dan je formalni model te odgovarajući optimizacijski problem. Dodatno su razmotreni različiti zahtjevi koji se još postavljaju na konačna rješenja te način njihove ugradnje u optimizacijski problem.

Zbog svoje složenosti, najčešći se problemi raspoređivanja nastavnih aktivnosti ne mogu rješavati iscrpnom pretragom kao niti jednostavnim heuristikama. U okviru ovog rada postavljena je hipoteza da su algoritmi odnosno metaheuristike evolucijskog računanja prikladne za rješavanje svih prethodno formalno definiranih problema raspoređivanja. Hipoteza je provjerena izradom niza metaheurističkih algoritama i provjerom njihove sposobnosti rješavanja stvarnih primjera. Razmotreni su načini paralelizacije ovih algoritama, počev od problemski specifičnih. Ostvaren je i hibridni paralelni algoritam evolucijskog računanja.

Definiran je formalni model sustava za potporu procesima raspoređivanja koji omogućava objavu napravljenih rasporeda te provođenje analiza zauzeća korisnika. Temeljem tog modela izgrađen je programski sustav čiji je rad ispitan u praksi.

Ključne riječi: raspoređivanje nastavnih obaveza, algoritmi evolucijskog računanja, paralelizacija algoritama, hibridni algoritam evolucijskog računanja.

Abstract

Scheduling School Activities using Evolutionary Computation

Problems of scheduling of school activities are ubiquitous and extremely important to all educational institutions. According to the computational complexity of these problems they usually belong to the class of $\mathcal{NP}$-hard problems. The scientific literature usually deals with simplified versions of several such problems. However, in practice, a number of problems that needs to be solved is much larger. In this thesis therefore an overview of a larger set of scheduling problems of school activities is given. For each problem a formal model is determined and an associated optimization problem is given. In addition, a various requirements are described that are usually additionally placed on the final schedule as well as the technique for its inclusion in optimization problem.

Because of complexity, the most common scheduling problems of school activities can not be solved by means of exhaustive search nor simple heuristics. In this thesis, we hypothesize that the evolutionary computation algorithms / metaheuristics are suitable for solving all previously formally defined scheduling problems. Hypothesis was verified by creating a series metaheuristics algorithms and checking their ability to solve real-world problem instances. In this thesis the parallelization of these algorithms is also researched, starting from problem-specific paralelization techniques. Also, the parallel hybrid evolutionary computation algorithm is described.

Finally, a formal model of the system is described which offers the support for publishing results of scheduling and which allows conducting the analysis of users business. Based on this model, a software system was built and tested in practice.

Keywords: scheduling of school activities, evolutionary computation algorithms, parallelization of algorithms, hybrid evolutionary computation algorithm.

Sadržaj

Popis slika

Popis tablica

Izvorni tekstovi programa

Poglavlje 1

Uvod i motivacija

Organizacija te izvođenje nastave na sveučilištu velik je i iznimno složen problem. Pronalazak prikladnih rješenja važan je zadatak jer loša rješenja imaju utjecaj na tisuće studenata, stotine nastavnika i nastavnog osoblja, kvalitetu izvođenja nastave te najvažnije – kvalitetu savladavanja i polaganja gradiva. Uporaba računala, posebice uz danas dostupnu veliku procesnu moć, može bitno doprinijeti kvalitetnom rješavanju ovih problema.

Problem organizacije nastave možemo razložiti na više manjih, ali opet povezanih podproblema: (i) uporaba računala u svrhu poučavanja studenata te računalne provjere njihova znanja, (ii) uporaba računala kao potpora bržoj pripremi i provođenju pisanih provjera znanja te (iii) uporaba računala kao pomoć za izradu kvalitetnijih rasporeda nastavnih obaveza.

Preliminarna istraživanja o uporabi računala u svrhu poučavanja studenata te računalne provjere njihova znanja opisana su u okviru radova [Čupić and Mihajlović, 2010, Čupić, 2008, Glavinić et al., 2008], gdje su ujedno načinjeni i opisani konkretni sustavi koje svakodnevno koriste stotine studenata. U tim radovima razrađena je platforma koja može poslužiti kao temelj za daljnja istraživanja u rješavanju opisanog problema.

Istraživanje uporabe računala kao potpore bržoj pripremi i provođenju pisanih provjera znanja kao i razvoj programske podrške započeto je u okviru projekta informacijske tehnologije [Čupić, 2009] kojemu je cilj razvoj odgovarajućih alata otvorenog koda. Ovdje se javlja niz problema koji traže odgovarajuća rješenja: kako brzo i efikasno izraditi i ocijeniti provjere znanja [Čupić, 2010, Šnajder et al., 2008], kako analizirati dobivene rezultate i temeljem njih korigirati pitanja [Čupić et al., 2009b], te kako omo-

gućiti strojno očitavanje provjera znanja [Bonačić et al., 2009, Čupić, 2010]. Pronalazak još kvalitetnijih rješenja zahtjeva još dosta istraživanja i izrade potpornih programskih rješenja.

Konačno, jedan od segmenata projekta [Čupić, 2009] zahtjeva i izradu programskih rješenja koja će pomoći da se provjere znanja izvedu na takav način da im svi studenti mogu prisustvovati. Stoga je izrada prikladnih rasporeda od presudnog značaja. Štoviše, problem raspoređivanja prisutan je u svim segmentima ljudskog djelovanja i od iznimnog je značaja za redovno funkcioniranje društva. Neki od tipičnih primjera su pronalaženje optimalnog načina transporta roba, raspoređivanje poslova po strojevima u tvornicama, raspoređivanje poslova u računalnim sustavima te raspoređivanje grupa ljudi (primjerice izrada satnice predavanja, rasporeda laboratorijskih vježbi te rasporeda provjera znanja na sveučilištu). Ovi se problemi mogu rješavati različitim tehnikama, a jedna od pogodnih tehnika su algoritmi evolucijskog računanja. Uporaba takvih rješenja u organizaciji nastave opisana je u [Čupić et al., 2010, Čupić and Franović, 2010]. Uporaba evolucijskih algoritama za izradu rasporeda laboratorijskih vježbi opisana je u [Bratković et al., 2009, Dino Matijaš et al., 2010] dok je primjena evolucijskih algoritama na izradu rasporeda provjera znanja opisana u [Čupić et al., 2009a]. Svi ti radovi ukazuju na značaj problema raspoređivanja te naglašavaju potrebu za pronalaskom visoko-kvalitetnih rješenja.

1.1 Postavljeni ciljevi

Imajući u vidu prethodno opisanu situaciju, u okviru ove doktorske disertacije naglasak rada stavljen je upravo na probleme raspoređivanja nastavnih aktivnosti. Prvo pitanje koje je analizirano stoga je upravo pitanje vrsta problema raspoređivanja nastavnih obaveza koji se javljaju u obrazovnim institucijama. U okviru drugog poglavlja dan je pregled problema raspoređivanja koji se danas tipično mogu naći u znanstvenoj literaturi. Kako ovaj rad pokazuje, to je samo mali podskup problema s kojima se svakodnevno susreće administrativno i nastavno osoblje i za koje su nužno potrebna prikladna rješenja. Velik dio danas dostupnih radova su radovi koji rješavaju pojednostavljene verzije problema. Takve verzije problema danas nisu od pretjerano velike koristi jer se u praksi od rasporeda zahtjeva još niz dodatnih uvjeta koje ovi radovi ne uzimaju u obzir. Ovaj zaključak iznosi i Barry McCollum u radu [McCollum, 2006], prikladno nazvanom *Ras-*

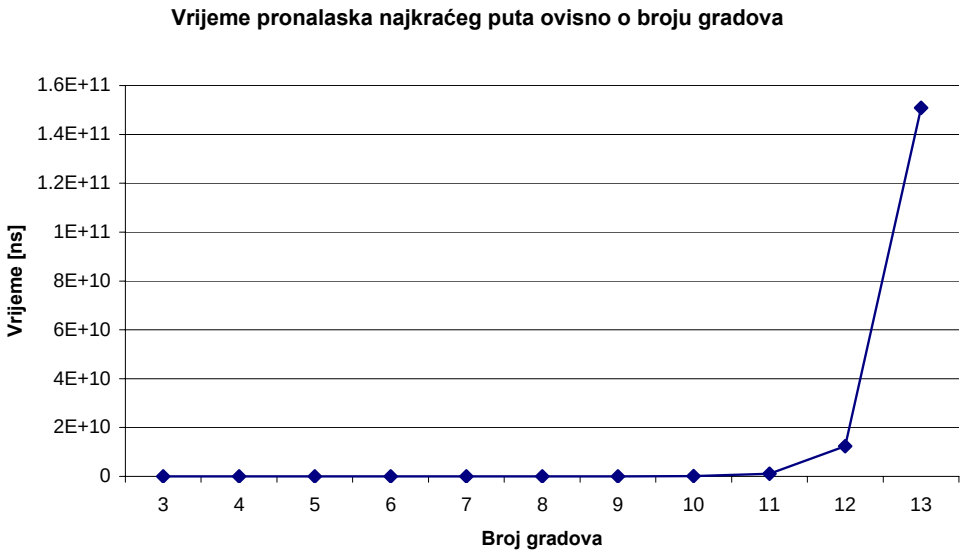
poređivanje na sveučilištu: premošćivanje jaza između istraživanja i prakse. Opravdanje koje se najčešće navodi je mogućnost uspoređivanja različitih pristupa na standardiziranom skupu ispitnih podataka. Naime, činjenica je da se zahtjevi na rasporede razlikuju od zemlje do zemlje i od sveučilišta do sveučilišta; do današnjeg dana nitko još nije uspio načiniti zajednički jezik kojim bi bilo moguće opisivati svu tu raznolikost i potom izgraditi priklade djelotvorne algoritme. Stoga je učestala praksa definirati niz pojednostavljenja i algoritme potom optimirati tako da rješavaju takve specifične probleme koje je praksi rijetko moguće iskoristi.

U okviru ove disertacije dan je pregled niza problema raspoređivanja te su istražene njihove specifičnosti i načini rješavanja.

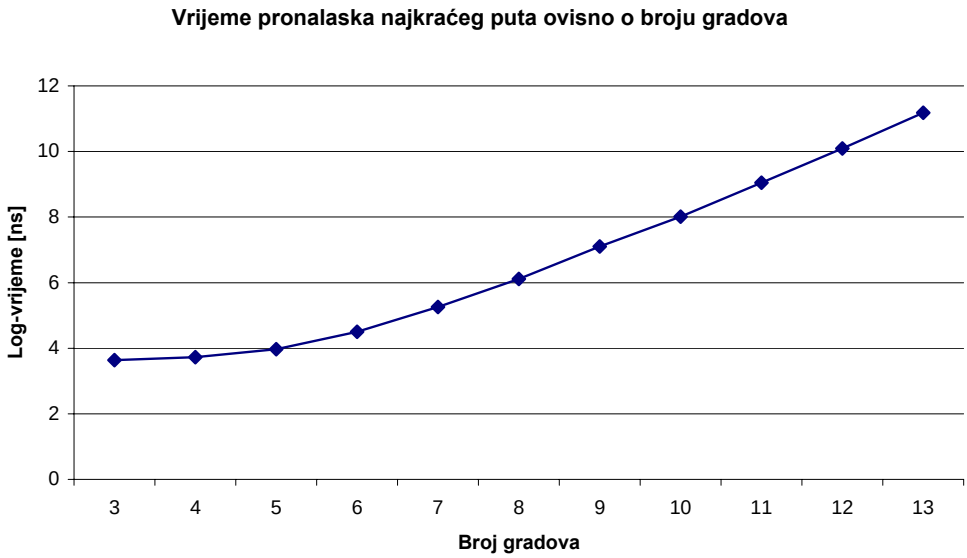
Sljedeći postavljeni cilj je istraživanje mogućnosti primjene jednog određenog načina rješavanja na sve opisane probleme; polazna pretpostavka je da su algoritmi evolucijskog računanja prikladni za dobivanje kvalitetnih rješenja svih spomenutih problema. Kao što je pokazano u poglavlju 2, problemi izrade rasporeda tipično su $\mathcal{NP}$-potpuni problemi. To znači da je provedba optimizacijskog procesa iscrpnim pretraživanjem (engl. *brute force*) neizvediva za najveći broj slučajeva iz stvarnog života, jer je broj kombinacija koje je potrebno provjeriti prevelik.

Kao ilustraciju složenosti ovakvih problema razmotrimo poznati problem trgovačkog putnika. Na stolnom računalu je načinjen niz mjerenja za probleme od 3 do 13 gradova. Pri tome je korišteno računalo s AMD Turion 64 procesorom na 1.8 GHz i s 1 GB radne memorije; algoritam iscrpne pretrage bio je ostvaren u programskom jeziku Java. Algoritam je problem s 11 gradova rješavao u manje od 5 sekundi. Vrijeme potrebno za rješavanje problema s 12 gradova iznosilo je približno 12.3 sekunde, dok je za 13 gradova bilo potrebno poprilici 2,5 minute. Očekivano vrijeme rješavanja problema s 14 gradova je oko pola sata, za 15 gradova oko 7,6 sati, dok bi za 16 gradova trebalo oko 4,7 dana. Ovo je lijepo vidljivo na slici 1.1 odnosno na slici 1.2 gdje je skala ordinate logaritamska. Detaljnijom analizom ovog problema te samih podataka eksperimenta, lako je utvrditi da je složenost problema faktorijelna pa je jasno da je problem koji se sastoji od primjerice 150 gradova primjenom tehnike grube sile praktički nerješiv; problemi raspoređivanja s kojima se susrećemo su upravo takve vrste. U teoriji grafova, problem trgovačkog putnika odgovara pronalasku Hamiltonovog ciklusa u grafu.

U situacijama gdje u raspoloživom vremenu nije moguće doći do egzaktnih rješenja



Slika 1.1: Ovisnost vremena pretraživanja kod iscrpne pretrage o broju gradova kod TSP-a



Slika 1.2: Ovisnost vremena pretraživanja kod iscrpne pretrage o broju gradova kod TSP-a uz logaritamsku skalu po ordinati

odnosno pronaći rješenje za koje se postiže optimum, kao prikladnim se je pokazalo koristiti heurističke metode ili kraće, heuristike. *Heuristike* su algoritmi koji pronalaze rješenja koja su dovoljno dobra, imaju relativno nisku računsku složenost (tipično govorimo o polinomijalnoj složenosti) ali ne nude nikakve garancije da će uspjeti pronaći optimalno rješenje. Heuristike dijelimo na konstrukcijske heuristike te na algoritme lokalne pretrage.

Konstrukcijski algoritmi rješenje problema grade dio po dio (bez povratka unatrag) sve dok ne izgrade kompletno rješenje. Tipičan primjer je algoritam najbližeg susjeda (engl. *nearest-neighbor procedure*). Na problemu trgovačkog putnika, ovaj algoritam započinje tako da nasumice odabere početni grad, i potom u turu uvijek odabire sljedeći najbliži grad. *Algoritmi lokalne pretrage* rješavanje problema započinju od nekog početnog rješenja, koje potom pokušavaju inkrementalno poboljšati. Ideja je da se definira skup jednostavnih izmjena koje je moguće obaviti nad trenutnim rješenjem, čime se dobivaju susjedna rješenja. U najjednostavnijoj verziji, algoritam za trenutno rješenje pretražuje skup svih susjednih rješenja i bira najboljeg susjeda kao novo trenutno rješenje (govorimo o metodi uspona na vrh, (engl. *hill-climbing method*)). Ovo se ponavlja tako dugo dok kvaliteta rješenja raste.

Ovako jednostavne (problemski specifične) heuristike probleme izrade rasporeda ne rješavaju dovoljno dobro. Stoga je drugi smjer koji se danas istražuje uporaba metaheuristika. Metaheuristika [Glover and Kochenberger, 2003, Talbi, 2009] je skup algoritamskih koncepata koji koristimo za definiranje heurističkih metoda primjenjivih na širok skup problema. Možemo reći da je metaheuristika heuristika opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema području u prostoru rješenja u kojem se nalaze dobra rješenja [Dorigo and Stützle, 2004]. Primjeri metaheuristika su simulirano kaljenje, tabu pretraživanje, evolucijsko računanje i slični.

Stoga je zanimljivo pitanje – mogu li se metaheuristički algoritmi iz porodice algoritama evolucijskog računanja uspješno koristiti za rješavanje svih problema koji su promatrani u ovome radu?

Dodatno, postavlja se i pitanje je li dobro koristiti nasumičnu pretragu, te da li, kada i u kojoj mjeri uvesti dodatne informacije u algoritam. Uporaba dodatne informacije u postupku pretraživanja čini razliku između slijepog i usmjerenog pretraživanja. U nedostatku bilo kakvih smjernica, algoritmi mogu samo nasumično generirati moguća

rješenja. Stoga bi se moglo pretpostaviti da će uvođenjem dodatnih informacija algoritmi raditi bolje. Međutim, uvođenjem dodatnih informacija pretraga se fokusira i time se smanjuje prostor pretraživanja. To u nekim slučajevima može djelovati nepovoljno na postupak pretraživanja (primjerice, u prisustvu velikog broja lokalnih ekstrema).

Ovakvim razmišljanjem dolazi se i do sljedećeg pitanja: *koji je algoritam pretraživanja onda najbolji*? Na našu sreću (ili ne), danas znamo odgovor na ovo pitanje. Wolpert i Macready su u svojim radovima dokazali da nema najboljeg algoritma. Dapače, dokazali su da su svi algoritmi pretraživanja – upravo prosječno dobri [Wolpert and Macready, 1995, 1997]:

> *All algorithms that search for an extremum of a cost function perform exactly the same, according to any performance measure, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A.*

U ovoj disertaciji stoga je fokus stavljen na puno uži skup problema i puno uži skup funkcija cilja; promatraju se tipični problemi raspoređivanja koji se javljaju prilikom organizacije nastave i nastavnih aktivnosti, a specifični primjeri bit će problemi raspoređivanja s kojima je autor u doticaju na Fakultetu elektrotehnike i računarstva. Ideja je pokazati da se evolucijsko računarstvo može primjeniti na rješavanje ove vrste problema te da se mogu dobiti zadovoljavajuća rješenja.

Nadalje, u okviru disertacije istražene su mogućnosti paralelizacije predloženih algoritama koji se temelje na metaheristikama. Prvi korak je razmatranje paralelizacije na razini pojedinih algoritma. Drugi smjer istraživanja bio je usmjeren prema ideji da se različiti algoritmi međusobno dobro nadopunjuju, te da se paralelizacija koja se radi tako da se u postupak uključi više različitih metaheuristika ponaša barem jednako dobro a ponekad i bolje u odnosu na algoritme koji koriste samo jednu vrstu metaheuristike. Naime, kada metaheuristički algoritam zapne u nekom lokalnom optimumu, implicitna pretpostavka je da, zbog činjenice da druga metaheuristika koristi drugačiji skup operatora, drugom algoritmu to neće biti lokalni optimum već rješenje koje se dalje može unaprijediti.

Konačno, pred kraj disertacije dan je osvrt na podatkovni model koji je potpora za objavu načinjenih rasporeda i koji time nudi mogućnost pripreme podataka koje

je potom moguće koristiti kao ulaz drugih algoritama koji rješavati kasnije probleme raspoređivanja.

1.2 Organizacija disertacije

Ova disertacija organizirana je na sljedeći način. Prvo poglavlje sadrži uvodna razmatranja u kojima se daju obrisi problema koji se riješavaju kroz disertaciju te koja je motivacija. U okviru prvog poglavlja dan je kratak pregled radova autora disertacije objavljenih u časopisima i na znanstvenim skupovima s međunarodnom recenzijom koji predstavljaju podlogu na kojoj je započet rad opisan u ovoj disertaciji. Zatim je dan pregled postavljenih ciljeva. Načinjen je kratak osvrt na probleme raspoređivanja nastavnih obaveza te na njihovu kompleksnost koja je tipično $\mathcal{NP}$-potpuna. Kao metode za dobivanje dovoljno kvalitetnih no ne nužno i optimalnih rješenja uvodene su heuristike te metaheuristike. Analizirano je pitanje prikladnosti različitih algoritama u svjetlu *No-free-lunch* teorema. Izložena je ideja paralelizacije te izgradnje hibridnog paralelnog algoritma evolucijskog računanja.

Drugo i treće poglavlje su pregledna poglavlja. U drugom poglavlju opisani su problemi raspoređivanja koji se susreću u znanstvenoj literaturi; opisani su i najčešće korišteni pristupi za njihovo rješavanje. Najprije su definirane dvije vrste problema raspoređivanja: raspoređivanje s ciljem minimizacije troškova i raspoređivanje s ciljem optimalnog korištenja raspoloživih sredstava. Definirani su pojmovi mekih i tvrdih ograničenja koja se primjenjuju pri rješavanju problema raspoređivanja. Dan je kratak pregled metoda za rješavanje problema raspoređivanja. Potom su definirana i opisana tri problema raspoređivanja nastavnih aktivnosti koji su najviše zastupljeni u znanstvenoj literaturi: problem izrade rasporeda predavanja u školama, problem izrade rasporeda predavanja na sveučilištu te problem izrade rasporeda provjera znanja. Za svaki od tih problema dana je i formalna definicija.

U trećem poglavlju načinjen je pregled najzastupljenijih algoritama evolucijskog računanja. Opisani su genetski algoritam, genetsko programiranje, algoritam diferencijske evolucije, porodica mravljih algoritama, algoritam roja čestica te algoritmi umjetnih imunoloških sustava. U okviru tog opisa razmotreni su načini prikaza rješenja koji se koriste prilikom implementacije algoritama evolucijskog računanja kao i operatori koje koriste pojedini algoritmi. Opisani su različiti načini odabira jedinki iz populacije kao

i njihove razlike. Rad algoritama ilustriran je na jednostavnim primjerima.

U četvrtom poglavlju navedene su formalne definicije problema koji se obrađuju u ovoj disertaciji kao i znanstveni radovi autora u kojima su neki od njih već publicirani na međunarodnim znanstvenim skupovima. Definirano je ukupno osam problema raspoređivanja: jednostavno raspoređivanje, raspoređivanje neraspoređenih studenata po predavanjima, uklanjanje konflikata u satnici predavanja na razini studenata, raspoređivanje laboratorijskih vježbi, raspoređivanje provjera znanja, raspoređivanje prostorija za provjere znanja, raspoređivanje timova te izrada prezentacijskih grupa za seminare. Svaki od tih problema raspoređivanja formalno je definiran te je naveden skup čvrstih ograničenja. Kako tako definirani problemi u praksi najčešće nemaju rješenja, problemi raspoređivanja svedeni su na optimizacijske probleme koji se zasnivaju na skupu čvrstih ograničenja koja treba poštivati te na skupu mekih ograničenja čiju količinu kršenja treba minimizirati. Za pojedine vrste problema razmotrena su moguća meka ograničenja te su dane njihove formalne definicije.

U petom poglavlju razmotrena je primjena algoritama evolucijskog računanja na odabrane probleme raspoređivanja koji su opisani u četvrtom poglavlju. Za rješavanje problema jednostavnog raspoređivanja dan je opis zajedničkih metoda koje se koriste pri implementaciji različitih algoritama evolucijskog računanja za rješavanje navedenog problema. Potom je opisana primjena (a) genetskog algoritma, (b) algoritma roja čestica, (c) algoritma *Max-Min* mravlji sustav, (d) jednostavnog imunološkog algoritma te (e) algoritma klonske selekcije na rješavanje navedenog problema. Definirana je struktura za prikaz rješenja kao i način vrednovanja dobivenih rješenja. Rad algoritama ispitan je na problemu raspoređivanja te su uspoređena rješenja. Za potrebe rješavanja problema izrade rasporeda provjera znanja razmotrena je uporaba genetskog algoritma, algoritma diferencijske evolucije te algoritma *Max-Min* mravlji sustav. Opisane su korištene globalne strukture podataka, struktura podataka korištena za prikaz rješenja, način vrednovanja rješenja te način provođenja lokalne pretrage. Ispitana je primjenjivost navedenih algoritama na jednom problemu izrade rasporeda provjera znanja. Provedeni su eksperimenti bez i s uporabom lokalne pretrage te je načinjena usporedba dobivenih rezultata. Na kraju poglavlja dan je osvrt na primjenu algoritama evolucijskog računanja na druge probleme raspoređivanja nastavnih obaveza.

U šestom poglavlju razmotrena je paralelizacija algoritama evolucijskog računanja

primijenjenih na probleme raspoređivanja nastavnih aktivnosti. Objašnjeni su razlozi za paralelizaciju algoritama, vrste paralelizacije te cijena paralelizacije. Paralelizacija algoritama evolucijskog računanja ostvarena je na problemu paralelizacije algoritama za rješavanje problema izrade rasporeda provjera znanja uz opasku da su analizirani načini primjenjivi i na druge probleme raspoređivanja nastavnih aktivnosti. Najprije je ostvarena paralelizacija na razini pojedinih operatora koja spada u problemski specifične paralelizacije. Prikazan je primjer paralelizacije operatora vrednovanja. Provedeno je ispitivanje ubrzanja koje se postiže paralelizacijom operatora vrednovanja na računalu s jednim procesorom, dva procesora, tri procesora i četiri procesora a s obzirom na veličinu posla. Potom je razmotrena paralelizacija na razini populacije koja je provedena za genetski algoritam te algoritam *Max-Min* mravlji sustav. Za oba je algoritma provedeno ispitivanje dobivenih ubrzanja na jednom, dva, tri i četiri procesora. Sljedeće je analiziran pristup paralelizaciji na razini algoritama. Uz topologiju dvorazinskog prstena provedeno je ispitivanje sustava algoritama sastavljenih od 12 genetskih algoritama, od 12 algoritama roja čestica, od 12 algoritama *Max-Min* mravlji sustav te na kraju od 12 jednostavnih imunoloških algoritama. Zatim je izgrađen paralelni heterogeni algoritam evolucijskog računanja. Provedena su ispitivanja na primjeru jednostavnog raspoređivanja te na primjeru izrade rasporeda provjera znanja. U slučaju jednostavnog raspoređivanja analizirane su topologije sastavljene od genetskog algoritma, algoritma Max-Min mravlji sustav, algoritma roja čestica, jednostavnog imunološkog algoritma te njihove kombinacije. U slučaju izrade rasporeda provjera znanja ispitane su kombinacije sastavljene od genetskog algoritma, algoritma Max-Min mravlji sustav, jednostavnog imunološkog algoritma te algoritma harmonijske pretrage. U oba slučaja pokazano je da određene kombinacije algoritama postižu usporedive ili bolje rezultate nego homogeni algoritmi.

U sedmom poglavlju definiran je model podataka prikladan za objavu te provođenje analiza nad rezultatima raspoređivanja nastavnih obaveza. Definirani model podataka prikladan je za uporabu u pojednostavljenom sustavu za upravljanje kolegijima. Model podataka prikazan je odgovarajućim dijagramom razreda jezika UML. Kako bi se provjerila prikladnost definirnog modela podataka, izrađena je prototipna implementacija sustava za upravljanje kolegijima *Ferko* koja je ukratko opisana. Sustav *Ferko* korisnicima nudi prikaz osobnih kalendara obaveza s poveznicama na odgovarajuće nastavne

aktivnosti. Djelatnicima se također nudi izrada analize zauzeća studenata kao i prikaz zauzeća odnosno kalendare svih prostorija koje su registrirane u sustavu. Za svaki kalendar nudi se dohvat i u standardnom obliku iCal.

Konačno, osmo poglavlje sadrži zaključna razmatranja. Dan je sažetak istraživanja provedenog u okviru ove disertacije i osvrt na provedene eksperimente. Ponovljeni su doprinosi te su navedene smjernice za daljnja istraživanja.

Poglavlje 2

Pregled problema raspoređivanja nastavnih obaveza

Pojam raspoređivanja odnosi se na više različito definiranih problema. Prema [Wren, 1996], razlikujemo nekoliko problema raspoređivanja, od kojih su u nastavku prenesene definicije za dva tipična.

> *Raspoređivanje s ciljem minimizacije troškova (engl.* scheduling*)* je dodjela sredstava (resursa) objektima koja se događa u prostoru i vremenu, koja poštuje zadani skup ograničenja i koja se radi na način koji minimizira ukupnu cijenu uporabe dodijeljenih sredstava. Čest primjer je raspoređivanje transporta ili usmjeravanje dostavnih vozila pri čemu se pokušava minimizirati broj korištenih vozila ili broj potrebnih vozača, te unutar tog minimuma još se dodatno pokušava smanjiti ukupna cijena. Drugi primjer je raspoređivanje poslova kojim se pokušava minimizirati broj korištenih vremenskih odjeljaka ili neko fizičko sredstvo.
>
> *Raspoređivanje s ciljem optimalnog korištenja raspoloživih sredstava (engl.* timetabling*)* je dodjela postojećih sredstava (resursa) skupu objekata koja se događa u prostoru i vremenu te koja poštuje zadani skup ograničenja i koja pokušava zadovoljiti u što većoj mjeri skup poželjnih svojstava takve dodjele. Primjeri su izrada rasporeda predavanja, izrada rasporeda provjera znanja i slično.

Nešto jednostavniju definiciju problema raspoređivanja daju Nau i suradnici [Nau et al., 2004]:

> Problem raspoređivanja je kako obaviti zadani skup akcija koristeći ograničen skup sredstava u ograničenom vremenskom periodu.

Ograničenja koja pri tome spominjemo mogu se podijeliti u dva razreda (primjerice, [Burke et al., 1997]):

- čvrsta ograničenja (engl. *hard constraints*) te
- meka ograničenja (engl. *soft constraints*).

Čvrsta ograničenja su ograničenja koja nužno moraju biti zadovoljena da bi načinjeni raspored bio valjan, odnosno prihvatljiv. Primjer ovakvog ograničenja je da se neko sredstvo ne može koristiti u isto vrijeme na dva mjesta (npr. student ne može istovremeno biti raspoređen na dva ispita; u nekom vremenskom periodu mora biti dovoljno resursa za obavljanje svih zakazanih događaja – npr. dovoljno dvorana za održavanje zakazanih ispita; i sl).

Meka ograničenja su ograničenja koja je poželjno zadovoljiti u što većoj mjeri; međutim, ako neka ograničenja nije moguće zadovoljiti, i takav raspored je valjan i prihvatljiv. Primjeri takvih ograničenja su: posao koji stroj obavlja treba biti koncentriran u što je moguće dulje periode, raspored predavanja trebao bi biti što je moguće više koncentriran (uz što je moguće manje prekida), raspored ispita treba biti takav da niti jedna dva ispita koja imaju zajedničkog studenta nisu blizu, i sl.

Iscrpan pregled tehnika za rješavanje problema raspoređivanja s ciljem minimizacije troškova može se pogledati u [Pinedo, 2008]. U okviru ove disertacije ta se vrsta problema ne razmatra. Problemi koji se razmatraju u ovoj disertaciji su problemi raspoređivanja s ciljem optimalnog korištenja raspoloživih sredstava.

Kada se govori o problemima raspoređivanja nastavnih obaveza, tipični primjeri problema koji se obrađuju u literaturi su izrada rasporeda predavanja za škole, izrada rasporeda predavanja na fakultetima, izrada rasporeda provjera znanja i slično. Svaki od tih problema ima svoje specifičnosti [Schaerf, 1999]. Primjerice, prilikom pripreme rasporeda provjera znanja jedna ili više provjera mogu biti smještene u istu dvoranu. No isto tako u slučaju provjere koju pohađa velik broj učenika jedna provjera može biti

raspodijeljena u više dvorana. Također, ako student ima više provjera znanja, poželjno je da one u vremenu budu što više raspršene, kako bi se smanjio stres kojem su studenti izloženi. S druge strane, prilikom izrade rasporeda predavanja jedan se kolegij smješta u jednu dvoranu (uz pretpostavku da na kolegiju postoji samo jedna grupa studenata) i poželjno je da raspored bude što kompaktniji, kako ne bi bilo rupa u rasporedima studenata.

S obzirom na uočene sličnosti i razlike, u posljednje se vrijeme razmišlja o izradi jednog sveobuhvatnog modela ili jezika za opis problema raspoređivanja [Zervoudakis and Stamatopoulos, 2000, Reis and Oliveira, 2000, Ranson and Ahmadi, 2007]. Međutim, do danas se taj pristup nije pokazao pretjerano uspješnim. Razloga ima više. Na svakoj od institucija koje rješavaju problem raspoređivanja postoje različiti zahtjevi koji se moraju poštivati ili optimirati pa je praktički nemoguće predvidjeti sve što bi se moglo zahtjevati. Stoga nije moguće niti ponuditi odgovarajući način opisivanja svih zahtjeva. Čak i u slučaju da se načini dovoljno općenit model, algoritam koji bi tražio rješenje na tako univerzalnoj razini bio bi ekstremno neučinkovit jer ne bi u obzir uzimao specifičnosti pojedinih problema koje mogu smanjiti prostor pretraživanja i ubrzati pretragu. Zbog složenosti problema raspoređivanja ovo je dovoljan razlog da u trenutku pisanja ove disertacije još uvijek nema nikakvih značajnijih i raširenijih implemetacija tog tipa sveobuhvatnog rješenja.

Potvrdu činjenice da su na svakom sveučilištu zahtjevi barem malo drugačiji možemo naći i u mnoštvu radova koji opisuju upravo specifične probleme tih sveučilišta i način njihovog rješavanja, npr. raspored predavanja škola u Grčkoj [Beligiannis et al., 2008], raspored predavanja na *University of Waterloo* [Carter, 2001], raspored predavanja na Spanish University: School of Telecommunications Engineering, *Universidade de Vigo* [Santiago-Mozos et al., 2005], određivanje kojem će se studentu dozvoliti upis kojeg kolegija temeljem njegovih želja i preferenci na Eindhoven University of Technology [van den Broek et al., 2007], raspored završnih ispita na *United States Military Academy in West Point* [Wang et al., 2010], kod kojeg je specifičnost da raspored nije moguće načiniti tako da svaki kolegij ima ispit u samo jednom terminu, pa se optimira raspored ispita na način da se minimizira broj termina u kojima će se održavati ispit iz istog kolegija.

Kada se razmatraju tipični problemi raspoređivanja, poznato je da su oni uobičajeno $\mathcal{NP}$-potpuni problemi. Tako su, primjerice, za specifičnu vrstu problema izrade rasporeda predavanja to dokazli Even i suradnici [Even et al., 1976] svođenjem prethodno poznatog $\mathcal{NP}$-potpunog problema 3-SAT [Garey and Johnson, 1979] na tu varijantu problema izrade rasporeda.

Unatoč (ili zahvaljujući) ovoj složenosti, odavno postoji težnja da se rješavanje ovih problema automatizira jer je ručno dobivanje bilo kakvih valjanih rješenja (uopće ne uzimajući u obzir dodatna meka ograničenja) vrlo dugotrajan i zamoran posao koji često nije niti provediv. Prema [Burke and Petrovic, 2002, Carter, 1986, Carter and Laporte, 1996, 1998], tehnike rješavanja ovih problema možemo podijeliti na četiri pristupa:

- slijedne metode (engl. *sequential methods*),
- metode grupiranja (engl. *cluster methods*),
- pristupi temeljeni na ograničenjima (engl. *constraint-based approaches*) te
- metaheurističke metode (engl. *meta-heuristic methods*).

Kod slijednih metoda redoslijed kojim će se događaji dodavati u raspored određuje se temeljem domenski specifične heuristike, i potom se tim redosljedom smještaju u raspored u neki od termina u kojem ne krše tvrda ograničenja. U ovom pristupu problem se najčešće preslikava u problem bojanja grafova [de Werra, 1985], pri čemu su događaji (kolegiji) čvorovi grafa, termini su boje a dva čvora grafa su međusobno povezana lukom ako dijele barem jednog studenta. Izrada rasporeda tada se svodi na pronalazak takve podjele boja po čvorovima da niti jedna dva čvora koja su međusobno povezana lukom nisu obojana istom bojom. Za utvrđivanje redosljeda Koristi se nekoliko heuristika [Brélaz, 1979, Carter and Laporte, 1996], poput redanja po broju lukova koji izlaze iz nekog čvora, redanja po težinskoj sumi broja lukova koji izlaze iz nekog čvora (težina je broj dijeljenih studenata), davanje prvenstva čvoru za koji je preostalo najmanje termina u koje se čvor može smjestiti a da ne krši tvrda ograničenja, davanje prvenstva čvoru koji ima najviše konflikata s do tada razmještenim kolegijima i slično. Primjer recentnijih pristupa u rješavanju ovih problema su [Burke et al., 2007, Ochoa et al., 2009].

Metode grupiranja polaze od pretpostavke da je lakše riješiti manje probleme, pa u tom smislu pokušavaju događaje razdijeliti u manje grupe koje zadovoljavaju tvrda

ograničenja i koje se potom smještaju u termine tako da se pazi na meka ograničenja [White and Chan, 1979]. Malo drugačiji pristup u kojem se, umjesto grupiranja svih kolegija, grupiraju samo kolegiji koji zadovoljavaju postavljen skup ograničenja čime se ipak donekle smanjuje prostor pretraživanja i povećava vjerojatnost pronalaska boljih rješenja opisan je u [Kendall and Li, 2008]. Slični metodama grupiranja su i pristupi dekompozicije koji dijele složeni problem na jednostavnije koje potom rješavaju jednostavnim metodama [Burke and Newall, 1999, Carter, 1983, Qu and Burke, 2007]. Primjerice, [Qu and Burke, 2007] skup kolegija dijeli u dva podskupa: tipično manji *težak podskup* i tipično veći *lagan podskup* koji se potom raspoređuju. Nažalost, pokazalo se je da ovakvi pristupi daje rješenja relativno slabe kvalitete, s obzirom na postavljena meka ograničenja.

Pristupi temeljeni na ograničenjima modeliraju problem raspoređivanja skupom varijabli (tipično događaji, npr. ispiti kolegija) kojima treba pridijeliti vrijednosti (primjerice, termine, dvorane i slično) uz zadovoljavanje skupa postavljenih ograničenja [Brailsford et al., 1999, White, 2001].

Pristupi temeljeni na metaheurističkim metodama su pristupi koji za rješavanje problema raspoređivanja koriste algoritme poput simuliranog kaljenja, tabu pretrage, evolucijskog računanja i slično. Ovi algoritmi kreću ili s jednim rješenjem ili s populacijom rješenja i inkrementalno grade nova rješenja temeljem starih koja u prosjeku postaju sve bolja i bolja. Radova iz ovog područja ima mnoštvo, npr. [Burke and Ross, 1997, Burke and Carter, 1997, Colorni et al., 1998, Burke and Erben, 2000, Burke and Causmaecker, 2002, Burke and Trick, 2004, Burke and Rudová, 2006, Chiarandini et al., 2006, Lewis, 2008].

Primjena metaheurističkih postupaka za rješavanje problema raspoređivanja pokazuje dosta potencijala. Naime, ovi algoritmi, iako ne garantiraju pronalazak optimalnog rješenja (pa u tom smislu nisu kompletni), pokazali su se djelotvornima u rješavanju problema kod kojih prostor pretraživanja raste eksponencijalno [Lobo et al., 2000]. Tako su problemi raspoređivanja rješavani simuliranim kaljenjem (npr. [Duong and Lam, 2004, Zhang et al., 2010]), *tabu* pretragom (npr. [Di Gaspero and Schaerf, 2001, Alvarez-Valdes et al., 2002, Aladag et al., 2009]), mravljim algoritmima (npr. [Socha et al., 2002, 2003, Dowsland and Thompson, 2005, Eley, 2007]), algoritmom harmonijske pretrage (npr. [Al-Betar et al., 2010, Geem, 2010]), genetskim algoritmom (npr. [Burke et al.,

1994, Erben and Keppler, 1996, Wilke et al., 2002, Pillay and Banzhaf, 2010]), algoritmom roja čestica [Shiau, 2011] i drugima. Usporedba više metaheurističkih algoritama na problemu izrade rasporeda provjera znanja dana je u [Azimi, 2004].

U novije vrijeme mogu se primijetiti još dva pravca u razvoju algoritama za rješavanja problema raspoređivanja: jedno su algoritmi koji se temelje na prilagodbi prethodnih prihvaćenih rješenja (engl. *case-based reasoning approaches*) gdje je ideja koristiti veliku bazu prethodnih dobrih rješenja i temeljem nje odabrati kao početno rješenje rješenje onog problema koji najsličniji problemu koji treba riješiti pa krenuti od tuda; drugi pravac je razvoj hiperheuristika.

Hiperheuristike (engl. *hyper-heuristics*) su algoritmi koji rade na višoj razini i koje odabiru prikladne heuristike na nižoj razini koje će se koristiti za rješavanje konkretnog problema [Denzinger et al., 1997, P. and E., 2000, Burke et al., 2003a,b]. Za razliku od heuristika i metaheuristika koje pretražuju prostor rješenja problema, hiper-heuristike pretražuju i optimiraju prostor heuristika. Također možemo reći da, prema [Burke et al., 2003a], hiperheuristikama možemo smatrati heuristike koje odabiru prikladne heuristike ili kao algoritme pretraživanja koji istražuju prostor algoritama za rješavanje problema. Neki primjeri hiperheuristika primijenjenih na problem rješavanja rasporeda su [Burke et al., 2007, Bader-El-Den et al., 2009].

U nastavku ovog poglavlja opisani su najčešći problemi raspoređivanja obrađivani u literaturi. Prema [Schaerf, 1999], radi se o problemu izrade rasporeda predavanja u osnovnim i srednjim školama (engl. *school timetabling*), o problemu izrade rasporeda predavanja na sveučilištima (engl. *course timetabling*) te o problemu izrade rasporeda provjera znanja (engl. *examination timetabling*).

2.1 Problem izrade rasporeda predavanja u školama

Problem izrade rasporeda predavanja u školama podrazumijeva izradu tjednog rasporeda za sve razrede u školi, pri čemu treba izbjeći situaciju u kojoj isti učitelj treba biti u istom trenutku u dva ili više razreda, te situaciju u kojoj u jednom razredu istovremeno treba biti više učitelja [Schaerf, 1999]. Formalno, problem je definiran na sljedeći način [Junginger, 1986, Schaerf, 1999].

Potrebno je pronaći x_{ijk}, pri čemu su $(i = 1 \dots m; j = 1 \dots n; k = 1 \dots p)$, tako da vrijedi:

$$\sum_{k=1}^{p} x_{ijk} = r_{ij} \qquad (i = 1 \ldots m; j = 1 \ldots n) \tag{2.1}$$

$$\sum_{j=1}^{n} x_{ijk} \leq t_{ik} \qquad (i = 1 \ldots m; k = 1 \ldots p) \tag{2.2}$$

$$\sum_{i=1}^{m} x_{ijk} \leq c_{jk} \qquad (j = 1 \ldots n; k = 1 \ldots p) \tag{2.3}$$

$$x_{ijk} = 0 \text{ ili } 1 \qquad (i = 1 \ldots m; j = 1 \ldots n; k = 1 \ldots p) \tag{2.4}$$

pri čemu je $x_{ijk} = 1$ ako razred c_i ima dodijeljenog učitelja t_j u terminu k, a 0 inače. U izrazima (2.1) do (2.4) m je broj razreda, n je broj učitelja a p je broj termina.

Ograničenje definirano izrazom (2.1) osigurava da svaki učitelj svakom razredu održi zadani broj predavanja (r_{ij} je broj predavanja koji razredu c_i treba održati učitelj t_j). Vrijednost r_{ij} bit će za učitelja j različita od nule samo za one razrede i u kojima taj učitelj treba održati predavanja; za ostale slučajeve vrijednost će biti nula. Ograničenje definirano izrazom (2.2) osigurava da je svaki učitelj u nekom terminu u najviše jednom razredu. Varijable t_{ik} pri tome mogu poprimiti ili vrijednost 1 ako učitelj t_i može predavati u terminu k ili 0 ako je u terminu k učitelj t_i nedostupan. Ograničenje definirano izrazom (2.3) osigurava da svaki razred u nekom terminu ima najviše jedno predavanje (tj. jednog učitelja). Varijable c_{jk} pri tome mogu poprimiti ili vrijednost 1 ako razred c_j može imati predavanje u terminu k ili 0 ako je u terminu k razred c_j nedostupan.

Jednostavnim proširenjem ovog modela moguće je definirati i unaprijed zadana predavanja. Unaprijed zadana predavanja podrazumijevaju da je prije procesa raspoređivanja već definiran djelomični raspored (za neke se učitelje točno zada u kojem terminu drže predavanje kojem razredu). Kako bi se ovo omogućilo, dovoljno je dodati ograničenja (2.5):

$$x_{ijk} \geq p_{ijk} \qquad (i = 1 \ldots m; j = 1 \ldots n; k = 1 \ldots p) \tag{2.5}$$

pri čemu je $p_{ijk} = 1$ ako razred c_i treba imati učitelja t_j u terminu k, a inače je 0.

Ova problem je $\mathcal{NP}$-potpun, i to su dokazali Even i suradnici [Even et al., 1976] svođenjem prethodno poznati $\mathcal{NP}$-potpuni problem 3-SAT [Garey and Johnson, 1979].

Ovako definiran problem niti na koji način ne uzima u obzir kvalitetu dobivenog rasporeda, u smislu potencijalne raspršenosti predavanja jednog razreda kroz čitav dan, u smislu mogućnosti da se definiraju više i manje poželjni termini za neka predavanja i slično. Naime, uobičajen je slučaj da postoji više rješenja koji zadovoljavaju sva nametnuta tvrda ograničenja (a u opisanom slučaju to bi bila sva ograničenja od (2.1) do (2.5)). Među svim tim rješenjima konačni je cilj odabrati (odnosno pronaći) upravo ono koje je najkvalitetnije po ostalim poželjnim karakteristikama. Primjerice, ako je potrebno definirati da postoje termini u kojima nije poželjno da razred ima neko predavanja, [Junginger, 1986] predlaže dodavanje sljedećeg ograničenja:

$$min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{p} d_{ijk} \cdot x_{ijk} \tag{2.6}$$

pri čemu je d_{ijk} neki veliki broj (kazna) ako nije poželjno da u terminu k razred c_i ima učitelja t_j, a inače je 0. Kako izraz (2.6) zahtjeva minimizaciju ukupne sume, u idealnom slučaju će za svaki i, j, k za koji je $d_{ijk} > 0$ odgovarajući x_{ijk} biti 0, ili ako to nije moguće, takvih će biti minimalan broj.

Današnja je situacija u školama daleko složenija u odnosu na ovaj jednostavan (a već $\mathcal{NP}$-potpun) problem: učitelj može predavati jedan ili više predmet, više razreda može odjednom slušati neki predmet kod određenog nastavnika, postoji problem s raspoloživim dvorama te kabinetnom nastavom gdje se nastava iz pojedinih predmeta održava isključivo u kabinetima tog predmeta; postoji mogućnost dijeljenja razreda u nekim terminima (primjer u Hrvatskoj su razredi u kojima dio učenika sluša Etiku a dio Vjeronauk pa se u vrijeme održavanja tog predmeta razred dijeli i jedan dio ili oba odlaze u druge prostorije, koje tada moraju biti slobodne, i slično).

Osim navedenih nedostataka, spomenimo još jedan – pretpostavka s kojom se je krenulo je da su razredi unaprijed definirani, pa njih treba razmjestiti. Iako je u tipičnim osnovnim i srednjim školama ovo istina za drugi i više razrede, činjenica je da upisom u školu prvi razredi unaprijed nisu ničim zadani. Škole ovo najčešće rješavaju tako da nekom nasumičnom metodom (ili po poznanstvima ili iskazanim željama tko bi htio biti s kime u razredu) utvrde grupe i potom se kreće u postupak izrade rasporeda u nadi da će se isti uspjeti generirati. Sa stanovišta optimizacijskog procesa koji se pri tome koristi, jasno je da bi bilo bolje dopustiti optimalno stvaranje grupa uz koje će raspored biti mogući i što je moguće kvalitetniji.

Drugo što također treba istaknuti je da su svi ovi problemi inherentno problemi *višekriterijske optimizacije* [Deb, 2009]; naime, kada postavljamo zahtjeve za kvalitetom rasporeda, tih zahtjeva tipično ima više: primjerice, da svaki pojedini nastavnik nema previše raspršen raspored (ovih zahtjeva ima, dakle, koliko i nastavnika), da svaki razred ima lijepo kontinuirani raspored (zahtjeva ima koliko i razreda) i sl. Međutim, gotovo se nikada ne pristupa rješavanju sa stanovišta višekriterijske optimizacije. Umjesto toga, problem se svodi na jednokriterijsku optimizaciju uvođenjem jedne težinske sume zadanih kriterija. Iako čest, ovaj pristup ima svojih nedostataka, a možda je najvažniji nedostatak kontrole nad iznosom najgoreg prekršenog ograničenja; primjerice, moguća je situacija u kojoj dva razreda imaju srednje raspršene rasporede, ili pak situacija u kojoj jedan razred uopće nema raspršenja a drugi ima ogromno raspršenje; u težinskoj sumi ova oba rasporeda bit će jednako dobra iako je jasno da to nije slučaj.

U svom preglednom radu, [Schaerf, 1999] navodi niz tehnika koje se koriste za rješavanje problema rasporeda predavanja u školama: direktna primjena jednostavnih heurističkih postupaka, svođenje na problem bojanja grafa, tehnike toka kroz mrežu, genetski algoritmi, simulirano kaljenje, logičko programiranje (npr. jezik Prolog), tehnike koje direktno rješavaju probleme zadovoljavanja ograničenja, tabu pretraga te kombiniranje više heuristika.

2.2 Problem izrade rasporeda predavanja na sveučilištu

Problem izrade rasporeda predavanja na sveučilištima (engl. *university course timetabling*) u određenoj je mjeri sličan problemu izrade rasporeda u školi [Schaerf, 1999]; temeljna je razlika što su razredi u školama disjunktni s obzirom na učenike (učenik pripada samo jednom razredu), dok kolegiji na sveučilištu u općem slučaju nisu disjunktni. Sljedeću razliku čine uska grla: dok u tipičnom rasporedu predavanja škole razred ima svoju dvoranu pa uska grla predstavljaju učitelji, sveučilišni raspored obiluje mnoštvom kolegija koje treba razmjestiti u bitno manji skup dvorana koje su pri tome još i različitih veličina, pa je tu usko grlo dostupnost odgovarajuće velike dvorane.

Pojednostavljeno, možemo reći da je problem izrade rasporeda predavanja na sveučilištu problem raspoređivanja kolegija u ograničen broj slobodnih prostorija te u ograničenom vremenskom razdoblju. Pri tome treba paziti da predavanja dva kolegija koji dijele studente ne budu zakazana u istom terminu, jer tada dijeljeni studenti neće moći

biti na oba predavanja.

Osnovni formalni model [de Werra, 1985] zahtjeva pronalazak takvih y_{ik}, gdje je $(i = 1 \ldots q; k = 1 \ldots p)$, za koje su zadovoljena sljedeća ograničenja:

$$\sum_{k=1}^{p} y_{ik} = k_i \qquad (i = 1 \ldots q) \tag{2.7}$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \qquad (k = 1 \ldots p) \tag{2.8}$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \qquad (l = 1 \ldots r; k = 1 \ldots p) \tag{2.9}$$

$$y_{ik} = 0 \text{ ili } 1 \qquad (i = 1 \ldots q; k = 1 \ldots p) \tag{2.10}$$

Pri tome su kolegiji $K_1, \ldots K_q$, a broj predavanja koje kolegij K_i treba održati je k_i. Kolegiji koji dijele studente razvrstani su u r grupa $S_1, \ldots, S_r$; pri čemu se u konkretnoj grupi S_i nalazi jedan ili više kolegija od kojih niti jedna dva ne smiju imati predavanja u istom terminu (drugim riječima, sve kolegije iz grupe S_i treba zakazati u različite termine). Ukupni broj termina je p a l_k je najveći broj kolegija koji se može zakazati u terminu k (primjerice, jer ima toliko slobodnih dvorana).

Varijabla $y_{ik} = 1$ ako se predavanje kolegija K_i održava u terminu k, a $y_{ik} = 0$ inače; ovih varijabli (a time i ograničenja (2.10)) ima ukupno $q \cdot p$. Ovaj model ne uzima u obzir broj studenata koji su upisali pojedini kolegij, veličine dostupnih dvorana, činjenicu da jedan nastavnik može držati predavanja na više kolegija (pa time opet ti kolegiji ne smiju ići istovremeno jer nastavnik ne može biti na dva mjesta odjednom), činjenicu da predavanja različitih predmeta mogu trajati različito dugo, činjenicu da pojedini predmeti moraju ići na propisani način (primjerice, blok od 2 sata jedan dan i blok od 3 sata neki drugi dan) i slično.

Ograničenja (2.7) osiguravaju da će svaki kolegij održati potreban broj predavanja (ima ih ukupno q). Ograničenja (2.8) osiguravaju da u niti jednom terminu neće biti zauzeto više prostorija no što ih je dostupno (ima ih ukupno p). Konačno, ograničenja (2.9) osiguravaju da niti jedna dva konfliktna kolegija nemaju zakazana predavanja u isto vrijeme (ima ih ukupno $r \cdot p$).

Definirani skup ograničenja je takav da u valjanom rasporedu sva ograničenja moraju biti zadovoljena – radi se o skupu tvrdih ograničenja. Ovaj problem pripada razredu

$\mathcal{NP}$-potpunih problema.

Jedno od predloženih mekih ograničenja koje je predložio [de Werra, 1985] je dodatno maksimizirati:

$$max \sum_{i=1}^{q} \sum_{k=1}^{p} d_{ik} \cdot y_{ik} \tag{2.11}$$

gdje je d_{ik} poželjnost da kolegij K_i ima predavanje u terminu k. Domena od d_{ik} pri tome nije ograničena. Primjerice, može se uzeti $d_{ik} \in [-1, 1]$ uz dogovor da $d_{ik} = -1$ znači da je takvo predavanje jako nepoželjno a da $d_{ik} = 1$ znači da je takvo predavanje jako poželjno.

Umjesto uporabe skupa konfliktnih kolegija, [Tripathy, 1992] koristi matricu konflikata $C_{q \times q}$, gdje c_{ij} predstavlja broj studenata koje dijele kolegiji K_i i K_j, odnosno kaznu smještaja ta dva kolegija u isti termin. Potom rješavanje ovog problema promatra kao optimizacijski problem smanjivanja ukupne kazne.

Varijanti ovog problema ima mnoštvo; tako [Schaerf, 1999] navodi nedostupnost nastavnika u nekom terminu, prethodna fiksno zakazana predavanja, više grupa za predavanja na pojedinim velikim kolegijima i slično. Na kolegijima koji predavanja imaju za više grupa (pa svaka grupa ima predavanja u različitim terminima) postoji problem kako rasporediti studente po tim grupama tako da ukupni raspored bude moguć. Ovaj je problem poznat kao *podproblem grupa* (engl. *grouping subproblem, student sectioning*).

Načina rješavanja ovog problema također ima mnogo. Primjerice, [Schaerf, 1999] navodi redukciju na problem bojanja grafova pa primjenu odgovarajućih algoritama, cjelobrojno linearno programiranje, tehnike mrežnog toka, tabu pretragu, pristupe temeljene na pravilima, pristupe temeljene na programiranju logike ograničenja (engl. *constraint logic programming*), genetske algoritme, logičko programiranje (uporabom jezika Prolog), simulirano kaljenje i druge.

Grupa autora [Rossi-Doria et al., 2003] daje drugačiju definiciju problema izrade rasporeda predavanja na sveučilištu: predavanja su modelirana skupom događaja E koje treba rasporediti u 5 dana gdje svaki dan ima 9 termina, što je ukupno 45 termina; definira se skup dvorana R u kojima se mogu odvijati predavanja, skup studenata S koji pohađaju predavanja, te skup značajki F prisutnih u dvoranama a koje predavanja pojedinih kolegija mogu tražiti. Svaki student pohađa jedno ili više predavanja a dvorane imaju dodatno i definirane kapacitete. Raspored je valjan ako su sva preda-

vanja smještena u termine i prostorije i ako su pri tome zadovoljena sljedeća čvrsta ograničenja:

- niti jedan student ne pohađa više od jednog predavanja u jednom trenutku,
- dodijeljene dvorane su dovoljno velike za sve studente koji pohađaju predavanje i imaju sve značajke zahtjevane od tog kolegija, te
- u nekom trenutku je u dvorani uvijek zakazano najviše jedno predavanje.

Uz ove, Rossi-Doria i suradnici definiraju i meka ograničenja kojima penaliziraju sljedeće situacije:

- student ima predavanje u zadnjem terminu u danu,
- student ima više od dva predavanja za redom, te
- student u danu ima samo jedno predavanje.

Temeljem ovog opisa autori u [Rossi-Doria et al., 2003] nadalje definiraju niz konkretnih primjera problema te uspoređuju različite metaheurističke metode primjenjene na te probleme. Pri tome koriste odnosno uspoređuju genetski algoritam, algoritam mravlje kolonije, iterativnu lokalnu pretragu [Lourenço et al., 2003], simulirano kaljenje te tabu pretragu. Rezultati usporedbe pokazuju da na manjim problemima najbolja rješenja daje algoritam iterativne lokalne pretrage nakon čega odmah slijede algoritmi simuliranog kaljenja te algoritam mravlje kolonije. Genetski algoritam na takvim primjerima daje loša rješenja, a tabu pretraga još lošija. Na srednje složenim problemima iterativno pretraživanje i dalje daje dobra rješenja dok genetski algoritam i tabu pretraga daju loša rješenja. Zanimljivost je da algoritam mravlje kolonije na ovim primjerima daje najlošija rješenja u odnosu na sve druge ispitane algoritme. Konačno, na teškim primjerima većina metaheuristika ima problema uopće pronaći valjana rješenja. Tako primjerice simulirano kaljenje nikada ne uspjeva pronaći valjana rješenja dok su drugi algoritmi nešto uspješniji.

Rad autora [McCollum, 2006] također sadrži pregled problema koji se javljaju pri izradi rasporeda predavanja kao i niz smjernica.

Osim navedenih pristupa, u literaturi se mogu pronaći i različiti drugi pristupi; primjerice, pristupi koji kombiniraju izradu rasporeda u proces koji se provodi više faza.

Primjer je rad [Kostuch, 2005], u kojem autori rješavaju upravo probleme iz [Rossi-Doria et al., 2003] pristupom koji radi u tri faze, i postiže zadovoljavajuće rezultate. U prvoj fazi algoritam pokušava konstruirati bilo kakvo valjano rješenje koristeći samo 40 od 45 raspoloživih termina (ne uzima zadnje termine u danu); tek ako to nije moguće, otvaraju se jedan po jedan preostalih 5 termina čiju uporabu meka ograničenja kažnjavaju. Za ovu fazu autori koriste algoritam bojanja grafova te algoritam maksimalnog podudaranja (engl. *maximum matching*). U drugoj fazi primjenjuje se simulirano kaljenje kako bi se popravila kvaliteta načinjenog rasporeda utvrđivanjem optimalnog redosljeda termina. Konačno, u trećem koraku, nakon što je prethodno utvrđen redosljed termina, rade se zamjene pojedinačnih predavanja po terminima, također uporabom simuliranog kaljenja.

Također zanimljiv pristup opisan je u radu [Chiarandini et al., 2006], gdje autori ciljano razvijaju kombinaciju algoritama koja radi dobro na konkretnom skupu problema koji rješavaju: probleme s međunarodnog natjecanja u izradi rasporeda predavanja[1]. Pri tome se isprobava velika količina različitih kombinacija algoritama; kombinacije algoritama koje ne rade dobro na tim problemima se odbacuju, preostale kombinacije algoritama se dodatno podešavaju i postupak eliminacije se ponavlja.

Konačni algoritam koji autori predlažu i koji je poslan na natjecanje je hibridni algoritam koji se temelji na uporabi konstrukcijskih heuristika i metaheuristika te koji u nekim slučajevima koristi brze lokalne metode smještanja predavanja u pojedine termine. Algoritam pri tome posebno rješava čvrsta ograničenja a posebno meka ograničenja. Za rješavanje čvrstih ograničenja algoritam koristi lokalnu pretragu i tabu pretragu. Kvaliteta tako pronađenih rješenja (dakle, meka ograničenja) popravlja se uporabom simuliranog kaljenja te spustom varijabilnim susjedstvom (engl. *Variable Neighbourhood Descent*). pretraživanja susjedstva varijabilne strukture. Spust varijabilnim susjedstvom je tehnika kod koje se algoritam lokalne pretrage primjenjuje slijedno na različite vrste susjedstva; primjena se ponavlja tako dugo dok se dobivaju poboljšanja uporabom bilo koje strukture susjedstva.

Temeljem rezultata eksperimenata autori su došli i do vrlo interesantnih zaključaka. Tako su primjerice genetski algoritam te algoritam mravlje kolonije iz razmatranja

[1] *The International Timetabling Competition* sponzorirano od međunarodne konferencije PATAT; detalji dostupni na web stranici `http://www.idsia.ch/Files/ttcomp2002/`, 2003.; posjećeno 18. ožujka 2011.

izbačeni vrlo rano; bolje ponašanje ovih metaheuristika dobiveno je tek uz dodavanje lokalne pretrage, tako da je umjesto čiste metaheuristike poželjna uporaba hibridnih verzija. Kod algoritama lokalne pretrage, pokazalo se je da su bolje one lokalne pretrage koje dozvoljavaju i modifikacije koje ne poboljšavaju niti ne pogoršavaju kvalitetu rješenja (drugim riječima, promjene u rješenju koje ne mjenjaju kvalitetu) u odnosu na algoritme lokalne pretrage koji isključivo prihvaćaju promjene koje poboljšavaju kvalitetu. Interesantno, ova spoznaja također je prirodno uključena u algoritmu diferencijske evolucije, o čemu će biti riječi kasnije u poglavlju 3.3. Autori dalje zaključuju kako na problemima izrade rasporeda uporaba varijabilnog susjedstva bitno unaprjeđuje algoritme lokalne pretrage i poboljšava rezultate. Sljedeći zaključak je da se više isplati zasebno rješavati čvrsta ograničenja a tek potom meka ograničenja (nasuprot pristupima koji uzimaju težinsku sumu oba pa rade odjednom optimizaciju i jednih i drugih). Također, autori zaključuju da se jednom postignuta valjanost rješenja u svakom slučaju treba očuvati pa kod operatora koji je mogu narušiti treba poduzeti odgovarajuće mjere kako bi se povratila valjanost (primjerice dodatnom lokalnom pretragom). Sljedeći zaključak je da tabu pretraga nije dobar algoritam za popravljanje mekih ograničenja[2] te da populacijski algoritmi ne pokazuju prednost u odnosu na algoritme koji rade s jednim rješenjem (s izuzetkom algoritma mravlje kolonije i jakom lokalnom pretragom). Konačno, još jedan od zanimljivih zaključaka je da se umjesto generiranja jednog početnog rješenja kod algoritama koji inače rade s jednim rješenjem ipak više isplati načiniti skup početnih rješenja primjenom različitih konstrukcijskih heuristika pa krenuti u postupak pretraživanja s najboljim rješenjem iz tog skupa. Za više detalja o opisanom pristupu zainteresirani se čitatelj upućuje na rad [Chiarandini et al., 2006].

2.3 Problem izrade raspored provjera znanja

Problem izrade raspored provjera znanja (engl. *Examination timetabling*) je problem kod kojeg se zahtjeva raspoređivanje ispita za skup kolegija (svaki kolegij ima jedan ispit) u ograničenom vremenskom periodu.

Prema [Schaerf, 1999], iako problem izrade rasporeda predavanja na sveučilištu i problem izrade rasporeda provjera znanja imaju nekih sličnosti, općeprihvaćene razlike

[2] napomena: zapažanje vrijedi za problem izrade rasporeda predavanja

su da za svaki kolegij postoji samo jedan ispit[3], uvjet izbjegavanja konflikata je čvrsto ograničenje, postoji niz različitih mekih ograničenja, u istoj prostoriji može biti više od jednog ispita i slično.

Formalni model najjednostavnije vrste problema ovog tipa dan je u nastavku. Potrebno je pronaći y_{ik} gdje su $(i = 1 \ldots q; k = 1 \ldots p)$ takve da vrijede sljedeća ograničenja:

$$\sum_{k=1}^{p} y_{ik} = 1 \qquad (i = 1 \ldots q) \tag{2.12}$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \qquad (k = 1 \ldots p) \tag{2.13}$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \qquad (l = 1 \ldots r; k = 1 \ldots p) \tag{2.14}$$

$$y_{ik} = 0 \text{ ili } 1 \qquad (i = 1 \ldots q; k = 1 \ldots p) \tag{2.15}$$

pri čemu $K_1, \ldots, K_q$ predstavlja q kolegija, odnosno njihove ispite. Slično kao kod rasporeda predavanja na sveučilištu, za svaki predmet se definira još i skup svih predmeta koji s njime dijele studente; uz zanemarenje jednočlanih skupova, takvih onda ima r i označit ćemo ih $S_1, \ldots, S_r$. Tako primjerice svi kolegiji koji su u skupu S_l moraju imati zakazan ispit u različitim terminima.

Varijabla $y_{ik} = 1$ ako se ispit kolegija K_i održava u terminu k, a $y_{ik} = 0$ inače; ovih varijabli (a time i ograničenja (2.15)) ima ukupno $q \cdot p$. Ovaj model ne uzima u obzir broj studenata koji pišu ispit pojedinih kolegija, veličine dostupnih dvorana, ograničenje na broj nastavnog osoblja dostupnog za čuvanje provjera, činjenicu da ispiti različitih predmeta mogu trajati različito dugo i slično.

Ograničenja (2.12) osiguravaju da će svaki kolegij imati dodijeljen točno jedan termin za ispit (ovih ograničenja ima ukupno q). Ograničenja (2.13) osiguravaju da u niti jednom terminu neće biti zakazano više od dozvoljenog broja ispita (ovih ograničenja ima ukupno p). Konačno, ograničenja (2.14) osiguravaju da niti jedna dva konfliktna kolegija nemaju zakazana provjeru znanja u isto vrijeme (ovih ograničenja ima ukupno $r \cdot p$).

Definirani skup ograničenja je takav da u valjanom rasporedu sva ograničenja moraju

[3] postoje situacije u kojima ovo ne vrijedi – primjer je dan u [Wang et al., 2010].

biti zadovoljena – dakle, radi se o skupu tvrdih ograničenja. Ovaj problem pripada razredu $\mathcal{NP}$-potpunih problema.

Jedno od mekih ograničenja koje se često koristi je dodatno minimizirati:

$$min \sum_{k=1}^{p-1} \sum_{l=1}^{r} \sum_{i,j \in S_l} y_{ik} \cdot y_{jk+1} \tag{2.16}$$

čime se zapravo minimizira broj pojava ispita dvaju kolegija koji dijele barem jednog studenta i koji su raspoređeni u dva susjedna termina. Modifikacije ovog ograničenja su brojati broj uključenih studenata (a ne samo broj pojava takvih situacija), zatim broj puta da se slično dogodi samo na većem razmaku u vremenu (dakle, uz provjeru termina k i $k+1$ gledati i parove k i $k+2$ i slično).

U preglednom radu [Schaerf, 1999] navodi se i niz varijanti ovog problema: nemogućnost održavanja ispita u pojedinim terminima, unaprijed zadani termini za neke ispite, izbjegavanje ispita prije i poslije ručka, izbjegavanje situacija u kojima student ima x ispita u y uzastopnih termina [Carter et al., 1994] i slično. Primjerice, pristup opisan u radu [Corne et al., 1993] kažnjava sljedeće situacije: student ima više od dva ispita u jednom danu, student ima dva ispita u uzastopnim terminima te student ima jedan ispit prije ručka a drugi poslije. U radu [Burke and Petrovic, 2002] dana je sistematizacija od 9 ograničenja koja se koriste za izradu rasporeda provjera znanja, a koja uključuju dostupne dvorane i njihove kapacitete, blizinu pojedinih ispita te poredak ispita. Većina radova danas se fokusira upravo na ovakva meka ograničenja, iako ima i iznimki [Asmuni et al., 2007, Petrovic et al., 2005]. U radu [McCollum, 2006] su opisani tipični koraci pri rješavanju problema izrade rasporeda provjera znanja kao i naputak kako modelirati rješenja.

Od tehnika koje se koriste za rješavanje ovog problema, [Schaerf, 1999] navodi direktnu primjenu heurističkih postupaka, redukciju na problem bojanja grafova, uporabu simuliranog kaljenja, uporabu genetskih algoritama kao i druge algoritme.

Poglavlje 3

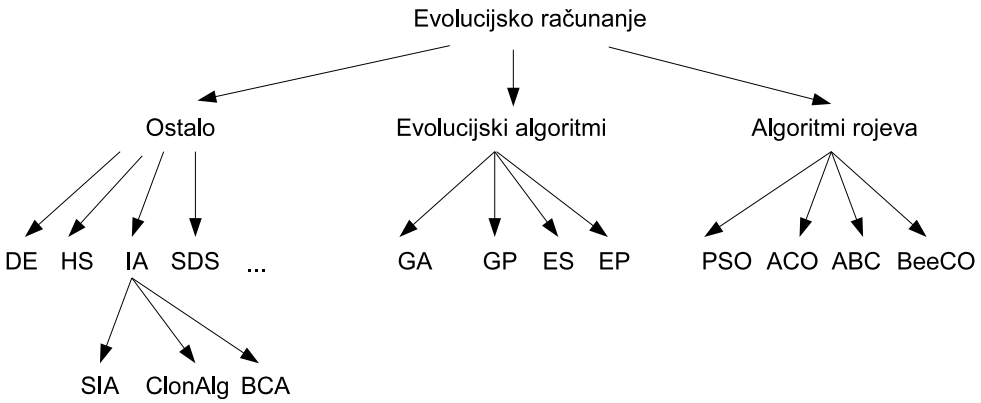
Algoritmi evolucijskog računanja

Evolucijsko računanje je grana umjetne inteligencije koja se, najvećim dijelom, bavi rješavanjem optimizacijskih problema. To je područje u razvoju već više od pola stoljeća: začetci sežu u kasne 50.-e [Back et al., 1997]. Evolucijsko računanje danas obuhvaća bogat skup raznorodnih algoritama od kojih je nekoliko istaknutijih opisano u ovom poglavlju. Evolucijsko računanje može se podijeliti u tri grane: *evolucijske algoritme*, *algoritme rojeva* te *ostale algoritme*. Ova podjela kao i odabrani algoritmi prikazani su na slici 3.1.

Područje evolucijskog računanja ujedno pripada i u područje metaheuristika [Michalewicz and Fogel, 2004, Talbi, 2009, Yang, 2008] – heuristika namijenjenih vođenju problemski specifičnijih heuristika u pokušaju da se pronađe potprostor u prostoru pretraživanja koji sadrži dobra rješenja.

U evolucijske algoritme danas uobičajeno ubrajamo evolucijske strategije, evolucijsko programiranje, genetski algoritam te genetsko programiranje. Utjecaj na ovo područje imali su radovi [Bremermann, 1962, Friedberg, 1958, Friedberg et al., 1959, Box, 1957], dok su temelj postavili radovi [Holland, 1962, Rechenberg, 1965, Schwefel, 1968, Fogel, 1962]. Danas o ovim algoritmima postoji obilje literature, a spomenut ćemo samo neke novije knjige: [Affenzeller et al., 2009, Deb, 2009, Sivanandam and Deppa, 2010, Langdon and Poli, 2010].

U algoritme rojeva ubrajamo *mravlje algoritme*, *algoritam roja čestica*, *algoritme pčela* i druge. Razvoj mravljih algoritama potaknut je eksperimentima biologa [Pasteels et al., 1987, Goss et al., 1989] o ponašanju mrava u prirodi. Popularizaciju te veliki poticaj za daljnja istraživanja mravlji algoritmi su doživjeli nakon objave rada [Colorni



Slika 3.1: Podjela evolucijskog računanja na glavne grane. Uz svaku granu prikazani su i odabrani algoritmi.

et al., 1991] te knjige [Dorigo and Stützle, 2004]. Opširni pregled mravljih algoritama može se pogledati u [Cordon et al., 2002]. Istaknutiji razvoj i primjena algoritma roja čestica počinje od sredine devedesetih, kada Eberhart i Kennedy definiraju sam algoritam [Kennedy and Eberhart, 1995] te knjigu [Kennedy et al., San Francisco, USA]. Od preglednih radova o algoritmu roja čestica valja istaknuti [Song and Gu, 2004, Banks et al., 2007, 2008]. Razvoj algoritama zasnovanih na pčelinjem ponašanju počinje 2004. godine objavom algoritma *Honey Bee Algorithm* [Nakrani and Tovey, 2004]; nakon toga se razvija algoritam virtualnih pčela [Yang, 2005], algoritam *Honey Bee Mating Optimization* [Haddad et al., 2006] te algoritam umjetne pčelinje kolonije [Karaboga and Basturk, 2007, 2008].

Granu *ostali algoritmi* čini još niz drugih algoritama koji ne pripadaju u prethodne dvije grane. Od značajnijih algoritama tu svakako pripadaju umjetni imunološki algoritmi, algoritam diferencijske evolucije te algoritam harmonijske pretrage. Podloga za definiranje umjetnih imunoloških sustava seže još u 1957. i čine je radovi Burneta [Burnet, 1957, 1959, 1978]. Najraširenije inačice algoritama, posebice onih temeljenih na principu klonske selekcije opisani su u [de Castro and Timmis, 2002, Cutello and Nicosia, 2005], a knjiga [Dasgupta and Niño, 2009] također nudi dobar uvod u područje. Algoritam diferencijske evolucije nastao je iz algoritma genetskog kaljenja (engl. *Genetic Annealing*) [Price, 1994]. Prva verzija algoritma opisana je u tehničkom izvještaju [Storn and Price, 1995], nakon čega su uslijedili i radovi [Storn and Price, 1996, Price and Storn, 1997, Storn and Price, 1997, Price, 1997, 1999]. Algoritam harmo-

nijske pretrage osmislili su Lee i Geem 2004. godine te su pokazali njegovu primjenu kako u optimizaciji funkcija kontinuiranih varijabli, tako i za rješavanje kombinatornih problema [Lee and Geem, 2004, 2005].

3.1 Genetski algoritam

L. J. Fogel, A. J. Owens i M. J. Walsh stvorili su 1966. evolucijsko programiranje [Fogel et al., 1966], I. Rechenberg 1973. i H.-P. Schwefel 1975. evolucijske strategije [Rechenberg, 1973, Schwefel, 1975], a H. J. Holland 1975. genetski algoritam [Holland, 1975]. Paralelno tome, tekao je i razvoj genetskog programiranja koji je 1992. popularizirao J. R. Koza [Koza, 1992]. Inspiracija za razvoj genetskog algoritma je Darwinova teorija o postanku vrsta [Darwin, 1859].

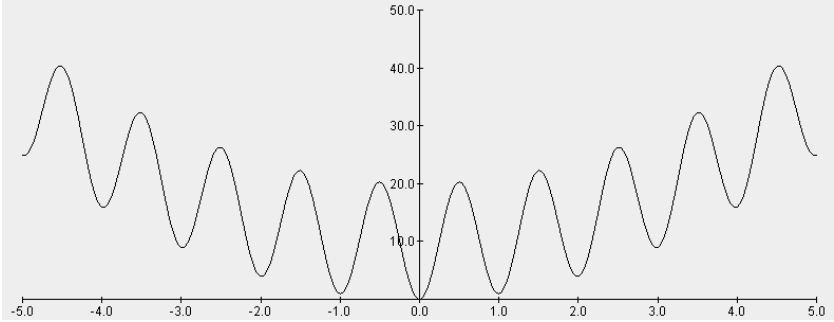
Darwin teoriju razvoja vrsta temelji na 5 postavki: (i) plodnost vrsta – potomaka uvijek ima više no što je potrebno, (ii) veličina populacije je približno stalna, (iii) količina hrane je ograničena, (iv) kod vrsta koje se seksualno razmnožavaju, nema identičnih jedinki već postoje varijacije, te (v) najveći dio varijacija prenosi se nasljeđem. Iz navedenoga slijedi da će u svakoj populaciji postojati borba za preživljavanje: ako potomaka ima više no što je potrebno a količina hrane je ograničena, sve jedinke neće preživjeti. Pri tome će one bolje i jače imati veću šansu da prežive i dobiju priliku dalje se razmnožavati. Prilikom razmnožavanja, djeca su u velikoj mjeri određena genetskim materijalom svojih roditelja, no nisu ista roditeljima – postoji određeno odstupanje.

Ova razmatranja osnova su za razvoj genetskog algoritma. Problemi koje rješavamo genetskim algoritmom tipično su optimizacijski problemi, koje možemo opisati na sljedeći način. Zadana je funkcija $f(\vec{x})$ gdje je $\vec{x} = (x_1, x_2, \ldots, x_n)$. Potrebno je pronaći $\vec{x}^*$ koji maksimizira (ili minimizira) funkciju f. Pogledajmo ovo na primjeru funkcije jedne varijable (slika 3.2) koja je definirana izrazom:

$$f(\vec{x}) = 10 + x^2 - 10 \cdot cos(2 \cdot \pi \cdot x). \tag{3.1}$$

Iz slike je jasno vidljivo da funkcija ima jedan globalni minimum za $x = 0$, $f(x) = 0$.

Kako genetski algoritam radi? Postoji puno različitih izvedbi, no generalna ideja je sljedeća. Algoritam koristi *populaciju* jedinki. Svaka jedinka je jedno moguće rješenje zadanog problema (u spomenutom slučaju, to će biti vrijednost varijable x). Jedinke



Slika 3.2: Višemodalna funkcija jedne varijable.

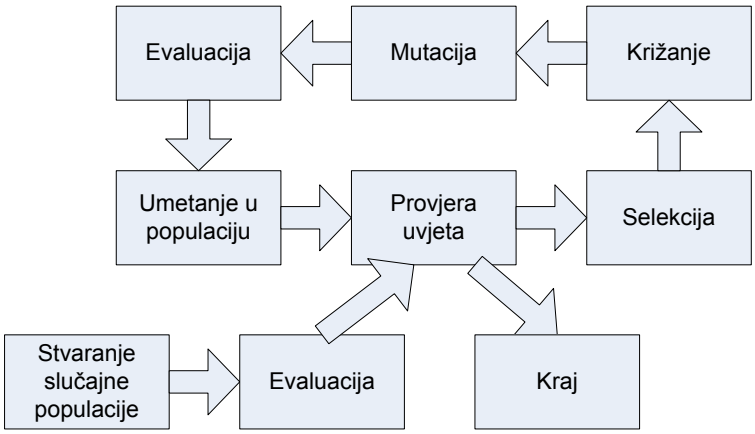
još nazivamo i *kromosomima*. Svakoj jedinki možemo odrediti njezinu *dobrotu* (engl. *fitness*). Dobrotu jedinke odredit ćemo tako da izračunamo vrijednost funkcije u toj točki (x); što je vrijednost funkcije veća, to je jedinka lošija (pretpostavka je da rješavamo problem minimizacije funkcije). Operatorom *selekcije* (engl. *selection*) biraju se iz populacije jedinke koje postaju roditelji. Roditelji pomoću operatora *križanja* (engl. *crossover*) stvaraju djecu, čime se emulira izmjena genetskog materijala (engl. *recombination*). Nad djecom potom djeluje operator *mutacije* (engl. *mutation*). Konačno, operatorom *zamjene* (engl. *reinsertion*) djeca ulaze u populaciju rješenja, čime se zatvara ciklus rada algoritma.

3.1.1 Vrste genetskog algoritma

Genetski algoritmi dijele se na *sekvencijske* i *paralelne*. Paralelni genetski algoritmi omogućavaju rad u višedretvenim i raspodijeljenim okruženjima i oni su obrađeni u kasnijim poglavljima. Sekvencijski genetski algoritam može se podijeliti na dvije tipične izvedbe [Alba and Dorronsoro, 2008]: *eliminacijski genetski algoritam* (engl. *steady-state genetic algorithm*) te *generacijski genetski algoritam* (engl. *generational genetic algorithm*). Ove obje izvedbe opisane su u nastavku.

3.1.1.1 Eliminacijski genetski algoritam

Ova vrsta genetskog algoritma prikazana je na slici 3.3. U svakom koraku (može se još koristiti i termin generacija) iz čitave se populacije odabiru dva roditelja nad kojima se izvodi križanje čime nastaje novo dijete. Dijete se potom mutira i ubacuje u populaciju. Kako veličina populacije mora biti stalna, ovo ubacivanje izvodi se tako da dijete



Slika 3.3: Eliminacijski genetski algoritam.

zamijeni neku jedinku iz populacije (primjerice, najgoru).

Selekcija svakog od roditelja može se obaviti nekim od operatora selekcije; primjerice, proporcionalnom selekcijom ili turnirskom selekcijom. Jednako tako, odabir jedinke koja će biti u populaciji biti zamijenjena nastalim djetetom također se može obaviti nekim od operatora selekcije, pri čemu se operator tako prilagodi da veću vjerojatnost odabira daje lošijim jedinkama. Izvedba ove vrste genetskog algoritma opisana je pseudokodom 3.4. Posebnost ove vrste genetskog algoritma je da nastalo dijete već u sljedećem trenutku može postati roditelj koji će se križati sa svojim pretcima i dalje stvarati potomke.

```
P = stvori_početnu_populaciju(VEL_POP)
evaluiraj(P)
ponavljaj_dok_nije_kraj
  odaberi R1 i R2 iz P
  D = križaj(R1, R2)
  mutiraj D
  evaluiraj D
  umetni D u P operatorom zamjene
kraj
```

Slika 3.4: Pseudokod eliminacijskog genetskog algoritma.

3.1.1.2 Generacijski genetski algoritam

Za razliku od prethodno opisane vrste kod koje ne postoji čista granica između roditelja i djece, generacijski genetski algoritam iz koraka u korak iz populacije roditelja stvara novu populaciju djece koja potom postaju roditelji – stara populacija roditelja time odmah izumire. Shematski prikaz ove vrste genetskog algoritma prikazan je na slici 3.5.

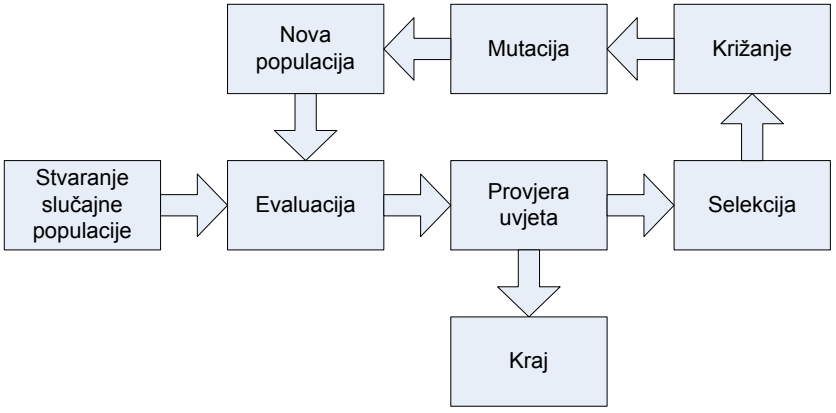
Generacijski genetski algoritam iz trenutne populacije operatorima selekcije, križanja i mutacije gradi novu pomoćnu populaciju sastavljenu isključivo iz djece (slika 3.6).

Onog trenutka kada se izgradi nova populacija čija je veličina jednaka staroj, stara populacija roditelja se odbacuje, a novoizgrađena populacija djece postaje trenutna populacija.

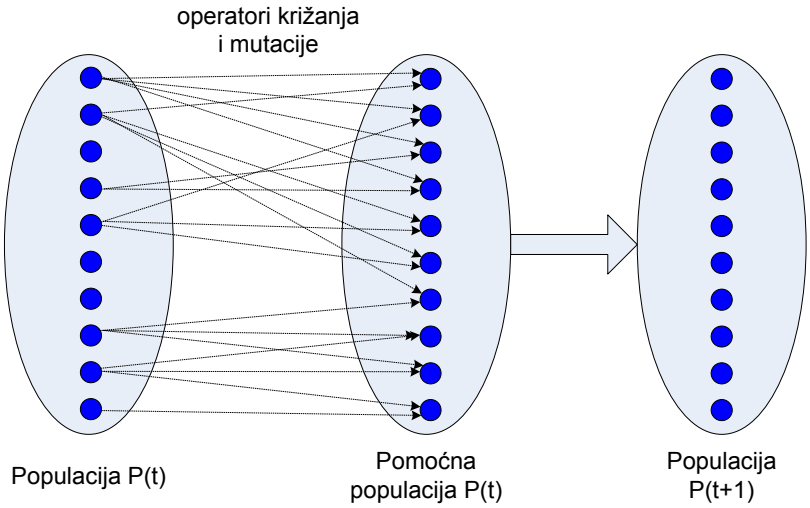
Izvedba ove vrste genetskog algoritma dana je pseudokodom 3.7.

Direktnom implementacijom prikazanog pseudokôda dobiva se algoritam koji nema elitizma (odnosno najbolja se jedinka ne čuva). Stoga je sasvim moguće da sva nastala djeca budu lošija od najboljeg roditelja pa dolazi do gubitka najboljeg pronađenog rješenja. Ovome se može doskočiti tako da se najprije u populaciju djece iskopira najbolji roditelj, a ostatak populacije potom izgradi na uobičajeni način.

Prilikom rješavanja problema genetskim algoritmom, potrebno je definirati način na koji kromosom kodira rješenje, način na koji se obavlja križanje, način na koji se obavlja mutacija te način na koji se odabiru jedinke za razmnožavanje.



Slika 3.5: Generacijski genetski algoritam.



Slika 3.6: Korak generacijskog genetskog algoritama.

```
P = stvori_početnu_populaciju(VEL_POP)
evaluiraj(P)
ponavljaj_dok_nije_kraj
  nova_populacija P' = ∅
  ponavljaj_dok_je veličina(P')<VEL_POP
    odaberi R1 i R2 iz P
    {D1, D2} = krizaj(R1, R2)
    mutiraj D1, mutiraj D2
    dodaj D1 i D2 u P'
  kraj
  P = P'
kraj
```

Slika 3.7: Pseudokod generacijskog genetskog algoritma.

3.1.2 Prikaz rješenja

Najjednostavniji način prikaza rješenja je nizom bitova (engl. *bit-string*). Svaki kromosom (tj. jedinka) sastoji se od jednakog i fiksnog broja bitova. Neka za potrebe ovog razmatranja broj bitova n bude 10. Prostor koji pretražujemo također je ograničen na područje $[x_{min}, x_{max}]$. Primjerice, neka je $[x_{min}, x_{max}] = [-5, 5]$. Potrebno je definirati način na koji se niz nula i jedinica iz kromosoma preslikava u rješenje, tj. način na koji se obavlja dekodiranje. Najjednostavniji način je niz nula i jedinica u kromosomu promatrati kao binarno zapisan cijeli broj. Kako je kromosom sastavljen od n bitova, cijeli brojevi koje on može predstavljati su brojevi od 0 do $2^n - 1$, što u slučaju da je

$n = 10$ znači od 0 do 1023. Najmanji broj (0) pri tome predstavlja x_{min}, najveći broj (1023) predstavlja x_{max}, a bilo koji broj između najmanjeg i najvećeg predstavlja neki broj između x_{min} i x_{max}. Najčešće se uzima linearno skaliranje, pa ako se vrijednost cijelog broja zapisanog u kromosomu označi s k, pripadna vrijednost rješenja dobiva se izrazom:

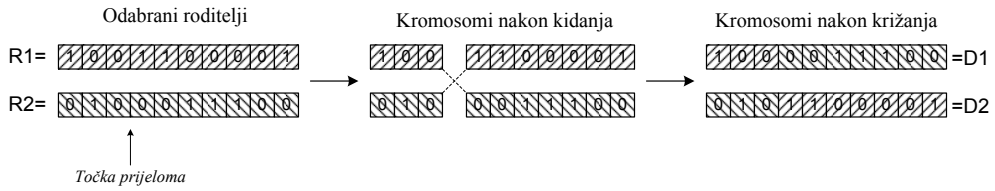
$$x = x_{min} + \frac{k}{2^n - 1} \cdot (x_{max} - x_{min}). \tag{3.2}$$

3.1.3 Operator križanja

Nakon što su odabrana dva roditelja, operatorom križanja stvaraju se njihovi potomci. Operator križanja često se opisuje parametrom p_c – vjerojatnošću da se križanje doista dogodi (uobičajene vrijednosti su između 60% i 90%). Križanje je moguće načiniti na više načina. U nastavku su dani primjeri koji provođenjem križanja uvijek daju dva djeteta.

3.1.3.1 Križanje s jednom točkom prekida

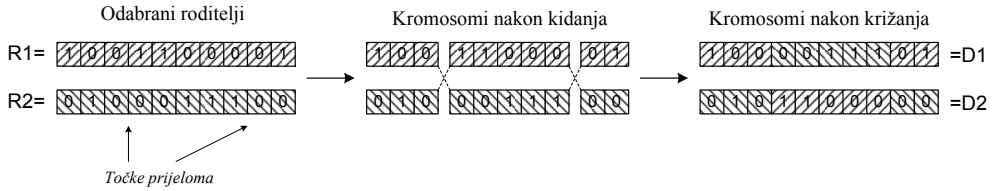
Kromosomi-roditelji poslože se jedan ispod drugoga (lijevi dio slike 3.8). Posredstvom slučajnog mehanizma odabere se točka kidanja oba kromosoma. Potom se stvaraju dva djeteta $D1$ i $D2$: prvo dobiva prvi dio kromosoma roditelja $R1$ i drugi dio kromosoma roditelja $R2$, a drugo dijete dobiva prvi dio kromosoma roditelja $R2$ i drugi dio kromosoma roditelja $R1$. Čitav postupak ilustriran je na slici 3.8.



Slika 3.8: Križanje s jednom točkom prijeloma.

3.1.3.2 Križanje s t-točaka prekida

Križanje s t-točaka prekida općenitiji je slučaj prethodno opisanog križanja, gdje t može biti između 1 i $k - 1$ (gdje je k broj bitova kromosoma). Primjerice, slučaj za $t = 2$, te s točkama prijeloma nakon 3. i 8. bita prikazan je na slici 3.9.



Slika 3.9: Križanje s t-točaka prijeloma.

3.1.3.3 Uniformno križanje

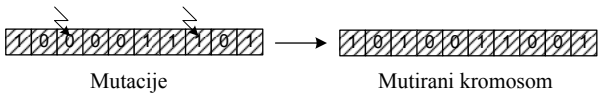
Uniformno križanje može se promatrati kao križanje s $k-1$ točkom prekida (gdje je k broj bitova kromosoma). Kromosomi oba roditelja raspadnu se nakon svakog bita i potom svako dijete bira što će uzeti od kojeg roditelja. Ovo se u računalnim programima najčešće izvodi na sljedeći način:

```
D1 = R1 · R2 + r · (R1 ⊕ R2)
D2 = R1 · R2 + !r · (R1 ⊕ R2)
```

pri čemu znak '$\cdot$' predstavlja operator *logičko-I* nad pojedinim bitovima, znak '$+$' *logičko-ILI* nad pojedinim bitovima, znak '$\oplus$' *logičko-isključivo-ILI* nad pojedinim bitovima, te uskličnih operator komplementa nad pojedinim bitovima. r je pri tome slučajno generirani niz od k bitova. Lako je uočiti da će logičko-I na mjestima gdje su bitovi u oba roditelja isti tu vrijednost prekopirati u dijete, a operator logičko-isključivo-ILI na onim mjestima na kojima su bitovi roditelja različiti odabrati slučajno generirani iz r (ili komplement od r kod djeteta 2) i njega prekopirati u dijete.

3.1.4 Operator mutacije

Operator mutacije nad binarnim kromosomima djeluje tako da posredstvom slučajnog mehanizma nekim bitovima promijeni vrijednost. Operator se najčešće opisuje vjerojatnošću promjene bita p_m, koja uobičajeno iznosi između 1% i 5% (a često je i manja). Djelovanje je prikazano na slici 3.10.



Slika 3.10: Djelovanje mutacije.

3.1.5 Operator selekcije

Selekcija je postupak kojim se odabiru jedinke iz populacije, najčešće u svrhu razmnožavanja. Najjednostavniji način selekcije – slučajni odabir jedinki iz populacije nije dobro rješenje; da bi genetski algoritam napredovao, veću šansu trebaju imati bolje jedinke. Stoga se za selekciju često koriste sljedeće tehnike: *proporcionalna selekcija* te *turnirska selekcija*.

Dobrota svake pojedine jedinke uobičajeno se procjenjuje "mjerenjem" funkcijom koju optimiramo. Primjerice, ako obavljamo maksimizaciju funkcije, česta je praksa dobrotu jedinke $\vec{x}_i$ poistovjetiti s iznosom funkcije u točki koju jedinka predstavlja:

$$\text{fitness}(i) = f(\vec{x}_i). \tag{3.3}$$

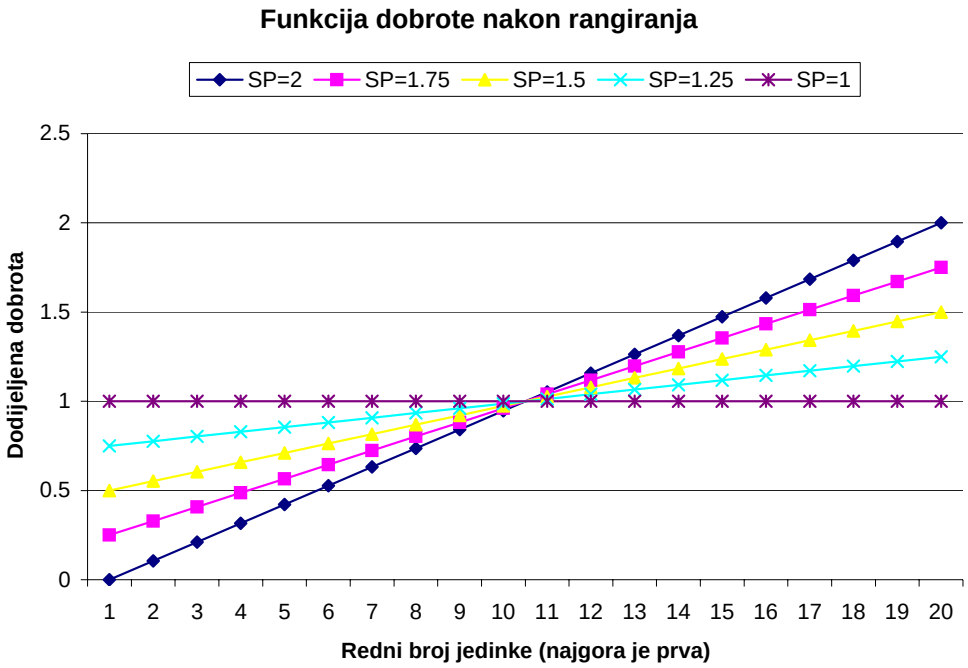
Međutim, često se pokazuje da je određivanje dobrote jedinke rangiranjem bolje rješenje. U tu svrhu, populacija se najprije sortira prema vrijednosti funkcije koja se optimira (dakle prema $f(\vec{x}_i)$). Najlošija jedinka (u slučaju maksimizacije, to je ona koja ima najmanji iznos $f(\vec{x}_i)$) pri tome dolazi na 1. mjesto, a najbolja na zadnje mjesto. i-toj se jedinki kao dobrota tada postavlja vrijednost definirana sljedećim izrazom:

$$\text{fitness}(i) = 2 - \text{SP} + 2 \cdot (\text{SP} - 1) \cdot \frac{i-1}{n-1} \tag{3.4}$$

gdje je SP parametar koji opisuje selekcijski pritisak, i može biti u intervalu $[1, 2]$, a n je veličina populacije (vidi sliku 3.11 za $n = 20$ i različite vrijednosti SP-a).

Uz SP = 1 svim se jedinkama, neovisno o njihovoj stvarnoj dobroti, pridjeljuje ista vrijednost dobrote. Ovo će rezultirati time da sve jedinke – i dobre i loše – imaju jednaku šansu biti odabrane, što neće rezultirati usmjerenim pretraživanjem prostora mogućih rješenja. Suprotna se situacija pojavljuje uz SP = 2: najgoroj jedinki pridjeljena je vrijednost 0, čime se ta jedinka nikada neće birati kao potencijalni roditelj, dok najbolja jedinka ima najveći iznos pridijeljene dobrote. Za vrijednosti SP-a između 1 i 2 omjer dodijeljene dobrote najboljem i najgorem rješenju $\text{fitness}(\vec{x}_{best})/\text{fitness}(\vec{x}_{worst})$ mijenja se od 1 (potpuno slučajna pretraga) do ∞ (izrazito usmjerena pretraga). Stoga se ovim parametrom može fino podešavati koliki naglasak algoritam stavlja na najbolje pronađeno rješenje.

Uz veliki selekcijski pritisak, algoritam će mnogo više vremena raditi s najboljom

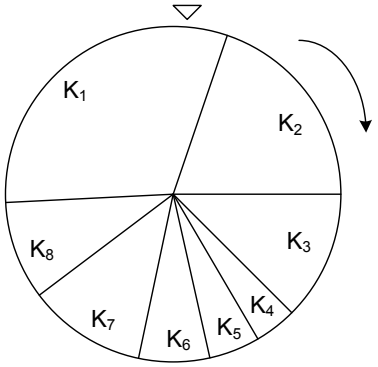


Slika 3.11: Ovisnost dobrote jedinke o parametru SP kod rangiranja. Primjer ilustrira slučaj $n = 20$ te nekoliko različitih vrijednosti parametra SP.

pronađenom jedinkom, čime će se u određenoj mjeri ponašati kao algoritam lokalne pretrage – radit će fino pretraživanje u okolici trenutno najboljeg rješenja. Problem koji se pri tome javlja je sljedeći – što ako to rješenje nije globalni optimum? U tom slučaju postoji mogućnost da će algoritam "zaglaviti" u lokalnom optimumu, pa govorimo o preranoj odnosno prebrzoj konvergenciji. Smanjivanjem selekcijskog pritiska i manje dobra rješenja imat će veću vjerojatnost sudjelovanja u generiranju potomstva. Time će se osigurati da algoritam obavlja šire pretraživanje prostora rješenja čime se povećava i vjerojatnost pronalaska još boljih rješenja a smanjuje vjerojatnost prerane konvergencije. Posljedica smanjenja selekcijskog pritiska je veća robusnost algoritma; no istovremeno povećava se i vrijeme potrebno da algoritam konvergira ka stvarnom globalnom optimumu – jednom kada ga pronađe.

3.1.5.1 Proporcionalna selekcija

Ova selekcija još je poznata pod nazivom *Roulette-wheel selection*. Ideja je sljedeća: sve se jedinke postave na kolo, tako da bolje jedinke imaju veću površinu na kolu (slika 3.12). Potom se kolo zavrti. Odabrana je ona jedinka na kojoj se kolo zaustavilo. S



Slika 3.12: Ilustracija proporcionalne selekcije za slučaj 8 jedinki čije su dobrote prikazane tablicom 3.1.

obzirom da jedinki pripada to veći dio dio oboda kola što ima veću dobrotu, ponavljamo li eksperiment puno puta, bolje će jedinke biti češće birane od lošijih, i ta će vjerojatnost biti upravo proporcionalna dobroti same jedinke.

Programski, ovo se može postići tako da se sve jedinke preslikaju na kontinuirani dio pravca duljine 1. Pri tome svaka jedinka dobiva dio proporcionalan njezinoj dobroti. Ako s $\text{fit}(i)$ označimo dobrotu i-te jedinke, a s $\text{len}(i)$ označimo duljinu segmenta koji pripada toj jedinki, vrijedi:

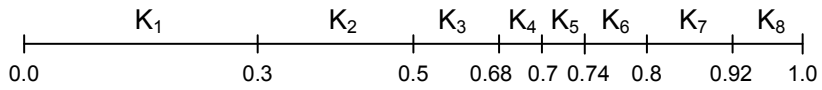
$$\text{len}(i) = \frac{\text{fit}(i)}{\sum_{j=1}^{n} \text{fit}(j)}. \tag{3.5}$$

Primjerice, neka se populacija sastoji od 8 jedinki čije su dobrote prikazane u tablici 3.1. Normirana duljina segmenta svake jedinke direktno odgovara i vjerojatnosti odabira te jedinke. Postavljenjem svih jedinki na segment pravca, dobiva se situacija prikazana na slici 3.13.

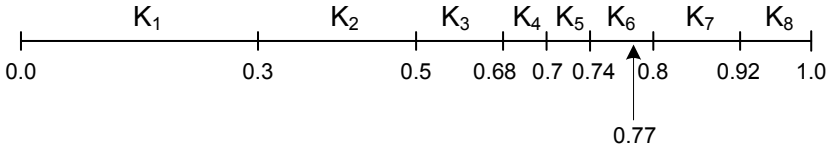
Jedinka se odabire tako tako da se generira slučajni broj iz intervala $[0, 1]$ i zatim pogleda u čije je područje taj broj upao. Primjerice, ako je generiran broj 0.77, odabrana

Tablica 3.1: Primjer razdiobe vjerojatnosti selekcije temeljem dobrote kod proporcionalne selekcije

Jedinka	1	2	3	4	5	6	7	8
Dobrota jedinke	6	4	3.6	0.4	0.8	1.2	2.4	1.6
Vjerojatnost odabira	0.3	0.2	0.18	0.02	0.04	0.06	0.12	0.08



Slika 3.13: Ilustracija proporcionalne selekcije jediničnim pravcem za slučaj 8 jedinki čije su dobrote prikazane tablicom 3.1.



Slika 3.14: Ilustracija proporcionalne selekcije jediničnim pravcem za slučaj 8 jedinki čije su dobrote prikazane tablicom 3.1; odabran je broj $0,77$ što odgovara jedinki K_6.

je jedinka $K6$ (slika 3.14).

Ovakva izvedba proporcionalne selekcije ima problem koj se javlja pri velikim iznosima funkcije dobrote. Ovo je ilustrirano na primjeru prikazanom u tablici 3.2.

Tablica 3.2: Problem skale kod proporcionalne selekcije

Jedinka	1	2	3	4
Dobrota jedinke	1007	1008	1005	1008
Vjerojatnost odabira	0.124768	0.124892	0.12452	0.124892

Jedinka	5	6	7	8
Dobrota jedinke	1003	1011	1017	1012
Vjerojatnost odabira	0.124272	0.125263	0.126007	0.125387

Radi se o primjeru maksimizacije funkcije pri čemu se za potrebe ovog primjera radi sa samo 8 jedinki, a funkcija je takva da joj je maksimum neki relativno veliki broj (primjerice $2 \cdot 10^5$). U nekom trenutku populacija se sastoji od jedinki prikazanih u tablici 3.2. Uvidom u podatke vidi se da je trenutno najbolje rješenje ono pod brojem 7. Međutim, vjerojatnost njegovog odabira gotovo je jednaka vjerojatnosti odabira najgoreg rješenja u populaciji, i iznosi nešto više od 12%. Problem koji je ovdje istaknut je osjetljivost prikazanog algoritma odabira na skalu. Ako su funkcije dobrote relativno

mali brojevi koji se pri tome dosta razlikuju, algoritam radi zadovoljavajuće. Međutim, kako vrijednosti dobrote jedinki rastu, tako se smanjuje relativna razlika između njih i vjerojatnosti odabira najgore i najbolje jedinke postaju ujednačene. Rješenje ovog problema leži ili u uporabi relativne dobrote (kao djelotvorna dobrota svake jedinke uzme se razlika između dobrote jedinke i dobrote najgore jedinke, što je bitno manje osjetljivo na skalu) odnosno u uporabi prethodno opisanog rangiranja, gdje se jedinkama temeljem njihovom ranga u populaciji dodijeli dobrota (što je potpuno neosjetljivo na skalu, ali zahtjeva sortiranje populacije) i potom se tako definirana dobrota koristi za proporcionalnu selekciju.

3.1.5.2 k-turnirska selekcija

k-turnirska selekcija iz populacije posredstvom slučajnog mehanizma odabire k jedinki i potom uzima najbolju (ili najgoru – ovisno za što se odabir koristi). Ukoliko je potrebno odabrati n roditelja, postupak turnirske selekcije ponavlja se n puta. Čest je slučaj uporabe 3-turnirske selekcije gdje se nasumice odabiru 3 jedinke i potom uzima najbolja od njih. Kako bi se dobila dva roditelja koja će se potom križati, potrebno je obaviti dva turnira.

U praksi se često koristi i dodatno pojednostavljenje 3-turnirske selekcije u kombinaciji s eliminacijskim genetskim algoritmom: kako bi se dobila dva roditelja za križanje, nasumice se iz populacije izvuku 3 jedinke. Za roditelje se uzimaju dvije bolje a najlošiju se jedinka odbaci i zamijeni nastalim djetetom. Time se umjesto obavljanja tri turnira sve riješi samo jednim turnirom.

3.1.6 Drugi prikazi kromosoma

Binarni kromosomi prikladan su prikaz kada se radi o jednostavnijim problemima. Vrlo često, međutim, radi se o problemima gdje bi cijena kodiranja odnosno dekodiranja rješenja u/iz binarnog prikaza bila potpuno neprihvatljiva. Primjerice, rješava li se problem trgovačkog putnika genetskim algoritmom, kao kromosom često se koristi polje cijelih brojeva čija veličina odgovara broju gradova, a na i-tom mjestu se nalazi redni broj grada koji u i-tom koraku treba posjetiti. Dakako, u tom slučaju treba razviti i odgovarajuće operatore križanja i mutacije. Primjerice, mutacija bi mogla odabrati dva elementa polja i zamijeniti im redoslijed. S križanjem ovdje treba biti posebno oprezan,

jer se ne smije dopustiti da se jednostavnim prijelomom kromosoma roditelja dobije dijete koje neke gradove posjećuje dva puta a neke nikada.

Ako je rješenje problema koji rješavamo više decimalnih brojeva, umjesto binarnog prikaza možemo koristiti polje decimalnih brojeva. U tom slučaju operator križanja na i-to mjesto u dijete može staviti aritmetičku sredinu vrijednosti koje su bile na i-tom mjestu u oba roditelja. Mutaciju pak možemo implementirati tako da svakom elementu polja dodamo neki slučajno generirani broj izvučen iz normalne distribucije.

3.2 Genetsko programiranje

Genetsko programiranje tehnika je izuzetno srodna genetskom algoritmu. Razlika je u tome što kromosom predstavlja program koji je rješenje zadanog problema. Tipična struktura podataka koja se kod genetskog programiranja koristi za prikaz kromosoma je stablo.

Pogledajmo to na jednostavnom primjeru. Mjerenjem izlaza nekog procesa snimljena je njegova ulazno-izlazna karakteristika (tablica 3.3).

Zanima nas algebarski izraz koji najbolje opisuje tabeliranu funkciju (grafički prikaz

Tablica 3.3: Primjer funkcije jedne varijable za aproksimaciju genetskim programiranjem

x	f(x)	x	f(x)
0	4	1.6	-5.93108
0.1	3.892383	1.7	-4.70869
0.2	2.578716	1.8	-2.32285
0.3	0.657845	1.9	0.472592
0.4	-1.0855	2	2.816585
0.5	-1.96498	2.1	4.031366
0.6	-1.63934	2.2	3.846619
0.7	-0.23462	2.3	2.480139
0.8	1.700922	2.4	0.548066
0.9	3.393531	2.5	-1.16275
1	4.122921	2.6	-1.97962
1.1	3.485439	2.7	-1.58571
1.2	1.548364	2.8	-0.13352
1.3	-1.15971	2.9	1.809713
1.4	-3.82031	3	3.465504
1.5	-5.59958		

dan je na slici 3.15).

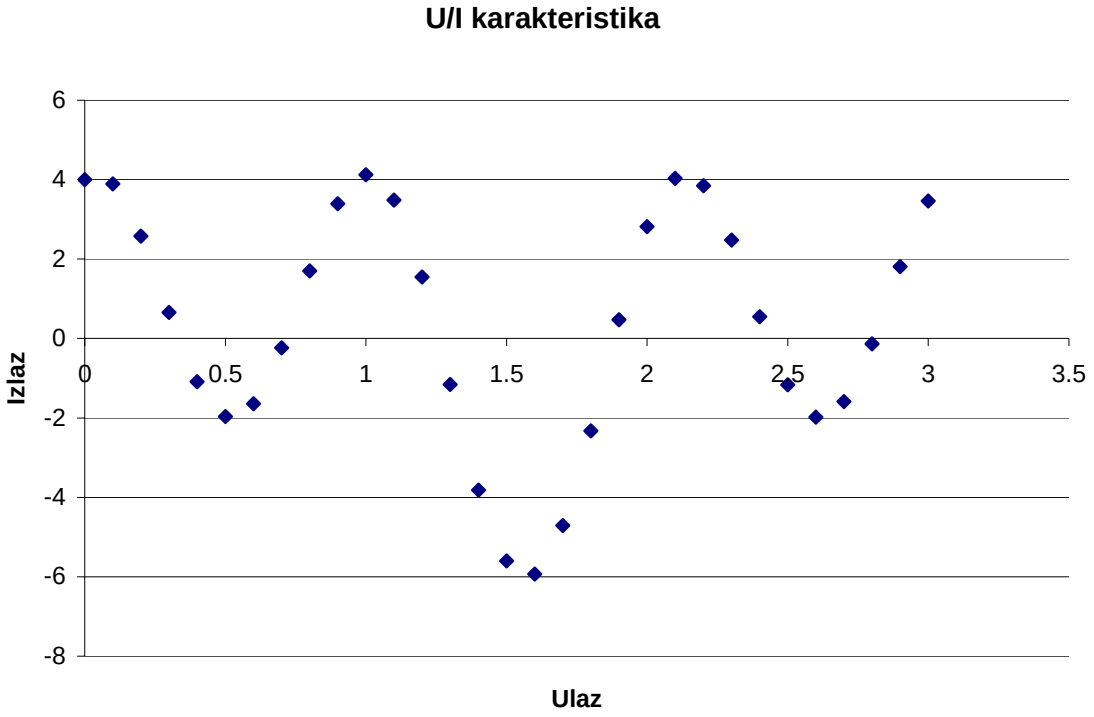
Zadatak genetskog programiranja je pronaći funkciju koja dobro aproksimira ove vrijednosti. Sjetimo se da se svaka funkcija može prikazati operatorskim stablom. Primjerice, funkciju:

$$f(x) = 2 \cdot sin(3 \cdot x) + 4 \cdot cos(6 \cdot x) \tag{3.6}$$

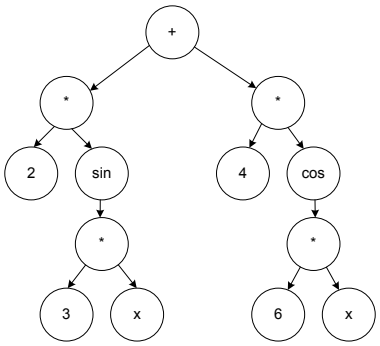
je moguće prikazati stablom (slika 3.16).

Kromosomi kod genetskog programiranja upravo su operatorska stabla, čiji čvorovi mogu biti, primjerice, aritmetičke operacije ($+$, $-$, $\cdot$, $/$), ugrađene funkcije (sin, cos), konstante (bilo koji cijeli ili decimalni broj) te varijable. Što se više vrsta čvorova dozvoli, veća je šansa da se pronađe adekvatno stablo koje odgovara zadanoj funkciji, ali raste i vrijeme potrebno za njegov pronalazak.

Evaluacija kromosoma u našem slučaju može se izvesti kao ukupna suma razlike izlaza funkcije u promatranoj točki i izmjerenog podatka (ili možemo računati srednje



Slika 3.15: Grafički prikaz ulazno-izlazne karakteristike sustava; vrijednosti odgovaraju onima prikazanima tablicom 3.3.



Slika 3.16: Prikaz funkcije operatorskim stablom.

kvadratno odstupanje), po svim točkama za koje imamo zadane podatke. Križanje dvaju roditelja tipično se radi tako da se neko podstablo jednog roditelja zamijeni nekim podstablom drugog roditelja (slika 3.17).

Operator mutacije može se izvesti na više načina – primjerice, slučajnom zamjenom čvora, brisanjem podstabla i zamjenom s novim slučajno stvorenim podstablom, zamjenom redosljeda djece i sl. Jedan primjer prikazan je na slici 3.18.

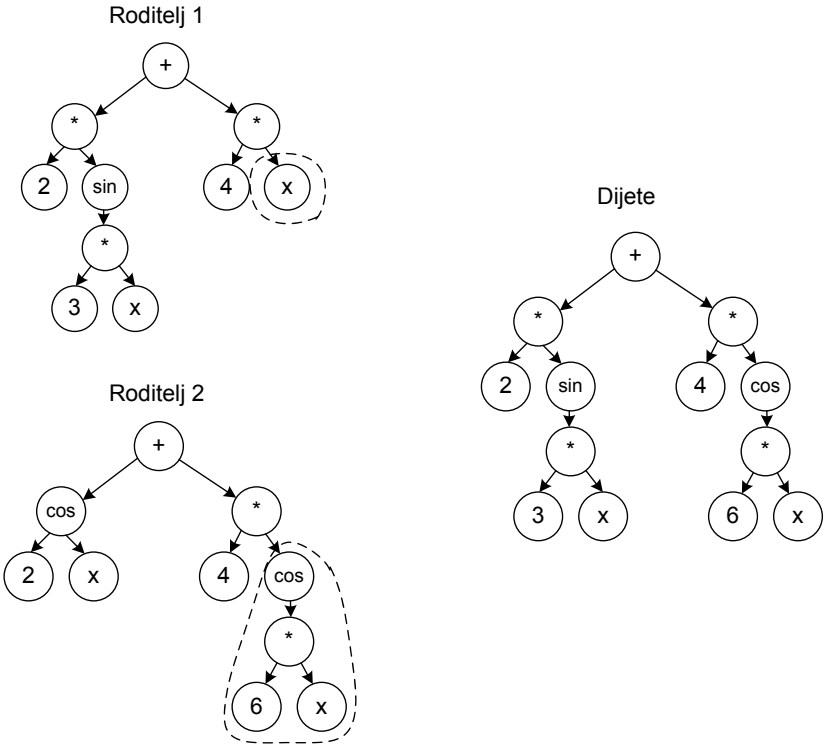
Jednom kada smo definirali sve potrebno (način prikaza kromosoma, način evaluacije, djelovanje operatora križanja i mutacije) postupak je dalje standardan (jednak kao kod klasičnog genetskog algoritma).

Uočimo također da problem koji rješavamo genetskim programiranjem ne mora biti ovako jednostavan. Zamislimo sljedeći primjer: imamo parove podataka (*slučajno stvoreno polje brojeva*, *sortirano polje brojeva*). Sjetimo li se da se svaki program može prikazati sintaksnim stablom – mogli bismo definirati dozvoljene vrste čvorova, i potom tražiti genetskim programiranjem program koji nesortirana polja prevodi u sortirana. Kromosom bi tada doslovno bio program koji bismo uporabom jezičnog procesora mogli prevesti, pokrenuti i evaluirati njegov rad.

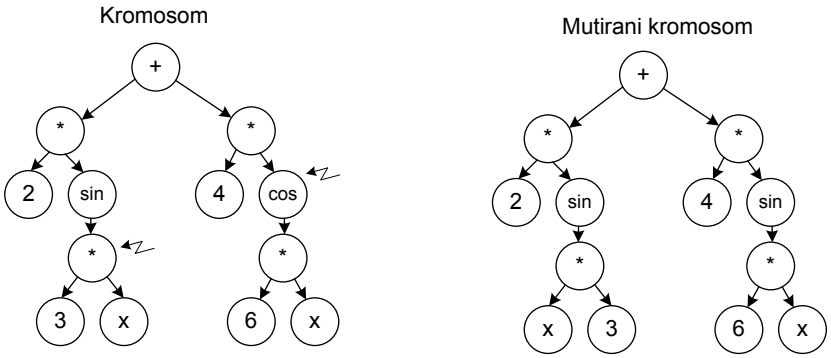
3.3 Diferencijska evolucija

Algoritam diferencijske evolucije, iako ne pripada među biološki inspirirane algoritme, ima dosta sličnosti s genetskim algoritmom. To je također populacijski algoritam koji nove potomke stvara uporabom operatora *diferencijacije*.

Razvoj algoritma diferencijske evolucije može se pratiti još od 1994. godine, kada je



Slika 3.17: Izvedba križanja kod genetskog programiranja.



Slika 3.18: Izvedba mutacije kod genetskog programiranja.

Storn u časopisu *Dr. Dobbs Journal* opisao algoritam genetskog kaljenja (engl. *Genetic Annealing*) [Price, 1994]. Godinu dana kasnije, prva verzija algoritma diferencijske evolucije opisana je u tehničkom izvještaju [Storn and Price, 1995], nakon čega je uslijedio niz radova [Storn and Price, 1996, Price and Storn, 1997, Storn and Price, 1997, Price, 1997, 1999].

Prva izvedba algoritma bila je načinjena za rješavanje optimizacijskih problema funkcija u kontinuiranom prostoru. Stoga algoritam sva rješenja tretira kao D-dimenzijske vektore. Osnovna ideja algoritma je modificirati jedinke trenutne populacije linearnom kombinacijom drugih jedinki iste populacije. Ovakav pristup ima često poželjno svojstvo *adaptivnog koraka pretraživanja*; naime, kako se modifikacija jedinki radi skaliranom diferencijom dviju slučajno odabranih jedinki, u slučaju kada je populacija dosta raspršena, i promjene će biti dosta velike. S druge strane, kada se populacija preseli u blizinu optimuma, diferencije između svih jedinki će biti male pa će i modifikacije biti male, čime se pospješuje brzo napredovanje prema optimumu, odnosno konvergencija.

U svrhu pojašnjenja rada algoritma neka je pretpostavka da je potrebno rješiti optimizacijski problem pronalaska vektora $\vec{x}$ koji minimizira zadanu funkciju $f(\vec{x})$; neka je pri tome vektor $\vec{x}$ D-dimenzijski, tj. $\vec{x} = (x_0, x_1, \ldots, x_{D-1})$.

3.3.1 Stvaranje početne populacije

Algoritam diferencijske evolucije je populacijski algoritam. Neka je s n označen broj jedinki populacije koje je potrebno stvoriti. Jedinke su D-dimenzijski vektori pri čemu su dozvoljene vrijednosti koje j-ta komponenta vektora $\vec{x}_i$ može poprimiti ograničene s $b_{L,j} \leq (\vec{x}_i)_j \leq b_{U,j}, 0 \leq j \leq D-1$. Uz takve uvjete stvaranje početne populacije opisano je pseudokodom 3.19. Oznaka `x(j,i)` pri tome označava j-tu komponentu vektora $\vec{x}_i$. Funkcija `slucajno(a,b)` za decimalne brojeve a i b vraća slučajni decimalni broj iz raspona $[a, b)$.

```
ponavljaj za i iz 0 do n-1
  ponavljaj za j iz 0 do D-1
    x(j,i)=bL(j) + slucajno(0.0,1.0) * (bU(j)-bL(j))
  kraj
kraj
```

Slika 3.19: Stvaranje početne populacije.

3.3.2 Diferencijska mutacija

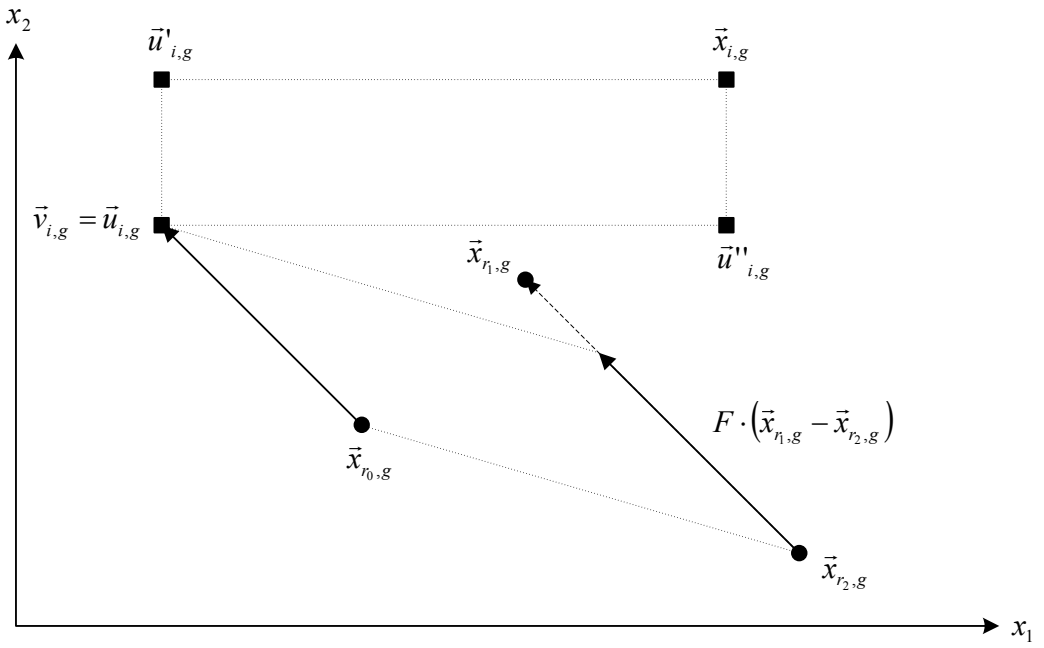
Prvi korak u stvaranju nove jedinke je stvaranje vektora *mutanta* (engl. *mutant vector*) za svaku jedinku generacije g. Taj će vektor potom biti iskorišten kako bi se križanjem stvorila nova jedinka koja će postati kandidat za ubacivanje u generaciju $g+1$. Pretpostavimo da se promatra i-ta jedinka populacije. Taj vektor $\vec{x}_{i,g}$ (i-to rješenje u generaciji g) predstavlja ciljni vektor (engl. *target vector*). Uz tako fiksirani i posredstvom slučajnog mehanizma biraju se još tri jedinke iz populacije generacije g. Neka su odabrane jedinke s indeksima r_0, r_1 i r_2. Sva četiri broja (i, r_0, r_1 i r_2) pri tome moraju biti različita. Vektor $\vec{x}_{r_0,g}$ predstavlja bazni vektor (engl. *base vector*). Vektor mutant $\vec{v}_i, g$ tada je definiran kao zbroj baznog vektora i skalirane razlike preostala dva vektora, što prikazuje izraz (3.7).

$$\vec{v}_{i,g} = \vec{x}_{r_0,g} + F \cdot (\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) \tag{3.7}$$

Grafički prikaz izgradnje vektora mutanta za slučaj dvodimenzijskih vektora prikazan je na slici 3.20; skalirana razlika vektora $F \cdot (\vec{x}_{r_1,g} - \vec{x}_{r_2,g})$ dodana je baznom vektoru $\vec{x}_{r_0,g}$ čime je izgrađen vektor mutant $\vec{v}_{i,g}$. Taj se vektor mutant potom koristi u postupku križanja kako je opisano u nastavku. Ulaz u operator diferencijske mutacije su tri vektora $\vec{x}_{r_0,g}$, $\vec{x}_{r_1,g}$ i $\vec{x}_{r_2,g}$ a izlaz je vektor $\vec{v}_{i,g}$.

3.3.3 Križanje

Nakon što je za neki fiksirani ciljni vektor $\vec{x}_{i,g}$ izgrađen vektor mutant $\vec{v}_{i,g}$, provodi se postupak križanja kako bi se izgradio *probni vektor* $\vec{u}_{i,g}$ (engl. *trial vector*). Križanje se pri tome provodi između vektora mutanta i ciljnog vektora, i to na sljedeći način. Definira se vjerojatnost križanja $0 \leq C_r \leq 1$ koja predstavlja vjerojatnost da j-ta komponenta probnog vektora bude preuzeta od vektora mutanta. U slučaju da se j-ta komponenta ne preuzme iz vektora mutanta, preuzet će se od ciljnog vektora $\vec{x}_{i,g}$. Kako bi se osiguralo da se uvijek barem jedna komponenta preuzme iz vektora mutanta, na početku se provođenja križanja posredstvom slučajnog mehanizma odabire se indeks komponente koja će sigurno biti preuzeta (označimo ga s j_{rand}). Križanje se tada može opisati izrazom (3.8).



Slika 3.20: Djelovanje operatora diferencijske mutacije i križanja.

$$\vec{u}_{j,i,g} = \begin{cases} v_{j,i,g} & \text{ako je slucajno}(0.0, 1.0) \leq C_r \text{ ili } j == j_{rand} \\ x_{j,i,g} & \text{inace} \end{cases} \tag{3.8}$$

Grafički prikaz križanja ciljnog vektora $\vec{x}_{i,g}$ i vektora mutanta $\vec{v}_{i,g}$ za slučaj dvodimenzijskih vektora također je prikazan na slici 3.20. Teoretski, moguća su tri rezultata križanja – vektor $\vec{u}_{i,g}$ koji je sve komponente preuzeo iz vektora mutanta, vektor $\vec{u}'_{i,g}$ koji je prvu komponentu preuzeo iz vektora mutanta a drugu iz ciljnog vektora te vektor $\vec{u}''_{i,g}$ koji je prvu komponentu preuzeo iz ciljnog vektora a drugu iz vektora mutanta. Ulaz u operator križanja su vektori $\vec{v}_{i,g}$ i $\vec{x}_{i,g}$ a izlaz je jedan od tri vektora $\vec{u}_{i,g}$, $\vec{u}'_{i,g}$ ili $\vec{u}''_{i,g}$.

3.3.4 Operator selekcije

Nakon što se za svaki ciljni vektor vektor izgradi po jedan probni vektor, probni vektori se vrednuju. Operator selekcije primjenjuje se na parove (ciljni vektor, probni vektor), pri čemu se u sljedeću generaciju propušta probni vektor ako mu je dobrota bolja ili jednaka dobroti ciljnog vektora; u suprotnom propušta se ciljni vektor. Ako se obavlja postupak minimizacije funkcije f, djelovanje operatora selekcije može se opisati izrazom

(3.9).

$$\vec{x}_{i,g+1} = \begin{cases} u_{i,g} & \text{ako je } f(u_{i,g}) \leq f(x_{i,g}) \\ x_{i,g} & \text{inace} \end{cases} \tag{3.9}$$

Razlog propuštanja probnog vektora u sljedeću generaciju ako je dobrotom jednak ciljnom vektoru je izbjegavanje stagnacije. Naime, ako je probni vektor jednako dobar kao i ciljni vektor, kvaliteta rješenja se ne mjenja, a u sljedeću generaciju se uvodi raznolikost koja će možda pridonijeti daljnjem napretku postupka optimizacije.

3.3.5 Konačni oblik algoritma diferencijske evolucije

Nakon prolaska kroz osnovne operatore koje koristi algoritam diferencijske evolucije, sada možemo opisati i cjelokupni algoritam. Nakon što se izgradi početna populacija algoritam svaku jedinku trenutne populacije g proglašava ciljnim vektorom; za njega gradi vektor mutant te križanjem dobiva probni vektor. Ako se populacija sastoji od n jedinki, ovo će rezultirati s n probnih vektora. Nakon što su izgrađeni svi probni vektori, uspoređuju se dobrote odgovarajućih ciljnih vektora i probnih vektora (po parovima) te se u sljedeću generaciju propuštaju bolji. Time je izgrađena populacija $g+1$ i postupak se ponavlja.

Implementacija ovog algoritma prikazana je pseudokodom 3.21, a grafički prikaz dan je na slici 3.22.

3.3.6 Strategije generiranja probnih vektora

Prethodno opisani način generiranja probnog vektora samo je jedan od niza mogućih. U knjizi iz 2005. Price i suautori navode četiri strategije generiranja probnih vektora [Price et al., 2005].

- DE/rand/1/bin
- DE/best/1/bin
- DE/target-to-best/1/bin
- DE/rand/1/either-or

Prvi parametar nakon DE govori o načinu na koji se bira bazni vektor, broj govori koliko se vektorskih razlika koristi a zadnji parametar govori kako se provodi križanje.

```
P = stvori_početnu_populaciju(n)
evaluiraj(P)
ponavljaj_dok_nije_kraj
  ponavljaj za i iz 0 do n-1
    ponavljaj r0=slucajno(0,n) dok r0 u {i}
    ponavljaj r1=slucajno(0,n) dok r1 u {i,r0}
    ponavljaj r2=slucajno(0,n) dok r2 u {i,r0,r1}
    jrand = slucajno(0,D)
    ponavljaj za j iz 0 do D
      v(j,i)=x(j,r0)+F*(x(j,r1)-x(j,r2))
    kraj
    ponavljaj za j iz 0 do D
      ako je slucajno(0,1)<=Cr ili j==jrand tada
        u(j,i)=v(j,i)
      inače
        u(j,i)=x(j,i)
      kraj
    kraj
  kraj
  ponavljaj za i iz 0 do n-1
    ako je f(u(i)) < f(x(i)) tada
      x(i) = u(i)
    kraj
  kraj
kraj
```

Slika 3.21: Algoritam diferencijske evolucije.

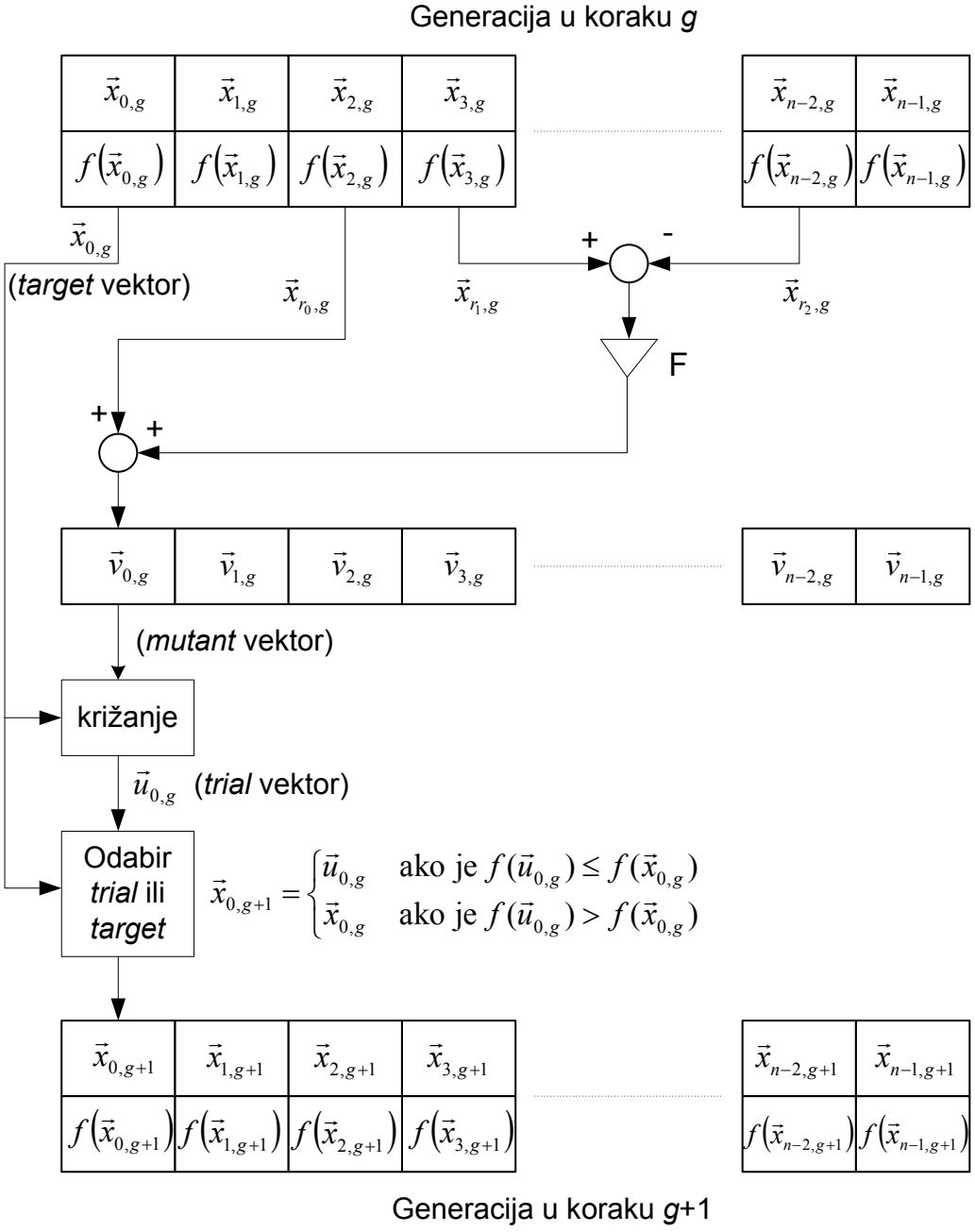
3.3.6.1 Strategija DE/rand/1/bin

Kod ove strategije kao bazni vektor odabire se slučajna jedinka iz populacije ($\vec{x}_{r_0,g}$). Koristi se jedna vektorska razlika a križanje se provodi uporabom zadane vjerojatnosti križanja C_r. Može se pisati:

$$\vec{v}_{i,g} = \vec{x}_{r_0,g} + F \cdot (\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) \tag{3.10}$$

Križanje se provodi prema izrazu (3.8) koji je ponovno naveden u nastavku.

$$\vec{u}_{j,i,g} = \begin{cases} v_{j,i,g} & \text{ako je slucajno}(0.0, 1.0) \leq C_r \text{ ili } j == j_{rand} \\ x_{j,i,g} & \text{inace} \end{cases} \tag{3.11}$$



Slika 3.22: Grafički prikaz algoritma diferencijske evolucije.

3.3.6.2 Strategija DE/best/1/bin

Kod ove strategije kao bazni vektor koristi se trenutno najbolja jedinka iz populacije ($\vec{x}_{\text{best}}$). Koristi se jedna vektorska razlika a križanje se provodi uporabom zadane vjerojatnosti križanja C_r. Preporuča se umjesto fiksne vrijednosti za F u svaku dimenziju uvesti malo odstupanje, tako da se prilikom za j-tu dimenziju koristi $F_j = F + 0.001 \cdot (\text{slucajno}_j(0.0, 1.0) - 0.5)$. Može se pisati:

$$\vec{v}_{i,g} = \vec{x}_{\text{best}} + F \cdot (\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) \tag{3.12}$$

Križanje se provodi prema izrazu (3.8) koji je ponovno naveden u nastavku.

$$\vec{u}_{j,i,g} = \begin{cases} v_{j,i,g} & \text{ako je slucajno}(0.0, 1.0) \leq C_r \text{ ili } j == j_{rand} \\ x_{j,i,g} & \text{inace} \end{cases} \tag{3.13}$$

3.3.6.3 Strategija DE/target-to-best/1/bin

Kod ove strategije kao bazni vektor odabire se ciljni vektor kojem se dodaje skalirana razlika najbolje jedinke iz populacija i ciljnog vektora ($\vec{x}_{i,g} + F \cdot (\vec{x}_{\text{best}} - \vec{x}_{i,g})$). Koristi se jedna vektorska razlika a križanje se provodi uporabom zadane vjerojatnosti križanja C_r. Može se pisati:

$$\vec{v}_{i,g} = \vec{x}_{i,g} + F \cdot (\vec{x}_{\text{best}} - \vec{x}_{i,g}) + F \cdot (\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) \tag{3.14}$$

Križanje se provodi prema izrazu (3.8) koji je ponovno naveden u nastavku.

$$\vec{u}_{j,i,g} = \begin{cases} v_{j,i,g} & \text{ako je slucajno}(0.0, 1.0) \leq C_r \text{ ili } j == j_{rand} \\ x_{j,i,g} & \text{inace} \end{cases} \tag{3.15}$$

3.3.6.4 Strategija DE/rand/1/either-or

Rad Price-a i drugih [Price et al., 2005] upućuje na postojanje problema kod koji je optimalan način generiranja problnih vektora uporaba čiste diferencijske mutacije; s druge pak strane postoje problemi kod kojih je optimalan način generiranja problnih vektora uporaba čiste rekombinacije. Stoga je definirana strategija *either-or* koja u skladu s propisanom vjerojatnosti P_F generira probne vektore uporabom diferencijske mutacije – izraz 3.16, a u $(1 - P_F)$ posto slučajeva probne vektore generira uporabom

rekombinacije – izraz 3.17.

$$\vec{u}_{i,g} = x_{r_0,g} + F \cdot (x_{r_1,g} - x_{r_2,g}) \tag{3.16}$$

$$\vec{u}_{i,g} = x_{r_0,g} + K \cdot (x_{r_1,g} + x_{r_2,g} - 2 \cdot x_{r_0,g}) \tag{3.17}$$

Pri tome se preporuča koristiti $K = 0.5 \cdot (F + 1)$.

Vitaliy Feoktistov u knjizi daje još puno koncizniju podjelu strategija za generiranje vektora mutanata [Feoktistov, 2006]. Prema njemu, sve se strategije mogu zapisati u općem obliku kao:

$$\omega = \beta + F \cdot \delta \tag{3.18}$$

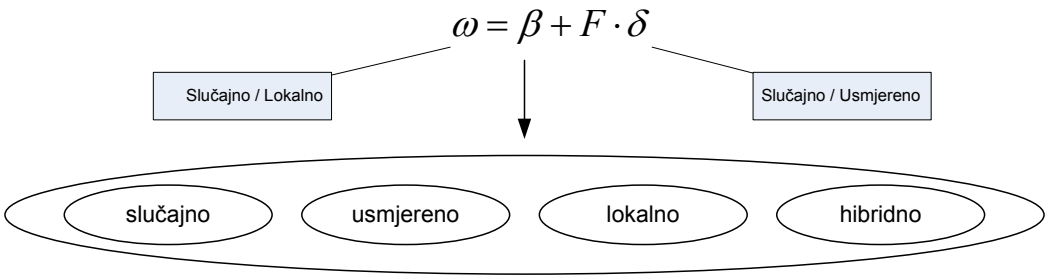
gdje je ω oznaka za vektor mutant, β oznaka za bazni vektor a δ oznaka za vektor diferencije. Različitim načinima odabira parametara β i δ moguće je dobiti različite razrede strategija, kao što je to ilustrirano na slici 3.23.

Četiri razreda koja definira Feoktistov navedena su u nastavku.

1. Razred *RAND* čine strategije kod kojih se pri stvaranju vektora mutanta ne uzima u obzir vrijednost evaluacijske funkcije.

2. Razred *RAND/DIR* čine strategije kod kojih se pri stvaranju vektora mutanta u obzir uzima vrijednost evaluacijske funkcije kako bi se odredio dobar smjer. Time se imitira gradijentni spust.

3. Razred *RAND/BEST* čine strategije kod kojih se pri stvaranju vektora mutanta u obzir uzima trenutno najbolje rješenje. Često ovakvo pretraživanje sliči nasu-



Slika 3.23: Algoritam diferencijske evolucije – opći oblik strategije generiranja mutanta.

mičnom pretraživanju okolice najboljeg rješenja.

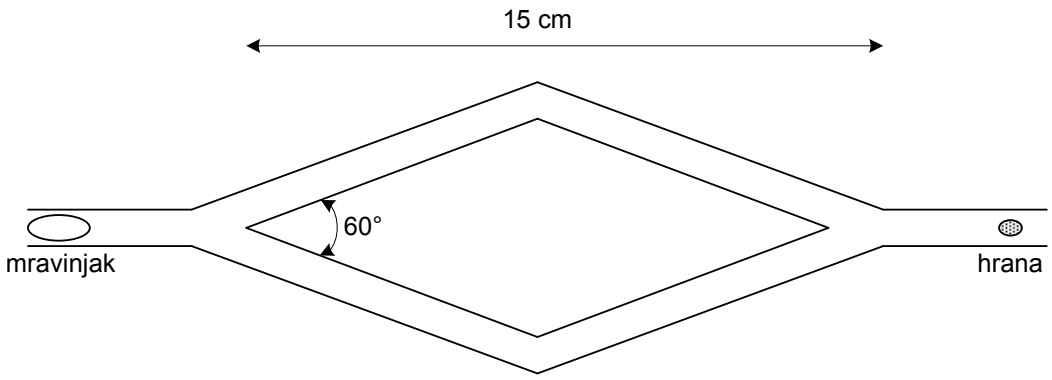
4. Razred *RAND/BEST/DIR* kombinira svojstva prethodne dvije grupe.

Primjere za svaku od ovih grupa čitatelj može potražiti u [Feoktistov, 2006].

3.4 Mravlji algoritmi

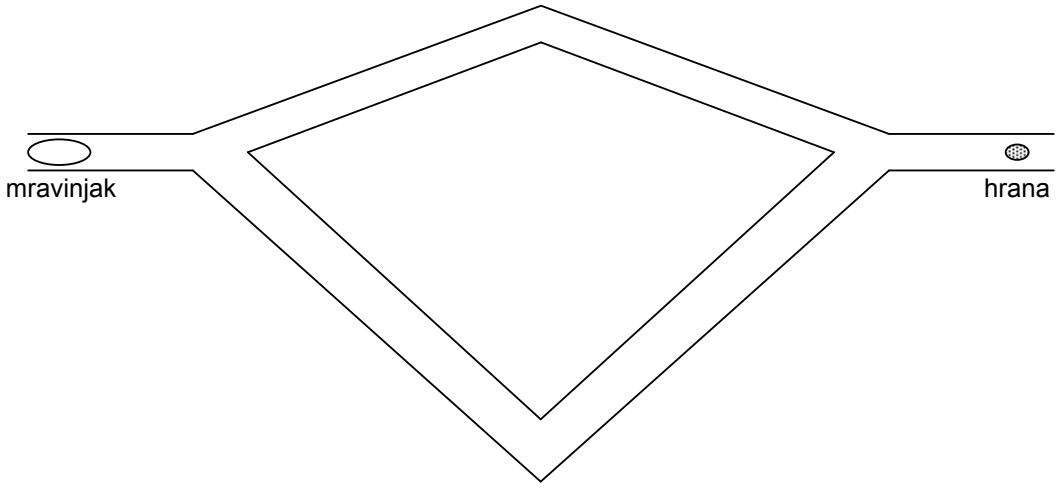
Mravi su izuzetno jednostavna bića u odnosu na čovjeka. Pa ipak, zahvaljujući socijalnim interakcijama ostvaruju vrlo kompleksna ponašanja. Stoga je mnoštvo različitih znanstvenih disciplina iskazalo interes za proučavanje mravljih kolonija. U okviru ove disertacije mravi su zanimljivi jer u određenom smislu uspješno rješavaju jednu vrstu optimizacijskog problema. Naime, uočeno je da mravi uvijek pronalaze najkraći put između izvora hrane i njihove kolonije, što im omogućava da hranu dopremaju maksimalno brzo. Ovakvo ponašanje mravi postižu unatoč tome što uopće nemaju razvijen osjetilo vida ili kod vrsta koje ga imaju on je vrlo loš.

Kako bi istražili ponašanje mrava, Deneubourg i suradnici [Goss et al., 1989, Denebourg et al., 1990] načinili su niz eksperimenata koristeći dvokraki most postavljen između mravinjaka i hrane (slika 3.24). Pri tome su promatrali vrstu mrava *Argentine ant, Iridomyrmex humilis*.



Slika 3.24: Eksperiment dvokrakog mosta (prvi).

Mravi prilikom kretanja ne koriste osjet vida, već je njihovo kretanje određeno socijalnim interakcijama s drugim mravima [Bonabeau et al., 1997a,b]. Na putu od mravinjaka do hrane, te na putu od hrane do mravinjaka, svaki mrav ostavlja kemijski trag – feromone. Mravi imaju razvijen osjet feromona, te prilikom odlučivanja kojim putem



Slika 3.25: Eksperiment dvokrakog mosta (drugi).

krenuti tu odluku donose obzirom na jakost feromonskog traga koji osjećaju. Tipično, mrav će se kretati smjerom jačeg feromonskog traga.

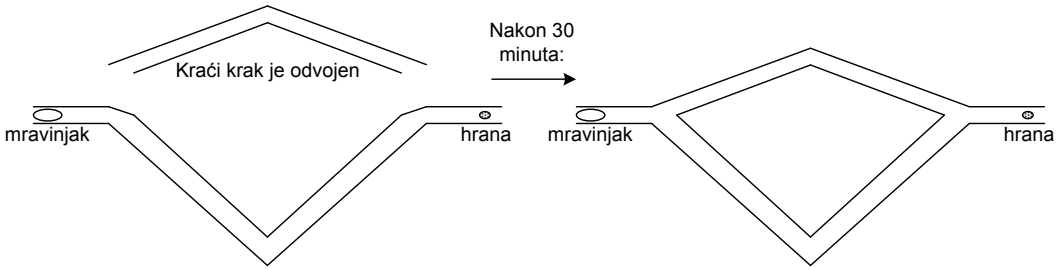
Deneubourg i suradnici prilikom eksperimenata varirali su duljine krakova mosta. U prvom eksperimentu oba su kraka bila jednako duga. Mravi su na početku u podjednakom broju krenuli preko oba kraka. Nakon nekog vremena, međutim, dominantni dio mrava kretao se je samo jednim krakom (slučajno odabranim). Objašnjenje je sljedeće. Na početku, mravi slučajno odabiru jedan ili drugi krak, i krećući se njima ostavljaju feromonski trag. U nekom trenutku dogodi se da nekoliko mrava više krene jednim od krakova (uslijed slučajnosti) i tu nastane veća koncentracija feromona. Privučeni ovom većom količinom feromona, još više mrava krene tim krakom što dodatno poveća količinu feromona. S druge strane, kako drugim krakom krene manje mrava, količina feromona koja se osvježava je manja, a feromoni s vremenom i isparavaju. Ovo u konačnici dovodi do situacije da se stvori jaki feromonski trag na jednom kraku, i taj feromonski trag privuče najveći broj mrava. Eksperiment je ponovljen više puta, i uočeno je da u prosjeku u 50% slučajeva mravi biraju jedan krak, a u 50% slučajeva biraju drugi krak.

Sljedeći eksperiment načinjen je s dvokrakim mostom kod kojeg je jedan krak bio dvostruko dulji od drugoga (slika 3.25). U svim eksperimentima pokazalo se da najveći broj mrava nakon nekog vremena bira kraći krak.

Ovakvo ponašanje na makroskopskoj razini – pronalazak najkraće staze između hrane i mravinjaka rezultat je interakcija na mikroskopskoj razini – interakcije između

pojedinih mrava koji zapravo nisu svjesni "šire slike" [Camazine et al., 2001, Haken, 1983, Nicolis and Prigogine, 1977]. Takvo ponašanje temelj je onoga što je danas poznato pod nazivom *izranjajuća inteligencija* (engl. *emerging intelligence*).

Konačno, kako bi se provjerila dinamika odnosno sposobnost prilagodbe mrava na promjene, načinjen je treći eksperiment (slika 3.26).



Slika 3.26: Eksperiment dvokrakog mosta (treći).

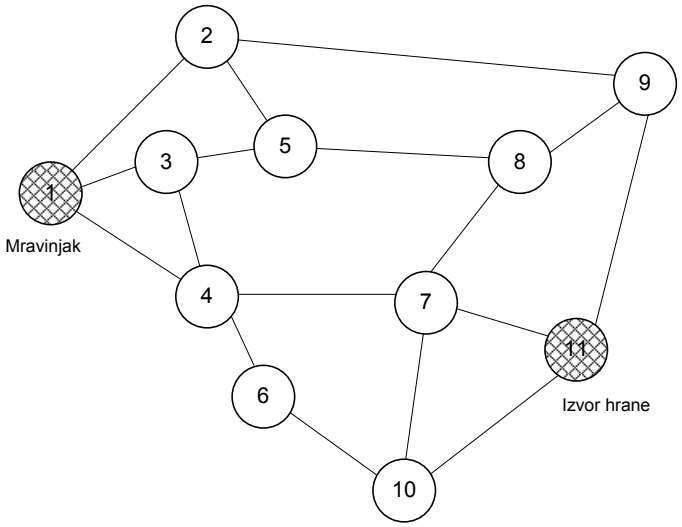
Kod ovog eksperimenta mravinjak i izvor hrane najprije su spojeni jednokrakim mostom (čiji je krak dugačak). Mravi su krenuli tim krakom do hrane i natrag. Nakon 30 minuta kada se je situacija stabilizirala, mostu je dodan drugi, dvostruko kraći krak. Međutim, eksperiment je pokazao da se je najveći dio mrava i dalje nastavio kretati duljim krakom, zahvaljujući stvorenom jakom feromonskom tragu.

3.4.1 Pojednostavljeni matematički model

Matematički model koji opisuje kretanje mrava dan je u [Dorigo and Stützle, 2004]. Mravi prilikom kretanja u oba smjera (od mravinjaka pa do izvora hrane, te od izvora hrane pa do mravinjaka) neprestano ostavljaju feromonski trag. Štoviše, neke vrste mrava na povratku prema mravinjaku ostavljaju to jači feromonski trag što je izvor pronađene hrane bogatiji. Simulacije ovakvih sustava, posebice uzme li se u obzir i dinamika isparavanja feromona, izuzetno su kompleksne. Međutim, ideje i zakonitosti koje su ovdje uočene našle su primjenu u nizu mravljih algoritama – u donekle pojednostavljenom obliku.

Ideja mravljih algoritama ilustrirana je na sljedećem primjeru. Između mravinjaka i izvora hrane nalazi se niz tunela (slika 3.27, tuneli su modelirani bridovima grafa).

Ideja algoritma je jednostavna. U fazi inicijalizacije, na sve se bridove postavi ista (fiksna) količina feromona. U prvom koraku, mrav iz mravinjaka (čvor 1) mora odlučiti



Slika 3.27: Primjer pronalaska hrane.

u koji će čvor krenuti. Ovu odluku donosi na temelju vjerojatnosti odabira brida p_{ij}^k:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha}{\sum_{l \in N_i^k} \tau_{il}^\alpha}, & \text{ako je } j \in N_i^k \\ 0, & \text{ako je } j \notin N_i^k \end{cases} \tag{3.19}$$

pri čemu τ_{ij} predstavlja vrijednost feromonskog traga na bridu između čvorova i i j, a α predstavlja konstantu. Skup N_i^k predstavlja skup svih indeksa svih čvorova u koje je u koraku k moguće prijeći iz čvora i. Konkretno, u našem slučaju $N_1^1 = 2, 3, 4$. Ako iz čvora i nije moguće prijeći u čvor j, vjerojatnost će biti 0. Suma u nazivniku prethodnog izraza ide po svim bridovima koji vode do čvorova u koje se može stići iz čvora i.

Nakon što odabere sljedeći čvor, mrav ponavlja postupak sve dok ne stigne do izvora hrane (čvor 11). Rješenje problema ovom se tehnikom gradi dio po dio – mravlji algoritmi pripadaju porodici *konstrukcijskih algoritama*.

Jednom kada stigne do izvora hrane, mrav zna koliki je put prešao. Umjetni mravi feromone tipično ostavljaju na povratku, i to na način da je količina feromona proporcionalna dobroti rješenja (odnosno obrnuto proporcionalna duljini puta). Ažuriranje se radi za sve bridove kojima je mrav prošao, i to prema izrazu:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau^k \tag{3.20}$$

gdje je:

$$\Delta\tau^k = \frac{1}{L}, \tag{3.21}$$

pri čemu je L duljina pronađenog puta. Isparavanje feromona modelirano je izrazom:

$$\tau_{ij} \leftarrow \tau_{ij} \cdot (1 - \rho) \tag{3.22}$$

gdje je ρ brzina isparavanja (iz intervala 0 do 1). Isparavanje se primjenjuje na sve bridove grafa.

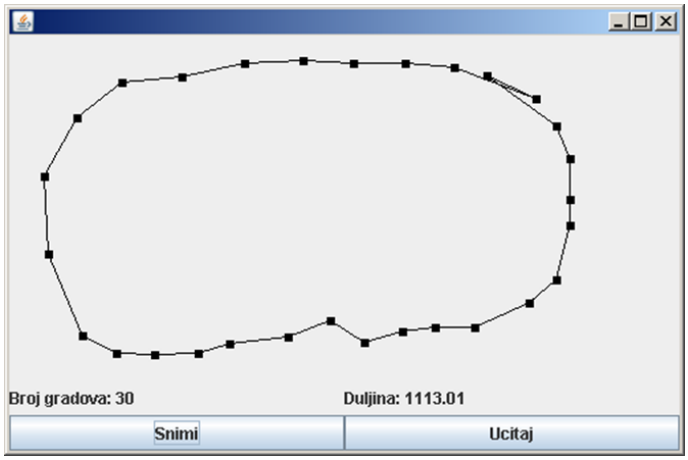
Ovaj pristup, dakako, ima svojih problema. Primjerice, kada mrav dinamički gradi put, ako se u svakom čvoru dopusti odabir bilo kojeg povezanog čvora, moguće je dobiti cikluse, što je nepoželjno. Također, ažuriranje feromonskog traga nakon svakog mrava pokazalo se je kao loše.

Temeljeći se na opisanim principima moguće je napisati jednostavan optimizacijski algoritam, što je prikazano pseudokodom 3.28.

```
ponavljaj dok nije kraj
  ponovi za svakog mrava
    stvori rješenje
    vrednuj rješenje
  kraj ponovi
  odaberi podskup mrava
  ponovi za odabrane mrave
    azuriraj feromonske tragove
  kraj ponovi
  ispari feromonske tragove
kraj ponavljanja
```

Slika 3.28: Jednostavni mravlji algoritam.

Algoritam radi s populacijom od m mrava, pri čemu u petlji ponavlja sljedeće. Svih m mrava pusti se da stvore rješenja i ta se rješenja vrednuju (izračunaju se duljine puteva). Pri tome, svaki mrav prilikom izgradnje pamti do tada pređeni put i kada treba birati u koji sljedeći čvor krenuti, automatski odbacuje čvorove kroz koje je već



Slika 3.29: Rješenje TSP-a dobiveno jednostavnim mravljim algoritmom.

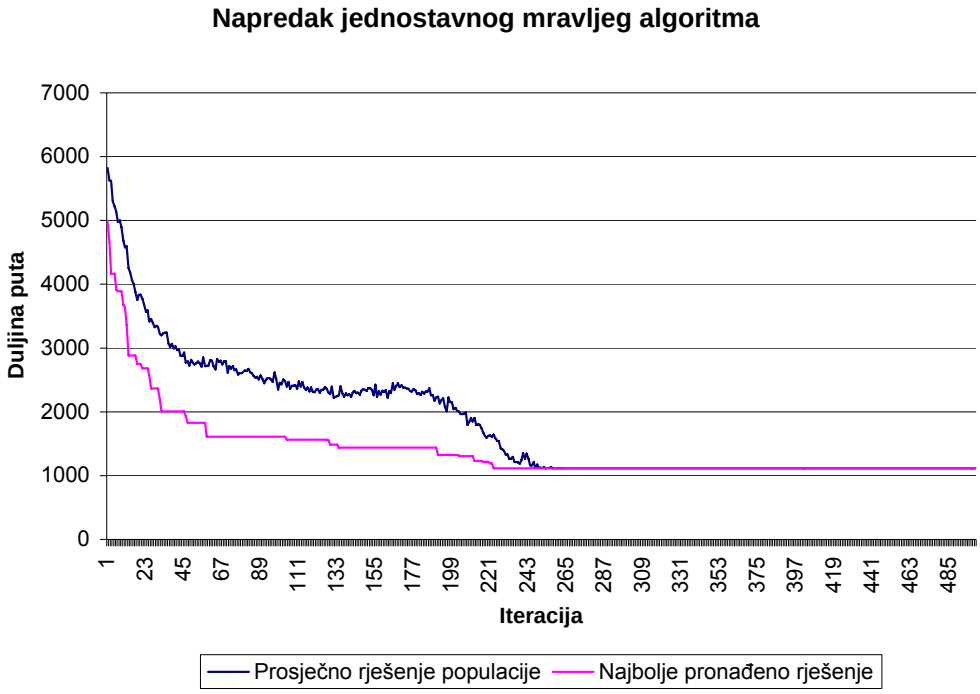
prošao (skupovi N_{ij}^k). Tek kada je svih m mrava stvorilo prijedloge rješenja, odabire se $n \leq m$ mrava koji će obaviti ažuriranje feromonskih tragova. Pri tome n može biti jednak m, što znači da svi mravi ažuriraju feromonske tragove. Kako se je takav pristup pokazao kao loša praksa, bolje je pustiti samo bolji podskup ili čak samo najboljeg mrava da obavi ažuriranje. Potom se primjenjuje procedura isparavanja feromona i postupak se ciklički ponavlja.

Kako bi se ilustrirao rad ovog algoritma, napisana je implementacija u Javi i to na problemu trgovačkog putnika s 30 gradova (TSP je NP težak problem; dobiveno rješenje kao i rješavani problem prikazani su na slici 3.29).

Rezultati su dobiveni uz sljedeće parametre: $m = 40$, $\rho = 0.2$, $\alpha = 1$, maksimalni broj iteracija iznosi 500. Feromonski tragovi svih bridova izvorno su postavljeni na $\frac{1}{5000}$. Sa slike se može uočiti da pronađeno rješenje nije idealno, ali je vrlo blizu optimalnog. Kvaliteta pronađenih rješenja kroz epohe prikazana je na slici 3.30, gdje se može pratiti najbolje pronađeno rješenje kao i prosječno pronađeno rješenje.

Osim opisanog jednostavnog algoritma, u porodicu mravljih algoritama ubrajaju se:

- mravlji sustav,
- elitistički mravlji sustav,
- rangirajući mravlji sustav,
- Max-Min mravlji sustav te drugi.



Slika 3.30: Napredak jednostavnog mravljeg algoritma.

U nastavku su opisani složeniji mravlji algoritmi koji se danas koriste za rješavanje problema.

3.4.2 Algoritam mravlji sustav

Algoritam *mravlji sustav* (engl. *Ant System*) predložili su Dorigo i suradnici [Dorigo and Stützle, 2004, Dorigo et al., 1991, Colorni et al., 1992]. Prilikom inicijalizacije grafa, na sve bridove deponira se količina feromona koja je nešto veća od očekivane količine koju će u jednoj iteraciji algoritma deponirati mravi. Koristi se izraz:

$$\tau_0 = \frac{m}{C^{nn}} \tag{3.23}$$

gdje je C^{nn} najkraća duljina puta pronađena nekim jednostavnim algoritmom (poput algoritma najbližeg susjeda). Ideja je zapravo dobiti kakvu-takvu procjenu duljine puta od koje će mravi dalje tražiti bolja rješenja, te ponuditi optimalnu početnu točku za rad algoritma. Prema [Dorigo and Stützle, 2004], uz premali τ_0 pretraga će brzo biti usmjerena prema području koje su mravi slučajno odabrali u prvoj iteraciji, a uz preveliki τ_0 količine feromona koje mravi ostavljaju u svakoj iteraciji bit će premale da bi

mogle usmjeravati pretragu, pa će se morati potrošiti puno iteracija kako bi mehanizam isparavanja uklonio višak feromona.

Rad algoritma započinje tako što m mrava stvara rješenja problema. U slučaju TSP-a, mravi se slučajno raspoređuju po gradovima iz kojih tada započinju konstrukciju rješenja. Prilikom odlučivanja u koji grad krenuti, umjesto prethodno prikazanog pravila, mravi koriste *slučajno proporcionalno pravilo* (engl. *random proportional rule*) koje uključuje dvije komponente: jakost prethodno deponiranog feromonskog traga te vrijednost heurističke funkcije. Vjerojatnost prelaska iz grada i u grad j određena je izrazom:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{l \in N_i^k} \tau_{il}^\alpha \cdot \eta_{il}^\beta}, & \text{ako je } j \in N_i^k \\ 0, & \text{ako je } j \notin N_i^k. \end{cases} \tag{3.24}$$

η_{ij} je pri tome heuristička informacija koja govori koliko se čini da je dobro iz grada i otići u grad j. Ovo je tipično informacija koja je poznata unaprijed, i u slučaju problema trgovačkog putnika računa se kao $\eta_{ij} = \frac{1}{d_{ij}}$, gdje je d_{ij} udaljenost grada i od grada j. Parametri α i β pri tome određuju ponašanje samog algoritma. Ako je $\alpha = 0$, utjecaj feromonskog traga se poništava, i pretraživanje se vodi samo heurističkom informacijom. Ako je pak $\beta = 0$, utjecaj heurističke informacije se poništava, i ostaje utjecaj isključivo feromonskog traga, što često dovodi do prebrze konvergencije suboptimalnom rješenju (gdje mravi slijede jedan drugoga po relativno lošoj stazi). Dobri rezultati postižu se uz $\alpha \approx 1$ i β između 2 i 5.

Nakon što su svi mravi načinili rješenja, rješenja se vrednuju. Potom se pristupa isparavanju feromona sa svih bridova, prema izrazu:

$$\tau_{ij} \leftarrow \tau_{ij} \cdot (1 - \rho). \tag{3.25}$$

Nakon isparavanja, svi mravi deponiraju feromonski trag na bridove kojima su prošli, i to proporcionalno dobroti rješenja koje su pronašli:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{3.26}$$

pri čemu je:

$$\Delta\tau_{ij} = \begin{cases} \frac{1}{C^k}, & \text{ako je brid } i\text{-}j \text{ na stazi } k\text{-tog mrava} \\ 0, & \text{inace} \end{cases} \tag{3.27}$$

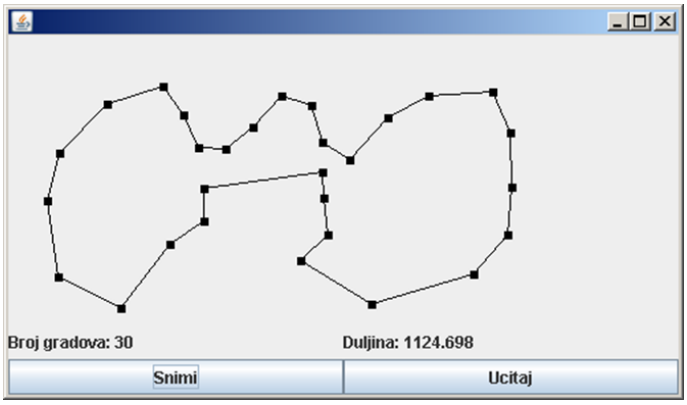
Implementacija ovog algoritma dana je pseudokodom 3.31.

```
ponavljaj dok nije kraj
  ponovi za svakog mravca
    stvori rješenje
    vrednuj rješenje
  kraj ponovi
  ispari feromonske tragove
  ponovi za sve mrave
    azuriraj feromonske tragove
  kraj ponovi
kraj ponavljanja
```

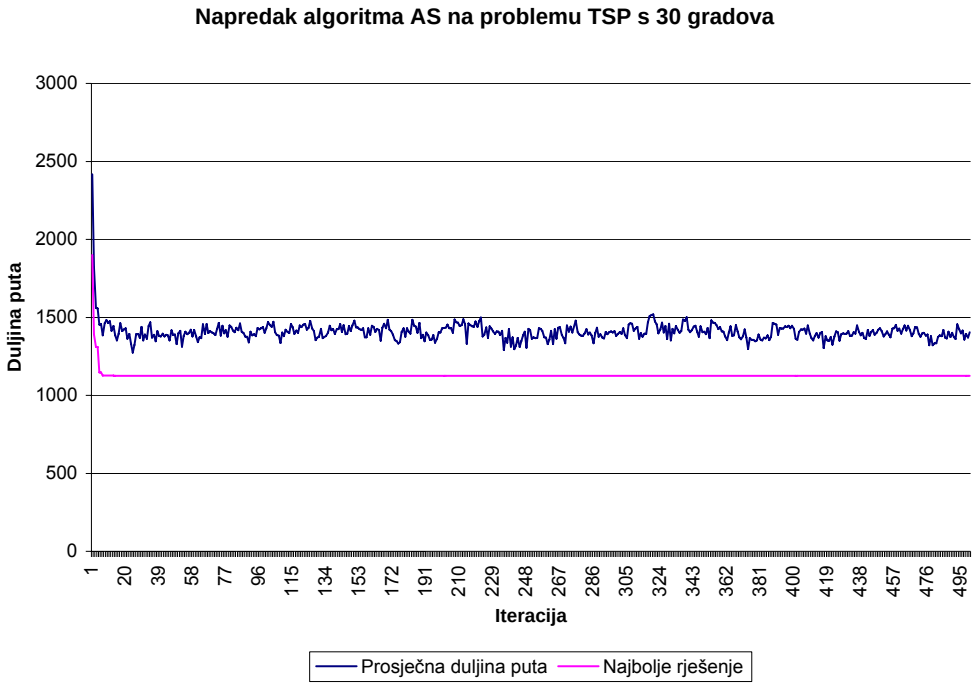
Slika 3.31: Pseudokod algoritma mravlji sustav.

U svrhu ilustracije rada algoritma načinjen je program u programskom jeziku Java i pokrenut nad problemom trgovačkog putnika s 30 gradova. Parametri algoritma su redom: $m = 30$, $\alpha = 1$, $\beta = 2$, $\rho = 0.5$. Algoritam je zaustavljen nakon 500 iteracija. Rezultati su prikazani na slikama 3.32 i 3.33.

Elitistička verzija algoritma (engl. *Elitist Ant System* – EAS) predložena je kao poboljšanje algoritma u [Dorigo et al., 1991, 1996]. Kod ove verzije dodatno se pamti globalno najbolje pronađeno rješenje. U svakoj iteraciji to se rješenje također koristi za ažuriranje feromonskih tragova s određenom težinom e. Ovo je moguće predočiti postojanjem još jednog mrava (označenog s bs) koji u svakoj iteraciji prođe upravo



Slika 3.32: Rješenje TSP-a dobiveno algoritmom mravlji sustav.



Slika 3.33: Napredak algoritma mravlji sustav.

najboljim zapamćenim putem. Pravilo ažuriranja feromona tada dobiva još jedan član pa glasi:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e \cdot \Delta\tau_{ij}^{bs}. \tag{3.28}$$

U [Dorigo and Stützle, 2004] se kao vrijednost konstante e navodi upravo broj mrava m.

Prema [Bullnheimer et al., 1999], dodatno poboljšanje donosi *algoritam rangirajućeg mravljeg sustava* (engl. *Rank-based Ant System*). Kod ovog algoritma, nakon što svi mravi stvore rješenja, mravi se sortiraju prema kvaliteti rješenja. Ažuriranje feromona obavlja samo najboljih $w-1$ mrava (i to proporcionalno svojem rangu), te mrav koji čuva globalno najbolje rješenje (koje ulazi s najvećim utjecajem):

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{w-1} (w-k)\Delta\tau_{ij}^{k} + w \cdot \Delta\tau_{ij}^{bs}. \tag{3.29}$$

Svaki mrav pri tome ažurira samo bridove koji su dio njegovog puta (za sve ostale bridove $\Delta\tau_{ij}^{k} = 0$).

Max-Min mravlji sustav (MMAS) [Stützle and Hoos, 1997, 2000, 1999] još je jedno unaprjeđenje algoritma *mravlji sustav*, koje uvodi 4 modifikacije.

1. Samo najbolji mrav smije obavljati ažuriranje feromona. Postoje varijante algoritma koje ažuriranje dopuštaju samo globalno najboljem mravu, varijante algoritma koje ažuriranje dopuštaju samo najboljem mravu u trenutnoj iteraciji, te varijante algoritma koje ažuriranje dopuštaju i globalno najboljem i najboljem u svakoj iteraciji, i to različitim intenzitetom.

2. Kako bi se spriječila prerana stagnacija algoritma zbog prethodne modifikacije, uvodi se gornja i donja granica na jakost feromonskog traga: $\tau_{ij} \in [\tau_{\min}, \tau_{\max}]$.

3. Inicijalizacija algoritma feromonski trag na svim bridovima postavlja na njihovu maksimalnu vrijednost. Ovo zajedno s niskom stopom isparavanja (predlaže se $\rho = 0.02$) osigurava da mravi na početku obavljaju široko pretraživanje prostora rješenja.

4. Svaki puta kada algoritam dođe u stagnaciju, odnosno kada nema poboljšanja u kvaliteti rješenja unutar zadanog broja iteracija, obavlja se reinicijalizacija feromonskih tragova na njihove maksimalne vrijednosti.

Gornja granica pri tome se postavlja na vrijednost:

$$\tau_{\max} = \frac{1}{\rho \cdot C^{bs}} \tag{3.30}$$

i ažurira svaki puta kada se pronađe novo najbolje rješenje C^{bs}. Donja granica određena je parametrom a prema izrazu:

$$\tau_{\min} = \frac{\tau_{\min}}{a}, \tag{3.31}$$

gdje je a parametar koji je potrebno izabrati. Pri svakoj promjeni gornje granice, algoritam automatski mijenja i donju granicu.

Danas postoji još niz drugih varijanti mravljih algoritama, a zainteresirani čitatelj se upućuje na [Dorigo and Stützle, 2004].

3.5 Algoritam roja čestica

Algoritam roja čestica (engl. *Particle Swarm Optimization*) otkriven je sasvim slučajno, pri pokušaju da se na računalu simulira kretanje jata ptica. C. W. Reynolds u svom radu [Reynolds, 1987] promatra jato ptica kao sustav čestica, gdje svaka čestica (tj. ptica) svoj let ravna prema sljedećim pravilima: (i) izbjegavanje kolizija s bliskim pticama, (ii) usklađivanje brzine leta s bliskim pticama te (iii) pokušaj ostanka u blizini drugih ptica. Inspirirani ovim i sličnim radovima, R. C. Eberhart i J. Kennedy shvaćaju da se takav sustav može koristiti kao optimizator te svoje ideje objavljuju 1995. godine u dva temeljna rada [Kennedy and Eberhart, 1995, Eberhart and Kennedy, 1995]. Eberhart, Simpson i Dobbins 1996. godine izdaju knjigu [Eberhart et al., 1996] o uporabi algoritma roja čestica kao univerzalnog optimizacijskog alata. Tako je, primjerice, u knjizi opisana vrlo uspješna primjena algoritma roja česta za treniranje umjetne neuronske (točnije, unaprijedne umjetne neuronske mreže ili višeslojnog perceptrona), gdje se algoritam koristi kao zamjena algoritma Backpropagation. Konačno, 1997. godine Eberhart i Kennedy objavljuju rad o prilagodbi algoritma za rad nad diskretnim domenama [Kennedy and Eberhart, 1997]. Algoritam je u određenoj mjeri inspiriran i sociološkim interakcijama između pojedinaca u populaciji, gdje svaki pojedinac pamti svoje do tada pronađeno najbolje rješenje problema, ima uvid u najbolje pronađeno rješenje svojih susjeda i pretraživanje usmjerava uzimajući u obzir obje komponente.

3.5.1 Opis algoritma

Algoritam roja čestica je populacijski algoritam. Populacija se sastoji od niza jedinki (čestica) koje lete kroz višedimenzijski prostor koji pretražuju, i pri tome svoj položaj mijenjaju temeljem vlastitog iskustva, te iskustva bliskih susjeda (čime se modeliraju socijalne interakcije između jedinki). Prilikom određivanja smjera kretanja, svaka jedinka u određenoj mjeri uzima u obzir svoje do tada pronađeno najbolje rješenje (*individualni faktor*), te najbolje pronađeno rješenje svoje bliske okoline (*socijalni faktor*). Utjecaj koji svaka od ovih komponenti ima uvelike određuje ponašanje same jedinke: radi li istraživanje prostora stanja (ukoliko je dominantni individualni faktor) ili fino podešavanje pronađenog rješenja (ukoliko je dominantni socijalni faktor). Na ovaj način sam algoritam kombinira globalno pretraživanje prostora stanja te lokalnu pretragu kojom se obavlja fino podešavanje rješenja.

Pseudokod 3.34 prikazuje ideju algoritma roja čestica. Algoritam koristi populaciju veličine `VEL_POP`, te pretražuje `DIM`-dimenzijski prostor rješenja. Pri tome x sadrži trenutne pozicije svih čestica a v njihove brzine. Polja x i v su dvodimenzijska: imaju onoliko redaka koliko ima čestica, te onoliko stupaca koliko rješenje ima dimenzija. Čestice pretražuju ograničeni prostor rješenja, a granice se čuvaju u poljima $xmin$ i $xmax$. Brzina svake čestice (te u svakoj dimenziji) ograničena je svojom minimalnom i maksimalnom vrijednosti (primjerice, od -5 do +5), što čuvaju polja $vmin$ i $vmax$.

Algoritam započinje inicijalizacijom populacije. Svaka se čestica smješta na neku slučajno odabranu poziciju, i dodjeljuje joj se neka slučajno odabrana brzina.

Slijedi glavni dio algoritma koji se ponavlja tako dugo dok se ne ispuni uvjet zaustavljanja (pronalazak dovoljno dobrog rješenja ili dostizanje maksimalnog broja iteracija).

- Za svaku se česticu izračuna vrijednost funkcije u točki koju čestica predstavlja.
- Za svaku se česticu provjeri njeno do tada zapamćeno najbolje rješenje i njeno novo pronađeno rješenje. Ako je novo bolje, pamti se kao novo najbolje rješenje te čestice. U pseudokodu ovo se pamti u dvodimenzijskom polju *pbest* (engl. *particles best solution*). Potrebno je pamtiti rješenje i vrijednost funkcije u tom rješenju.
- U čitavoj populaciji se pronađe najbolje rješenje i ako je prethodno zapamćeno globalno rješenje lošije, ažurira se na novo pronađeno. U pseudokodu ovo se pamti u polju *gbest* (engl. *global best solution*). Potrebno je pamtiti rješenje i vrijednost funkcije u tom rješenju.
- Za svaku česticu se obavlja ažuriranje trenutne brzine a potom i položaja. Najprije se obavi ažuriranje brzine tako da se na trenutnu brzinu doda individualna komponenta modulirana faktorom individualnosti (c_1) i slučajnim brojem iz intervala $[0, 1]$ te socijalna komponenta modulirana faktorom socijalnosti (c_2) i slučajnim brojem iz intervala $[0, 1]$. Potom se provjeri je li brzina izvan dozvoljenog raspona, i ako je, korigira se. Konačno, u skladu s novom brzinom ažurira se položaj čestice.

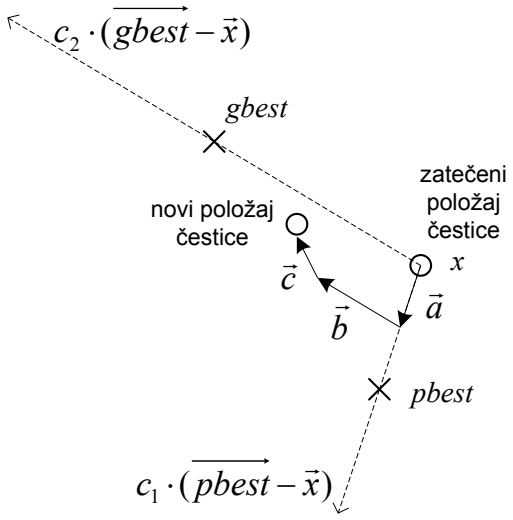
Kao što je vidljivo iz opisanoga, ažuriranje se obavlja prema sljedećim izrazima:

$$v_{i,d} = v_{i,d} + c_1 \cdot \text{rand}() \cdot (\text{pbest}_{i,d} - x) + c_2 \cdot \text{rand}() \cdot (\text{gbest}_d - x). \tag{3.32}$$

```
// inicijaliziraj_populaciju:
za i = 1 do VEL_POP
  za d iz 1 do DIM
    x[i][d] = random(xmin[d], xmax[d])
    v[i][d] = random(vmin[d], vmax[d])
  kraj
kraj

ponavljaj dok nije kraj

  // evaluiraj_populaciju:
  za i = 1 do VEL_POP
    f[i] = funkcija(x[i]);
  kraj

  // ima li čestica svoje bolje rješenje?
  za i = 1 do VEL_POP
    ako je f[i] bolji od pbest_f[i] tada
      pbest_f[i] = f[i]
      pbest[i] = x[i]
    kraj
  kraj

  // ima li čestica globano najbolje rješenje?
  za i = 1 do VEL_POP
    ako je f[i] bolji od gbest_f[i] tada
      gbest_f[i] = f[i]
      gbest[i] = x[i]
    kraj
  kraj

  // ažuriraj brzinu i poziciju čestice
  za i = 1 do VEL_POP
    za d iz 1 do DIM
      v[i][d] = v[i][d] + c1*rand()*(pbest[i][d]-x[i][d])
                        + c2*rand()*(gbest[d]-x[i][d])
      v[i][d] = iz_opsega(v[i][d], vmin[d], vmax[d])
      x[i][d] = x[i][d] + v[i][d]
    kraj
  kraj
kraj
```

Slika 3.34: Pseudokod algoritma roja čestica.

Faktori c_1 i c_2 uobičajeno se postavljaju na 2.0. Veća vrijednost faktora c_1 omogućit će veći stupanj individualnosti jedinke i time poticati istraživanje prostora, dok veća



Slika 3.35: Grafički prikaz pomaka čestice kod algoritma roja čestice.

vrijednost faktora c_2 jači naglasak stavlja na najbolje rješenje koje je čitav kolektiv do tada pronašao, i time osigurava detaljnije istraživanje okoline tog rješenja. Slika 3.35 jednostavnim vektorskim prikazom ilustrira "sile" koje djeluju na kretanje čestice, i pretpostavlja da se sve dimenzije vektora razlike *pbest* i x množe istim slučajno generiranim brojem (ista je pretpostavka i za vektor razlike *gbest* i x). Također, slika je nacrtana uz $c_1 = 2.0$ i $c_2 = 2.0$.

Prema formuli za ažuriranje brzine, nova brzina rezultat je triju komponenti: vektora $\vec{a}$, $\vec{b}$ i $\vec{c}$:

$$\vec{a} = c_1 \cdot \text{rand}() \cdot \vec{(}\text{pbest} - \vec{x}), \tag{3.33}$$

$$\vec{b} = c_2 \cdot \text{rand}() \cdot \vec{(}\text{gbest} - \vec{x}), \tag{3.34}$$

$$\vec{c} = \vec{v}. \tag{3.35}$$

Vektor $\vec{a}$ predstavlja pomak prema najboljem rješenju koje je pronašla jedinka, odgovarajuće skaliranom. Vektor $\vec{b}$ predstavlja pomak prema najboljem rješenju kolektiva, također odgovarajuće skaliranom. Vektor $\vec{c}$ poprima staru trenutnu vrijednost brzine, i zapravo predstavlja inerciju same čestice. Rezultantna brzina suma je sva tri djelovanja: čestica se po inerciji dalje nastavlja gibati, i pri tome brzinu djelomično modificira uslijed privlačenja svojeg i kolektivnog najboljeg rješenja.

3.5.2 Utjecaj parametara i modifikacije algoritma

Da bi opisani algoritam mogao biti pokrenut, nužno je odrediti vrijednosti svih parametara. Stoga je važno znati kakav je njihov utjecaj na rad algoritma.

Ograničenje brzine nužno je jer bez njega algoritam može doći u divergentno stanje. Stavi li se ograničenje brzine preveliko, čestica može preletiti preko područja dobrih rješenja. Stavi li se pak ograničenje brzine na premalu vrijednost, može se dogoditi situacija da čestica ostane zatočena u lokalnim optimumima – kako je brzina premala, čestica se više ne može se oteti utjecaju lokalnog optimuma. Prema [Eberhart and Shi, 2001], *vmax* se tipično stavlja na 10% do 20% raspona prostora koji se pretražuje.

Konstante c_1 i c_2 modeliraju jakost privlačne sile između najboljih rješenja i čestice – što veći broj, to je veća privlačna sila pa će čestica moći manje istraživati.

Veličina populacije tipično se kreće od 20 do 50. Naravno, postoje i problemi kod kojeg se do zadovoljavajućeg rješenja dolazi tek s većim populacijama.

3.5.2.1 Dodavanje faktora inercije

Proces pretraživanja prostora uobičajeno se podijeliti u dva koraka. U prvom korak obavlja se grubo pretraživanje kako bi se locirala "zanimljiva" područja, a potom se u drugom koraku obavlja fino pretraživanje unutar lociranih kandidata. Kako bi se osiguralo ovakvo ponašanje, izraz za ažuriranje brzine modificira se na sljedeći način:

$$v_{i,d} = w(t) \cdot v_{i,d} + c_1 \cdot \text{rand}() \cdot (\text{pbest}_{i,d} - x) + c_2 \cdot \text{rand}() \cdot (\text{gbest}_d - x). \tag{3.36}$$

Inercijska komponenta brzine množi se vremenski promjenjivim faktorom w. Inicijalno, w se postavlja na vrijednost blizu 1 (primjerice, 0.9), a s povećanjem broja iteracija t vrijednost se smanjuje prema nekoj minimalnoj vrijednosti. Ako su s w_{max} i w_{min} označene željena maksimalna i minimalna vrijednost, te ako je s T označena iteracija u kojoj težinski faktor treba pasti na w_{min}, za ažuriranje se može koristiti sljedeći izraz:

$$w(t) = \begin{cases} \frac{t}{T} \cdot (w_{min} - w_{max}) + w_{max}, & t \leq T \\ w_{min}, & t > T. \end{cases} \tag{3.37}$$

Ažuriranje pozicije čestice radi se na uobičajeni način.

3.5.2.2 Stabilnost algoritma

Prethodno je spomenuto da je jedan od načina da se pokuša osigurati stabilnost algoritma (u smislu da se spriječi divergencija) ograničavanje iznosa brzine. Matematička analiza samog algoritma (vidi [Clerc, 1999]) pokazuje da se stabilnost može postići ako se izraz za ažuriranje brzine modificira dodavanjem *faktora ograničenja* (engl. *constriction factor*) K:

$$v_{i,d} = K \cdot \left[v_{i,d} + c_1 \cdot \text{rand}() \cdot (\text{pbest}_{i,d} - x) + c_2 \cdot \text{rand}() \cdot (\text{gbest}_d - x) \right] \tag{3.38}$$

gdje je K funkcija parametara c_1 i c_2 i definiran je prema izrazu:

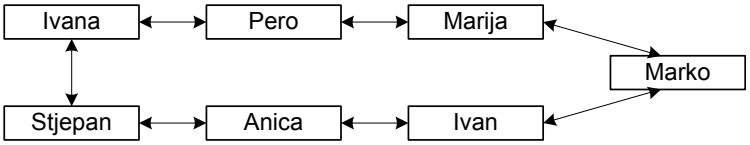
$$K = \frac{2}{\left| 2 - \phi - \sqrt{\phi^2 - 4 \cdot \phi} \right|}. \tag{3.39}$$

Pri tome je $\phi = c_1 + c_2$, $\phi > 4$. Prema [Eberhart and Shi, 2001], ϕ se tipično postavlja na 4.1 ($c_1 = 2.05$, $c_2 = 2.05$), što daje za K vrijednost 0.729.

3.5.2.3 Lokalno susjedstvo

Kod algoritma roja čestica opisanog pseudokodom 3.34 svaka čestica osjeća utjecaj dviju sila. Prva sila je vlastito iskustvo, odnosno najbolje rješenje koje je sama čestica pronašla. Druga sila je iskustvo kolektiva, odnosno najbolje rješenje koje je pronašao čitav kolektiv. Ovakva vrsta algoritma roja čestica još se naziva *potpuno-informirani algoritam roja čestica* (engl. *Fully informed PSO*). Nezgodno svojstvo ove inačice algoritma je mogućnost prerane konvergencije – čim jedna jedinka nađe potencijalno dobro rješenje, sve su jedinke automatski privučene k tom rješenju, što može spriječiti temeljito istraživanje prostora stanja i pronalazak eventualno još boljih rješenja.

Kako bi se tomu doskočilo, razvijena je inačica algoritma kod koje jedinkama nije dostupna informacija o globalno najboljem rješenju. Umjesto toga, uveden je topološki uređaj u populaciju. Za svaku se jedinku zna tko su joj susjedi, što je pojašnjeno u primjeru u nastavku. Neka se populacija sastoji od sljedećih jedinki: Ivana, Pero, Marija, Marko, Ivan, Anica i Stjepan. Susjedstvo se definira preko poznanstava: Ivana zna Peru



Slika 3.36: Primjer definiranog susjedstva.

i Stjepana, pa su Pero i Stjepan njezini susjedi i Ivana će komunicirati samo s njima. Pero zna Ivanu i Mariju, pa su Ivana i Marija njegovi susjedi i Pero će komunicirati samo s njima. Slično se može načiniti i za ostale jedinke (slika 3.36; susjedi su osobe direktno povezane strelicama).

Prilikom pretraživanja prostora stanja, kada jedinki zatreba kolektivno najbolje rješenje, jedinka će to rješenje izračunati tako da pogleda svoje najbolje rješenje i najbolja rješenja svojih susjeda – globalna informacija jedinki nije dostupna. Primjerice, kada Ivana treba utvrditi kamo dalje, za utvrđivanje najboljeg rješenja kolektiva pogledat će svoje najbolje rješenje, najbolje rješenje Pere i najbolje rješenje Stjepana. Ovdje je važno uočiti da se susjedstvo ne definira prema blizini u prostoru rješenja. Prilikom pretraživanja, Ivana i Pero mogu se udaljiti, i Ivani Marija može doći i puno bliža no što je Pero; međutim, Marija time neće postati susjeda Ivani.

Prilikom pisanja implementacije ove vrste algoritma roja čestica, potrebno se odlučiti na koji će se način definirati susjedstvo, jer to nije jednoznačno. Jedan od čestih načina je definiranje parametra n_s (veličina susjedstva) pri čemu se čestice slože u niz. Ako je $n_s = 1$, svaka je čestica isključivo susjed sama sebi. Ako je $n_s = 2$, susjedi od `čestica(i)` su `čestica(i-1)`, `čestica(i)` te `čestica(i+1)`. Ako je $n_s = 3$, susjedi od `čestica(i)` su sve čestice od `čestica(i-2)` do `čestica(i+2)`, itd. U tom slučaju, kolektivno najbolje rješenje označava se s *lbest* (engl. *local best*). Izraz za ažuriranje brzine tada umjesto *gbest* koristi *lbest(i)*, gdje je *lbest(i)* najbolje rješenje susjedstva i-te čestice:

$$v_{i,d} = v_{i,d} + c_1 \cdot \text{rand}() \cdot (\text{pbest}_{i,d} - x) + c_2 \cdot \text{rand}() \cdot (\text{lbest}_{i,d} - x). \tag{3.40}$$

Prema [Eberhart and Shi, 2001], prikladne veličine susjedstva su oko 15% ukupne veličine populacije. Ako je s `VEL_SUS` označen ukupan broj čestica koje čine susjedstvo,

vrijedi:

$$\text{VEL\_SUS} = 1 + 2 \cdot n_s. \tag{3.41}$$

Za više informacija na ovu temu čitatelj se upućuje na nedavno objavljeni rad [Montes de Oca et al., 2009].

3.5.3 Primjer rada algoritma

Primjer u nastavku prikazuje optimizaciju funkcije:

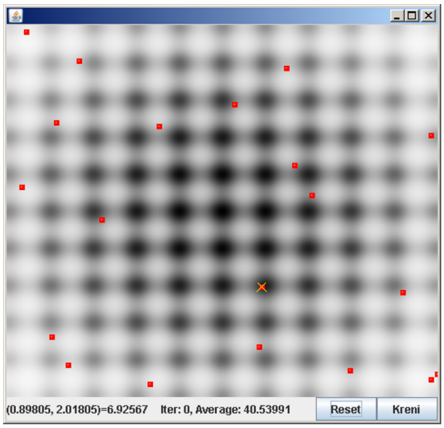
$$f(x_1, x_2, \ldots, x_n) = 10 \cdot n + \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot cos(2 \cdot \pi \cdot x_i)\right). \tag{3.42}$$

Konkretno, traži se vektor x za koji funkcija poprima minimalnu vrijednost. Analitičko rješenje je poznato: to je ishodište u kojem je vrijednost funkcije 0. Ova funkcija uzeta je stoga što ima niz lokalnih optimuma koji postupak pretraživanja globalnog optimuma bitno otežavaju. Prikaz rada algoritma za slučaj $D = 2$ prikazan je na slici 3.37, dok slika 3.38 prikazuje ovisnost najboljeg i prosječnog rješenja populacije u ovisnosti o epohi algoritma.

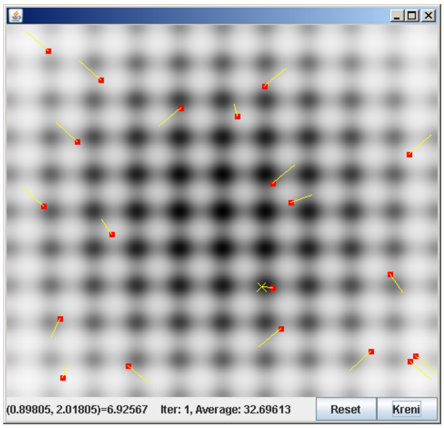
3.6 Algoritmi umjetnih imunoloških sustava

Darwinova teorija o razvoju vrsta poslužila je kao inspiracija za jednu veliku porodicu algoritama poput genetskog algoritma. Temeljna premisa ove teorije je seksualno razmožavanje kod kojega nove jedinke dobivaju genetski materijal od oba roditelja i dodatno bivaju mutirane. Pod utjecajem okoline, bolje i prilagođenije jedinke imaju veću šansu za daljnje razmožavanje, što u konačnici dovodi do izumiranja lošijih jedinki, i postupnom prilagođavanju vrste okolini u kojoj živi. Ovi principi iskorišteni su za izgradnju genetskog algoritma koji proces evolucije simulira uporabom dva genetska operatora: križanjem koje temeljem genetskog materijala dvaju roditelja stvara novog potomka, te mutacijom koja u genetski kôd djeteta unosi slučajne izmjene.

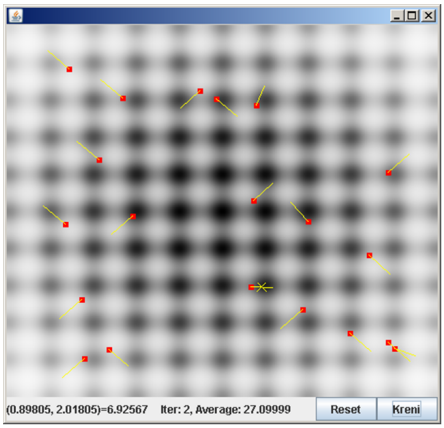
Međutim, ovo nije jedini način preživljavanja koji postoji u prirodi. Pogleda li se malo dublje u jednog pojedinca, otkriva se mehanizam koji, primjerice, čovjeku omogućava preživljavanje – njegov imunološki sustav. Prilikom rođenja, čovjekov imunološki



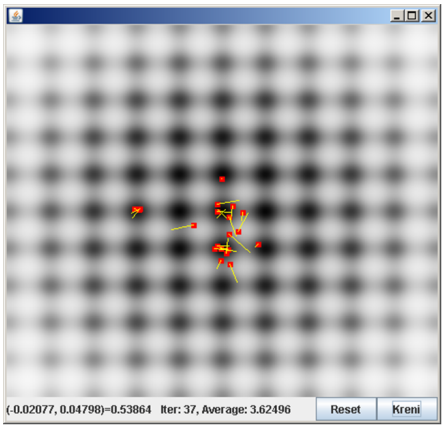
(a) Početno stanje



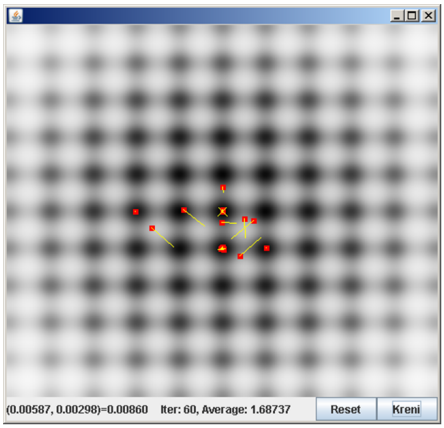
(b) Nakon prvog koraka



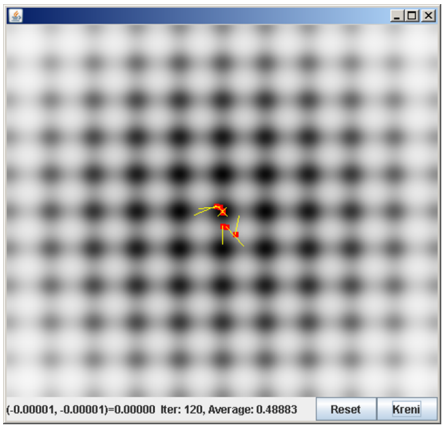
(c) Nakon drugog koraka



(d) Nakon 37. koraka

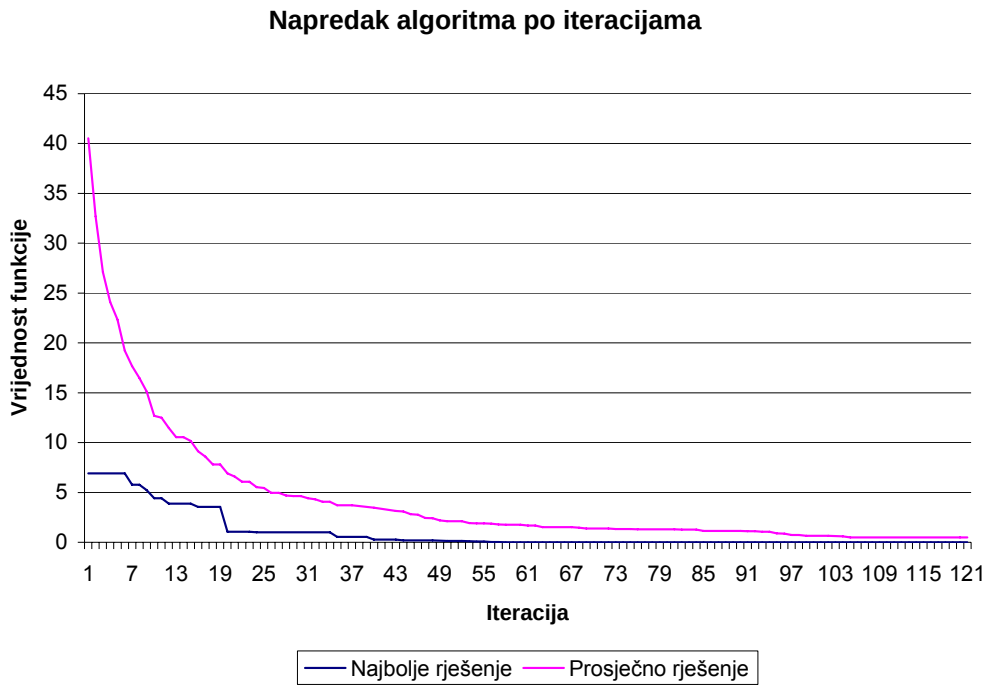


(e) Nakon 60. koraka



(f) Nakon 120. koraka

Slika 3.37: Prikaz rada algoritma roja čestica



Slika 3.38: Ovisnost najboljeg i prosječnog rješenja o iteraciji kod algoritma roja čestica.

sustav već raspolaže određenim mehanizmima za borbu protiv poznatih antigena (napadača). Ukoliko u tijelo uđu antigeni, tijelo raspolaže posebnom vrstom stanica koje su okruže antigen i potom ga unište. Ovaj mehanizam ljude štiti od velikog broja bolesti.

Posebnost imunološkog sustava je sposobnost prilagodbe i sposobnost učenja, što čovjeku omogućava obranu od bolesti kojima prethodno nije bio izložen te stjecanje imuniteta. Ako urođeni imunološki sustav ne prepozna i ne uništi antigene, aktivirat će se drugi dio imunološkog sustava koji je zadužen za reakciju na specifične antigene (ovo je proces koji može potrajati i nekoliko dana). Za obranu od antigena u tijelu su zadužene B-stanice koje luče antitijela (B-stanice proizvode se u koštanoj srži). Ulaskom antigena u tijelo dio B-stanica započet će lučenje antitijela. Antitijela imaju receptore kojima se mogu povezati s antigenom, a do ovog povezivanja će doći ukoliko su antitijelo i receptor kompatibilni. Mjera ove kompatibilnosti naziva se afinitetom. Povezivanje antigena s antitijelom stimulira B-stanicu koja je proizvela antitijelo i B-stanica započinje proces dijeljenja (mitoza), čime nastaje velika količina klonova B-stanice koji sazrijevaju i luče nove količine antitijela kao odgovor na nastalu infekciju. Prilikom procesa kloniranja B-stanice dolazi do nasumičnih mutacija u genetskom kodu B-stanice. Uslijed ovih mutacija dio nastalih klonova proizvodit će antitijela koja će imati još veći

afinitet prema antigenu, što će pak uzrokovati novi ciklus kloniranja tih stanica. Nakon nekog vremena, na ovaj način u tijelu će se razviti B-stanice koje mogu vrlo djelotvorno odgovoriti na specifični antigen koji je uzrokovao infekciju. Uništenjem antigena, međutim, dio B-stanica ostat će u tijelu, i ukoliko se nakon nekog vremena opet naiđe na isti antigen (ponovni razvoj infekcije), odgovor imunološkog sustava sada će biti daleko jači i efikasniji no što je to bilo prvog puta. Opetovanim izlaganjem istim antigenima, tijelo će u konačnici razviti brz i djelotvoran odgovor na taj antigen i time postaje imuno na tu bolest. Zanimljivost ovog procesa je činjenica da je čitav postupak usavršavanja B-stanica vođen isključivo slijepim nasumičnim mutacijama receptora.

Važno je napomenuti da prethodni opis predstavlja samo grupo pojednostavljenje složenih interakcija koje se odvijaju u imunološkom sustavu – no i to je dovoljno za opisati osnovne imunološke algoritme. Teoriju koja objašnjava kako radi imunološki sustav razvio je Burnet [Burnet, 1957, 1959, 1978] počev još davne 1957. godine. Područje *umjetnih imunoloških sustava* (engl. AIS – *Artificial Immune Systems*) bavi se razvojem modela i apstrakcija imunološkog sustava i njihovom primjenom u algoritmima za rješavanje problema u znanosti i inženjerstvu [de Castro and Timmis, 2002]. U ovoj disertaciji razmotrit će se samo *algoritmi klonske selekcije* (engl. CSA – *Clonal Selection Algorithms*). To su algoritmi koji pripadaju razredu *imunoloških algoritama* (engl. IA – *Immunological Algorithms*) za čiji je razvoj iskorišten prethodno opisani *princip klonske selekcije* (engl. *Clonal Selection Principle*) [Cutello and Nicosia, 2005, de Castro and Timmis, 2002]. Važno je napomenuti da zbog jednostavnosti, većina algoritama ne uvodi distinkciju između pojmova B-stanica i antitijelo, te ta dva pojma tretira kao jedan.

3.6.1 Jednostavni imunološki algoritam

Jednostavni imunološki algoritam (engl. SIA – *Simple Immunological Algorithm*) autora Cutello i Nicosia razvijen je 2002. godine, i opisan u [Cutello and Nicosia, 2002a,b, 2005]. Algoritam može služiti za izradu klasifikatorskih sustava te za optimizaciju. Ovdje je opisan s aspekta optimizacijskog algoritma.

SIA je populacijski algoritam koji radi s populacijom antitijela. Pri tome je svako antitijelo zapravo jedno rješenje problema koji se optimira. Antigen u ovom kontekstu predstavlja samu funkciju čiji se optimum traži. Afinitet pojedinog antitijela (rješenja)

prema antigenu (funkciji) tada je predstavljen kvalitetom (tj. dobrotom; engl. fitness) samog rješenja. Ukoliko se radi o maksimiziranju funkcije, tada je afinitet najčešće jednak upravo iznosu same funkcije u promatranom rješenju.

Izvorni opis algoritma za kodiranje rješenja koristi binarne nizove duljine l bitova (direktna analogija je binarno kodiranje kromosoma kod genetskog algoritma); međutim, postupak je proširiv na bilo kakvu reprezentaciju rješenja. Izvedba algoritma dana je pseudokodom 3.39.

```
SIA(Ag, l, d, dup)
  t = 0
  inicijaliziraj P(0) = {x1, x2, ..., xd}
  evaluiraj(P(0), Ag)
  ponavljaj dok nije zaustavi(P(t))
    Pclo = kloniraj(P(t), dup)
    Phyp = hipermutiraj(Pclo)
    evaluiraj(Phyp, Ag)
    P(t+1) = odaberi(Phyp + P(t), d)
    t = t+1
  kraj ponavljanja
kraj
```

Slika 3.39: Pseudokod jednostavnog imunološkog algoritma.

Parametri algoritma su:

- Ag – funkcija koju se optimira,
- l – broj bitova rješenja,
- d – veličina populacije rješenja, te
- dup – broj klonova svakog rješenja.

Algoritam započinje stvaranjem inicijalne populacije antitijela (rješenja) $P^{(0)}$ veličine d. U slučaju da se rješenja prikazuju binarno, ovo znači slučajno stvaranje d sljedova nula i jedinica, svaki duljine l. Potom se računa afinitet svakog antitijela (tj. rješenja se vrednuju).

Zatim se ulazi u petlju koja se ponavlja tako dugo dok uvjet zaustavljanja nije zadovoljen. To, primjerice, može biti pronalazak dovoljno dobrog rješenja ili prekoračenje maksimalnog broja iteracija.

U petlji se radi sljedeće. Svako antitijelo klonira se *dup* puta. Time iz populacije $P^{(t)}$, gdje je t oznaka iteracije, nastaje populacija klonova P^{clo} veličine $d \cdot dup$. Svako antitijelo iz populacije P^{clo} prolazi potom kroz proces hipermutacije čime nastaje nova populacija P^{hyp} iste veličine. Operator hipermutacije predstavlja nasumičnu promjenu receptora antitijela u pokušaju da se antitijelo bolje prilagodi povezivanju s antigenom. Kod ovog algoritma hipermutacija nasumično mijenja jedan bit na nasumično odabranoj poziciji (ili u slučaju nekog drugog načina predstavljanja rješenja obavlja jednu jednostavnu promjenu). Potom se za sve jedinke iz populacije P^{hyp} računaju afiniteti (vrednuju se nastala rješenja). Na kraju se iz unije populacija $P^{(t)}$ i P^{hyp} izabire d antitijela najvećeg afiniteta koji postaju nova populacija $P^{(t+1)}$ za sljedeći prolaz kroz petlju. Zbog takvog načina rada algoritam je automatski elitistički: ako su hipermutacijom nastala samo gora rješenja od trenutno najboljeg iz $P^{(t)}$, to najbolje će sigurno biti preneseno u novu populaciju i time očuvano.

Analiza složenosti samog algoritma može se pogledati u [Cutello and Nicosia, 2005].

U svrhu demonstracije rada algoritma na nekom malo složenijem problemu načinjena je implementacija u programskom jeziku Java koja uporabom ovog algoritma rješava problem trgovačkog putnika s 30 gradova.

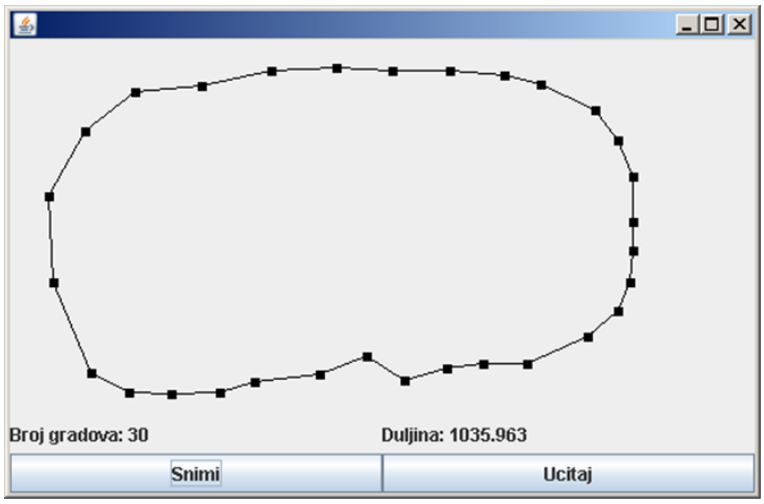
Antitijelo je pri tome predstavljeno kao vektor od 30 elemenata. Element na i-tom mjestu je cijeli broj koji predstavlja indeks grada koji trgovački putnik treba posjetiti u i-tom koraku. Operator hipermutacije izveden je tako da nasumično odabere dva koraka i zamijeni gradove koje trgovački putnik posjećuje u tim koracima.

Program je pokrenut uz sljedeće parametre:

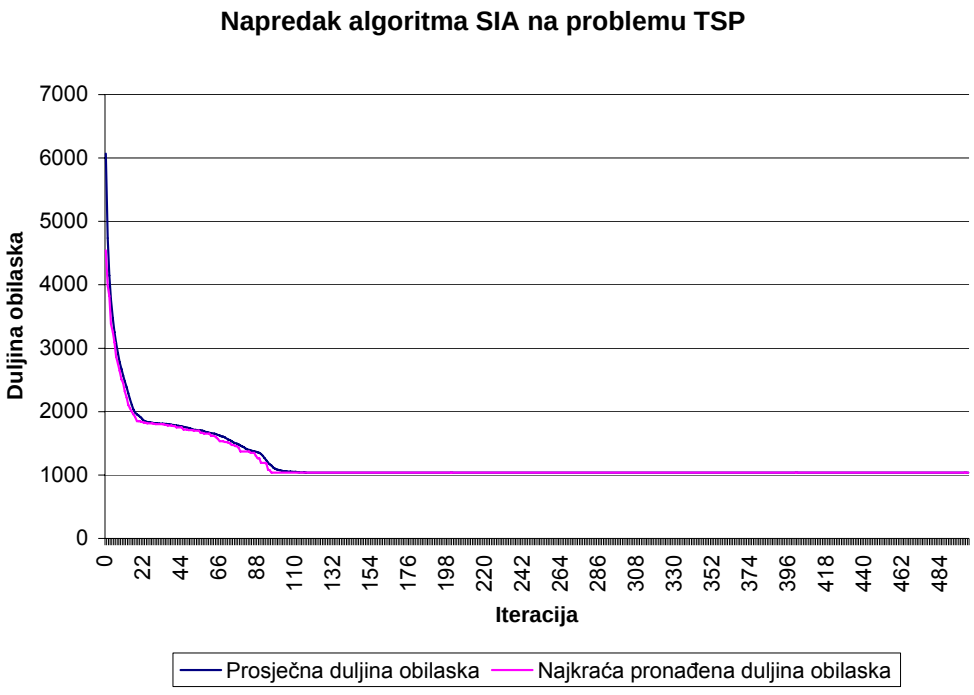
- broj antitijela u populaciji: 50,
- broj klonova za svaku stanicu: 10 te
- maksimalni broj iteracija: 500.

Na testnom primjeru (slika 3.40) teorijski najkraći put iznosi 1035,963 km, što je algoritam i pronašao. Kretanje prosječnog rješenja populacije kao najboljeg rješenja ovisno o broju iteracija algoritma prikazuje slika 3.41.

U nastavku slijedi opis algoritma CLONALG.



Slika 3.40: Problem obilaska 30 gradova riješen algoritmom SIA.



Slika 3.41: Napredak algoritma SIA na problemu TSP s 30 gradova.

3.6.2 Algoritam CLONALG

CLONALG je kratica od *Clonal Selection Algorithm*. Ovaj algoritam predložili su 1999. i kasnije razradili (2000., 2002.) De Castro i Von Zuben [de Castro and Zuben, 1999, 2000, 2002]. Opis algoritma dan je pseudokodom 3.42.

```
CLONALG(Ag, n, d, beta)
  t = 0
  inicijaliziraj P(0) = {x1, x2, ..., xd}
  ponavljaj dok nije zaustavi(P(t))
    evaluiraj(P(t), Ag)
    P(t) = odaberi(P(t), n)
    Pclo = kloniraj(P(t), beta)
    Phyp = hipermutiraj(Pclo)
    evaluiraj(Phyp, Ag)
    P' = odaberi(Phyp, n)
    Pbirth = stvoriNove(d)
    P(t+1) = zamijeni(P', Pbirth)
    t = t+1
  kraj ponavljanja
  evaluiraj(P(t), Ag)
kraj
```

Slika 3.42: Pseudokod algoritma CLONALG.

Ideja algoritma je slična kao i kod SIA, pri čemu se antitijela podvrgavaju postupku kloniranja proporcionalno njihovom afinitetu te postupku hipermutacije obrnuto proporcionalno njihovom afinitetu. Antitijela su nizovi od l elemenata pri čemu su elementi realni brojevi pa se svako antitijelo može promatrati kao točka u l-dimenzijskom prostoru.

Parametri algoritma su sljedeći:

- Ag – antigen odnosno funkcija koju optimiramo,
- n – broj antitijela u populaciji,
- d – broj novih jedinki koje ćemo u svakom koraku slučajno stvoriti i dodati u populaciju te
- β – parametar koji određuje veličinu populacije klonova.

Algoritam započinje stvaranjem početne populacije antitijela $P^{(0)}$. Antitijela se stvaraju nekim slučajnim mehanizmom. Potom se ulazi u petlju koja se ponavlja dok

uvjet zaustavljanja nije zadovoljen (autori izvorno kao uvjet zaustavljanja jedino navode fiksan broj iteracija).

U petlji se najprije računa afinitet svih antitijela obzirom na antigen (vrednuju se rješenja obzirom na funkciju koja se optimira). Potom se operatorom *odaberi* odabiru antitijela koja će se klonirati. Izvorno, algoritam odabire svih n antitijela (pa u tom slučaju ovo naprosto možemo preskočiti), no dozvoljava se i odabir manjeg broja.

Pristupa se procesu kloniranju odabranih antitijela (stvara se populacija P^{clo}). Pri tome je broj klonova pojedinog antitijela direktno proporcionalan afinitetu tog antitijela: što je afinitet veći, to je broj klonova veći. Ukupni broj klonova koji će ući u populaciju klonova P^{clo} označen je s N_C i određen parametrom β prema izrazu:

$$N_C = \sum_{i=1}^{n} \lfloor (\beta \cdot n)/i \rfloor. \tag{3.43}$$

Umjesto funkcije poda $\lfloor x \rfloor$ može se koristiti i klasično zaokruživanje na najbliži cijeli broj.

Nakon što je završen proces kloniranja, pristupa se hipermutacijama klonova. Pri tome je intenzitet hipermutacije obrnuto proporcionalan afinitetu antitijela. Izvorno, autori predlažu da se koristi vjerojatnosna distribucija određena izrazom:

$$p = e^{-\rho \cdot f} \tag{3.44}$$

gdje je ρ korisnički definiran parametar, a f normalizirana vrijednost afiniteta. Izraz koji se također često koristi je:

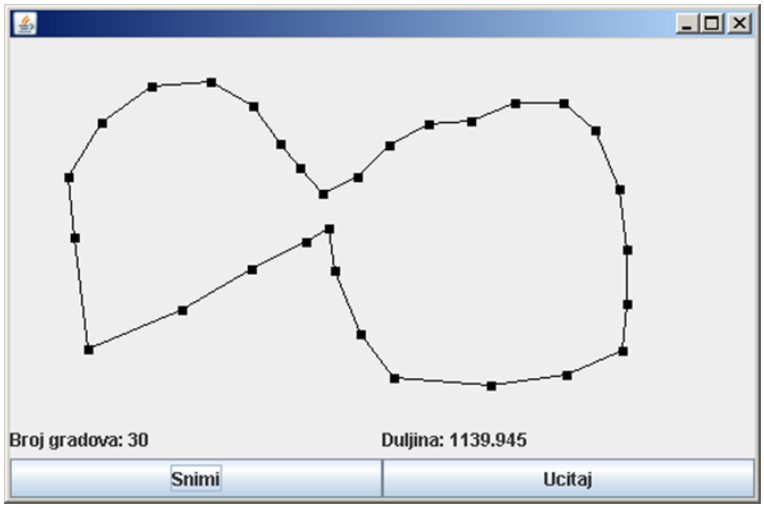
$$p = \frac{1}{\rho} e^{-f} \tag{3.45}$$

Nakon što je operatorom hipermutacije stvorena populacija P^{hyp}, računaju se afiniteti svih stvorenih antitijela obzirom na antigen. Potom se novu populaciju odabire n najboljih antitijela iz populacije P^{hyp}. Dodatno, kako bi se diverzificirala populacija (unio novi genetski materijal), stvara se d novih antitijela (slučajnim mehanizmom), i tih d antitijela zamjenjuje d antitijela s najmanjim afinitetom.

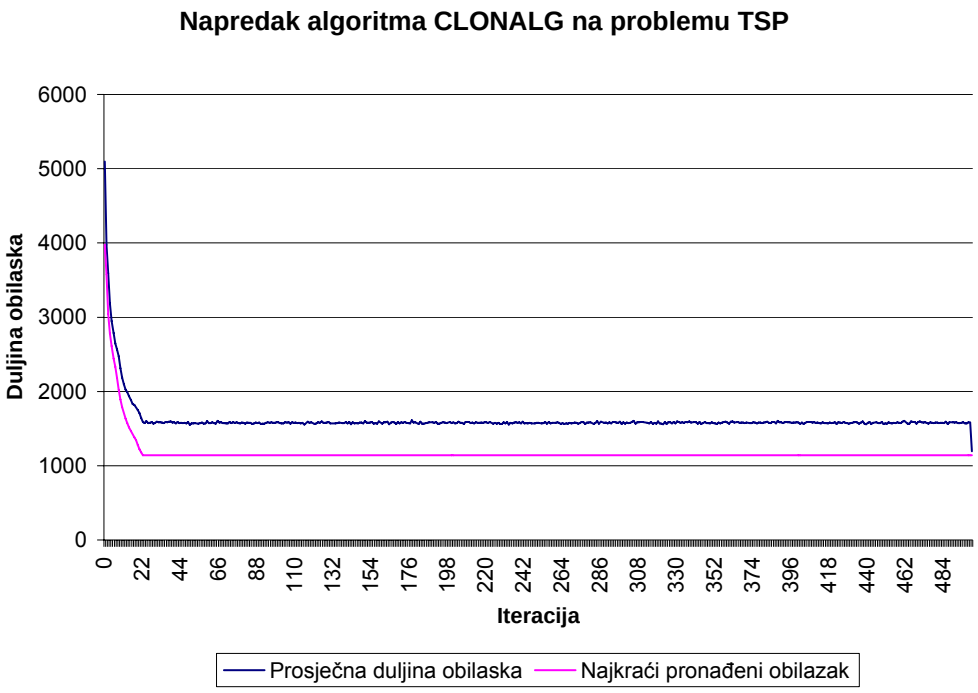
Analiza složenosti samog algoritma može se pogledati u [Cutello and Nicosia, 2005]. Ona je veća od prethodno opisanog algoritma jer uključuje i sortiranja kompletnih populacija.

Kako bi se provjerio rad algoritma, načinjena je implementacija u programskom jeziku Java. Rezultat izvođenja algoritma prikazan je na slici 3.43, dok je napredak algoritma po iteracijama prikazan na slici 3.44.

Rezultati su dobiveni uz $n = 200$, $d = 20$ i $\beta = 5$.



Slika 3.43: Problem obilaska 30 gradova riješen algoritmom CLONALG.



Slika 3.44: Napredak algoritma CLONALG na problemu TSP s 30 gradova.

Način implementacije operatora hipermutacije u tom primjeru opisan je u nastavku. Kako je svaki antigen predstavljen kao niz indeksa gradova, mutacija se izvodi zamjenom gradova koje treba posjetiti u dva slučajno odabrana koraka. Ovih se zamjena pri tome obavlja to više, što je afinitet antitijela manji. Odabrano je da se broj zamjena kreće od 1 za najbolje antitijelo pa sve do $1+l\cdot\rho$ za najlošije antitijelo (gdje je l broj gradova, ρ pozitivna konstanta manja od 1). Postavljen je zahtjev da točan broj (k) bude određen izrazom:

$$k = 1 + l \cdot \left(1 - e^{-r/\tau}\right) \tag{3.46}$$

pri čemu je r rang antitijela (0 za najbolje antitijelo, 1 za sljedeće, itd). Promatra li se k kao funkciju od r, iz zahtjeva: $k(0) = 0$ i $k(n-1) = 1 + l \cdot \rho$ slijedi

$$\tau = -\frac{n-1}{ln(1-\rho)}. \tag{3.47}$$

Za potrebe eksperimenta odabrano je $\rho = 0.25$, što uz $n = 200$ daje $\tau \approx 691.736$ (maksimalni rang jednak je $n-1$).

Kako bi se očuvalo najbolje rješenje, jedan klon antitijela ranga 0 direktno je propušten u populaciju P^{hyp} (na tom jednom klonu najboljeg primjerka zabranjena je mutacija).

3.6.3 Pregled korištenih operatora

U nastavku su ukratko navedene vrste operatora koje se danas uobičejeno koriste kod imunoloških algoritama. Detaljna razmatranja ovih operatora kao i dokaz konvergencije imunoloških algoritama nalazi se u [Cutello et al., 2007].

3.6.3.1 Operator kloniranja

Operator kloniranja zadužen je za stvaranje populacije klonova iz postojeće populacije antitijela. Operator statičkog kloniranja (engl. *static cloning operator*) [Cutello and Nicosia, 2002a] svako antitijelo iz izvorne populacije klonira *dup* puta, čime stvara populaciju klonova P^{clo} veličine $d \cdot dup$. Operator proporcionalnog kloniranja (engl. *proportional cloning operator*) [de Castro and Zuben, 2002] za svako antitijelo stvara broj klonova koji je proporcionalan afinitetu antitijela (tako da bolja antitijela dobiju više klonova). Vjerojatnosno kloniranje (engl. *probabilistic cloning*) [Cutello et al., 2003]

definira parametar p_c (engl. *clonal selection rate*) temeljem kojeg iz izvorne populacije bira antitijela koja će biti klonirana.

3.6.3.2 Operatori mutacije

Operator mutacije djeluje nakon kloniranja i tipično u populaciju unosi slijepe promjene nad genima. Broj mutacija koje će se načiniti nad jednim antitijelom određen je potencijalom slučajne mutacije (engl. *random mutation potential*) [Cutello et al., 2005].

Kod statičke hipermutacije (engl. *static hypermutation*) broj mutacija ne ovisi o funkciji dobrote $f(x)$ već je ograničen nekom konstantom c.

Kod proporcionalne hipermutacije (engl. *proportional hypermutation*) broj mutacija nad jednim antitijelom proporcionalan je funkciji dobrote $f(x)$ te je određen izrazom $(f(x) - f^*) \cdot (c \cdot l)$. Pri tome je f^* minimalna funkcija dobrote antitijela iz trenutne populacije.

Kod inverzno proporcionalne hipermutacije (engl. *inversely proportional hypermutation*) broj mutacija nad jednim antitijelom obrnuto je proporcionalan funkciji dobrote $f(x)$ te je određen izrazom $(1 - \frac{f^*}{f(x)}) \cdot (c \cdot l) + (c \cdot l)$ gdje je f^* minimalna funkcija dobrote antitijela iz trenutne populacije.

Kod hipermakromutacije (engl. *hypermacromutation*) broj mutacija ne ovisi niti o funkciji dobrote, niti o konstanti c. Mutacija se radi tako da se slučajno odaberu dva indeksa i i j tako da vrijedi $1 \leq i < j \leq l$ (gdje je l broj gena) i potom se s određenom vjerojatnošću mutiraju svi geni u nizu počev od i-tog pa do j-tog.

3.6.3.3 Operator starenja

Starenje je operator koji osigurava da antitijelo ne ostane predugo u populaciji.

Kod statičkog operatora starenja (engl. *static pure aging operator*) definira se vrijeme τ_B kao broj iteracija koje antitijelo može preživjeti u populaciji. Po isteku tog vremena, antitijelo se briše iz populacije, neovisno o njegovoj dobroti. Kod ove vrste algoritama, operator kloniranja klonovima prepisuje starost roditelja. Potom, nakon faze vrednovanja klonovima koji su bolji od svojih roditelja starost se resetira na 0. Elitistička verzija algoritma postiže se tako da se u svakoj iteraciji starost najboljeg antitijela također resetira na 0 (čime se osigurava da najbolje antitijelo nikada ne bude uništeno).

Kod stohastičkog operatora starenja (engl. *stohastic aging operator*) antitijelo iz populacije može biti izbrisano i prije isteka τ_B, što je definirano vjerojatnošću preživljavanja koja se smanjuje povećanjem starosti antitijela. Elitistička verzija algoritma tada se dobiva tako da se vjerojatnost preživljavanja za najbolje antitijelo uvijek postavi na 1.

3.6.4 Druga područja

U okviru ovog odjeljka od imunoloških algoritama prikazano je samo jedno usko područje koje se temelji na primjeni principa klonske selekcije. Radi potpunosti pregleda, nužno je spomenuti da danas postoje četiri glavna područja istraživanja: algoritmi negativne selekcije (engl. *negative selection algorithms* – NSA), algoritmi imunoloških mreža (engl. *immune network algorithms* – INA), algoritmi zasnovani za teoriji opasnosti (engl. *danger theory algorithms* – DTA) te algoritmi klonske selekcije (engl. *clonal selection algorithms*).

Poglavlje 4

Formalne definicije odabranih problema raspoređivanja

U okviru ovog poglavlja dane su formalne definicije problema raspoređivanja nastavnih obaveza koji se razmatraju u nastavku ove disertacije. U poglavlju 2 dan je kratak pregled problema raspoređivanja nastavnih obaveza o kojima se uobičajeno piše u znanstvenoj literaturi. Jedan od dominantnih problema iz te kategorije je problem izrade rasporeda predavanja na sveučilištu. Kako je taj problem ekstenzivno studiran i kako je već pokazano da su algoritmi evolucijskog računanja prikladni za njegovo rješavanje, ovdje dalje nije razmatran. U okviru ove disertacije razmatraju se problemi raspoređivanja nastavnih aktivnosti navedeni u nastavku, od kojih neki uopće nisu zastupljeni u znanstvenoj literaturi, a oni koji jesu, zastupljeni su u pojednostavljenom obliku.

1. *Jednostavno raspoređivanje.* Karakterizira ga jednostavnost u smislu da su termini fiksni i treba samo rasporediti studente uzimajući u obzir prethodna zauzeća. Varijanta je definiranje lista termina kojima studenti moraju prisustvovati.

2. *Raspoređivanje neraspoređenih studenata po predavanjima.* Naknadno upisane studente treba uklopiti u satnicu predavanja poštujući ograničenja na kapacitete grupa i postojeći nepromjenjiv raspored predavanja.

3. *Uklanjanje konflikata u satnici za predavanja na razini studenata.* Studentima kojima nakon izrade rasporeda predavanja ostanu konflikti treba promjeniti razmještaj po grupama (uz minimalnu promjenu) poštujući i dalje ograničenja na kapacitete grupa.

4. *Raspored laboratorijskih vježbi.* Na ovaj problem možemo gledati kao na vrstu općenitog problema izrade rasporeda predavanja na sveučilištu, gdje treba stvoriti razdiobu po grupama te dodjelu termina i studenata po terminima poštujući prethodna zauzeća studenata predavanjima.

5. *Raspored provjera znanja.* Kolegije koji imaju ispite treba rasporediti po unaprijed zadanom skupu termina pazeći na kapacitete termina, konflikte između studenata i niz drugih ograničenja.

6. *Raspored prostorija za provjere znanja.* Podrazumijeva dodjelu prostorija provjerama znanja iz unaprijed zadanog skupa prostorija u skladu s preferencijama kolegija. Radi se nakon što je za svaki kolegij definiran termin u kojem se održava njegova provjera znanja.

7. *Raspoređivanje timova.* Problem karakterizira postojanje unaprijed definiranih timova studenata te njima dodijeljenih asistenta koji ih vode i provode ispitivanje. Timove treba razmjestiti u unaprijed definiran skup termina tako da nemaju preklapanja s drugim postojećim obavezama i tako da asistent u jednom terminu ima najviše jedan dodijeljeni tim koji treba ispitati.

8. *Izrada prezentacijskih grupa za seminare.* Ovo je primjer problema višekriterijske optimizacije gdje seminarske mikrogrupe treba spojiti u veće prezentacijske grupe tako da čitavoj grupi ostane što je moguće više slobodnih termina za prezentacije te da se očuva "prosječna" kvaliteta prezentacijskih grupa.

4.1 Jednostavno raspoređivanje

Pod pojmom jednostavno raspoređivanje (engl. *grouping subproblem, student sectioning*) smatramo problem koji se sastoji od jednog događaja koji se odvija u nekoliko alternativnih slijedova događaja. Najjednostavniji slučaj je prikazan na slici 4.1; postoji jedan događaj koji ima 4 moguća termina u kojem ga je moguće odraditi; termini su pri tome unaprijed definirani, i način njihovog odabira nije dio ovog problema. Svakog studenta iz skupa studenata S pri tome treba rasporediti u jedan od ta četiri termina (T1-A, T1-B, T1-C i T1-D), pazeći pri tome tome na prethodna zauzeća tog studenata te na maksimalni broj studenata koji je moguće smjestiti u pojedini termin. Jednos-

tavan pristup rješavanju ovog problema bio bi iscrpnom pretragom: svakom studenatu s_i iz skupa S kardinaliteta n možemo odabrati jedan od termina t_j iz skupa termina T kardinaliteta m. Iscrpna pretraga znači isprobavanje svih mogućih kombinacija kojih u ovom slučaju ima:

$$m \cdot m \cdot \cdots \cdot m = m^n. \tag{4.1}$$

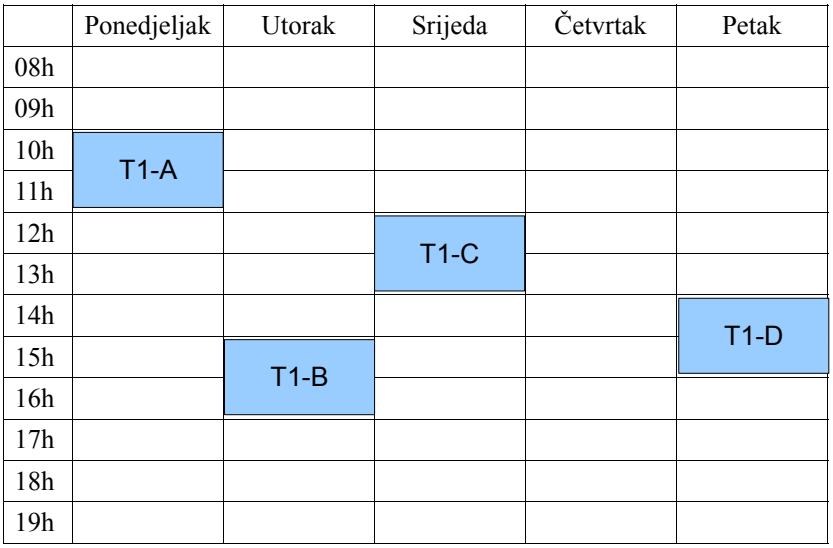
U nekom stvarnom primjeru za $n = 672$ studenta te $m = 4$ slijedi da je broj kombinacija $\approx 10^{405}$. Dakako, velik dio tih kombinacija bit će neispravan jer ne poštuje ograničenje kapaciteta termina a tek preostali dio će biti rješenja koja poštuju ograničenja termina ali imaju problema s konfliktima studenata s njihovim prethodnim zauzećima.

Općenitiji slučaj koji se javlja u praksi ilustriran je slikom 4.2. Zadan je događaj koji se sastoji od dva termina (primjerice, predavanja); pri tome jedan nastavnik prvo predavanje ima u ponedjeljak od 10h do 12h a drugo u srijedu od 12h do 14h, a drugi nastavnik ima prvo predavanje u utorak od 15h do 17h a drugo predavanje u petak od 14h do 16h. Za ovakav problem kažemo da ima slijed od dva termina, te da ima dva alternativna slijeda. Studente pri tome treba rasporediti tako da budu pridijeljeni na jedan od slijedova; nije dozvoljeno prvo predavanje odraditi kod jednog nastavnika a drugo kod drugog. Kombinatorička složenost problema ne ovisi o broju termina u slijedu, već samo o broju alternativnih slijedova. Tako su termini prvog slijeda na slici 4.2 označeni slovom A a drugog slijeda slovom B.

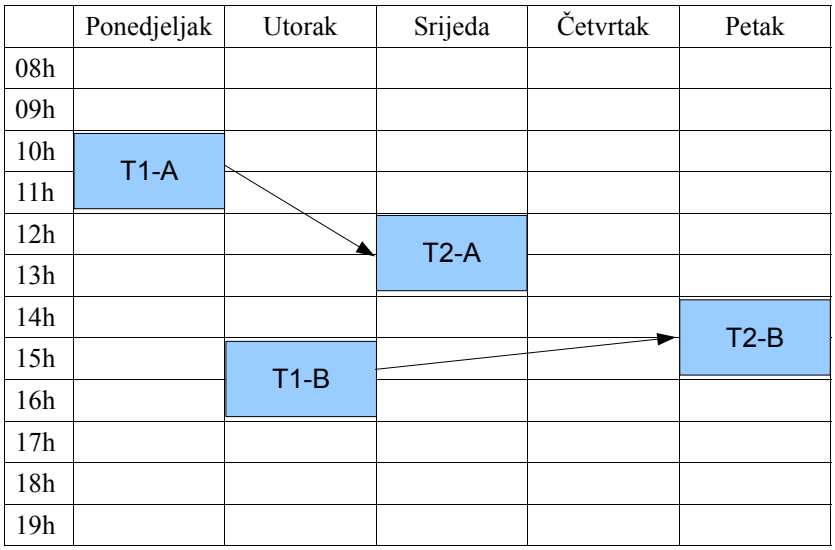
Također, za razliku od situacije prikazane na slikama 4.1 i 4.2, termini slijedova ne moraju nužno biti ograničeni na jedan tjedan već je moguća situacija gdje se termini protežu kroz dva ili više tjedna. Postojeća zauzeća studenata pri tome će iz tjedna u tjedan, u općem slučaju, biti različita.

Uz prethodna pojašnjenja, slijedi formalna definicija opisanog problema.

Zadan je skup studenata S kardinaliteta n. Za svakog studenta $s_i \in S$ zadan je skup postojećih zauzeća $P_i = \{p_{i,1}, \ldots, p_{i,l_i}\}$ gdje je l_i broj prethodnih zauzeća studenta s_i; tipično, zauzeća $p_{i,j}$, $i \in \{1, n\}$, $j \in \{1, \ldots, l_i\}$ će imati datum zauzeća, vrijeme početka zauzeća te vrijeme kraja zauzeća (npr. `2010-03-18 12:00 13:30`). U praksi, ovaj će podatak biti proširen dodatnim informacijama (primjerice, dodijeljenom prostorijom) koje će omogućiti naknadnu objavu rasporeda; međutim, za formalnu definiciju ovog problema te nam informacije nisu bitne.



Slika 4.1: Primjer jednostavnog rasporeda (1) – događaj ima samo jedno predavanje u četiri alternativna termina.



Slika 4.2: Primjer jednostavnog rasporeda (2) – događaj ima dva predavanja, i postoje dvije serije termina.

Zadan je skup slijedova $E = \{e_1, \ldots, e_m\}$ gdje je m kardinalitet tog skupa. Svakom slijedu e_1 pridijeljen je k-člani skup termina $T_i = \{t_{i,1}, \ldots, t_{i,k}\}$, pri čemu je svaki termin $t_{i,j}$, $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, k\}$ opisan datumom, vremenom početka i vremenom kraja; pri tome su termini $t_{i,a} \in T_i$ i $t_{i,b} \in T_i$, $a \neq b$, vremenski disjunktni. Termini različitih slijedova, dakako, ne moraju biti disjunktni (jer mogu ići istovremeno ali u različitim prostorijama). Svaki slijed ima jednak broj termina: k.

Za svaki slijed $e_i \in E$ definiran je kapacitet C_i; to je najveći broj studenata koji je moguće pridijeliti tom slijedu. U praksi, ovo će ograničenje biti rezultat raspoloživog kapaciteta prostorija koje su rezervirane za svaki termin $t_{i,j}$ slijeda e_i. Štoviše, ako su različiti termini smješteni u različite prostorije, kapacitet slijeda C_i bit će određen kapacitetom najmanje pridijeljene prostorije.

U svrhu dobivanja balansiranog rasporeda, idealno bi bilo postaviti sva ograničenja kapaciteta na jednaku vrijednost, tj. $\forall i \in \{1, \ldots, m\}$, $C_i = \lceil \frac{n}{m} \rceil$ (iako nije nužno).

Označimo s $x_{i,j}$ varijablu koja poprima vrijednost 1 ako je i-tom studentu dodijeljen j-ti slijed a 0 inače; ovih varijabli ima ukupno $n \times m$. Neka je zadana funkcija *preklapanje*(A, B) čiji su argumenti dva skupa termina a rezultat broj minuta preklapanja između tih skupova termina. Potrebno je pronaći takvu dodjelu vrijednosti varijablama $x_{i,j}$ da vrijedi:

$$\sum_{j=1}^{m} x_{i,j} = 1 \qquad (i = 1, \ldots, n) \tag{4.2}$$

$$\sum_{i=1}^{n} x_{i,j} \leq C_j \qquad (j = 1, \ldots, m) \tag{4.3}$$

$$x_{i,j} = 0 \text{ ili } 1 \qquad (i = 1, \ldots, n; j = 1, \ldots, m) \tag{4.4}$$

$$x_{i,j} \cdot \textit{preklapanje}(P_i, e_j) = 0 \qquad (i = 1, \ldots, n; j = 1, \ldots, m) \tag{4.5}$$

Ograničenja definirana izrazom (4.2) osiguravaju da je svakom studentu dodijeljen točno jedan slijed; ovih ograničenja ima ukupno n. Ograničenja definirana izrazom (4.3) osiguravaju da niti jednom slijedu nije dodjeljeno više studenata no što je kapacitet tog slijeda; ovih ograničenja ima ukupno m. Ograničenja definirana izrazom (4.4) osiguravaju da vrijednosti varijabli budu iz domene varijabli; ovih ograničenja ima ukupno $n \times m$. Konačno, ograničenja definirana izrazom (4.5) osiguravaju da načinjenom dodjelom studenti nemaju konflikata s njihovim prethodnim zauzećima; ovih ograničenja

također ima $n \times m$; međutim, besmisleno ih je sve provjeravati jer je u $n \times m - n$ slučajeva vrijednost $x_{i,j} = 0$ pa je tada i ograničenje zadovoljeno. Umjesto ograničenja (4.5) može se koristiti ograničenje (4.6):

$$\text{ako je } x_{i,j} = 1 \text{ tada } \mathit{preklapanje}(P_i, e_j) = 0 \qquad (i = 1, \ldots, n; j = 1, \ldots, m) \tag{4.6}$$

iz kojeg je jasnije da je provjeru potrebno raditi samo za n načinjenih dodjela, jer će, zbog ograničenja (4.2), (4.3) i (4.4) u samo n slučajeva $x_{i,j}$ biti jednaka 1.

4.1.1 Optimizacijski problem

Prethodna formalna definicija problema je takva da će, ili imati valjano rješenje, ili rješenje neće postojati. Ako rješenje ne postoji, uz pretpostavku da razlog tome nisu loše zadana ograničenja (dakle, da nije zadan slučaj poput $\sum_{i=1}^{m} C_i < n$ ili nešto slično), razlog će biti nemogućnost raspodjele studenata na način da se izbjegnu preklapanja sa svim prethodnim obavezama.

U tom slučaju, prethodnom problemu se može pristupiti tako da se definira mjera nezadovoljenja ograničenja (4.5) ili (4.6). Neka je funkcija $\mathit{kazna}(\mathbf{X})$:

$$\mathit{kazna}(\mathbf{X}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(x_{i,j} \cdot \mathit{preklapanje}(P_i, e_j) \right). \tag{4.7}$$

Sada se uz prethodno definirana ograničenja (4.2)-(4.4) može tražiti takva dodjela vrijednosti varijablama $x_{i,j}$ (odnosno matricu $\mathbf{X}$) koja minimizira funkciju $\mathit{kazna}(\mathbf{X})$.

Valjano rješenje, ako postoji, imat će $\mathit{kazna}(\mathbf{X}) = 0$; s druge pak strane, ako valjanog rješenja nema, tada će se rješavanjem ovako postavljenog optimizacijskog problema dobiti rješenje koje je najbolje moguće.

4.1.2 Inačice problema

Osim prethodne relaksacije ograničenja (4.5) što je rezultiralo optimizacijskim problemom minimizacije funkcije (4.7), moguće je definirati još neke relaksacije. Jedan od primjera je relaksacija ograničenja (4.3) tako da se dopusti prekoračenje kapaciteta slijeda u nadi da će se time razriješiti problem preklapanja nastalog rasporeda s prethodnim zauzećima studenata. Tako se umjesto izraza (4.3) može definirati funkcija

prekapacitiranost($\mathbf{X}$) na sljedeći način:

$$prekapacitiranost(\mathbf{X}) = \sum_{j=1}^{m} \left(-min(C_j - \sum_{i=1}^{n} x_{i,j}, 0) \right). \tag{4.8}$$

Drugi pristup je promatrati balansiranost raspodjele studenata. Naime, prethodno je već spomenuto da bi bilo dobro kada bi se moglo osigurati da broj studenata raspoređenih u svaki slijed bude upravo $C_i = \lceil \frac{n}{m} \rceil$, uz pretpostavku da su prostorije dodijeljene pojedinim slijedovima prikladne veličine da takva razdioba bude moguća. Ako su stvarni kapaciteti veći te ih se umjetno spusti na ovu vrijednost, javlja se rizik da više nije moguće načiniti raspored bez preklapanja. Stoga bi se umjesto umjetnog spuštanja kapaciteta mogao definirati očekivani kapacitet slijeda C_i^* kako slijedi:

$$C_i^* = n \cdot \frac{C_i}{\sum_{j=1}^{m} C_j} \qquad (i = 1, \ldots, m). \tag{4.9}$$

Za slučaj potencijalno mogućeg rasporeda, dakle kada je $\sum_{j=1}^{m} C_j \geq n$, uvijek je ispunjeno $C_i^* \leq C_i$. Sada se funkcija *nebalansiranost*((X)) može definirati na sljedeći način:

$$nebalansiranost(\mathbf{X}) = \sum_{j=1}^{m} \left| C_j^* - \sum_{i=1}^{n} x_{i,j} \right|. \tag{4.10}$$

U izrazu (4.10) za svaki se slijed gleda apsolutna razlika između očekivanog broja studenata i dodijeljenog broja studenata, i ta se razlika pridodaje u ukupno mjeru nebalansiranosti. Zadatak optimizacije može biti ovu mjeru smanjiti na najmanju moguću. Za slučaj jednakih kapaciteta slijedova, dakle za $C_1 = C_2 = \ldots = C_m = C$ očekivani srednji kapacitet slijeda prelazi u $\frac{n}{m}$:

$$C_i^* = n \cdot \frac{C_i}{\sum_{j=1}^{m} C_j} = n \cdot \frac{C}{m \cdot C} = \frac{n}{m}. \tag{4.11}$$

Osim navedenog, moguće je postaviti uvjet i na kvalitetu dobivenog rasporeda sa stanovišta svakog pojedinog studenta. Neka je definirana funkcija *udaljenost*(x, T) gdje je x termin (dakle zadan datumom, početkom i krajem) a T proizvoljan skup termina (svaki zadan datumom, početkom i krajem). Neka funkcija *udaljenost*(x, T) vraća udaljenost (u minutama) od termina x do najbližeg termina iz skupa termina T, odnosno minimum između kraja najbližeg termina iz T koji završava prije početka termina x i

početka termina x te između početka najbližeg termina iz T koji počinje nakon kraja x i kraja od termina x. Uporabom ovako definirane funkcije može se definirati funkcija $nekvaliteta(\mathbf{X})$, koja procjenjuje koliko se loše načinjeni raspored uklapa u postojeća zauzeća svakog studenta:

$$nekvaliteta(\mathbf{X}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(x_{i,j} \cdot \sum_{t \in e_j} udaljenost(t, P_i \cup e_j \setminus \{t\}) \right). \tag{4.12}$$

Uz ovako definiranu funkciju $nekvaliteta(\mathbf{X})$, jedan od ciljeva minimizacije može biti pronaći takvo rješenje koje minimizira uočenu nekvalitetu. U idealnom slučaju, ako je moguće pronaći valjani raspored (dakle onaj kod kojeg je minimum izraza (4.7) jednak 0), te raspored koji nema kršenja kapaciteta slijedova (dakle kod kojeg je minimum izraza (4.8) jednak 0), kriterij (4.12) može poslužiti za daljnje poboljšanje rasporeda kako bi studenti imali zauzeća koja se nastavljaju na njihova postojeća. U konačnici, ovaj se problem može svesti na minimizaciju višekriterijske funkcije, pri čemu su kriteriji:

- $kazna(\mathbf{X})$ – mjera u kojoj su količini razriješena preklapanja dodijeljenih termina i postojećih obaveza pojedinih studenata,
- $prekapacitiranost(\mathbf{X})$ – mjera u kojoj su mjeri zadovoljena ograničenja kapaciteta slijedova,
- $nebalansiranost(\mathbf{X})$ – mjera u kojoj su mjeri zadovoljena očekivanja da broj studenata u slijedu bude jednak prosječnom očekivanom kapacitetu slijedova, te
- $nekvaliteta(\mathbf{X})$ – mjera koliko se loše načinjeni raspored uklapa u postojeća zauzeća studenata.

Ovakvom definicijom problema omogućava se njegovo rješavanje bilo primjenom algoritama višekriterijske optimizacije, bilo svođenjem na jednokriterijsku optimizaciju pa uporaba odgovarajućih algoritama.

4.2 Raspoređivanje neraspoređenih studenata po predavanjima

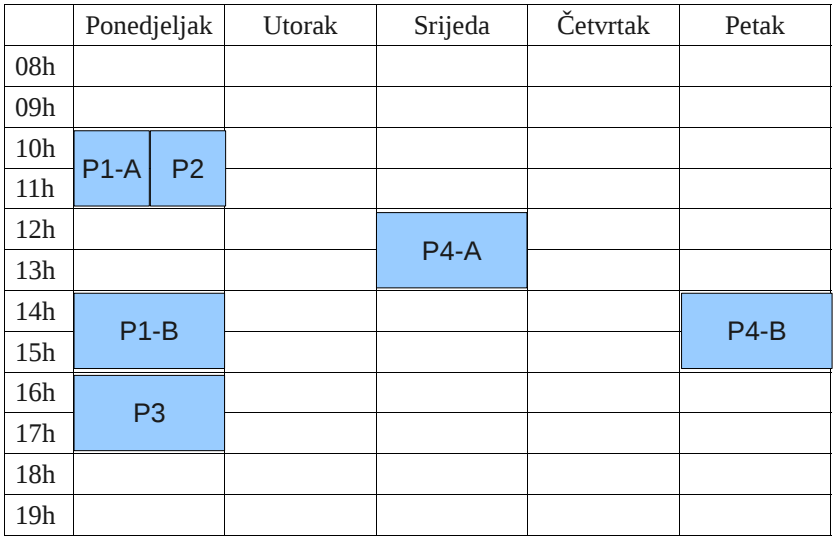
Jedan od problema koji se svaki semestar javlja na sveučilištu je naknadni upis studenata. Naime, administrativni procesi na početku semestra tako su postavljeni da se do trenutka početka nastave završe upisi studenata na kolegije, generiraju rasporedi predavanja za grupe za predavanja, te razmjeste studenti u grupe za predavanja, na način koji će što je moguće većem broju studenata omogućiti pohađanje predavanja; tako je to barem u teoriji.

Praksa je nažalost nešto drugačija, pa se zna dogoditi da određen postotak studenata do zadanog roka ne bude upisan; bilo zbog nepredane prijave, opravdane spriječenosti ili nekog trećeg razloga. Posljedica je pojava određenog broja studenata koje je potrebno upisati nakon što je završena izrada rasporeda predavanja i nakon što su oformljene i popunjene grupe za predavanja.

Problem koji se ovdje javlja je na koji način studente koji se naknadno upisuju rasporediti u grupe za predavanja. Naime, kako su procesi raspoređivanja predavanja završeni, i kako su svim grupama dodijeljeni termini za predavanja te prostorije u kojima se ta predavanja odvijaju, broj studenata koje je moguće smjestiti u pojedinu grupu određen je razlikom između kapaciteta prostorije i broja studenata koji su već smješteni u tu grupu. Koristi li se tehnika *prvi-upis-prvi-bira*, lagano se može dogoditi situacija da odabir jednog studenta koji je mogao birati između više mogućih termina onemogući sljedećeg studenta da načini odabir uz koji neće imati preklapanja (uz pretpostavku da više ne može birati grupe koje su popunjene). Ovo postaje očito ako se razmotri primjer raspored predavanja prikazan slikom 4.3.

Za potrebe primjera prikazan je raspored za četiri kolegija (P_1-P_4). Svi kolegiji imaju po 2 sata predavanja; pri tome su, zbog velikog broja studenata, na kolegijima P_1 i P_4 otvorene dvije grupe (A i B). Neka u svakoj od grupa P_1-A i P_1-B ima jedno slobodno mjesto. Neka na drugim kolegijima ima više slobodnih mjesta. Potrebno je obaviti upis dva studenta. Prvi student upisuje predmete P_1, P_3 i P_4, dok drugi student upisuje predmete P_1, P_2 i P_4.

Ako se smještanje studenata u grupe za predavanja obavi redosljedom upisa, prvi bi student mogao, primjerice, odabrati da predmet P_1 sluša u grupi B, predmet P_3 u

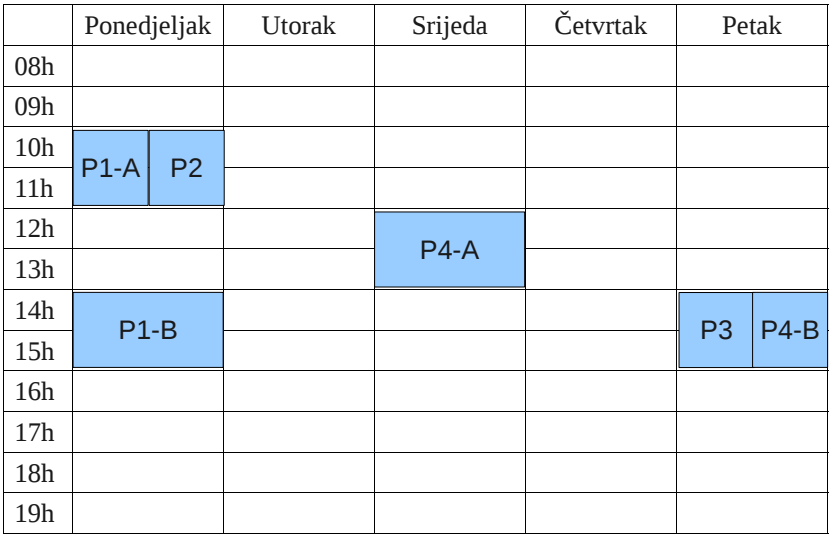


Slika 4.3: Raspoređivanje naknadno upisanih studenata – primjer rasporeda četiri kolegija.

jedinom terminu u kojem se drži, te predmet P_4 u grupi B. Nakon ovog odabira, drugi student više ne može izabrati raspored u kojem može pohađati sva predavanja. Naime, kako je prethodni student zauzeo jedino slobodno mjesto u grupi P_1-B, ovaj student na kolegiju P_1 može izabrati isključivo grupu P_1-A koja ima predavanja istovremeno kada su i predavanja kolegija P_2 koji je student također upisao. Međutim, da je prvi student načinio drugačiji odabir (ili da je drugi student prvi birao), mogao se je načiniti takav izbor uz koji oba studenta mogu složiti takav raspored uz koji će moći pohađati sva upisana predavanja.

Prethodni primjer, zbog svoje bi jednostavnosti mogao navesti na krivi zaključak – da je dovoljno samo "pametno" složiti redosljed kojim studenti biraju grupe. Da to nije tako, ilustrirano je na slici 4.4.

Neka je situaciju slična prethodnom primjeru: prvi student upisuje P_1, P_3 i P_4, dok drugi student upisuje predmete P_1, P_2 i P_4. Također, neka je u svakoj od grupa P_1-A, P_1-B, P_4-A i P_4-B preostalo po jedno slobodno mjesto. Neka sada bira prvi student i neka odabere P_1-B, P_3 i P_4-A; drugi student uz ovakav odabir može birati samo P_1-A, P_2 i P_4-B čime ima koliziju predavanja P_1-A i P_2. Neka se zamjeni redosljed pa dopusti da drugi student bira prvi. Neka on odabrere P_1-B, P_2 i P_4-A; sada drugi student može odabrati samo P_1-A, P_3 i P_4-B, čime ima koliziju predavanja P_3 i P_4-B. Naravno, postoji i rješenje gdje oba studenta mogu slušati sva upisana predavanja: prvi mora



Slika 4.4: Raspoređivanje naknadno upisanih studenata – primjer rasporeda četiri kolegija.

odabrati P_1-A, P_3 i P_4-A a drugi P_1-B, P_2 i P_4-B.

Opisani slučajevi niti po čemu nisu posebni; dapače, u prisustvu više studenata broj kombinacija u kojima netko nekoga može blokirati raste, i postavlja se pitanje kako načiniti ovaj odabir. Problem je opisan u radu [Čupić et al., 2010], a ideja je zasebno raditi upise ionako zakašnjelih studenata i čim se skupi "kritična" masa, pokrenuti postupak raspoređivanja koji će ovaj problem pokušati riješiti na način da ga gleda cjelovito.

Kao dopunu prethodnog opisa treba spomenuti da se ovakav problem ne mora javljati samo sa studentima koji kasne s upisom svih predmeta. Često se javlja i situacija da se uspješno i na vrijeme završi upis jednog dijela predmeta, a zakasni samo s nekim od predmeta (primjerice zbog predaje molbe za naknadnu promjenu upisanog predmeta ili zbog drugih razloga). U takvom slučaju student već ima djelomično fiksiran raspored predavanja i odabir za preostale predmete treba načiniti tako da se dobije raspored u kojem student može slušati sve upisane kolegije.

Formalno, ovaj problem može se opisati na sljedeći način. Zadan je skup kolegija $C = \{c_1, \ldots, c_o\}$. Svaki kolegij ima predavanja tijekom semestra organiziranih u jedan ili više slijedova termina (u konačnici, svaki slijed predstavlja jednu grupu za predavanja). Pri tome je kolegiju c_i pridružen skup slijedova $E_i = \{e_{i,1}, e_{i,2}, \ldots\}$, pri čemu je svakom slijedu $e_{i,j}$ pridružen skup termina $T_{i,j} = \{t_{i,j,1}, t_{i,j,2}, \ldots\}$. Kardinaliteti skupo-

va slijedova E_i i E_j, $i \neq j$ pri tome ne moraju biti jednaki. Masovniji kolegiji će imati više slijedova, manji kolegiji manje. Za svaki slijed $e_{i,j}$ kolegija c_i definiran je kapacitet slijeda $C_{i,j}$ – broj studenata koji mogu biti pridruženi na taj slijed, te zauzeće slijeda $N_{i,j}$ – to je broj studenata koji su već smješteni u taj slijed. Broj slobodnih mjesta u slijedu tada je jednak $C_{i,j} - N_{i,j}$.

Neka je $S = \{s_1, \ldots, s_n\}$ skup studenata koji su neraspoređeni na barem jednom kolegiju. Neka su s $u_{i,j}$ označeni indikatori upisa predmeta; $u_{i,j} = 1$ ako je student s_i upisao kolegij c_j i na njemu je neraspoređen; inače je $u_{i,j} = 0$. Za svakog studenta $s_i \in S$ zadan je skup postojećih zauzeća $P_i = \{p_{i,1}, \ldots, p_{i,l_i}\}$ gdje je l_i broj prethodnih zauzeća studenta s_i. Za studenta s_i koji nije raspoređen niti na jednom kolegiju, odgovarajući skup prethodnih zauzeća bit će prazan skup, tj. $P_i = \emptyset$. Ako je student prethodno bio raspoređen na nekim kolegijima, skup P_i bit će jednak uniji pridruženih slijedova na upisanim i raspoređenim kolegijima.

Problem razmještanja studenata u grupe za predavanja sada se može definirati kako slijedi. Neka je $x_{i,j,k} = 1$ ako je studentu s_i na predmetu c_j dodijeljen slijed $e_{j,k}$; inače je $x_{i,j,k} = 0$. Sljedeća ograničenja moraju vrijediti:

$$\sum_{k=1}^{|E_j|} x_{i,j,k} = u_{i,j} \qquad (i = 1, \ldots, n; j = 1, \ldots, o) \tag{4.13}$$

$$\sum_{i=1}^{n} x_{i,j,k} \leq C_{j,k} - N_{j,k} \qquad (j = 1, \ldots, o; k = 1, \ldots, |E_j|) \tag{4.14}$$

$$x_{i,j,k} = 0 \text{ ili } 1 \qquad (i = 1, \ldots, n; j = 1, \ldots, m; k = 1, \ldots, |E_j|) \tag{4.15}$$

$$\mathit{preklapanje}(P_i^*) = 0 \qquad (i = 1, \ldots, n) \tag{4.16}$$

Ograničenja definirana izrazom (4.13) osiguravaju da je student s_i na kolegiju c_j smješten točno u jedan slijed (ako je upisao taj kolegij), odnosno da na tom kolegiju nije razmješten ako kolegij nije upisao. Ograničenja definirana izrazom (4.14) osiguravaju da je broj studenata razmještenih u slijed $e_{j,k}$ kolegija c_j i dalje manji ili jednak broju slobodnih mjesta u tom slijedu. Izraz (4.15) ograničava domenu varijabli $x_{i,j,k}$ na 0 ili 1. Ograničenja zadana izrazom (4.16) osiguravaju da u načinjenoj dodjeli student nema preklapanja s prethodnim postojećim obavezama kao niti preklapanja između dodijeljenih termina. Pri tome je kao argument P_i^* funkciji *preklapanje*() predan multiskup čiji

su elementi skupovi svih postojećih zauzeća tog studenta. Ovaj multiskup definiran je na sljedeći način:

$$P_i^* = \{P_i\} \cup \{e_{j,k} \,|x_{i,j,k} = 1; j = 1, \ldots, m; k = 1, \ldots, |E_j|\} \tag{4.17}$$

gdje je P_i skup prethodnih zauzeća studenta s_i; simbol $\cup$ ovdje se koristi za dodavanje elemenata u multiskup. Ono što se nadodje su skupovi termina pridruženi onim slijedovima $e_{j,k}$ na koje je student s_i smješten, tj. za koje je $x_{i,j,k} = 1$. Time je na sintaksnoj razini promijenjena definicija ove funkcije (u odnosu na definiciju iz prethodnog odjeljka), ali smisao ostaje isti.

4.2.1 Optimizacijski problem

Ograničenja (4.13) i (4.15) će se uvijek tretirati kao čvrsta ograničenja: ako je student upisao kolegij c_j, on na tom kolegiju negdje mora biti razmješten. Isto tako, domena varijable $x_{i,j,k}$ je jasno definirana. Stoga ako nije moguće načiniti razdiobu studenata po grupama na takav način da vrijede sva ograničenja (4.13)-(4.16), mjesta za relaksaciju mogu se tražiti u ograničenjima (4.14) i (4.16).

Ograničenje (4.14) zahtjeva da se ne prekorači kapacitet slijeda, jer za to postoje objektivni razlozi (primjerice, prostorija u kojoj se održava nastavna aktivnost nema dovoljno mjesta). U praksi, ako je baš nužno, ovaj se problem ponekad može riješiti tako da se proširi kapacitet prostorije (primjerice, dodavanjem još jedne klupe). Također, iako bi u idealnom svijetu svi studenti uvijek bili na svim nastavnim aktivnostima, praksa pokazuje da ovisno od situacije do situacije jedan dio studenata ne dolazi na sve termine nastavnih aktivnosti. Ova činjenica može poslužiti kao opravdanje za dozvoljavanje određene količine prekapacitiranosti termina. Dakako, što je prekapacitiranost nekog slijeda veća, to će teže biti osigurati izvođenje nastavnih aktivnosti, pa porast svakako treba kažnjavati nekom nelinearnom funkcijom. Stoga se umjesto čvrstog ograničenja (4.14) može definirati sljedeća vrijednost:

$$\mathit{prekapacitiranost}(\mathbf{X}) = \sum_{j=1}^{o} \sum_{k=1}^{|E_j|} max(0, \sum_{i=1}^{n} x_{i,j,k} + N_{j,k} - C_{j,k})^{\alpha} \tag{4.18}$$

Neka je drugi argument funkcije $max(0, \rho)$ korištene u sumi označen s ρ. Njegova vrijednost je razlika između ukupnog broja studenata koji su dodijeljeni u promatrani

slijed i kapaciteta slijeda. Ako je studenata manje no što je kapacitet, razlika će biti negativna, dakle $\rho < 0$ pa će vrijediti $max(0, \rho) = 0$. Tek ako je broj studenata veći od kapaciteta slijeda, ρ će biti pozitivan i odgovarat će broju studenata iznad kapaciteta. Tada će vrijediti $max(0, \rho) = \rho$. Parametar α omogućava nelinearno kažnjavanje prekapacitiranosti. Primjerice, uz $\alpha = 2$ prekapacitiranost od jednog studenata će povećati iznos funkcije *prekapacitiranost*($\mathbf{X}$) za jedan, ali prekapacitiranost od tri studenta iznos funkcije povećat će čak za devet. Prikladnim odabirom ovog parametra moguće je podešavati koliko se strogo kažnjavaju veća prekoračenja kapaciteta. Uz $\alpha = 1$ situacija u kojoj su tri slijeda prekapacitirana svaki za jedno mjesto ili situacija u kojoj je samo jedan slijed prekapacitiran za tri mjesta bile bi tretirane kao jednako prihvatljive – to u praksi nije slučaj i zbog toga je preporučljivo koristiti vrijednosti $\alpha > 1$.

Nakon što je prekapacitiranost definirana na ovaj način, problemu raspoređivanja može se pristupiti kao optimizacijskom problemu kojemu je cilj minimizirati iznos funkcije *prekapacitiranost*($\mathbf{X}$). Ako postoji valjani raspored s obzirom na čvrsta ograničenja (4.13)-(4.16), tada će postojati i takav $\mathbf{X} = \mathbf{X}^*$ u kojem će vrijediti *prekapacitiranost*($\mathbf{X}^*$) $= 0$. Ako valjano rješenje ne postoji, tada će se minimizirajući izraz (4.18) pronaći najbolja moguća razdioba studenata uz koju svi i dalje mogu pratiti nastavne aktivnosti svih upisanih kolegija.

Drugi razlog za nepostojanje valjanog rasporeda je nemogućnost zadovoljenja ograničenja (4.16). Ako je nastupio takav slučaj, tada je očito nemoguće načiniti razdiobu studenata po grupama (na bilo kakav način) uz koju svi studenti mogu pratiti nastavne aktivnosti svih upisanih kolegija. U praksi, do ovakvih situacija zna doći ako student upisuje dva manja izborna kolegija koje inaće nitko ne upisuje oba, pa oba kolegija imaju samo po jedan slijed aktivnosti, i ti slijedovi idu paralelno (uz djelomično ili potpuno preklapanje). Osim ovako očitih razloga, postoje i drugi kombinatorne prirode zbog kojih može doći do pojave preklapanja.

Kako je zadatak raspoređivanja neraspoređenih studenata po grupama za predavanja dati najbolje moguće rješenje koje će se potom primijeniti, umjesto čvrstog ograničenja (4.16) može se definirati vrijednost:

$$ukupnoPreklapanje(\mathbf{X}) = \sum_{i=1}^{n} preklapanje(P_i^*)^{\beta}. \tag{4.19}$$

Ukupno preklapanje pri tome se definira kao zbroj preklapanja svih studenta, pri

čemu se preklapanje svakog od studenata u zbroj dodaje s potencijom β. Pri tome je preporuka koristiti $\beta > 1$ čime se osigurava izbjegavanje situacije u kojoj malo studenata ima veliku količinu preklapanja a ostali malo preklapanja, već bi količina preklapanja trebala biti ravnomjernije raspoređena. Povećanjem vrijednosti parametra β više će se kažnjavati veći iznosti preklapanja. Primjerice, neka dva studenta imaju preklapanje pri čemu jedan ima 2 sata preklapanja a drugi 4 sata preklapanja. Uz $\beta = 1$ ukupno će preklapanje biti $2^1 + 4^1 = 6$ a uz $\beta = 3$ ukupno će preklapanje biti $2^3 + 4^3 = 72$. U situaciji u kojoj oba studenta imaju po 3 sata preklapanja, uz $\beta = 1$ ukupno će preklapanje biti $3^1 + 3^1 = 6$ a uz $\beta = 3$ ukupno će preklapanje biti $3^3 + 3^3 = 54$. Uz $\beta = 1$ obje su situacije jednake; uz $\beta = 3$ prva situacija u kojoj jedan student ima veći iznos preklapanja lošija je od druge situacije. Kako je prilikom izrade rasporeda poželjno izbjeći situacije u kojoj dio studenata ima izrazito loše rasporede, slijedi preporuka uporabe $\beta > 1$ čime se veći iznosi preklapanja više kažnjavaju. Ovakav način nelinearnog kažnjavanja korišten je u nastavku i u drugim slučnim situacijama.

Uz ovako definiranu funkciju, problem pronalaska rasporeda može se promatrati kao optimizacijski problem kojemu je zadatak pronaći onaj $\mathbf{X}$ koji će osigurati postizanje minimuma funkcije *ukupnoPreklapanje*($\mathbf{X}$).

U slučaju da postoji valjano rješenje $\mathbf{X} = \mathbf{X}^*$, minimum funkcije *ukupnoPreklapanje*($\mathbf{X}^*$) će biti upravo 0. U slučaju da valjano rješenje ne postoji, ideja je pronaći najbolje moguće s obzirom na postavljeno ograničenje.

Konačno, čak i ako se čvrsta ograničenja (4.14) i (4.16) zamijene mekim ograničenjima (4.18) i (4.19), postavlja se pitanje jesu li sva rješenja koja minimiziraju funkcije (4.18) i (4.19) jednako vrijedna? S pedagoškog stanovišta, odgovor na ovo pitanje je ne – potrebno je pogledati kako rasporedi svakog od studenata doista izgleda, odnosno koliko je kvalitetan. Kvalitetu se pri tome može mjeriti brojem sati prekida u rasporedu tijekom svakog dana. Primjerice, ako student u ponedjeljak ima nastavnu aktivnost od 8h do 10h, pa sljedeću od 13h do 14h i zatim još jednu od 15h do 16h, tijekom tog dana student ima dva prekida; prvi od 10h do 13h u trajanju 3 sata te drugi od 14h do 15h u trajanju od 1 sata. Ukupna količina prekida u satima tada iznosi $3 + 1 = 4$ sata. Prelazak iz jednog dana u drugi dan pri tome se ne računa kao prekid.

Neka je definirana funkcija *kolicinaPrekida*(P_i^*) kao funkcija koja računa ukupnu količinu prekida u satima za studenta s_i, gdje je P_i^* definiran na uobičajeni način.

Prekidi se pri tome računaju tek nakon što se načini unija svih termina iz multiskupa P_i^*. Uz ovako definiranu funkciju, može se definirati sljedeće meko ograničenje:

$$ukupnoPrekida(\mathbf{X}) = \sum_{i=1}^{n} kolicinaPrekida(P_i^*)^{\gamma}. \tag{4.20}$$

Faktor γ pri tome omogućava kontroliranje razdiobe prekida između pojedinih studenata, i preporuka je koristiti $\gamma > 1$. Uz ovako definiranu funkciju $ukupnoPrekida(\mathbf{X})$, kao jedan od zahtjeva optimizacije može se postaviti i pronalazak takvog rasporeda koji će minimizirati funkciju $ukupnoPrekida(\mathbf{X})$, odnosno koji će osigurati da studenti imaju kompaktne rasporede, čime će im ostati više vremena za druge nastavne ali i nenastavne aktivnosti.

U konačnici, problem raspoređivanja neraspoređenih studenata po predavanjima može se prikazati kao višekriterijski optimizacijski problem koji mora zadovoljiti čvrsta ograničenja (4.13) i (4.15), te koji bi morao u određenoj mjeri zadovoljiti meka ograničenja:

1. minimizirati prekapacitiranost slijedova, odnosno izraz (4.18),
2. minimizirati količinu konflikata dodijeljenih nastavnih aktivnosti svakog studenta, odnosno izraz (4.19), te
3. postići što je moguće kompaktniji raspored, odnosno minimizirati količinu prekida u rasporedu svakog studenata, odnosno izraz (4.20).

Pri tome redosljed kojim su meka ograničenja ovdje navedena ne mora odgovarati važnosti zadovoljenja tih kriterija.

4.3 Uklanjanje konflikata u satnici za predavanja na razini studenata

Konflikti u rasporedu predavanja mogu se javiti iz više razloga. Primjerice, oni mogu biti posljedica načina na koji je načinjen raspored predavanja te smještanje studenata u grupe za predavanja. Druga mogućnost je da je studentima dana sloboda da modificiraju inicijalni smještaj u grupe, pa da uspiju provesti samo djelomične promjene čime im mogu ostati kolizije. Koji god da je razlog za nastanak ovakvog stanja, postavlja se

pitanje je li moguće uočene kolizije razriješiti dodatnim izmjenama zatečenog smještaja u grupe za predavanje, i to je upravo srž ovog problema.

Neka vrijede sljedeće pretpostavke:

- načinjen je raspored predavanja, odnosno za svaku grupu na svakom kolegiju utvrđeni su termini u kojima grupa ima predavanja;
- načinjen je razmještaj studenata po grupama za predavanja te
- svi su studenti razmješteni, odnosno nema studenata koji su upisani na kolegij ali im još nije dodijeljena grupa za predavanja.

Kako je ovaj problem u određenoj mjeri sličan prethodno opisanom problemu, koriste se iste oznake gdje je to moguće.

Formalno, ovaj problem može se opisati na sljedeći način. Zadan je skup kolegija $C = \{c_1, \ldots, c_o\}$. Svaki kolegij ima predavanja tijekom semestra organiziranih u jedan ili više slijedova termina (u konačnici, svaki slijed predstavlja jednu grupu za predavanja). Pri tome je kolegiju c_i pridružen skup slijedova $E_i = \{e_{i,1}, e_{i,2}, \ldots\}$, pri čemu je svakom slijedu $e_{i,j}$ pridružen skup termina $T_{i,j} = \{t_{i,j,1}, t_{i,j,2}, \ldots\}$. Kardinaliteti skupova slijedova E_i i E_j, $i \neq j$ pri tome ne moraju biti jednaki. Masovniji kolegiji će imati više slijedova, manji kolegiji manje. Za svaki slijed $e_{i,j}$ kolegija c_i definiran je kapacitet slijeda $C_{i,j}$ – broj studenata koji mogu biti pridruženi na taj slijed, te zauzeće slijeda $N_{i,j}$ – to je broj studenata koji inicijalno smješteni u taj slijed. Broj slobodnih mjesta u slijedu tada je jednak $C_{i,j} - N_{i,j}$.

Neka je $S = \{s_1, \ldots, s_n\}$ skup samo onih studenata koji imaju barem jedan konflikt u rasporedu predavanja. Ne su s $u_{i,j,k}$ označeni indikatori inicijalnih dodjela u grupe za predavanja; $u_{i,j,k} = 1$ ako je student s_i upisao kolegij c_j i na njemu je je smješten u grupu (slijed) $e_{j,k}$; inače $u_{i,j,k} = 0$. Svaki student u općem slučaju nije upisan na svaki kolegij; stoga će za studenta s_i na kolegiju c_j postojati k za koji je $u_{i,j,k} = 1$ samo ako je taj student upisao taj kolegij i tada će postojati samo jedan k za koji je to ispunjeno (uz fiksirani i i j). Ako student s_i nije upisao kolegij c_j, tada će za svaki k uz tako fiksirano i i j vrijediti $u_{i,j,k} = 0$. Da bi problem bio dobro zadan, očekuje se dakle da će uvijek vrijediti:

$$0 \leq \sum_{k=1}^{|E_j|} u_{i,j,k} \leq 1 \qquad (i = 1, \ldots, n; j = 1, \ldots, o). \tag{4.21}$$

Prilikom rješavanja ovog problema pretpostavka je također da student u tom trenutku još nema nikakvih drugih zauzeća osim onih koja slijede iz smještaja u satnicu. Stoga se dalje pretpostavlja da svaki student $s_i \in S$ ima prazan skup postojećih zauzeća, tj. $P_i = \emptyset$, što je razlika u odnosu na prethodno definirani problem.

Za svakog studenta s_i iz S mogu se definirati i indikatorske zastavice $w_{i,j} \in \{0, 1\}$ koje govore ima li student s_i na kolegiju c_j prema inicijalnom razmještaju u grupe za predavanja barem jednu koliziju; ako ima, tada je $w_{i,j} = 1$ a ako nema, tada je $w_{i,j} = 0$. Ako je student upisao 5 kolegija, moguće je da ima kolizije na 2 kolegija a na preostala 3 nema, stoga je važna informacija o konfliktnim kolegijima za svakog studenta. Za svakog će studenta $s_i \in S$ vrijediti:

$$\sum_{j=1}^{o} w_{i,j} \geq 2. \tag{4.22}$$

Naime, kako je pretpostavka da skup S sadrži samo one studente koji imaju na barem jednom predmetu koliziju, tada sigurno uvijek vrijedi $\sum_{j=1}^{o} w_{i,j} \geq 1$. Međutim, kako po pretpostavci studenti nemaju nikakvih drugih prethodnih zauzeća te kako je implicitna pretpostavka da jedan kolegij nema konfliktan raspored sa samim sobom, očito je da ako kolizije postoje, one moraju biti rezultat konflikata između barem dva kolegija, dakle $\sum_{j=1}^{o} w_{i,j} \geq 2$.

Neka je $x_{i,j,k} = 1$ ako je studentu s_i na predmetu c_j dodijeljen slijed $e_{j,k}$; inače je $x_{i,j,k} = 0$ (opaska: inicijalni raspored može se rekonstruirati postavljanjem $x_{i,j,k} = u_{i,j,k}$). Problem izrade rasporeda tada se može svesti na problem dodjele vrijednosti $x_{i,j,k}$ tako da vrijede sljedeća ograničenja:

$$\sum_{k=1}^{|E_j|} x_{i,j,k} = \sum_{k=1}^{|E_j|} u_{i,j,k} \qquad (i = 1, \ldots, n; j = 1, \ldots, o) \tag{4.23}$$

$$N_{j,k} - \sum_{i=1}^{n} u_{i,j,k} + \sum_{i=1}^{n} x_{i,j,k} \leq C_{j,k} \qquad (j = 1, \ldots, o; k = 1, \ldots, |E_j|) \tag{4.24}$$

$$x_{i,j,k} = 0 \text{ ili } 1 \qquad (i = 1, \ldots, n; j = 1, \ldots, m; k = 1, \ldots, |E_j|) \tag{4.25}$$

$$\mathit{preklapanje}(P_i^*) = 0 \qquad (i = 1, \ldots, n) \tag{4.26}$$

Ograničenja definirana izrazom (4.23) osiguravaju da je student s_i na kolegiju c_j

smješten u točno onoliko slijedova u koliko je bio smješten i inicijalnim rasporedom (a to će biti 0 ako nije upisao taj kolegij odnosno 1 ako ga je upisao). Ograničenja definirana izrazom (4.24) osiguravaju da je broj studenata razmještenih u slijed $e_{j,k}$ kolegija c_j i dalje manji ili jednak broju slobodnih mjesta u tom slijedu. Kako je $N_{j,k}$ ukupni broj studenata koji su inicijalno smješteni u taj slijed, aktualni broj dobiva se tako da se od tog broja oduzmu svi studenti koji se razmještaju a koji su inicijalno bili smješteni na taj slijed i potom se dodaju studenti koji su rasporedom razmješteni na taj slijed. Ako nije bilo nikakvih promjena, tj. ako je $x_{i,j,k} = u_{i,j,k}$, tada će vrijediti $\sum_{i=1}^{n} u_{i,j,k} = \sum_{i=1}^{n} x_{i,j,k}$ pa će zapravo pisati $N_{j,k} \leq C_{j,k}$. Izraz (4.25) ograničava domenu varijabli $x_{i,j,k}$ na 0 ili 1. Ograničenja zadana izrazom (4.26) osiguravaju da u načinjenoj dodjeli student nema preklapanja između dodijeljenih termina predavanja. P_i^* je pri tome definiran kao multiskup na uobičajen način:

$$P_i^* = \{e_{j,k} \,|x_{i,j,k} = 1; j = 1, \ldots, m; k = 1, \ldots, |E_j|\}\,. \tag{4.27}$$

Uz ovako definiran problem, rasporedi studenata koji nemaju konflikata neće se mijenjati jer ti studenti nisu niti uključeni u skup studenata S koji se razmatraju. Njihov utjecaj na ovaj problem vidi se jedino u vrijednostima inicijalnog broja studenata razmještenog po grupama, tj. $N_{j,k}$. Studentima kojima se mijenjaju grupe trenutna definicija dozvoljava kompletnu promjenu rasporeda predavanja. Ovome se može doskočiti na sljedeći način. Neka je definiran:

$$\Psi_i(\mathbf{X}) = \sum_{j=1}^{o} \frac{1 - w_{i,j}}{2} \sum_{k=1}^{|E_j|} |x_{i,j,k} - u_{i,j,k}| \qquad (i = 1, \ldots, n) \tag{4.28}$$

kao broj promjena grupa studentu s_i na kolegijima na kojima nije imao konflikt; izraz $|x_{i,j,k} - u_{i,j,k}|$ pri tome označava apsolutnu vrijednost razlike. Neka je još definirano i:

$$\Psi(\mathbf{X}) = \sum_{i=1}^{n} \Psi_i(\mathbf{X})^{\delta} \tag{4.29}$$

kao ukupni broj promjena grupa svim studentima na kolegijima na kojima nisu imali konflikt. Pri tome je δ parametar koji utvrđuje u kojoj se mjeri kažnjava iznos broja promjena na razini pojedinog studenta. Uobičajeno je pri tome koristiti $\delta \geq 1$. Sada se

može postaviti čvrsto ograničenje:

$$\Psi(\mathbf{X}) = 0 \tag{4.30}$$

kojim se traži da se raspored modificira na takav način da student ne smije imati promjene na kolegijima na kojima nema konflikt. Time se zahtjava maksimalno čuvanje već načinjenog rasporeda.

4.3.1 Optimizacijski problem

Temeljem prethodno opisanih čvrstih ograničenja (4.23)-(4.26) i (4.30) u praksi gotovo sigurno neće biti moguće pronaći valjan raspored. Jedan od čestih razloga leži u činjenici da za razriješiti konflikt između dva kolegija barem na jednom od njih studentu treba promijeniti grupu. Time će najčešće ta promjena uzrokovati koliziju koje prije nije bilo. Stoga je prirodno odustati od čvrstog ograničenja (4.30) i njega pretvoriti u određen oblik mekog ograničenja. Jednom kada se to načini, gubi se kontrola nad brojem izmjena grupa koje se mogu dogoditi. Stoga se definira vrijednost:

$$\Theta_i(\mathbf{X}) = \sum_{j=1}^{o} \frac{w_{i,j}}{2} \sum_{k=1}^{|E_j|} |x_{i,j,k} - u_{i,j,k}| \qquad (i = 1, \ldots, n) \tag{4.31}$$

kao broj promjena grupa studentu s_i na kolegijima na kojima je imao konflikt te:

$$\Theta(\mathbf{X}) = \sum_{i=1}^{n} \Theta_i(\mathbf{X})^{\epsilon} \tag{4.32}$$

kao ukupni broj promjena grupa svim studentima na kolegijima na kojima su imali konflikt. Pri tome je ϵ parametar kojim se regulira preferirani način raspodjele ovih brojeva po studentima. Uobičajeno je $\epsilon \geq 1$. Ovi su podatci potrebni kako bi se moglo tražiti da se minimizira broj promjena grupa na kolegijima na kojima je student imao konflikt (bolje je promijeniti samo jednu grupu nego mijenjati obje) te broj promjena grupa na kolegijima na kojima student nije imao konflikt.

Osim ove izmjene, uobičajeno je čvrsta ograničenja (4.24) i (4.26) također pretvoriti u meka na isti način kako je to načinjeno kod prethodnog problema, s obzirom da će čvrsta ograničenja najčešće biti nemoguće zadovoljiti i dobiti valjan raspored.

Time se dobiva višekriterijski optimizacijski problem kod kojeg su ograničenja na-

vedena u nastavku:

1. minimizirati broj promjena grupa na konfliktnim kolegijima $\Theta(\mathbf{X})$, odnosno izraz (4.32),
2. minimizirati broj promjena grupa na kolegijima koji nisu konfliktni $\Psi(\mathbf{X})$, odnosno izraz (4.29),
3. minimizirati prekapacitiranost slijedova, odnosno izraz (4.18),
4. minimizirati količinu konflikata dodijeljenih nastavnih aktivnosti svakog studenta, odnosno izraz (4.19) pri čemu je P_i^* definiran izrazom (4.29), te
5. postići što je moguće kompaktniji raspored, odnosno minimizirati količinu prekida u rasporedu svakog studenata, odnosno izraz (4.20).

Redosljed kojim su meka ograničenja ovdje navedena ne mora odgovarati važnosti zadovoljenja tih kriterija.

4.4 Raspored laboratorijskih vježbi

Raspored laboratorijskih vježbi sljedeći je od problema opisan u nastavku. Ovaj problem u određenoj je mjeri sličan problemu izrade rasporeda predavanja na sveučilištu; najvažnije razlike su:

- istovremeno se radi stvaranje grupa i razdioba studenata u te grupe,
- studenti imaju prethodna zauzeća drugim nastavnim aktivnostima pa raspored treba ispreplesti oko tih zauzeća,
- prostorije imaju prethodna zauzeća drugim nastavnim aktivnostima pa raspored treba ispreplesti oko tih zauzeća,
- trajanje termina laboratorijskih vježbi može biti od 15 minuta pa do preko 5 sati što znači da se radi s finom granulacijom vremena, te
- postoji više kriterija na prihvatljivost prostorija, termina, broj studenata koji mogu ići paralelno i sl.

O ovim problemima i načinima njihova rješavanja već su objavljeni radovi [Bratković et al., 2009, Dino Matijaš et al., 2010, Čupić and Franović, 2010], pa će ovdje opis problema biti izložen u skladu s definicijama iz tih radova.

Problem izrade rasporeda laboratorijskih vježbi (engl. *laboratory exercise timetabling problem, LETP*) može se definirati kao uređena šestorka:

$$LETP = (T, L, R, E, S, C)\,, \tag{4.33}$$

gdje je T skup *vremenskih odsječaka* (engl. *time slot*) u kojima je moguće obaviti raspoređivanje, L je skup ograničenih sredstava koja su potrebna za izvođenje vježbi i koja se dijele na razini ustanove, R je skup prostorija u koje je moguće razmještati vježbe, E je skup događaja odnosno vježbi koje je potrebno razmjestiti, S je skup studenata koje je potrebno razmjestiti, te C je skup ograničenja koja treba zadovoljiti.

- Skup vremenskih odsječaka u kojima je moguće obaviti raspoređivanje označen je s T. Pretpostavka je da se trajanje svake vježbe može opisati kao višekratnik fiksnog vremenskog kvanta (primjerice, uz kvant od 15 minuta moguće je definirati vježbe koje traju 15 minuta, 30 minuta itd). Taj kvant označen je s t_q. Vremenski odsječak dodijeljen nekoj vježbi tada je definiran kao niz od jednog ili više uzastopnih kvanata. Uz mali iznos kvanta bit će moguće definirati vježbe širokog raspona trajanja (uz vrlo finu granulaciju); međutim, uz mali kvant prostor pretraživanja će biti ekstremno velik, pa će pronalazak rasporeda biti vrlo težak zadatak. Također, skup T ne mora i tipično ne predstavlja kontinuirani raspon vremena (primjerice, od ponedjeljka od 8h ujutro do petka u 20h); naime, zakazati laboratorijske vježbe u ponedjeljak u 23h ili u utorak u 04h ujutro uobičajeno neće biti prihvatljivo. Stoga će u ovaj skup ući termini iz raspona dana unutar kojeg se radi raspored što je najčešće od 8h pa do 20h (točne granice moguće je odabrati po potrebi).

- Skup ograničenih sredstava koji se koriste za laboratorijske vježbe označen je s L. To primjerice može biti uporaba programskog paketa *MatLab* koji se koristi u izvođenju laboratorijskih vježbi različitih kolegija. Kako je uporaba tog paketa moguća samo uz uporabu licenci a institucija ima ograničen broj licenci, očito je da se prilikom izrade rasporeda ne smije dogoditi situacija u kojoj je razmješteno

previše grupa studenata koje trebaju ove licence istovremeno jer se vježbe neće moći održati. Ovo dijeljenje nije dovoljno kontrolirati samo na razini jedne vježbe jednog kolegija već kontrolu treba raditi na razini čitavog rasporeda. Za svako sredstvo $l \in L$ broj dostupnih primjeraka sredstva označen je s $quantity_l$. Prilikom izrade rasporeda, pretpostavka je da svako dodijeljeno radno mjesto troši po jedan primjerak sredstva. Stoga je ovime dano ograničenje na maksimalni broj dodijeljenih radnih mjesta istovremeno koja troše ovo sredstvo.

- Svakoj prostoriji r pridijeljeni su sljedeći podatci: $(size_r, T_r)$, $size_r \in \mathbb{N}$, $T_r \subseteq T$, pri čemu:
  - Radno mjesto je definirano kao elementarno svojstvo prostorije. Svakoj prostoriji $r \in R$ pridijeljen je broj radnih mjesta koja su na raspolaganju u toj prostoriji – $size_r \in \mathbb{N}$. Primjerice, u računalnim laboratorijima, broj radnih mjesta bit će definiran brojem računala u prostoriji; ako prostorija ima 30 računala, definira se da ima 30 radnih mjesta.
  - Za svaku prostoriju $r \in R$ definiran je i skup termina $T_r \subseteq T$ u kojima je ta prostorija dostupna za raspoređivanje laboratorijskih vježbi; naime, moguće je da su prostorije u nekim periodima zauzete održavanjem drugih nastavnih aktivnosti pa se ne smiju dopustiti preklapanja između načinjenog rasporeda vježbi i prethodnih zauzeća prostorija.

- Događajem $e \in E$ smatra se održavanje jedne laboratorijske vježbe jednog kolegija. Događaj pri tome može biti razmješten u jedan termin (u jednu ili više prostorija) ili u više termina (opet u jednu ili više prostorija). Pojam termin se ovdje odnosi na kontinuirani slijed od jednog ili više kvanata čija duljina odgovara zadanom trajanju događaja. Primjerice, za događaj može biti predviđen jedan termin u ponedjeljak od 8h do 12h u dvije prostorije te jedan termin u srijedu od 11h do 15h u tri prostorije. Studenti koji moraju prisustvovati tom događaju tada će biti raspoređeni jednim dijelom u prvi termin a drugim dijelom u drugi termin. Svaki događaj ima sljedeća svojstva:
  - Događaj $e \in E$ ima trajanje, oznaka $dur_e \in \mathbb{N}$, definirano kao broj kvanata; primjerice prva laboratorijska vježba iz kolegija Umjetna inteligencija može

imati trajanje $dur_{AI} = 4$ kvanta, što bi uz kvant definiran kao 15 minuta značilo da traje 1 sat (60 minuta).

- Kako bi se moglo kontrolirati ukupno trajanje jednog događaja, definira se *dopustivi raspon događaja*, oznaka $span_e \in \mathbb{N}$. Taj raspon postavlja ograničenje na razliku između početka najranijeg termina tog događaja i kraja najkasnijeg termina tog istog događaja. Primjerice, neka je prvu laboratorijsku vježbu iz kolegija Umjetna inteligencija moguće raspoređivati kroz svih pet dana tjedna, te neka je definiran dopustivi raspon te vježbe kao 2 dana. Iz ovoga slijedi da ako je najraniji termin dodjeljen tom kolegiju smješten u ponedjeljak, najkasniji termin može biti smješten još u utorak; s druge strane, ako najraniji termin bude u srijedu, najkasniji termin može biti još u četvrtak. Umjesto ograničenja u danima, često se želi postaviti ograničenje u minutama – primjerice, neki masovni kolegij na jednoj vježbi piše samo kratku provjeru znanja na računalu u trajanju od 30 minuta. Kako kolegij ima dvostruko više studenata no što ima računala, uobičajeno će se kao dopustivi raspon događaja definirati period od 60 minuta. Time će se efektivno tražiti rezervacija svih računalnih prostorija u dva uzastopna termina te raspoređivanje svih studenata u jedan od ta dva termina.
- S obzirom da se predviđa da različiti kolegiji izvode različite vrste laboratorijskih vježbi, za svaki događaj je potrebno definirati skup prihvatljivih prostorija $R_e \subseteq R$. Naime, kolegiji koji izvode vježbe na računalu tražit će prostorije opremljene računalima dok će kolegiji koji izvode različite vrste električkih mjerenja tražiti prostorije opremljene odgovarajućim uređajima i slično.
- Za svaki događaj $e \in E$ moguće je definirati neprazni podskup $T_e \subseteq T$ dozvoljenih vremenskih slotova u koje je potrebno razmjestiti termine događaja. Primjerice, ako radi raspored za dane od ponedjeljka do petka, za neki se događaj može tražiti da bude raspoređen samo u utorak i četvrtak. Ovo je potrebno podržati kako bi se omogućila izrada rasporeda u situacijama u kojima nastavno osoblje nije dostupno u nekim terminima.
- Događaj može zahtjevati i jedno ili više ograničenih sredstava. Skup ograničenih sredstava koje zahtjeva događaj $e \in E$ označen je s $L_e \subseteq L$.

– Za održavanje svake laboratorijske vježbe u dodijeljenim terminima prisutni su i djelatnici koji izvode vježbu. Koliko je djelatnika raspoloživo za dežuranje u terminima događaja e označava se s $staff_e \in \mathbb{N}$. Koliko djelatnika treba dežurati u pojedinim termina ovisit će o dodijeljenim prostorijama; veće prostorije omogućavaju formiranje većih grupa studenata ali i zahtjevaju više prisutnih djelatnika. Broj djelatnika koje je na događaju e potrebno smjestiti u prostoriju r označen je s $usage_{e,r}$. Za prostorije r koje nisu u skupu dozvoljenih prostorija za događaj e, tj. za $r \notin R_e$ vrijednost $usage_{e,r}$ nije definirana.

– Na razini događaja ponekad je potrebno ograničiti najveći broj prostorija koje će odjednom biti dodijeljene tom događaju. Za događaj e najveći dopušteni broj istovremeno dodijeljenih prostorija označen je s $rooms_e \in \mathbb{N}$.

– U slučaju da tijekom jednog perioda raspoređivanja ide više laboratorijskih vježbi jednog kolegija, potrebno je omogućiti definiranje poretka izvođenja tih vježbi. Primjerice, ako kolegij ima dvije vježbe u istom tjednu, potrebno je omogućiti definiranje ograničenja poput: svi termini prve vježbe moraju biti gotovi prije no što počnu termini druge vježbe. Proširenje ovog zahtjeva je omogućiti definicije poput: između prve i druge vježbe trebaju biti barem dva dana razmaka. Kako bi se ovo podržalo, definirana je relacija $\succ_d$ nad parom događaja $\succ_d$: $E \times E$. Dva događaja e_1 i e_2 su u relaciji $e_2 \succ_d e_1$ ako i samo ako svaki termin događaja e_2 je raspoređen barem d dana nakon zadnjeg termina događaja e_1.

– Za svaki se događaj također treba definirati broj studenata koji se smještaju na jedno radno mjesto, što je označeno s $spw_e \in \mathbb{N}$. Već je prethodno definirano da se svaka prostorija sastoji od određenog broja radnih mjesta. Poznavanjem broja radnih mjesta u prostoriji te broja studenata koji se smještaju na jedno radno mjesto može se doći do ukupnog broja studenata koji se može smjestiti u tu prostoriju – što će biti umnožak tih parametara. U prostorijama opremljenim računalima uobičajeno je na jedno računalo smjestiti jednog ili eventualno dva studenta.

– Pojedini događaji mogu za održavanje svojih vježbi angažirati i studente demonstratore. Ako je to slučaj, tada će na tom kolegiju biti definiran skup

demonstratora S_e^d; taj će skup biti disjuntan sa skupom studenata S_e koji moraju odraditi tu vježbu. Na razini događaja tada je još definirano:

* broj odrada demonstratora $dem\_times(e)$; primjerice, moguće je tražiti da svaki demonstrator mora biti prisutan na dva termina,
* za svaku prihvatljivu prostoriju r događaja e, tj. $r \in R_e \subseteq R$ definiran je minimalno potreban broj demonstratora $dem\_required(e, r)$ koji moraju biti prisutni u toj prostoriji. Za prostorije $r \notin R_e \subseteq R$ vrijednost $dem\_required(e, r)$ nije definirana.

- Skup S je skup studenata koje treba rasporediti ili koji su demonstratori. Za svakog se studenta $s \in S$ definiraju sljedeća svojstva:
  - Skup vremenskih odsječaka u kojima je student slobodan $T_s \subseteq T$. Naime, kako se raspored laboratorijskih vježbi radi nakon izrade rasporeda predavanja, pretpostavka je da svaki student ima prethodno definirana zauzeća koja ovdje treba uzeti u obzir i tada ga ne raspoređivati i na laboratorijske vježbe (bilo kao studenta koji radi vježbu, bilo kao demostratora na vježbi ako je demonstrator).
  - Neprazni skup $E_s \subseteq E$ označava sve događaje kojima student s mora prisustvovati (odnosno događaje na kojima ga treba rasporediti).
  - Neprazni skup $E_s^d \subseteq E$ označava sve događaje kojima je student s demonstrator, pa ga na tim događajima treba rasporediti.
  - Studenti koji trebaju odraditi događaj e podijeljeni su u jednu ili više kategorija; skup kategorija na događaju e označen je s K_e. Studenti svake kategorije razmještaju se zasebno, odnosno u jednu prostoriju nije moguće smjestiti studente različitih kategorija. Korist kategorija je što dopuštaju različite načine izvođenja vježbe. Primjerice, moguće je imati jedan skup studenata koji će vježbu raditi u jednom programskom paketu i drugi skup studenata koji će istu vježbu raditi u drugom programskom paketu (što im je unaprijed poznato). Prilikom izrade rasporeda treba voditi računa da se studenti jedne kategorije razmjeste zasebno i ne miješaju sa studentima druge kategorije.

- Sva ograničenja koja izrada rasporeda laboratorijskih vježbi mora zadovoljiti su:
  - Prostorija u bilo kojem trenutku može biti pridijeljena samo jednom događaju.
  - Raspoređivanje se može raditi samo u terminima u kojima je prostorija slobodna.
  - Prostorija r se nekom događaju e može pridijeliti samo ako je u njegovom skupu prihvatljivih prostorija, tj. $r \in R_e$.
  - U prostoriju r dodijeljenu događaju e moguće je smjestiti najviše $size_r \cdot spw_e$ studenata koji nisu demonstratori (demonstratori, ako postoje, ne troše radna mjesta).
  - Događaj e može biti raspoređen samo u vremenske odsječke koji su za njega prihvatljivi, tj. u $T_e \subseteq T$, ako je T_e definiran; ako nije, događaj se može rasporediti u sve vremenske odsječke T.
  - Smještanjem termina događaja e u raspored, taj termin (a time i događaj) zauzima slijed od dur_e vremenskih odsječaka, pri čemu je dur_e zadano trajanje događaja e.
  - Događaj e mora biti raspoređen unutar dozvoljenog raspona $span_e$, ako je taj definiran.
  - Ako za dva događaja postoji definirana relacija uređaja $\succ_d$, tada ona mora biti zadovoljena u načinjenom rasporedu.
  - Ograničena sredstva $l \in L$ mogu biti istovremeno korištena na najviše $quantity_l$ radnih mjesta.
  - Ako se u nekom terminu događaja e koristi više prostorija, mora biti dovoljno nastavnog osoblja koje će moći odraditi tu vježbu.
  - Ako događaj e ima demostratore, u svakom terminu mora biti raspoređeno barem onoliko demonstratora koliko to zahtjevaju dodijeljene prostorije i njihova potrošnja demonstratora.
  - Demonstratori, ako postoje na događaju e, moraju odraditi upravo onoliko termina koliko je propisano s $dem\_times(e)$.
  - Događaj e može u jednom terminu imati zauzetih najviše $rooms_e$ prostorija.

– Studenti moraju biti raspoređeni na svim vježbama na kojima trebaju biti raspoređeni.
– Student može u nekom terminu biti raspoređen samo ako u to vrijeme već nema neko prethodno zauzeće.
– Student u jednom trenutku može biti samo na jednom događaju.

4.4.1 Optimizacijski problem

Rješenje problema koje bi zadovoljavalo sva prethodna ograničenja (ako ih se smatra čvrstim ograničenjima) često neće biti moguće pronaći. Stoga se neka od čvrstih ograničenja mogu prevesti u meka, kako bi se problem pronalaska dobrog rasporeda svelo na problem minimizacije kršenja mekih ograničenja. Prihvatljivi kandidati za ovo su ograničenja navedena u nastavku.

1. Studenti moraju biti raspoređeni na svim vježbama koje pohađaju.
2. Student može u nekom terminu biti raspoređen samo ako u to vrijeme već nema neko prethodno zauzeće.
3. Student u jednom trenutku može biti samo na jednom događaju.

Od prva dva ograničenja pri tome je dovoljno zahtjevati da samo jedno od njih postane meko ograničenje. Tako je, primjerice, u radu [Bratković et al., 2009] izabrano da se ograničenje (2) tretira kao meko ograničenje – opisani algoritam uvijek razmjesti sve studente (pa je ograničenje (1) uvijek zadovoljeno), ali cijena je pojava kolizija s drugim obavezama; optimizacijski je problem tada svesti ovu količinu kolizija na minimalnu, odnosno na nula za valjani raspored. Ako se valjani raspored ne uspije generirati, uobičajeno je prihvatljivo koristiti onaj raspored koji ima razumno mali broj preklapanja.

U radu [Dino Matijaš et al., 2010] načinjen je drugačiji odabir: ograničenje (1) je proglašeno mekim ograničenjem, a ograničenje (2) je tvrdo ograničenje. Kako se u tom radu opisuje konstrukcijski algoritam koji raspored gradi studenta po studenta, donesena je odluka da se student ubaci u raspored samo ako je to moguće načiniti bez ikakvih kolizija; stoga će za sve studente koji se uspiju ubaciti u raspored čvrsto ograničenje (2) biti zadovoljeno. Međutim, studenti koji se ne uspiju smjestiti u raspored ostat će neraspoređeni što je kršenje mekog ograničenja (1). Optimizacijski je problem tada minimizirati broj studenata koji nisu razmješteni.

Ograničenje (3) pri tome se uvijek može svesti na poopćenje ograničenja (2), tako da se kao (2) koristi ograničenje:

> Student može u nekom terminu biti raspoređen samo ako u to vrijeme već nema neko prethodno zauzeće ili zauzeće nekim drugim događajem.

U tom smislu, to se ograničenje može tretirati ili kao tvrdo, ili kao meko, ovisno o tome kako se postavi optimizacijski problem.

Osim ovih ograničenja, na raspored je moguće dodati još dva ograničenja koja će govoriti o kvaliteti načinjenog rasporeda, pa su ovdje opisana samo na idejnoj razini.

Kvaliteta rasporeda studenta može se definirati kao mjera kontinuiranosti rasporeda za svakog pojedinog studenata; naime, sa stajališta studenata načinjeni raspored nije jednak ako student ima nastavu ujutro pa prekid od 6 sati pa dva sata laboratorijskih vježbi, ili ako ima nastavu ujutro pa odmah u nastavku laboratorijske vježbe, čime ostatak dana studentu ostaje slobodan. Svakako bi trebalo preferirati rasporede koji se dobro uklapaju u postojeće studentske obaveze. Formalno, mjeru kvalitete rasporeda studenata možemo definirati na sličan način kako je to načinjeno kod problema raspoređivanja neraspoređenih studenata, izraz (4.20).

Kvaliteta rasporeda događaja može se definirati kao mjera kontinuiranosti i inertnosti rasporeda pojedinog događaja. Pri tome se pod pojmom mjera kontinuiranosti rasporeda događaja promatra broj prekida u zakazanim terminima; npr. nije isto ako događaj ima termin od 8h do 10h, pa od 12h do 14h, pa od 18h do 20h, ili ako taj isti događaj ima termine 8h do 10h, 10h do 12h i 12h do 14h. Ovaj drugi raspored svakako je sa stajališta djelatnika koji dežura u tim terminima puno kvalitetniji. Pod pojmom mjera inertnosti rasporeda promatramo tendenciju rasporeda da zadrži dodijeljene prostorije. Primjerice, ako neki događaj ima tri dodijeljena termina (i to čak potpuno kontinuirano), jedna je mogućnost da su sva tri zakazana u istoj prostoriji; druga situacija koja se u praksi zna dogoditi je da je prvi termin zakazan u prostoriji r_1, drugi u prostoriji r_2 a treći opet u prostoriji r_1. Ovakav raspored će djelovati vrlo frustrirajuće na djelatnike, posebice ako prostorija r_1 u terminu u kojem je događaj prebačen u r_2 ostaje prazna. Ovo nezadovoljstvo tipično će biti uzrokovano činjenicom da je promjena prostorije "skupa" operacija (treba isprazniti prethodnu prostoriju, pogasiti uređaje, zaključati je, vratiti ključ, dobiti ključ od nove prostorije itd). Opisana se situacija u određenoj mjeri može popraviti naknadnim pregledom rasporeda od strane ljudskog operatera, a prije

njegove objave. Situacije u kojima su prostorije r_1 i r_2 jednake kapacitetom i ako je jedna uvijek prazna kada je druga iskorištena mogu se razriješiti ručnim prebacivanjem svih termina u jednu od te dvije prostorije. Međutim, ako je dođe do ispreplitanja dva kolegija, ručne su korekcije već puno zahtjevnije, i svakako je preporučljivo da sam algoritam izrade rasporeda pokušava biti što je moguće više inertan pri dodjeli prostorija. Na koji se način točno može mjeriti kvaliteta rasporeda događaja vrlo je subjektivno pa stoga ovdje nećemo davati nikakav formalni opis.

4.5 Raspored provjera znanja

Raspored provjera znanja problem je koji u današnje doba ima mnoštvo različitih definicija, i jako je ovisan od institucije do institucije na kojoj se rješava. Primjerice, jedan od uobičajenijih opisa je: *potrebno je rasporediti ispite u raspoložive termine pri čemu se svaki ispit održava u samo jednom terminu i studenti mogu nazočiti svim ispitima koje trebaju polagati.* Međutim, već i ovakva jednostavna definicija danas ne vrijedi, jer ima primjera institucija gdje je broj termina dovoljno mali i izbornost upisa kolegija dovoljno velika da je izraditi takav raspored nemoguće, pa se traži *izrada rasporeda u kojem svaki kolegij ima ispite u minimalnom broju termina; idealno to je jedan termin, ali može i više* [Wang et al., 2010].

U ovoj disertaciji stoga se razmatraju problemi izrade rasporeda provjera znanja koji ima minimalno sljedeće karakteristike:

- zadan je fiksan skup termina i svi termini traju jednako,
- ispit kolegija potrebno je smjestiti u točno jedan od termina,
- studenti moraju moći nazočiti ispitima svih kolegija koje su upisali te
- termini imaju svoje kapacitete koje se ne smije prekoračiti.

Primjer ovakvog problema mogao bi biti sljedeći: postoji 135 kolegija koje je potrebno razmjestiti u 10 dana; svaki dan ima 4 termina u trajanju po 3 sata (od 9h do 12h, od 12h do 15h, od 15h do 18h te od 18h do 21h). Stvarno trajanje pojedinih ispita može biti i kraće od 3h (i često je); važno je samo da ne smije probiti tu granicu. Razmatranje ovakvog problema i načini njegova rješavanja dani su u [Čupić et al., 2009a, Čupić and Franović, 2010]. U nastavku je fokus stavljen na nadogradnju prethodno

opisanog modela koja će puno više pažnje posvetiti kvaliteti načinjenog rasporeda sa stajališta studenata.

Formalni model ovog problema dan je u nastavku. Potrebno je pronaći x_{jk} gdje su $(j = 1, \ldots, o; k = 1, \ldots, p)$ takve da vrijede sljedeća ograničenja:

$$\sum_{k=1}^{p} x_{jk} = 1 \qquad (j = 1, \ldots, o) \tag{4.34}$$

$$\left(\sum_{j=1}^{o} x_{jk} \leq 1 \wedge \sum_{j=1}^{o} x_{jk} \cdot n_j \leq l_k^h\right) \vee \left(\sum_{j=1}^{o} x_{jk} > 1 \wedge \sum_{j=1}^{o} x_{jk} \cdot n_j \leq l_k^s\right) \qquad (k = 1, \ldots, p) \tag{4.35}$$

$$\sum_{j \in S_l} x_{jk} \leq 1 \qquad (l = 1, \ldots, r; k = 1, \ldots, p) \tag{4.36}$$

$$x_{jk} = 0 \text{ ili } 1 \qquad (j = 1, \ldots, o; k = 1, \ldots, p) \tag{4.37}$$

pri čemu $c_1, \ldots, c_o$ predstavlja o kolegija, odnosno njihove ispite. Za svaki su kolegij c_j definirani još i skupovi S_j kao skupovi svih kolegija koji uključuju taj kolegij i međusobno dijele barem jednog upisanog studenata. Skup S definiran je kao skup tih skupova. Posljedica je stvaranje skupa S koji sadrži r elemenata koje ćemo označiti $S_1, \ldots, S_r$. Tako primjerice svi kolegiji koji su u skupu $S_l \in S$ moraju imati zakazan ispit u različitim terminima.

Varijabla $x_{jk} = 1$ ako se ispit kolegija c_j održava u terminu k, a $x_{jk} = 0$ inače; ovih varijabli (a time i ograničenja (4.37)) ima ukupno $o \cdot p$. Ograničenja (4.34) osiguravaju da svaki kolegij ima dodijeljen točno jedan termin za ispit (ovih ograničenja ima ukupno o).

Ograničenja (4.35) osiguravaju da u niti jednom terminu neće biti zakazano više od dozvoljenog broja ispita (ovih ograničenja ima ukupno p). Za svaki je termin k pri tome definiran njegov meki kapacitet n_k^s odnosno njegov čvrsti kapacitet n_k^h. Meki kapacitet termina predstavlja broj studenata koje je moguće smjestiti u taj termin na "ugodan" način. Tvrdi kapacitet termina predstavlja broj studenata koji je teoretski moguće smjestiti u taj termin ali uz vrlo nisku kvalitetu. Ideja uvođenja ova dva kapaciteta za svaki termin je sljedeća: ako je u neki termin smješteno više od jednog kolegija, tada će se inzistirati da ukupan broj studenata koji su dodijeljeni u taj termin bude manji ili jednak mekom kapacitetu termina. S druge pak strane, ako postoji nekoliko kolegija koji imaju već broj studenata od mekog kapaciteta, njima će se dopustiti da se smjeste

u termin koji ima zadovoljavajući tvrdi kapacitet ako su pri tome jedini u tom terminu. U izrazu (4.35) suma $\sum_{j=1}^{o} x_{jk}$ predstavlja broj kolegija koji su smješteni u termin k, a suma $\sum_{j=1}^{o} x_{jk} \cdot n_j$ ukupan broj studenata koji su smješteni u taj termin. Uporaba mekog i tvrdog kapaciteta omogućava da se prilikom izrade rasporeda ne uzimaju u obzir dostupnost pojedinačnih prostorija (i time bitno poveća prostor pretraživanja). Umjesto toga, temeljem analize raspoloživih prostorija na početku se za svaki termin može načiniti procjena mekog i tvrdog kapaciteta i dalje raditi s tim parametrima.

Konačno, ograničenja (4.36) osiguravaju da niti jedna dva konfliktna kolegija nemaju zakazana provjeru znanja u isto vrijeme (ovih ograničenja ima ukupno $r \cdot p$).

4.5.1 Optimizacijski problem

Opisani model sastoji se od četiri porodice ograničenja od kojih sva moraju biti zadovoljena da bi raspored bio valjan. Međutim, so sada u model nisu bili uzeti parametri koji utječu na kvalitetu rasporeda: blizina ispita za pojedine studente, procjena težine ispita i slično. Stoga se model može proširiti sljedećim parametrima.

Neka je zadan n-člani skup studenata S čiji su elementi studenti $\{s_1, \ldots, s_n\}$. Neka su definirane indikatorske varijable $u_{i,j}$ za koje vrijedi $u_{i,j} = 1$ ako student s_i treba nazočiti provjeri znanja c_j; inače je $u_{i,j} = 0$. Uz ovako definirane indikatorske varijable, broj studenata koji polažu pojedinu provjeru više nije potrebno zadati već se može izračunati prema:

$$n_j = \sum_{i=1}^{n} u_{i,j} \qquad (j = 1, \ldots, o). \tag{4.38}$$

Za svaki je kolegij definirana njegova subjektivna težina $w_j \in \mathbb{R}$, te godina na kojoj se kolegij predaje $y_j \in \mathbb{N}$. Za svakog je studenta s_i definiran upisani modul na studiju $s_i^m \in \mathcal{M}$, gdje je $\mathcal{M} = \{m_1, \ldots, m_q\}$ skup svih modula. Za potrebe ovog modela pretpostavlja se da svaki student ima pridijeljen točno jedan modul. U praksi, na različitim razinama studija studenti se vode pod različitim klasifikacijama (primjerice, upisani studij na prvoj godini preddiplomskog studija, upisani smjer na drugoj godini preddiplomskog studija, upisani modul na trećoj godini preddiplomskog studija, upisani profil na diplomskom studiju, i sl). Za potrebe ovdje opisanog modela a bez gubitka općenitosti sve te podjele možemo svesti na podjele po prikladno nazvanim modulima i dalje raditi s time. Svaki modul m_l ima pridijeljen skup kolegija $C_l^m \subseteq C$ koji pripadaju tom modulu. Pri tome presjek $C_{l_1}^m \cap C_{l_2}^m$ za $l_1 \neq l_2$ ne mora biti prazan skup, odnosno

jedan kolegij može biti pridijeljen na više modula. Konačno, neka je definiran skup R kao skup dvojki (c_a, c_b) koje predstavljaju kolegije za koje se posebno želi provjeravati raspršenost u generiranom rasporedu (ideja je pokušati osigurati da su svaka dva kolegija definirana kao par u tom skupu što je moguće više razmaknuta u načinjenom rasporedu). Jedan od načina definiranja ovog skupa je kao skup parova svih kolegija koji dijele barem ρ studenata (gdje je ρ korisnički zadan prag) a oba su istovremeno obavezna na nekom modulu. Ako je skup predmeta koji su obavezni na modulu m_l označen s C_l^{m*} pri čemu vrijedi $C_l^{m*} \subseteq C_l^m \subseteq C$, skup R može se definirati kao:

$$R = \{(c_a, c_b) \mid \sum_{i=1}^{n} (u_{i,a} \cdot u_{i,b}) \wedge \exists m_l . c_a \in C_l^{m*} \wedge c_b \in C_l^{m*}\}. \tag{4.39}$$

Uvođenje subjektivnih težina kolegija w_j omogućit će izradu rasporeda u kojem su teži kolegiji više razmaknuti; primjerice, ako postoje tri kolegija c_1, c_2 i c_3 težina $w_1 = 4$, $w_2 = 2$, $w_3 = 4$ koji dijele studente, bolji će raspored biti onaj kod kojeg je redosljed ispita u rasporedu c_1 pa c_2 pa c_3 nego onaj kod kojeg je redosljed ispita c_1 pa c_3 pa c_2 jer su time dva teška kolegija (c_1 i c_3) smještena jedan za drugim. U praksi, dobivanje subjektivnih težina kolegija rješava se uporabom anonimnih anketa koje studenti mogu popuniti na kraju godine; ti se podatci mogu potom koristiti za izradu rasporeda u sljedećoj godini.

Uvođenje informacije o godini kolegija omogućava da se na različit način kažnjava bliski smještaj dva predmeta koja dijele studente a s različitih su godina u odnosu na kažnjavanje bliskog smještaja dva predmeta koja dijele studente i pripadaju istoj godini; u prvom slučaju kazna bi trebala biti manja, kako bi se preferirala izrada rasporeda koji za studente koji redovno upisuju kolegije ima visoku kvalitetu.

Praćenje modula studenta te praćenje kolegija koji se drže na pojedinim modulima također omogućava izradu kvalitetnijih rasporeda time što se, ako se promatra neki student s_i, više kažnjava blizak smještaj dva kolegija c_a i c_b koji se oba drže na modulu na kojem je i taj student; manja se kazna koristi za kažnjavanje bliskog smještaja dva kolegija od kojih jedan pripada studentovom modulu a drugi ne ili pak ako niti jedan od kolegija ne pripada studentovom modulu. Ideja je osigurati da razmještaj kolegija unutar jednog modula bude što je moguće kvalitetniji.

Uvođenje skupa za dodatno razmicanje R omogućava da se dodatni naglasak stavi na provjeru odnosa odabranih parova kolegija; naime, uobičajeno se bliskost smještaja

dvaju kolegija kažnjava proporcionalno broju studenata koje dijele što za posljedicu ima činjenicu da su rasporedi za male module ugroženi rasporedima za masovnije module. Stoga se predviđa uporaba ovakvog mehanizma koji bi mogao pomoći da se neovisno o broju dijeljenih studenata stavi dodatni naglasak na razmicanje odabranih parova kolegija (u praksi, ovo primjerice može biti primijenjeno na teorijske kolegije profila i slično).

Konačno, potrebno je i definirati kvalitetu bliskosti razmještaja dva kolegija u rasporedu; u praksi se obično procjenjuje kazna. Primjerice, neka je za svaki termin t definiran $day(t)$ kao indeks dana u kojem se nalazi termin. Neka vrijedi da je $day(t) = d_0$ za termine koji su smješteni u prvi dan ispitnog razdoblja, $day(t) = d_0 + 1$ za termine koji su smješteni u drugi dan ispitnog razdoblja, itd. Još se jednostavniji slučaj dobiva postavljanjem $d_0 = 0$. Nadalje, neka je za svaki termin t definiran $index(t)$ kao redni broj termina unutar dodijeljenog dana. Primjerice, ako u danu postoje četiri termina, prvi ima $index(t) = 0$, drugi $index(t) = 1$, itd. Uporabom ovih informacija moguće je za svaki razmješteni kolegij c_j izbrojati koliko studenata ima još barem jedan ispit u istom danu kada je i termin kolegija c_j – neka je taj broj označen s K_j^0, odnosno koliko još studenata ima barem jedan ispit u sljedećem danu kada je i termin kolegija c_j – neka je taj broj označen s K_j^1. Funkciju kazne moguće je definirati kao funkciju oblika

$$\alpha \cdot \sum_{j=1}^{o} K_j^0 + \beta \sum_{j=1}^{o} \cdot K_j^1 \tag{4.40}$$

pri čemu bi faktor α trebao biti dosta veći od faktora β; ova oba faktora određuju koliko se kažnjava smještaj kolegija koji dijele studente u isti dan te smještaj kolegija koji dijele studente u susjedni dan. Po samoj je definiciji u vrijednosti K_j^0 i K_j^1 uključen i broj studenata koji su zahvaćeni ovim bliskim smještajem, pa takva definicija može uzrokovati da se više pažnje posveti masovnijim kolegijima, čemu onda protuteža može biti uporaba dodatno definiranog skupa za razmicanje R.

Koristeći informaciju o položaju termina u danu, ovako definirana kazna može se još dodatno podesiti. Primjerice, umjesto da se za studente kolegija c_j broji samo koliko ti studenti imaju još ispita u istom danu, može se definirati općenitija vrijednost $K_j^{0,h}$ koja govori koliko studenata kolegija c_j ima još ispite koji su u tom istom danu i to u terminu koji je na udaljenosti h, gdje se udaljenost h dva termina t_1 i t_2 koji pripadaju

istom danu računa kao $h = index(t_2) - index(t_1)$; slučaj $h < 0$ može se ignorirati da se izbjegne višestruko prebrojavanje. Sada se umjesto prethodne formule za izračun kazne može koristiti formula oblika:

$$\sum_{j=1}^{o} \sum_{h=0}^{v-1} \alpha_h \cdot K_j^{0,h} + \beta \sum_{j=1}^{o} \cdot K_j^1 \tag{4.41}$$

pri čemu je v broj termina u jednom danu, oznaka α_h je porodica konstanti koje kažnjavaju bliskost smještaja po terminima unutar jednog dana (pri čemu će uobičajeno vrijediti $\alpha_0 > \alpha_1 > \alpha_2 > \ldots > \alpha_v$), a β faktor koji definira kaznu za smještaj u sljedeći dan. Očekivan odnos ovih konstanti je $\alpha_0 > \ldots > \alpha_v > \beta$.

Daljnje poopćenje opisanog pristupa je uvesti funkciju kazne definiranu za proizvoljna dva termina koja je monotono padajuća s obzirom na vremenski razmak između termina; u tom slučaju potrebno je imati na raspolaganju točnu informaciju o datumu i vremenu početka svakog termina. Tada se može definirati funkcija $distance(t_i, t_j)$ kao funkcija koja vraća razliku (u satima) od početka termina t_1 do početka termina t_2. Uz tako definiranu funkciju, funkciju kazne za vremensku blikost smještaja može se definirati na sljedeći način:

$$tPenalty(t_i, t_j) = \alpha^{1-\kappa \frac{distance(t_i,t_j)}{24}}. \tag{4.42}$$

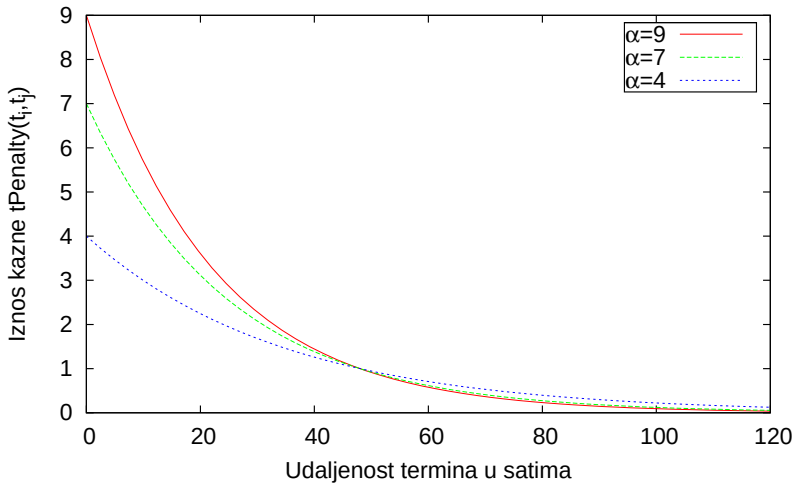
Funkcija $tPenalty(t_i, t_j)$ uz različite je vrijednosti parametra α te uz $\kappa = 0.5$ prikazana na slici 4.5.

Kažnjavanje vremenske bliskosti dva kolegija koja je potrebno dodatno razmaknuti (tj. koji se nalaze u skupu R) a koja su smještena u dane d_i i d_j može se modelirati na sljedeći način. Neka je s Δ označena razlika u danima, tj. $\Delta = |d_i - d_j|$. Neka je konstruirana monotono padajuća funkcija $d(\Delta)$:

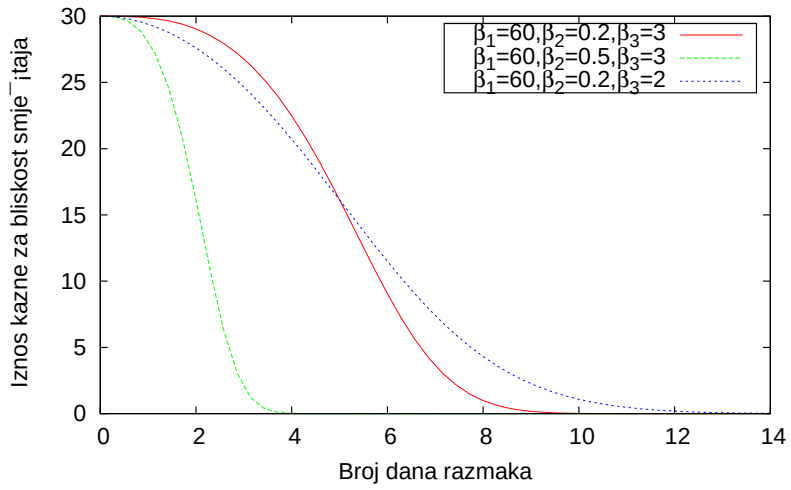
$$d(\Delta) = \frac{\beta_1}{1 + exp((\beta_2 \cdot \Delta)^{\beta_3})} \tag{4.43}$$

pri čemu su $\beta_1 \geq 0$, $\beta_2 \geq 0$ i $\beta_3 \geq 0$. Izgled ove porodice funkcija za različite je vrijednosti ovih parametara prikazan na slici 4.6.

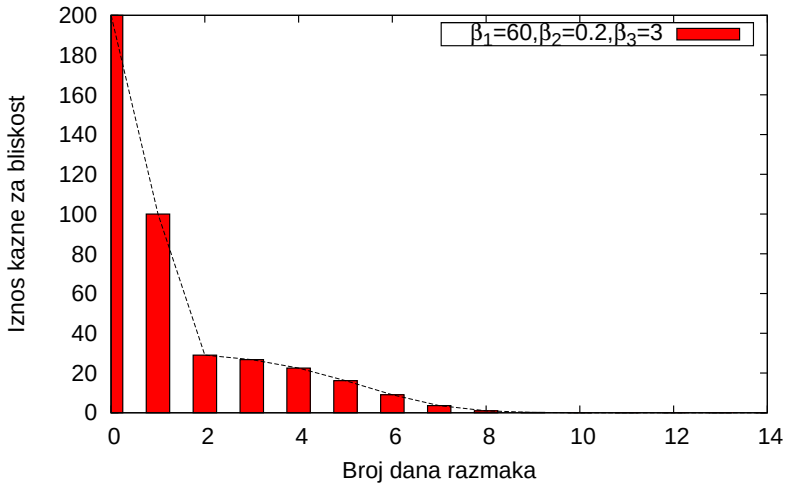
Ovakva se funkcija već može koristiti za funkciju kazne. Međutim, funkcija ima relativno mali omjer kazne za kolegije smještene u isti dan i kazne za kolegije smještene na razmaku od dva dana. Kako bi se dodatno naglasila kazna za smještaj para kolegija



Slika 4.5: Funkcija kazne vremenske bliskosti, za različite vrijednosti konstante α.



Slika 4.6: Funkcija kazne bliskosti u danima.



Slika 4.7: Konačni oblik funkcije kazne bliskosti u danima.

u isti dan ili odmah u susjedni, konačna funkcija kazne $dPenalty$ definirana je kao modifikacija fukcije d:

$$dPenalty(d_i, d_j) = \begin{cases} \gamma_1, & \text{ako je } \Delta = 0 \\ \gamma_2, & \text{ako je } \Delta = 1 \\ d(\Delta), & \text{ako je } \Delta \geq 2 \end{cases} \tag{4.44}$$

pri čemu se očekuje da za konstante γ_1 i γ_2 vrijedi: $\gamma_1 >> \gamma_2 > d(\Delta)$. Konstanta γ_1 omogućava definiranje visoke kazne za slučaj da su kolegiji smješteni u isti dan; konstanta γ_2 omogućava definiranje nešto niže kazne za slučaj da su kolegiji smješteni u susjedne dane; konačno, ako to nije slučaj, kazna se dodjeljuje u skladu s funkcijom $d(.)$. Izgled ove funkcije uz $\gamma_1 = 200$, $\gamma_2 = 100$, $\beta_1 = 60$, $\beta_2 = 0.2$ i $\beta_3 = 3$ prikazan je na slici 4.7.

Za izračun kvalitete rasporeda potrebno je još definirati način na koji se računa kazna između dva kolegija c_j i c_k koja dijele studente, odnosno na koji se način utvrđuje djelotvorni broj dijeljenih studenata $n^*_{j,k}$. Ideja je sljedeća: ako je veza između dva kolegija c_j i c_k student s_i kojemu su ta oba kolegija u u njegovom skupu predmeta modula (dakle, $c_j \in C^m_{s^m_i}$ i $c_k \in C^m_{s^m_i}$), djelotvorni broj dijeljenih studenata $n^*_{j,k}$ uvećava se za iznos δ_1 (primjerice, za $\delta_1 = 1$). Ako to nije slučaj te ako se oba kolegija drže na istoj godini, tj. $y_{c_j} = y_{c_k}$ koja je ujedno i godina na kojoj se predaje modul na kojem je student, djelotvorni broj dijeljenih studenata $n^*_{j,k}$ uvećava se za iznos $\delta_2 < \delta_1$, primjerice

za $\delta_2 = 0.5$. Ako niti to ne vrijedi, ali vrijedi da je jedan kolegij na godini na kojoj je student a drugi nije, djelotvorni broj dijeljenih studenata $n^*_{j,k}$ uvećava se za iznos $\delta_3 < \delta_2$, primjerice za $\delta_3 = 0.25$. Konačno, ako niti ovo nije zadovoljeno, djelotvorni broj dijeljenih studenata $n^*_{j,k}$ uvećava se za iznos $\delta_4 < \delta_3$, primjerice za $\delta_4 = 0.125$. Uz tako definiran djelotvorni broj dijeljenih studenata između kolegija c_j i c_k, definiramo kaznu $cPenalty(c_j, c_k)$ na sljedeći način:

$$cPenalty(c_j, c_k) = n^*_{j,k} \cdot min(w_j, w_k). \tag{4.45}$$

Kako bi se za par kolegija (c_j, c_k) izračunala kazna $cPenalty$, ideja je pogledati koliko taj par djelotvorno dijeli studenata, ali i kolika je subjektivna težina tih kolegija. Ako su u pitanju jedan lakši kolegij i jedan teži kolegij, ideja je da kazna bude manja nego za slučaj da su oba kolegija teška. Stoga se kao kazna $cPenalty$ uzima djelotvorni broj studenata pomnožen sa subjektivnom težinom lakšeg od dva kolegija.

Konačni izraz za kaznu $penalty(\mathbf{X})$ čitavog rasporeda tada se može definirati kako slijedi:

$$penalty(\mathbf{X}) = \sum_{(c_a, c_b) \in R} dPenalty(d_{c_a}, d_{c_b}) + \sum_{i=1}^{p} \sum_{j=i+1}^{p} tPenalty(t_i, t_j) \cdot r(t_i, t_j) \tag{4.46}$$

gdje je:

$$r(t_i, t_j) = \sum_{c_a | x_{a,i}=1} \sum_{c_b | x_{b,j}=1} cPenalty(c_a, c_b). \tag{4.47}$$

Prva suma u izrazu (4.46) predstavlja kaznu za blizak smještaj parova kolegija koje je potrebno razmicati i iznos te kazne ne ovisi o broju zajedničkih studenata koje imaju parovi kolegija niti o težini samih kolegija. Druga suma u izrazu (4.46) računa kaznu za svaka dva termina (uz $t_i < t_j$) kao kaznu uslijed blizine tih termina pomnoženu kaznom uslijed djelotvornog broja studenata koji dijele parovi kolegija smještenih u promatrane termine. Izraz zahtjeva da je $t_i < t_j$ jer se pretpostavlja je tvrdo ograničenje da u isti termin ne mogu biti smještena dva kolegija koja dijele barem jednog studenta biti zadovoljeno. Ako to nije slučaj, može se dopustiti i $t_i \leq t_j$ pri čemu bi funkciju (4.42) trebalo modificirani na sličan način kao što je to učinjeno za dobivanje funkcije (4.44) – za slučaj da je $distance(t_i, t_j) = 0$, umjesto predloženog izraza trebalo bi kao vrijednost kazne koristiti veliki pozitivan broj, a za slučaj da je $distance(t_i, t_j) > 0$ koristiti izraz

(4.46). Time će se optimizacijski proces najprije koncentrirati da riješi sve situacije gdje su kolegiji koji dijele studente smješteni u isti termin, a nakon toga će se optimirati kvaliteta prema drugim kriterijima.

4.5.2 Inačica problema

Jedna od inačica ovog problema je problem izrade rasporeda ponovljenih provjera znanja. Kako se očekuje da je veći dio studenata provjere položio tijekom redovnih termina provjera, broj termina u koje treba razmjestiti ponovljene provjere svih kolegija bitno je manji (obično 50% originalno raspoloživih termina). Kalendarski su termini definirani tako da počinju odmah nakon redovnih provjera znanja. Primjerice, ako redovne provjere završavaju u utorak, prvi termin za ponovljene provjere može biti već u srijedu. U takvom scenariju nije prihvatljivo da termin ponovljene provjere nekog kolegija bude odmah dan nakon originalne provjere. Zbog toga se pri izradi rasporeda ponovljenih provjera kao jedan od podataka uzima i originalno načinjeni raspored te se uvodi kažnjavanje blizine originalnog termina i ponovljenog termina za provjeru kolegija.

4.6 Raspored prostorija za provjere znanja

Problem izrade rasporeda prostorija problem je koji predstavlja logički nastavak problema izrade rasporeda provjera znanja. Jednom kada je riješen problem izrade rasporeda provjera znanja, sigurno je da je svakom kolegiju dodijeljen točno jedan termin, te da je na raspolaganju dovoljan broj prostorija za održavanje svih zakazanih provjera unutar jednog termina, uz pretpostavku da se prostorije zauzimaju racionalno. U praksi se je pokazalo da ovaj posljednji zahtjev ne mora biti nužno ispunjen te da kolegiji s malo studenata znaju zauzeti velike prostorije kako bi studenti "komforno" pisali provjeru i kako bi bili dovoljno daleko jedni od drugih da niti teoretski ne mogu prepisivati. Time pak za druge kolegije ne ostaje dovoljno mjesta kako bi održali provjeru koja je također dodijeljena u isti termin. Takve situacije, kada su se pojavljivale, obično su bile rješavane dogovorom više kolegija koji bi studente izmiješali u istu prostoriju, računajući da studenti koji pišu različite provjere pa budu izmiješani unutar prostorije mogu sjediti bitno bliže a da se ne dogodi prepisivanje. Međutim, da bi se ovakve situacije u potpunosti izbjegle, poželjno je unaprijed načiniti i kompletan raspored prostorija za svaku od provjera znanja.

Formalno, ovaj se problem može opisati na sljedeći način (prema [Čupić et al., 2010]).

- Definiran je vremenski disjunktan skup termina $T = \{T_1, T_2, ..., T_p\}$ u koje su raspoređene provjere znanja.
- Skup $C_k = \{c_{k,1}, c_{k,2}, \ldots\} \subseteq C$ predstavlja skup svih kolegija koji su raspoređeni u termin t_k, gdje je C skup svih kolegija koji imaju raspoređenu provjeru znanja. Ako se ovaj problem rješava kao nastavak na prethodno opisani problem, tada se skup C_k može definirati na sljedeći način: $C_k = \{c_j \mid x_{j,k} = 1, j = 1, \ldots, o\}$.
- Neka je n_j broj studenata koji trebaju nazočiti provjeri na kolegiju c_j. Ti brojevi mogu biti zadani ili se kao u prethodnom problemu mogu izračunati kao

$$n_j = \sum_{i=1}^{n} u_{i,j} \qquad (j = 1, \ldots, o). \tag{4.48}$$

- Skup $R = \{r_1, r_2, ..., r_l\}$ predstavlja skup svih prostorija u koje je moguće smještati ispite kolegija. Pretpostavka je da jedna prostorija može biti dodijeljena samo jednom kolegiju u jednom terminu – nema dijeljenja prostorija.
- Skup $R_i \subseteq R$ predstavlja skup prostorija koje se slobodne u terminu T_i, $T_i \in T$.
- Oznaka $\mathit{stud}(r_i, c_j)$ predstavlja broj studenata koji je kolegij c_j spreman staviti u prostoriju r_i, ako bi mu ta prostorija bila dodijeljena za provođenje provjere.
- Oznaka $\mathit{staff}(r_i, c_j)$ predstavlja broj djelatnika nastavnog osoblja koje bi kolegij c_j angažirao za čuvanje ispita u prostoriji r_i ako bi mu ta prostorija bila dana za provođenje ispita.
- Oznaka $\mathit{building}(r_i)$ predstavlja zgradu u kojoj je prostorija r_i smještena. Prilikom izrade rasporeda prostorija ideja je pripaziti da prostorije dodijeljene jednom kolegiju ne budu previše dislocirane.
- Oznaka $\mathit{floor}(r_i)$ predstavlja kat zgrade u kojoj se nalazi prostorija r_i.

Uvođenjem $\mathit{stud}(r_i, c_j)$ omogućeno je svakom kolegiju precizno podešavanje broja studenata za svaku potencijalno raspoloživu prostoriju. Naime, treba uvažiti činjenicu da sve prostorije nisu jednake – postoje prostorije s ravnim podom, prostorije s kosim

podom (poput amfiteatara) i slično. Ovisno o načinu na koji se provodi provjera, pojedini kolegiji u istu prostoriju mogu stavljati više ili manje studenata i upravo ovaj parametar daje djelatnicima svakog kolegija mogućnost finog podešavanja. Iz istog se razloga uvodi mogućnost definiranja $staff(r_i, c_j)$, kako bi svaki kolegij mogao dati realnu procjenu koliko će zapravo biti "skupo" tom kolegiju pridijeliti promatranu prostoriju.

Zadatak izrade rasporeda prostorija je za svaki kolegij c_j pronaći $AR_j \subseteq R$, odnosno skup prostorija koje su mu dodijeljene za provođenje njegove provjere znanja. Ideja je sljedeća. Za svaki termin T_i potrebno je pronaći particiju skupa R_i u disjunktne podskupove $\mathcal{R}_\rangle = \{R_{i,1}, ..., R_{i,\phi}, R_{unused}\}$, $\phi = |C_i|$, $R_{i,j} \cap R_{i,k} = \emptyset, \forall j \neq k$, tako da vrijedi $\forall c_j \sum_{r \in R_{i,j}} stud(r, c_j) \geq n_j$. Potrebno je dakle načiniti dekompoziciju skupa prostorija koje su raspoložive u promatranom terminu u skup disjunktnih skupova prostorija pridijeljenih svakom od kolegija koji imaju provjeru u tom terminu te izdvojiti sve nepotrebne sobe u zaseban skup. Pri tome suma kapaciteta prostorija dodijeljenih kolegiju c_j (kako ih vidi taj kolegij) mora biti dovoljna da je moguće smjestiti sve studente tog kolegija.

Uz taj zahtjev, dodaju se još dva. Prvo, za svaki termin $T_i \in T$ ukupni broj djelatnika koje je potrebno angažirati za čuvanje provjera u tom terminu treba biti minimalni:

$$minimize f(\mathcal{R}) = \sum_{c_{i,j} \in C_i, r \in R_{i,j}} staff(r, c_j).$$

Ovaj zahtjev automatski rezultira odbacivanjem svih prostorija koje su zauzete a zapravo predstavljaju višak s obzirom na broj studenata na kolegiju jer se njihovim uzimanjem ništa ne dobiva a troše se dodatni djelatnici.

Drugi zahtjev posljedica je uobičajene prakse da na većim kolegijima uvijek postoji barem jedan dodatni djelatnik – *šetač* koji tijekom provođenja provjere obilazi sve prostorije, odgovara na studentska pitanja te daje pojašnjenja u slučaju nejasnoća, pogrešaka ili drugih problema s ispitnim pitanjima. Za veće kolegije koji se provode u većim institucijama, bez dodatnih zahtjeva na raspored, generirani rasporedi ovaj zadatak mogu učini vrlo zahtjevnim jer je sasvim realan scenarij u kojem šetač treba obići četiri prostorije teoretski smještene u četiri različite zgrade i eventualno čak i ne sve u prizemlju. Ono što je svakako poželjno je dobiti prostornu kompaktnost dodijeljenih prostorija kako bi se izbjegli takvi scenariji.

Kompaktnost rasporeda može se postići minimizacijom ukupnog puta koji šetač mo-

ra načiniti kako bi obišao sve dodijeljene prostorije i opet se vratio do prve prostorije. Ovo je problem koji je u znanosti izuzetno dobro proučen i već je spomenut u poglevlju 1. Radi se o *Problemu trgovačkog putnika* [Johnson and McGeoch, 1997]. Kao što je poznato, taj je problem već sam po sebi $\mathcal{NP}$-potpuni problem. Kako nije u interesu konstruiranje problema čiji su već i potproblemi $\mathcal{NP}$-potpuni problemi, u okviru definicije ovog problema relaksira se postavljeni zahtjev u drugi koji također osigurava razumnu prostornu kompaktnost. Pri tome je ideja iskoristiti informacije koje postoje o prostorijama: zgradu u kojoj se prostorija nalazi te na kojem je katu u toj zgradi. Kvaliteta rasporeda prostorija jednog konkretnog kolegija tada se računa temeljem broja zgrada u kojima se nalaze dodijeljene prostorije te temeljem broja različitih katova na kojima su prostorije smještene. Što je više zgrada uključeno u raspored prostorija jednog kolegija, taj je raspored lošiji. Najbolje bi bilo imati sve prostorije na istom katu jedne zgrade. Ako to nije moguće, onda je sljedeće nešto lošije rješenje imati prostorije koje su na dva kata iste zgrade, itd. Najlošija su pak rješenja ona u kojima se prostorije nalaze u više zgrada. Na taj se način očekuje da manji kolegiji budu razmješteni na jednom katu, srednje veliki kolegiji na više katova iste zgrade a tek oni masovni na više katova u više zgrada.

Osim navedenih kriterija, praksa je pokazala da u prisustvu više tipova prostorija (ravni pod, srednje velike prostorije s kosim podom, veliki amfiteatri te konačno računalni laboratoriji u kojima se također mogu pisati provjere znanja iako nisu za to namijenjeni) djelatnici imaju preferencije prema pojedinim tipovima. Stoga je u model uključena i poželjnost prostorije, i to na sljedeći način. Ravne prostorije su najpoželjnije. Prepisivanje u njima je dosta teško i lagano ih je nadzirati. Kosi amfiteatri su manje poželjni; u njima je prepisivanja od studenata smještenih ispred olakšano i nadzor je otežan. Računalni laboratoriji najmanje su poželjni jer u njima studenti sjede poprilično blizu i tu je prepisivanje najlakše. Prilikom izrade rasporeda stoga bi trebalo načiniti najkvalitetniju moguću dodjelu prostorija uzimajući u obzir ova opažanja.

Ovako postavljen model problema izrade rasporeda prostorija za provjere znanja od svakog kolegija očekuje definiranje dva parametra za svaku prostoriju: broj studenata koje je kolegij spreman smjestiti u tu prostoriju te broj djelatnika potreban za čuvanje u toj prostoriji. Dakako, ovi se parametri inicijalno svi mogu postaviti na razumne pretpostavljene vrijednosti pa su intervencije potrebne samo na kolegijima koji doista

nešto žele korigirati.

Čvrsta ograničenja problema mogu se opisati na sljedeći način. Neka je indikatorska varijabla $v_{i,j,k} = 1$ ako je prostorija r_i kolegiju c_j dodijeljena u terminu t_k, a $v_{i,j,k} = 0$ inače. Neka je indikatorska varijabla $x_{j,k} = 1$ ako je ispit kolegija c_j smješten u termin t_k, a $x_{j,k} = 0$ inače (ovo je izlaz prethodno opisanog problema izrade rasporeda provjera znanja, i u ovom se problemu tretira kao konstantna vrijednost). Neka je $f_{i,k}$ indikatorska varijabla za koju vrijedi: $f_{j,k} = 1$ ako je u terminu t_k prostorija r_i slobodna, a $f_{j,k} = 0$ inače. Sljedeća ograničenja moraju vrijediti:

$$\sum_{k=1}^{p}\sum_{i=1}^{l} v_{i,j,k} \cdot (1 - x_{j,k}) = 0 \qquad (j = 1, \ldots, o) \tag{4.49}$$

$$\sum_{j=1}^{o} v_{i,j,k} \leq 1 \qquad (i = 1, \ldots, l; k = 1, \ldots, p) \tag{4.50}$$

$$\sum_{k=1}^{p}\sum_{i=1}^{l} v_{i,j,k} \cdot stud(i,j) \geq n_j \qquad (j = 1, \ldots, o) \tag{4.51}$$

$$\sum_{i=1}^{l}\sum_{j=1}^{o}\sum_{k=1}^{p} v_{i,j,k} \cdot f_{i,k} = 0 \tag{4.52}$$

$$v_{i,j,k} = 0 \text{ ili } 1 \qquad (i = 1, \ldots, l; j = 1, \ldots, o; k = 1, \ldots, p) \tag{4.53}$$

Ograničenje (4.49) osigurava da kolegij c_j nema dodijeljenih prostorija niti u jednom terminu koji nije njegov termin. Ograničenje (4.50) osigurava da je u svakom terminu prostorija dodijeljena najviše jednom kolegiju. Ograničenje (4.51) osigurava da je za svaki kolegij zauzeto dovoljno prostorija da se u njih mogu rasporediti svi studenti. Ograničenje (4.52) osigurava da su prostorije pridijeljene kolegijima samo ako su u tim terminima slobodne. Konačno, ograničenje (4.53) osigurava valjanu domenu vrijednosti varijabli $v_{i,j,k}$.

4.6.1 Optimizacijski problem

Kako bi se osiguralo da se u slučaju nepostojanja valjanog rasporeda ipak dobije maksimalno dobar raspored, ograničenje (4.51) može se promatrati kao meko ograničenje. Temeljem prethodno opisanih zahtjeva tada se mogu definirati sve komponente koje je potrebno uzeti u obzir prilikom definicije optimizacijskog problema.

- *Broj studenata koji načinjenim rasporedom nemaju gdje pisati provjeru znanja* – nakon izrade rasporeda moguće je da je nekim kolegijima dodijeljeno premalo prostorija.

- *Broj mjesta u prostorijama koja su nepopunjena* – nakon izrade rasporeda moguće je da je kolegiju dodijeljeno previše prostorija ili da su dodijeljene prevelike prostorije; rješenja u kojima kolegij od 10 studenata ima dodijeljenu prostoriju za 70 studenata potrebno je izbjegavati.

- *Ukupni broj djelatnika potreban za provođenje provjera znanja* – ako postoji mogućnost biranja između prostorije koja troši jednog djelatnika i prostorije koja troši dva djelatnika, prednost treba dati prostoriji koja troši jednog djelatnika.

- *Mjera prostorne kompaktnosti prostorija dodijeljenih kolegijima* – ova mjera procjenjuje koliko je opterećenje šetača koji mora obilaziti sve prostorije dodijeljene pojedinom kolegiju.

- *Mjera poželjnosti dodijeljenih prostorija kolegijima* – ova mjera procjenjuje koliko su prostorije dodijeljene kolegiju poželjne za čuvanje provjere znanja.

Jedan od načina izračuna mjere prostorne kompaktnosti i poželjnosti je kako opisan u nastavku. Neka je *preferabilityPenalty* početno postavljen na 0. Za svaku prostoriju koja je u nekom terminu dodijeljena nekom kolegiju, ta se vrijednost uvećava za ψ_1 ako je prostorija ravna, ψ_2 ako je prostorija amfiteatar odnosno za ψ_3 ako je prostorija laboratorij. U općem slučaju može se definirati skup vrsta prostorija Ψ, svakoj prostoriji pridijeliti njezinu vrstu te iznos kazne kada se koristi ta vrsta prostorije modelirati odgovarajućom konstantom ψ_x.

Neka je *locationPenalty* početno postavljen na 0. Za svaki raspoređeni kolegij c_i potrebno je izračunati n_b – broj zgrada u kojima se nalaze njemu dodijeljene prostorije, te n_f – broj katova na kojima se nalaze njemu dodijeljene prostorije. Za taj kolegij tada je potrebno uvećati *locationPenalty* za $\kappa_1 \cdot (n_b - 1)$ te za $\kappa_2 \cdot (n_f - 1)$.

Ukupna mjera prostorne kompaktnosti i poželjnosti tada se može izračunati kao zbroj ove dvije komponente, tj.:

$$spacePrefPenalty = preferabilityPenalty + locationPenalty. \tag{4.54}$$

Na stvarnom primjeru izrade rasporeda prostorija na Fakultetu elektrotehnike i računarstva, empirijski se je došlo do sljedećih vrijednosti: $\psi_1 = 0$, $\psi_2 = 9$, $\psi_3 = 15$, $\kappa_1 = 70$ te $\kappa_2 = 23$.

Uz ovako definirane kriterije, problem izrade rasporeda prostorija za provjere znanja je optimizacijski višekriterijski problem u okviru kojeg je potrebno:

- minimizirati broj studenata koji načinjenim rasporedom nemaju gdje pisati provjeru znanja,
- minimizirati broj mjesta u prostorijama koja su nepopunjena,
- mimimizirati ukupni broj djelatnika potreban za provođenje provjera znanja, te
- minimizirati mjeru prostorne kompaktnosti prostorija dodijeljenih kolegijima i mjeru poželjnosti dodijeljenih prostorija kolegijima, odnosno *spacePrefPenalty*.

Problem je moguće rješavati kao višekriterijski problem, ili ga je moguće primjenom odgovarajućih tehnika svesti na jednokriterijski i riješavati na uobičajen način.

4.7 Raspoređivanje timova

Problem raspoređivanja timova još je jedan od problema koji se javlja na akademskim institucijama [Čupić and Franović, 2010]. Problem se pojednostavljeno može opisati na sljedeći način. Studenti nekog kolegija razdijeljeni su timove i svakom je timu pridružen voditelj koji je djelatnik. Kako djelatnika na kolegiju ima manje no timova, svaki je djelatnik voditelj određenog broja timova. Timovi rade projekte i u određenim terminima trebaju doći do svojeg voditelja kako bi ih on ispitao. U tom smislu, raspoređivanje timova je problem u kojem je u skup unaprijed zadanih termina potrebno razmjestiti timove tako da niti jedan student u terminu koji je dodijeljen timu nema kolizija s prethodnim zauzećima (predavanjima, laboratorijskim vježbama i slično). Istovremeno, ako je u nekom terminu na raspolaganju više prostorija, ne smije se dogoditi situacija da u taj termin bude raspoređeno više timova istog voditelja jer voditelj troši čitav termin na ispitivanje jednog tima. Za svaki se tim zna broj studenata a za svaku prostoriju njezin kapacitet; u prostoriju se istovremeno može staviti i više timova, tako dugo dok je kapacitet prostorije veći ili jednak broju studenata koji su smješteni u nju. Smještanje

tima pri tome je atomarno – nije moguće da nekoliko studenata tima bude smješteno u jedan termin a ostatak u drugi (ili drugu prostoriju).

Formalno, ovaj se problem može opisati na sljedeći način. Neka je S skup svih studenata koji su razmješteni u timove. Za svakog studenta $s_i \in S$ zadan je skup postojećih zauzeća $P_i = \{p_{i,1}, p_{i,2}, \ldots\}$. Definiran je vremenski disjunktan skup termina $T = \{t_1, t_2, ..., t_p\}$ u koje je moguće raspoređivati timove. Definiran je skup prostorija $R = r_1, \ldots, r_l$ koje se mogu koristiti za raspoređivanje. Neka je $a_{i,k}$ indikatorska varijabla za koju vrijedi: $a_{i,k} = 1$ ako je u terminu t_k dozvoljeno koristiti prostoriju r_i za raspoređivanje, a $a_{j,k} = 0$ inače. Neka je definiran $stud(i)$ kao ukupni broj studenata koji se mogu smjestiti u prostoriju r_i. Definiran je skup timova $\tau = \{\tau_1, \ldots, \tau_o\}$ koje treba rasporediti. Neka $\tau_i^S \subset S$ označava podskup studenata koji su smješteni u tim τ_i. Razdioba studenata po timova je disjunktna odnosno ne postoje dva tima koja dijele studente. Neka je n_i broj studenata koji su pridijeljeni u tim τ_i. Neka je $D = 1, \ldots, h$ skup identifikatora voditelja timova (voditelja ukupno ima h). Neka je $voditelj(i) \in D$ funkcija koja za tim τ_i vraća indeks dodijeljenog voditelja. Neka je $x_{i,j,k}$ indikatorska varijabla za koju vrijedi $x_{i,j,k} = 1$ ako je tim τ_j smješten u termin t_k u prostoriju r_i, a $x_{i,j,k} = 0$ inače. Potrebno je pronaći vrijednosti za $x_{i,j,k}$ tako da vrijedi:

$$\sum_{i=1}^{l}\sum_{k=1}^{p} x_{i,j,k} = 1 \qquad (j = 1, \ldots, o) \tag{4.55}$$

$$\Big|\{voditelj(j) \mid x_{i,j,k} = 1; i = 1, \ldots, l; j = 1, \ldots, o\}\Big| = \sum_{i=1}^{l}\sum_{j=1}^{o} x_{i,j,k} \qquad (k = 1, \ldots, p) \tag{4.56}$$

$$\sum_{j=1}^{o}\sum_{k=1}^{p} x_{i,j,k} \cdot (1 - a_{i,k}) = 0 \qquad (i = 1, \ldots, l) \tag{4.57}$$

$$\sum_{j=1}^{o}\sum_{k=1}^{p} x_{i,j,k} \cdot n_j \leq stud(i) \qquad (i = 1, \ldots, l) \tag{4.58}$$

$$\sum_{i=1}^{l}\sum_{j=1}^{o}\sum_{k=1}^{p} x_{i,j,k} \cdot \sum_{s_a \in \tau_i^S} preklapanje(P_a, t_j) = 0 \tag{4.59}$$

$$x_{i,j,k} = 0 \text{ ili } 1 \qquad (i = 1, \ldots, l; j = 1, \ldots, o; k = 1, \ldots, p) \tag{4.60}$$

Pri tome ograničenje (4.55) osigurava da je svaki tim smješten u točno jedan termin u točno jednu prostoriju. Ograničenje (4.56) osigurava da u jednom terminu nema više

timova koji pripadaju istom voditelju. Najprije se konstruira skup svih identifikatora voditelja čiji su timovi razmješteni u bilo koju prostoriju u promatranom terminu i gleda se kardinalitet tog skupa. On mora biti jednak broju timova koji su razmješteni u tom terminu; ako to nije slučaj, više od jednog tima ima istog voditelja. Ograničenje (4.57) osigurava da je timu prostorija dodijeljenja samo ako se u tom terminu smije koristiti. Ograničenje (4.58) osigurava da je ukupni broj studenata smješten u prostoriju u nekom terminu manji je ili jednak kapacitetu te prostorije. Ograničenje (4.59) osigurava da studenti načinjenim razmještajem nemaju kolizija s prethodnim zauzećima. Konačno, ograničenje (4.60) osigurava valjanost vrijednosti varijabli $x_{i,j,k}$.

4.7.1 Optimizacijski problem

Uz pretpostavku da nije moguće doći do valjanog rasporeda, uzrok može biti kršenje bilo kojeg od prethodno navedenih ograničenja (4.55)-(4.60). Ograničenja (4.55), (4.56), (4.57), (4.58) i (4.60) su ograničenja od kojih se uobičajeno ne želi odstupiti (dakle čvrsta ograničenja). Međutim, ograničenje (4.59) predstavlja ograničenje koje se može pretvoriti u meko. Time je moguće definirati problem izrade rasporeda timova kao optimizacijski problem kod kojeg je cilj minimizirati količinu preklapanja studenata u timovima s postojećim obavezama.

Osim ovog kriterija, evidentno je da se u svrhu izrade kvalitetnijeg rasporeda mogu definirati još dva kriterija:

- *kontinuiranost rasporeda voditelja* koji bi više vrednovao rasporede u kojima voditelj ima manje prekida, te
- *kontinuiranost rasporeda studenta* koji bi više vrednovao rasporede u kojima se dodijeljeni termini dobro uklapaju u postojeće rasporede studenata. Ako su timovi pri tome prikladno odabrani, uvođenje ove mjere će imati smisla. Međutim, ako su timovi sastavljeni od heterogenih skupina studenata gdje različiti studenti inače imaju različite rasporede, ova mjera neće imati pretjeranog smisla.

Problem je moguće rješavati odgovarajućim algoritmima za rješavanje problema višekriterijske optimizacije. Alternativno, problem se može svesti na jednokriterijski i tada rješavati na uobičajen način.

4.8 Izrada prezentacijskih grupa za seminare

Problem izrade prezentacijskih grupa za seminare problem je slaganja studentskih makrogrupa grupa iz studentskih mikrogrupa pri čemu treba paziti na kvalitetu tako složenih makrogrupa. Recimo najprije nekoliko riječi o porijeklu ovog problema. Uvođenjem Bolonjskog procesa na Fakultet elektrotehnike i računarstva definiran je novi kolegij *Seminar*. Kao predradnju za pohađanje tog kolegija, svaki student (neovisno o upisanom smjeru) definira listu potencijalnih voditelja poredanih prema prioritetu. Temeljem takvih lista svih studenata i temeljem dotadašnjeg uspjeha samih studenata za svakog se voditelja definira mikrogrupa koju čine studenti koje će voditi na tom kolegiju. Posljedica ovakvog načina dodjele voditelja su mikrogrupe koje su sastavljene od studenata koji nastavu pohađaju po različitim rasporedima, turnusima i koji pripadaju raznorodnim smjerovima. Zbog toga je već i prilikom vođenja jedne mikrogrupe voditeljima pravi izazov pronaći termin kada su svi studenti njihove mikrogrupe slobodni.

Pred kraj kolegija, svaki od studenata treba pred drugim studentima održati prezentaciju posla koji je načinio tijekom semestra. Za potrebe ovih prezentacija, formiraju se prezentacijske makrogrupe – više mikrogrupa udružuje se u jednu veću makrogrupu. Primjerice, neka su 504 studenata razdijeljeni u 126 mikrogrupa (4 studenta po mikrogrupi); svaka mikrogrupa ima dodijeljenog jednog voditelja. Potrebno je oformiti 18 makrogrupa tako da se grupira 7 po 7 mikrogrupa; time svaka makrogrupa ima ukupno $7 \cdot 4 = 28$ studenata.

Pri provođenju ovakvog grupiranja potrebno je osigurati kvalitetu rasporeda prema sljedećim kriterijima.

1. *Ujednačenost veličina makrogrupa.* U općem slučaju, broj upisanih studenata na kolegij Seminar ne mora biti višekratnik od 4, a dakako niti broj studenata dodijeljenih svakom voditelju ne mora biti jednak; ovo za posljedicu ima postojanje mikrogrupa s različitim brojem studenata. Stoga je grupiranje to kvalitetnije što je broj studenata unutar svake makrogrupe bliži nekom ciljanom broju $\overline{n}$. Primjerice, može se zahtjevati da veličina makrogrupe bude što bliža broju 30. U makrogrupu pri tome nije moguće dodati samo pojedine studente iz mikrogrupa; ili ih se dodaje sve ili niti jedan.

2. *Maksimalno mogući broj slobodnih termina za prezentacije.* Pred kraj semestra

svaki student treba održati prezentaciju. Za makrogrupu od 30 studenata to znači da je potrebno pronaći 30 termina kada su svi studenti te makrogrupe slobodni kako bi se u tim terminima održale prezentacije. Problem je sljedeći: zauzeće jedne mikrogrupe zapravo je unija zauzeća svih studenata. Mikrogrupa je slobodna samo u terminima u kojima je svaki od studenata koji čine mikrogrupu slobodan. To pak znači da slobodni termini mikrogrupe odgovaraju presjeku slobodnih termina studenata koji čine tu mikrogrupu. Makrogrupe se sastoje od više spojenih mikrogrupa – stoga su slobodni termini makrogrupe jednaki presjeku slobodnih termina pripadnih mikrogrupa; ovaj skup uz loš način grupiranja uobičajeno teži ka praznom skupu. Stoga je prilikom grupiranja mikrogrupa potrebno voditi računa da se grupiranje obavi na način koji će ostaviti maksimalnu količinu slobodnih termina u svakoj makrogrupi.

3. *Ujednačenost razdiobe studenata po makrogrupama.* Ocjenjivanje studenata unutar svake makrogrupe radi se na način da jedan dio dobiva odličnu ocjenu, jedan dio ocjenu vrlo dobar, pa sve do ocjene dovoljan. Problem koji se s takvim pristupom javlja je sljedeći: neka je jedna makrogrupa sastavljena od studenata koji do tog trenutka spadaju u 5% studenata s najvišim prosjekom ocjena na fakultetu. Iako je za pretpostaviti da bi svi oni načinili odlične prezentacije, jedan dio bi ih dobio ocjenu odličan, jedan dio ocjenu vrlo dobar pa sve do dijela koji bi dobio ocjenu dovoljan. Da se izbjegnu ovakve situacije, prilikom stvaranja makrogrupa jedan od ciljeva svakako treba biti da u njoj postoji ravnomjerna raspodjela studenata (podjednak broj visokorangiranih, srednjerangiranih i niskorangiranih studenata). Kako se u makrogrupu međutim ne mogu uključivati pojedinačni studenti već samo čitave mikrogrupe, ovo nije jednostavan zadatak.

Heurističke metode za rješavanje ovog problema koje ne uzimaju izravno u obzir zahtjev da se osigura maksimalno mogući broj termina u kojima su makrogrupe slobodne opisane su u [Paunović, 2009]. Navedeni se problem uspješno može rješavati i tehnikama evolucijskog računanja i to direktno primjenom višekriterijske optimizacije ili pak svođenjem na problem jednokriterijske optimizacije.

Formalno, ovaj se problem može opisati na sljedeći način. Neka je skup M skup svih postojećih mikrogrupa, tj. $M = \{m_1, \ldots, m_n\}$. Svaki m_j pri tome odgovara skupu studenata koji je smješten u tu mikrogrupu. Broj studenata tada se može izračunati

kao:

$$N_S = \sum_{m \in M} |m|. \tag{4.61}$$

Očekivani broj makrogrupa tada je:

$$N_M = \left\lceil \frac{N_S}{\overline{n}} \right\rceil \tag{4.62}$$

gdje je $\overline{n}$ prosječni (ciljani) broj studenata smještenih u svaku makrogrupu. Uobičajeno, $\overline{n} >> \max_{m \in M} |m|$. Neka je s N_{M*} označen maksimalni dopustivi broj makrogrupa koji ćemo računati kao $N_{M*} = N_M \cdot \tau_1$ gdje je $\tau_1 \geq 1$ konstanta. Neka je s $N_{S,min} = N_S \cdot \tau_2$ označen minimalni broj studenata koji je dopustiv u nekoj makrogrupi a s $N_{S,max} = N_S \cdot \tau_3$ maksimalni broj studenata koji je dopustiv u nekoj makrogrupi; pri tome mora vrijediti $0 < \tau_2 \leq 1$ i $\tau_3 \geq 1$. Iznimno, kako kao maksimalni broj makrogrupa dozvoljavamo više grupa od N_M, dozvoljava se da neke od njih budu prazne, tj. da nemaju dodijeljenu niti jednu mikrogrupu.

Neka je definirana indikatorska varijabla $x_{i,j}$ za koju vrijedi $x_{i,j} = 1$ ako je mikrogrupa i pridijeljena makrogrupi j, a $x_{i,j} = 0$ inače. Moraju vrijediti sljedeća ograničenja:

$$\sum_{j=1}^{N_{M*}} x_{i,j} = 1 \qquad (i = 1, \ldots, n) \tag{4.63}$$

$$\sum_{i=1}^{n} x_{i,j} \cdot |m_i| \leq N_{S,max} \qquad (j = 1, \ldots, N_{M*}) \tag{4.64}$$

$$\sum_{i=1}^{n} x_{i,j} \cdot |m_i| \geq N_{S,min} \vee \sum_{i=1}^{n} x_{i,j} \cdot |m_i| = 0 \qquad (j = 1, \ldots, N_{M*}) \tag{4.65}$$

Ograničenje (4.63) osigurava da je svaka mikrogrupa smještena u točno jednu makrogrupu. Ograničenje (4.64) osigurava da broj studenata od kojeg se sastoji svaka makrogrupa ne prelazi dopušteni maksimalni broj studenata. Ograničenje (4.65) osigurava da je makrogrupa ili prazna ili ima barem onoliko studenata koliko je propisani minimum.

4.8.1 Optimizacijski problem

Uporabom ograničenja (4.63), (4.64) i (4.65) definirana du ograničenja koja svako rješenje mora zadovoljiti (čvrsta ograničenja). Međutim, uporabom te vrste ograničenja

nije moguće govoriti o kvaliteti rješenja. Stoga je potrebno definirati način kako mjeriti kvalitetu rješenja.

Neka vrijedi sljedeće: svi su studenti međusobno uspoređeni temeljem nekog kriterija (primjerice, uspjehom na razredbenom ispitu, dotadašnjim prosjekom ocjena i slično) te su temeljem tog kriterija rangirani. Ako ukupno imamo N_S studenata, broj različitih rangova pri tome ne mora biti jednak N_S jer mogu postojati jednako uspješni studenti koji će imati jednake rangove. Primjerice, ako postoje tri jednako uspješna najuspješnija studenta, sva tri će imati rang 1; prvi sljedeći student tada će imati rang 4 čime rangovi 2 i 3 neće biti dodijeljeni nikome. Za potrebe definiranja ovog problema pretpostavlja se da su efekti ovakve raspodjele rangova minimalni, odnosno da se neće dogoditi situacija u kojoj svi studenti imaju isti rang ili situacija u kojoj je broj iskorištenih rangova bitno manji od broja studenata.

Neka je s $r_{p\%}$ označen rang studenta od kojeg postoji $p\%$ bolje ili jednako rangiranih studenata; tako će primjerice rang $r_{20\%}$ odgovarati rangu onog studenta za kojeg postoji 20% jednako-ili-bolje rangiranih studenata i 80% lošije rangiranih studenata. Za svaku stvorenu makrogrupu ideja je provjeriti raspodjelu rangova studenata koji su smješteni u tu makrogrupu. Stoga se za svaku makrogrupu definira q podgrupa studenata. Neka je s $\mathcal{F}$ označen skup svih makrogrupa; $\mathcal{F} = \{f_1, f_2, \ldots, f_{N_{M*}}\}$. Studente koji su smješteni u f_j tada je moguće dobiti na sljedeći način:

$$\{\cup m_i | x_{i,j} = 1; i = 1, \ldots, n\}. \tag{4.66}$$

Neka je s $f_{i,j}$ označen skup studenata smještenih u makrogrupu f_i takvih da im za rang r vrijedi: $r_{\frac{j-1}{q}\cdot 100\%} < r \leq r_{\frac{j}{q}\cdot 100\%}$ pri čemu je $0 < j \leq q$. f_i je tada skup ovih podgrupa. Ako se odabere $q = 5$, tada će makrogrupa f_3 imati podgrupe $f_{3,20\%}$, $f_{3,40\%}$, $f_{3,60\%}$, $f_{3,80\%}$, $f_{3,100\%}$. Grupa $f_{3,20\%}$ će sadržavati sve studente koji su dodijeljeni u makrogrupu f_3 i koji pripadaju u 20% najboljih studenata u generaciji; grupa $f_{3,40\%}$ će sadržavati sve studente koji su dodijeljeni u makrogrupu f_3 i koji pripadaju u sljedećih 20% najboljih studenata u generaciji (koji jesu u 40% najboljih ali nisu u prvih 20% studenata), itd.

Temeljem ovih podataka može se definirati očekivani prosječni broj studenata u svakoj podgrupi. Primjerice, ako se broj makrogrupa fiksira upravo na N_M, tada se očekivani broj studenata N_{S*} u svakoj podgrupi može izračunati prema izrazu:

$$N_{S*} = \frac{N_S}{N_M} \cdot \frac{1}{q}. \tag{4.67}$$

Za svako predloženo rješenje može se utvrditi koliko je odstupanje stvarno dodijeljenog broja studenata s obzirom na očekivani broj studenata. Neka je s $\Theta(\phi)$ označena funkcija praga:

$$\Theta(\phi) = \begin{cases} 1, & \phi > 0 \\ 0, & \text{inače.} \end{cases} \tag{4.68}$$

Neka je najveći dozvoljeni broj makrogrupa $\leq N_{M*}$. U rješenju $\mathbf{X}$ stvarni broj makrogrupa je:

$$N_{M'} = \sum_{j=1}^{N_{M*}} \Theta\left(\sum_{i=1}^{n} x_{i,j}\right). \tag{4.69}$$

Temeljem tog broja, očekivani prosječni broj studenata u svakoj podgrupi u promatranom rješenju bit će:

$$N_{S'} = \frac{N_S}{N'_M} \cdot \frac{1}{q}. \tag{4.70}$$

Stoga se funkcija odstupanja od prosječnog očekivanog broja studenata u podgrupama može definirati na sljedeći način:

$$odstupanjaRangova(\mathbf{X}) = \sum_{j=1}^{N_{M*}} \left\{ \Theta\left(\sum_{i=1}^{n} x_{i,j}\right) \cdot \sum_{f_{i,j} \in f_j} \left| |f_{i,j}| - N_{S'} \right|^{\alpha_1} \right\}^{\alpha_2} \tag{4.71}$$

gdje su α_1 i α_2 konstante kojima se može utjecati na način kažnjavanja odstupanja unutar podgrupa te na razini pojedinih makrogrupa. Odstupanje od željenog prosječnog broja studenata unutar svake makrogrupe može se definirati na sljedeći način:

$$odstupanjaVelicina(\mathbf{X}) = \sum_{j=1}^{N_{M*}} \left\{ \Theta\left(\sum_{i=1}^{n} x_{i,j}\right) \cdot \left| \frac{N_S}{N_M} - \sum_{f_{i,j} \in f_j} |f_{i,j}| \right|^{\alpha_3} \right\}^{\alpha_4} \tag{4.72}$$

gdje su α_3 i α_4 konstante kojima se može utjecati na način kažnjavanja odstupanja unutar podgrupa te na razini pojedinih makrogrupa. Kvaliteta rješenja s obzirom na slobodne termine može se izračunati na sljedeći način:

$$slobodniTermini(\mathbf{X}) = \sum_{j=1}^{N_{M*}} \zeta\left(\Theta\left(\sum_{i=1}^{n} x_{i,j}\right), \xi(f_i)\right). \tag{4.73}$$

Funkcija $\xi(a)$ računa broj slobodnih termina u nekom zadanom razdoblju u kojem su svi studenti iz predanog skupa studenata a slobodni. To je broj termina u kojima prezentacijska makrogrupa sastavljena od navedenih studenata može održavati prezentaciju. Funkcija $\zeta(a, b)$ modelira utjecaj količine slobodnih termina pojedine makrogrupe na funkciju koja opisuje ukupnu količinu slobodnih termina. Uporaba klasičnog reda potencija kao što je to bio slučaj kod modeliranja prethodnih kazna ovdje nije potpuno primjeren. Naime, za neku makrogrupu koja se sastoji od 30 studenata poželjno je da ima što je moguće veći broj slobodnih termina, tako dugo dok je taj broj manji ili sumjerljiv s brojem 30, jer je to broj prezentacija koje treba odraditi. Međutim, daljnji porast broja slobodnih termina te makrogrupe u ukupnoj kvaliteti trebao bi imati manji utjecaj no porast broja slobodnih termina neke druge makrogrupe koja još nije dostigla potreban broj slobodnih termina. Dakako, ovo vrijedi samo za makrogrupe koje imaju dodijeljene studente. Stoga bi jedna moguća definicija ovakve funkcije bila:

$$\zeta(a, b) = \begin{cases} 0, & a = 0 \\ b^{\alpha_5} & b < \theta \\ \theta^{\alpha_5} + (b - \theta)^{\alpha_6} & \text{inače.} \end{cases} \tag{4.74}$$

Ako je broj studenata smještenih u promatranu makrogrupu jednak nuli, funkcija $\zeta()$ također vraća nula. Ako makrogrupa ima dodijeljene studente, i ako je broj termina manji od željenog praga, broj slobodnih termina potencira se na α_5; ako je broj slobodnih termina veći od praga, dio do praga potencira se na α_5 a ostatak na α_6, pri čemu želimo da je $\alpha_6 < \alpha_5$.

Uz ovako definirane funkcije, pronalazak dobrog rješenja svodi se na pronalazak rješenja koje zadovoljava čvrsta ograničenja te minimizaciju kriterijskih funkcija:

- $odstupanjaRangova(\mathbf{X})$ i
- $odstupanjaVelicina(\mathbf{X})$

te maksimizaciju kriterijske funkcije

- $slobodniTermini(\mathbf{X})$.

Rješavanju ovog problema može se pristupiti bilo uporabom višekriterijskih optimizacijskih tehnika bilo svođenjem na problem jednokriterijske optimizacije i uoprabom odgovarajućih tehnika za njihovo rješavanje.

Poglavlje 5

Evolucijski algoritmi primijenjeni na odabrane probleme raspoređivanja

U prethodnom poglavlju formalno je definirano nekoliko problema raspoređivanja. U okviru ovog rada za svaki od definiranih problema osmišljeni su i implementirani algoritmi zasnovani na evolucijskom računanju koji ih uspješno rješavaju. Za neke od problema algoritmi su osmišljeni i implementirani u okviru studentskih završnih i diplomskih radova. Svaki od ovih algoritama provjeren je i ispitan na nizu praktičnih problema koje je trebalo rješavati u proteklih pola desetljeća, i čija su rješenja korištena u praksi.

U ovom poglavlju opisani su algoritmi evolucijskog računanja primijenjeni na podskup tih problema.

5.1 Primjena algoritama evolucijskog računanja na problem jednostavnog raspoređivanja

Formalni model problema *jednostavnog raspoređivanja* naveden je u poglavlju 4.1. Radi se o problemu kod kojeg je unaprijed zadan skup studenata te skup slijedova termina. Osnovni je zadatak svakog studenta smjestiti u jedan od tih slijedova, pazeći pri tome da takvim smještajem student nije u koliziji s postojećim rasporedom nastavnih obaveza. Kod optimizacijske inačice ovog problema definirana su četiri kriterija koja je

potrebno zadovoljiti u što je mogućoj većoj mjeri (u ovom slučaju, kršenja se pokušavaju minimizirati):

- kazna($\mathbf{X}$) – mjeri u kojoj su količini razriješena preklapanja dodijeljenih termina i postojećih obaveza pojedinih studenata,
- prekapacitiranost($\mathbf{X}$) – mjeri u kojoj su mjeri zadovoljena ograničenja kapaciteta slijedova,
- nebalansiranost($\mathbf{X}$) – mjeri u kojoj su mjeri zadovoljena očekivanja da broj studenata u slijedu bude jednak prosječnom očekivanom kapacitetu slijedova te
- nekvaliteta($\mathbf{X}$) – mjeri koliko se loše načinjeni raspored uklapa u postojeća zauzeća studenata.

Za ovaj optimizacijski problem u nastavku je prikazano pet algoritama evolucijskog računanja koji ga rješavaju:

- eliminacijski genetski algoritam,
- algoritam roja čestica,
- *Max-Min* mravlji algoritam,
- jednostavan imunološki algoritam te
- algoritam klonske selekcije.

Za svaki od tih algoritama, prekapacitiranost dvorana je čvrsto ograničenje, odnosno zahtjeva se da uvijek vrijedi prekapacitiranost($\mathbf{X}$) $= 0$. Kriterij nebalansiranosti neće se razmatrati. Preostala dva ograničenja, kazna($\mathbf{X}$) i nekvaliteta($\mathbf{X}$) nastojat će se svesti na minimum.

Kako je za većinu ovih algoritama prikaz rješenja zajednički, prije opisa algoritama opisan je korišteni prikaz rješenja koristeći elemente objektno-orijentirane paradigme; konkretno, koristi se programski jezik Java.

Rješenje problema predstavljeno je parametriziranim razredom `Solution<P>`, gdje je `P` parametar (izvorni kod 5.1) koji predstavlja tip podatka korišten za pohranu rezultata

```
1 public class Solution<P> {
2   public int[] studentGroup;
3   public int[] groupSize;
4   public P penalty;
5   public Object context;
6 }
```

Izvorni tekst programa 5.1: Struktura podataka za prikaz rješenja.

vrednovanja tog rješenja. Uporabom parametriziranih razreda osigurano je razdvajanje strukture podataka korištene za prikaz rješenja i strukture podataka korištene za vrednovanje rješenja čime se dopušta implementacija različitih načina vrednovanja.

Polje studentGroup predstavlja vektor koji ima onoliko elemenata koliko se studenata raspoređuje; na i-toj lokaciji tog vektora nalazi se indeks slijeda u koji je student smješten (u nastavku ćemo koristiti sinonim *grupa*). Pomoćno polje groupSize je polje koje za svaku grupu čuva broj studenata koji su smješteni u tu grupu. Iako se taj broj može dobiti vrlo jednostavno analizom sadržaja polja studentGroup, redundantna pohrana ove informacije omogućit će brže izvođenje određenih evolucijskih operatora.

Varijabla penalty omogućava pohranu kvalitete rješenja. Kako u trenutku definiranja ovog razreda još nije poznat točan način kako će kvaliteta biti mjerena, iskorištena je mogućnost parametrizacije razreda kojom je točan tip moguće definirati u trenutku uporabe razreda. Konačno, varijabla context služi za pohranu dodatnih informacija u samo rješenje; varijabla je uključena u strukturu rješenja kako bi se podržao dio algoritama evolucijskog računanja kojima osnovni skup podataka koje pamtimo neće biti dovoljan.

Sljedeće što treba definirati je kako se rješenje vrednuje. U prikazu koji slijedi za svako se rješenje umjesto dobrote računa njegova kazna. Vrijednost izračunate kazne pohranjuje se u varijable razreda BasicPenalty, kako prikazuje izvorni kod 5.2.

```
1 public class BasicPenalty {
2   public int numberOfConflicts;
3   public double quality;
4 }
```

Izvorni tekst programa 5.2: Struktura podataka za prikaz kazne rješenja.

Varijabla numberOfConflicts čuva mjeru ukupnog broja konflikata. Ova mjera računa se kao zbroj mjera konflikata pojedinih studenata. Mjera konflikata pojedinog studenta

računa se kao broj termina slijeda u koji je student smješten a koji imaju preklapanje sa studentovim prethodno zakazanim nastavnim obavezama, neovisno o trajanju tog preklapanja. Tako primjerice, ako slijed ima 4 termina, mjera preklapanja studenta će biti broj iz skupa $\{0, 1, 2, 3, 4\}$.

Varijabla quality čuva ukupnu kaznu rasporeda uslijed njegove nekompaktnosti, pri čemu manji broj predstavlja kvalitetnije rješenje. Ukupna kazna rasporeda uslijed njegove nekompaktnosti dobiva se kao zbroj kazni zbog nekompaktnosti rasporeda pojedinih studenata. Kako bi se izračunala kazna uslijed nekompaktnosti rasporeda pojedinog studenta, u njegov postojeći raspored nastavnih aktivnosti ugrađuju se termini koje je student dobio smještanjem u jedan od slijedova termina. Za svaki se termin t_i iz dodijeljenog slijeda traži najbliži termin $t_{i,b}$ koji student ima isti dan a završava prije početka termina t_i, te najbliži termin $t_{i,a}$ koji student ima isti dan a počinje nakon kraja termina t_i. Ako $t_{i,b}$ i $t_{i,a}$ oba ne postoje, kazna zbog nekompaktnosti za termin $q(t_i)$ iznosi 0. U suprotnom, ako termin $t_{i,b}$ ne postoji, definira se da postoji i da mu je kraj u $-\infty$. Ako termin $t_{i,a}$ ne postoji, definira se da postoji i da mu je početak u $+\infty$. Neka je s t_1 označena razlika u minutama od kraja termina $t_{i,b}$ do početka termina t_i, a s t_2 razlika u minutama od kraja termina t_i do početka termina $t_{i,a}$. Kazna zbog nekompaktnosti rasporeda za termin t_i tada se računa prema izrazu:

$$q(t_i) = \frac{min(t_1, t_2)}{15}, \tag{5.1}$$

čime se kao iznos kazne zbog nekompaktnosti za taj termin uzima broj 15-minutnih intervala do najbliže postojeće studentove nastavne aktivnosti. Kazna zbog nekompaktnosti za promatranog studenta tada se definira kao zbroj kazni zbog nekompaktnosti za pojedine termine pridijeljenog slijeda.

Svi algoritmi koji su u nastavku prikazani za usporedbu dviju jedinki koriste sljedeće pravilo. Dvije jedinke imaju jednaku kaznu ako su im sve komponente kazne jednake. Jedinka i je bolja od jedinke j ako:

- i.numberOfConflicts manji od j.numberOfConflicts ili
- i.numberOfConflicts jednak j.numberOfConflicts i i.quality jednak j.quality.

Podatci temeljem kojih će se rješavati ovaj problem su:

1. popis identifikatora studenata koje treba rasporediti,

```
1  public class Data<P> {
2    public String[] allDates;
3    public String[] jmbags;
4    public Map<String,Integer> mapJmbagToIndex;
5    public Group[] groups;
6    public Map<String,Integer> mapGroupToIndex;
7    public P[][][] penalties;
8  }
9  public class Group {
10   public String name;
11   public int capacity;
12   public Term[] terms;
13 }
14 public class Term {
15   public String date;
16   public String startsAt;
17   public String endsAt;
18   public String room;
19 }
```

Izvorni tekst programa 5.3: Struktura podataka za prikaz nepromjenjivih podataka o problemu koji se rješava.

2. popis postojećih zauzeća studenata te

3. popis raspoloživih slijedova termina i ograničenja kapaciteta.

Temeljem ovih podataka moguće je izgraditi strukturu podataka sa statičkim podatcima, kako prikazuje izvorni kod 5.3.

Za pamćenje nepromjenjivih podataka o primjerku problema koji se rješava definiran je parametriziran razred Data. Polje allDates pri tome čuva datume svih dana u kojima postoji barem jedan termin u barem jednom slijedu. Polje jmbags je polje studentskih identifikatora a mapa mapJmbagToIndex omogućava brzi pronalazak pozicije identifikatora u polju. Polje groups je polje definiranih slijedova a mapa mapGroupToIndex omogućava brzi pronalazak pozicije slijeda u polju ako je poznato ime slijeda. Razred Group pri tome čuva parametre pojedinog slijeda, poput naziva, kapaciteta te niza pridruženih termina, gdje svaki termin ima definiran datum, početak, kraj te prostoriju.

Kako je kod ovog problema popis studenata kao i popis slijedova unaprijed zadan i nepromijenjiv, vrednovanje rješenja problema moguće je bitno ubrzati tako da se unaprijed izračunaju sve potrebne mjere kvalitete za pojedine studente. Tome upravo služi polje penalties, koje je dvodimenzijsko – za svakog studenta (indeks prve dimenzije) čuva parcijalne mjere kvalitete za smještaj tog studenta u svaki od postojećih slijedova (drugi indeks). To je zapravo matrica dimenzija *broj studenata* $\times$ *broj slijedova* pri

```
1  public class Population<P> implements Iterable<Solution<P>> {
2
3    // Privatne varijable
4    private Solution<P> members[];
5    private ISolutionFactory<P> factory;
6
7    // Konstruktor
8    public Population(int n, ISolutionFactory<P> factory);
9
10   // Ostale metode
11   public Solution<P> get(int index);
12   public void set(int index, Solution<P> solution);
13   public int size();
14   public Solution<P> createNewSolution();
15   public Solution<P> findBest(Comparator<Solution<P>> comparator);
16   public void sort(Comparator<Solution<P>> comparator);
17   public Iterator<Solution<P>> iterator();
18 }
```

Izvorni tekst programa 5.4: Struktura podataka za prikaz populacije rješenja.

čemu su elementi matrice odgovarajuće unaprijed izračunate kazne.

5.1.1 Pomoćni razredi

Svi algoritmi evolucijskog računanja koje ovdje opisujemo su populacijski algoritmi. Populacija je opisana razredom Population<P> kojeg prikazuje izvorni kod 5.4. Iz koda su izbačene implementacije pojedinih metoda.

U zaseban razred EvoService izdvojene su metode koje se dijele između više algoritama (izvorni kod 5.5).

Metoda initialize (...) ; za svaku poziciju od fromIndex do toIndex stvara novi slučajni primjerak rješenja koje zadovoljava čvrsta ograničenja, te rješenja vrednuje ako je predani evaluator različit od **null**. Metoda je korisna kako za inicijalizaciju čitavih populacija, tako i za inicijaliziciju dijelova populacije (što se koristi primjerice kod imunoloških algoritama).

Metoda choose (...) ; iz skupa cijelih brojeva od 1 do *size* bira onoliko različitih cijelih brojeva koliko je veliko polje predano kao prvi argument, i odabrane brojeve pohranjuje u to polje. Metoda se može koristiti na različitim mjestima, a primjer je k-turnirska selekcija gdje iz populacije od n jedinki treba odabrati k jedinki.

Metoda fill (...) u populaciju subpopulation kopira jedinke iz populacije population koje se nalaze na pozicijama predanim u polju indexes.

Metoda sort (...) koristeći predani komparator sortira predanu populaciju te upareno

```
1  public class EvoService {
2    public static <P> void initialize(
3      Data<P> data, Population<P> population,  int fromIndex,
4      int toIndex, RandomGen rgen, Evaluator<P> evaluator);
5
6    public static void choose(
7      int[] chosenIndexes, int size, RandomGen rgen);
8
9    public static <P> void fill(
10     Population<P> population,
11     Population<P> subpopulation, int[] indexes);
12
13   public static <P> void sort(
14     Population<P> population, int[] indexes,
15     Comparator<Solution<P>> comparator);
16
17   public static <P> void fixGroups(
18     Data<P> data, Solution<P> sol, RandomGen rgen);
19
20   protected static <P> void correctGroupCapacity(
21     Data<P> data, Solution<P> sol, int groupIndex,
22     int startFromGroupIndex, int startFromStudentIndex,
23     RandomGen rgen);
24
25   public static <P> void mutate(
26     Data<P> data, Solution<P> sol, RandomGen rgen,
27     double mutProb);
28
29   public static <P> void crossover(
30     Data<P> data, Solution<P> sol, RandomGen rgen,
31     Solution<P> parent1, Solution<P> parent2);
32 }
```

Izvorni tekst programa 5.5: Pomoćne metode korištene od strane više algoritama.

polje indeksa. Drugim riječima, sortiranje se provodi kao da se radi o sortiranju uređenih parova $(jedinka, indeks)$; zamjenom dviju jedinki u populaciji radi se automatski i zamjena indeksa u pridruženom polju indeksa.

Metoda fixGroups(...) provjerava jesu li u predanom rješenju prekršena čvrsta ograničenja, te ako za neku grupu jesu prekršena, pozivom metode correctGroupCapacity(...) rješava tu prekapacitiranost. Po završetku rada, predano rješenje sigurno zadovoljava čvrsto ograničenje o prekapacitiranosti grupa.

Metoda correctGroupCapacity(...) za grupu određenu argumentom groupIndex u rješenju sol obavlja korekciju broja dodijeljenih studenata. Tako dugo dok je grupa prekapacitirana, slučajnim se odabirom traži prva grupa u kojoj još ima mjesta (ako je startFromGroupIndex>=0, pretraga počinje od grupe čiji je to indeks); takva sigurno postoji jer je pretpostavka da su kapaciteti svih grupa dovoljni za razmještanje svih studenata. Stoga ako postoji prekapacitirana grupa, tada sigurno postoji i grupa koja ima slobodnih mjesta. Potom se počev od neke slučajne pozicije (ako je startFromStudentIndex<0) ili od pozicije startFromStudentIndex traži prvi student koji je smješten u tu prekapacitiranu grupu; on se potom prebacuje u prethodno pronađenu grupu sa slobodnim mjestom. Postupak se ponavlja tako dugo dok postoji prekapacitiranost grupe.

Metoda mutate(...) nad predanim rješenjem obavlja mutaciju. Pri tome se kao parametar predaje vjerojatnost mutacije grupe dodijeljene jednom studentu (terminologijom genetskog algoritma koji radi nad bitovnim nizovima, ovo bi odgovaralo vjerojatnosti mutacije jednog bita). Metoda garantira da će rješenje koje se na kraju dobije zadovoljavati čvrsta ograničenja jer će se po potrebi provesti dodatne korekcije (pozivima metode correctGroupCapacity(...)).

Metoda crossover (...) temeljem dva predana roditelja generira novo rješenje koje dio dodijeljenih grupa preuzima iz jednog roditelja a dio iz drugog. Operator križanja je implementacija uniformnog križanja. Metoda garantira da će rješenje koje se na kraju dobije zadovoljavati čvrsta ograničenja jer će se po potrebi provesti dodatne korekcije (pozivom metode fixGroups(...)).

5.1.2 Eliminacijski genetski algoritam

Implementaciju eliminacijskog genetskog algoritma za problem jednostavnog raspoređivanja prikazuje izvorni tekst programa 5.6.

```
1  public class TournamentGA<P> implements IAlgorithm<P> {
2
3    // Parametri i strukture podataka
4    private RandomGen rgen = new RandomGenDefault();
5    private Data<P> data;
6    private ISolutionFactory<P> solFactory;
7    private Evaluator<P> evaluator;
8    private Population<P> population;
9    private double mutProb;
10   private double crossProb;
11   private Comparator<Solution<P>> comparator;
12
13   // Radni podatci
14   private Solution<P> bestSolution;
15
16   // "Globalni" spremnik za operator selekcije
17   private int[] chosenIndexes;
18   private Population<P> chosenSolutions;
19
20   public TournamentGA(Data<P> data, ISolutionFactory<P> solFactory,
21     Evaluator<P> evaluator, Comparator<Solution<P>> comparator,
22     int popSize, int tournamentSize,
23     double mutProb, double crossProb) {
24
25     this.data = data;
26     this.solFactory = solFactory;
27     this.evaluator = evaluator;
28     this.comparator = comparator;
29
30     population = new Population<P>(popSize, solFactory);
31     EvoService.initialize(
32       data, population, 0, population.size()-1,
33       rgen, evaluator
34     );
35
36     chosenIndexes = new int[tournamentSize];
37     chosenSolutions = new Population<P>(tournamentSize, solFactory);
38
39     this.mutProb = mutProb;
40     this.crossProb = crossProb;
41
42     bestSolution = population.findBest(comparator);
43   }
```

Izvorni tekst programa 5.6: Genetski algoritam za rješavanje problema jednostavnog raspoređivanja.

```
45    public void singleStep() {
46      for(int i = 0; i < population.size(); i++) {
47        EvoService.choose(chosenIndexes, population.size(), rgen);
48        EvoService.fill(population, chosenSolutions, chosenIndexes);
49        EvoService.sort(chosenSolutions, chosenIndexes, comparator);
50
51        Solution<P> sol = solFactory.createNewSolution();
52        if(rgen.nextDouble()<crossProb) {
53          // Krizaj i mutiraj...
54          EvoService.crossover(
55            data, sol, rgen, chosenSolutions.get(0),
56            chosenSolutions.get(1)
57          );
58        } else {
59          // Samo mutiraj najboljeg...
60          sol.copyFrom(chosenSolutions.get(0));
61        }
62        EvoService.mutate(data, sol, rgen, mutProb);
63
64        evaluator.evaluate(sol);
65        population.set(chosenIndexes[chosenIndexes.length-1], sol);
66        if(comparator.compare(bestSolution, sol)>0) {
67          bestSolution = sol;
68        }
69      }
70    }
71  }
```

Nastavak izvornog koda s prethodne stranice.

U konstruktoru razreda obavlja se stvaranje i inicijalizacije populacije kao i pomoćne populacije za troturnirsku selekciju. Metoda singleStep (...) je metoda koja obavlja onoliko turnira koliko populacija ima jedinki (čime zapravo odrađuje ekvivalent jedne izmjene generacije kod generacijskog algoritma). Algoritam koristi pojednostavljenu verziju troturnirske selekcije: posredstvom slučajnog mehanizma odabiru se tri jedinke iz populacije. Dvije bolje stvaraju dijete koje se potom ubacuje u populaciju na mjesto trećeizabrane jedinke. Pri tome je vjerojatnost križanja određena parametrom *crossProb*. Ako se događa križanje, križaju se dvije bolje jedinke, dijete se mutira i ubacuje natrag u populaciju. Ako se križanje ne događa, tada se kao dijete radi kopija najbolje od izabranih jedinki, potom se dijete mutira i na kraju vraća u populaciju na mjesto najlošije od triju odabranih jedinki. Operator zamjene ne provjerava je li dijete lošije do jedinke koju izbacuju (međutim, algoritam je elitistički jer se uvijek zamjenjuje najgora od tri odabrane jedinke pa će najbolja sigurno ostati očuvana).

Rad algoritma prikazan je na problemu raspoređivanja gdje je 469 studenata potrebno razmjestiti u jedan od 3 slijeda termina. Svaki slijed sastoji se od dva termina.

Slijedovi se ukupno protežu kroz tri dana u tjednu, a u tih tri dana studenti imaju ukupno evidentirano 2239 postojećih zauzeća. Kapacitet svakog slijeda je 157 studenata, čime se u tri slijeda može razmjestiti ukupno 471 student. To znači da će svako stvoreno rješenje automatski zadovoljavati i uvjet balansiranosti.

Kretanja broja konflikata te kvalitete u najboljem rješenju kroz vrijeme prikazana su na slikama 5.1 i 5.2. Rezultati su dobiveni uz populaciju od 40 jedinki, vjerojatnost križanja 0.9 te vjerojatnost mutacije gena od 3 jedinične vjerojatnosti[1]. Iscrtani su svi rezultati kako bi se mogao steći dojam o tijeku optimizacije te raspršenosti pojedinih mjera. Slika 5.1a pri tome prati količinu kolizija koja preostaje u rasporedu i tu se vidi da je funkcija monotono padajuća. Primjer na kojem je algoritam pokrenut u optimumu ima 4 kolizije koje se ne daju razriješiti. Slika 5.1b prikazuje promjene u kazni zbog nekompaktnosti rasporeda. Tijekom prvog dijela optimizacije kazna zbog nekompaktnosti rasporeda raste na račun činjenice da algoritam uspjeva smanjivati količinu kolizija. Uz primjereno smanjen broj kolizija algoritam potom uspjeva pronaći sve kvalitetnija i kvalitetnija rješenja koja imaju sve manju kaznu zbog nekompaktnosti rasporeda.

Na slici 5.2 prikazani su isti podatci samo u sažetom obliku. Slika 5.2a prikazuje prosječan broj konflikata. Uz svaku je točku grafa prikazano i odstupanje u obliku jedne standardne devijacije na više i na niže. Na isti način generirana je i slika 5.2b na kojoj je te parametre moguće pratiti za kvalitetu rasporeda.

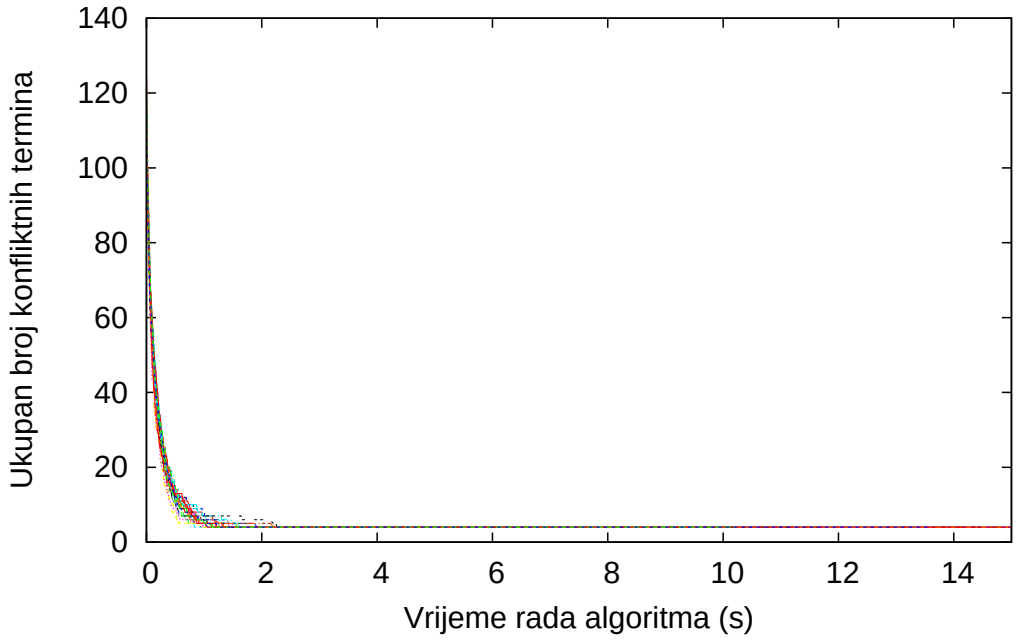
Kao što se vidi sa slika 5.1 i 5.2, genetski algoritam uspješno rješava zadani problem.

5.1.3 Algoritam roja čestica

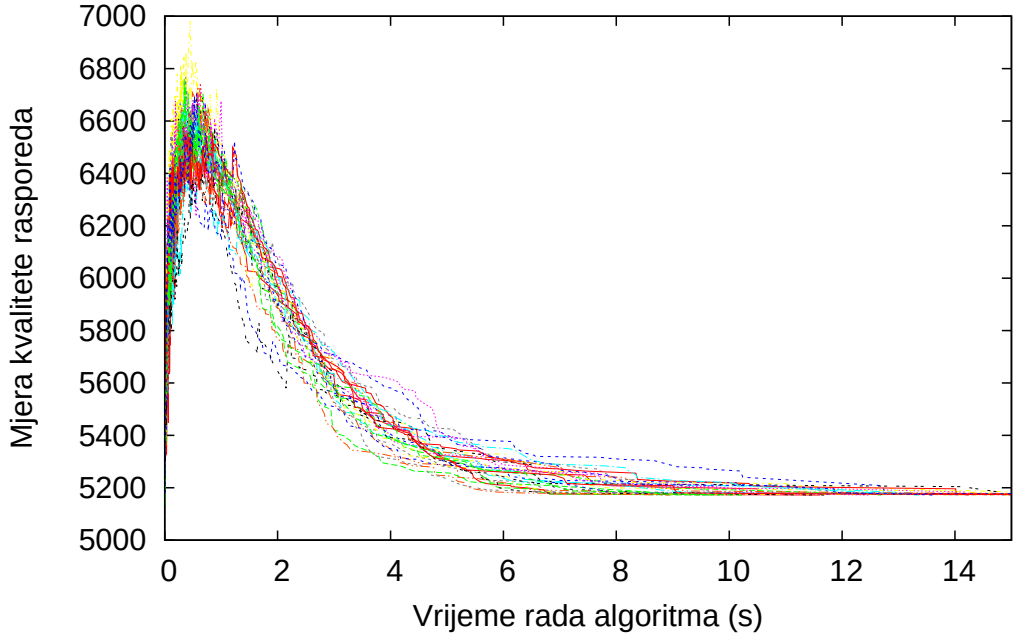
Implementaciju algoritma roja čestica za problem jednostavnog raspoređivanja prikazuje izvorni kod 5.7. Radi se o implementaciji s lokalnim susjedstvom pri čemu je korištena topologija prstena. Čestica i ima kao direktne susjede čestice $n+i-1 \; mod \; n$ i $n+i+1 \; mod \; n$, gdje je n ukupan broj čestica u populaciji. Veličina čitavog susjedstva čestice i definirana je parametrom ϵ koji se predaje konstruktoru. Time svaka čestica ima kao susjede sve čestice od $n+i-\epsilon \; mod \; n$ do $n+i+\epsilon \; mod \; n$.

Kako je problem koji ovdje rješavamo diskretan i kako svakom od studenata možemo pridijeliti jednu od svega nekoliko grupa, direktna implementacija prototipnog algorit-

[1] U ovom radu jedinična vjerojatnost mutacije gena za kromosom koji se sastoji od n gena definirana je kao $\frac{1}{n}$.

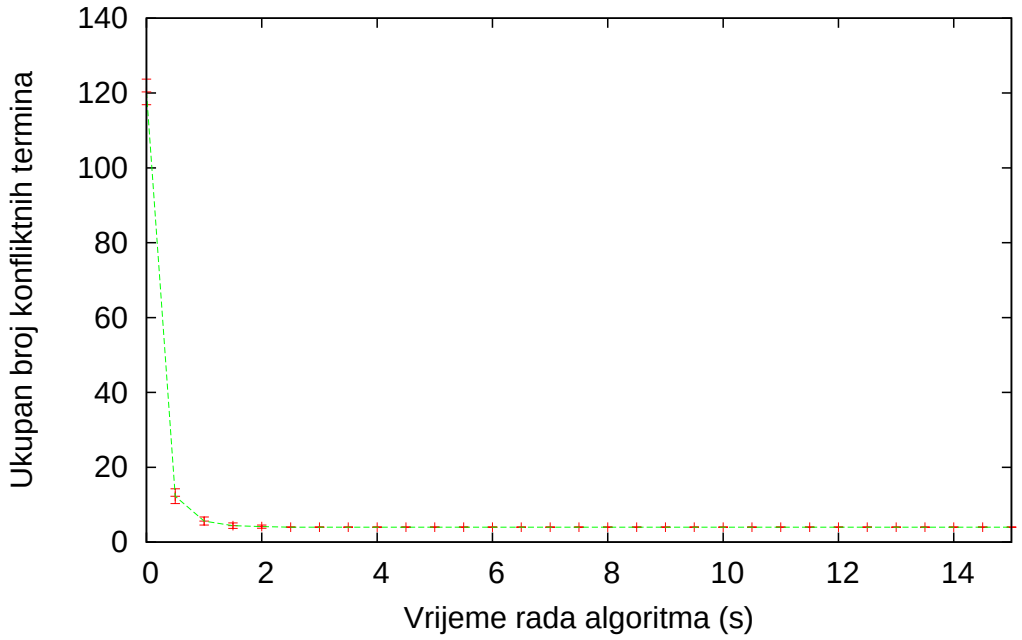


(a) Broj konflikata

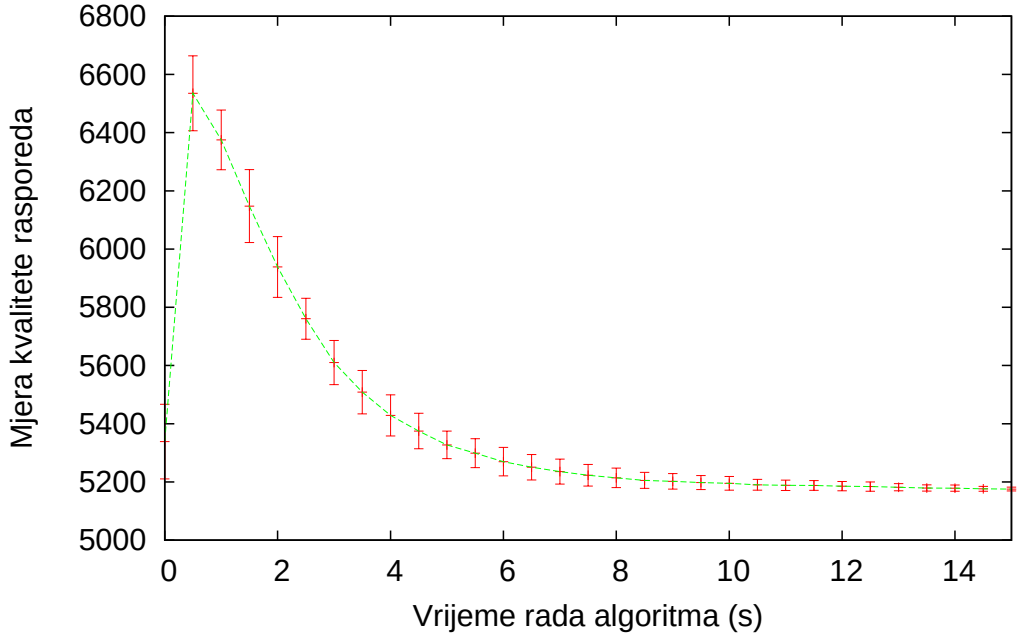


(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.1: Evolucija rješenja primjenom genetskog algoritma za rješavanje problema izrade rasporeda.



(a) Broj konflikata



(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.2: Evolucija rješenja primjenom genetskog algoritma za rješavanje problema izrade rasporeda – prosječni parametri.

ma roja čestica pokazuje izuzetno slabe rezultate. Naime, izvorni algoritam roja čestica osmišljen je za rješavanje optimizacijskih problema nad kontinuiranim domenama. Umjesto toga, ovdje je prikazana prilagođena verzija algoritma koja radi u cjelobrojnoj domeni. Algoritam ne koristi konstante C_1 i C_2 koje određuju individualnu i socijalnu komponentu niti vektore razlike između trenutnog rješenja i najboljih rješenja. Umjesto toga, u duhu ideje algoritma roja čestica, svaka čestica ima svoju trenutnu poziciju, pamti svoje najbolje pronađeno rješenje, ima definirano lokalno susjedstvo i pristup najboljem rješenju susjedstva kao i pristup globalno najboljem rješenju (što je slučaj kod potpuno informiranog algoritma roja čestica, engl. *Fully-informed PSO*).

Ažuriranje pozicije čestice obavlja se u skladu s definiranim vjerojatnostima. Za svakog studenta u 1% slučajeva grupa se bira slučajno. U preostalih 99% slučajeva postupak je sljedeći. U 10% slučajeva čestica zadržava grupu koju ima za tog studenta. U preostalih 90%, postupak je sljedeći. U 33% slučajeva grupa se preuzima iz globalno najboljeg rješenja; u 33% slučajeva grupa se preuzima iz najboljeg rješenja susjedstva a u preostalim slučajevima grupa se preuzima iz najboljeg rješenja te čestice. Ovo je jasnije prikazano u izvornom kodu 5.7 u unutrašnjoj petlji metode `singleStep()`. Varijabla `context` strukture koja predstavlja jedno rješenje (odnosno ovdje jednu česticu) iskorištena je za pamćenje reference na najbolje ikada pronađeno rješenje te čestice.

Kretanje broja konflikata te kvalitete rasporeda kroz vrijeme za 30 eksperimenata prikazano je na slici 5.3; zbirni rezultati prikazani su na slici 5.4. Rješavan je problem koji je prethodno opisan u podpoglavlju o genetskom algoritmu, a rezultati su dobiveni uz populaciju od 40 čestica te vrijednost parametra $\epsilon = 10$.

5.1.4 Algoritam Max-Min mravlji sustav

Implementaciju algoritma *Max-Min* mravlji sustav za problem jednostavnog raspoređivanja prikazuje izvorni tekst programa 5.8. Algoritam koristi dva pomoćna polja. Polje `heuristics` je polje dimenzija *broj studenata* $\times$ *broj grupa* koje na lokaciji $[i, j]$ sadrži vrijednost heurističke informacije smještaja studenta i u grupu j. Polje `trails` je polje jednakih dimenzija koje na lokaciji $[i, j]$ sadrži vrijednost feromonskog traga smještaja studenta i u grupu j.

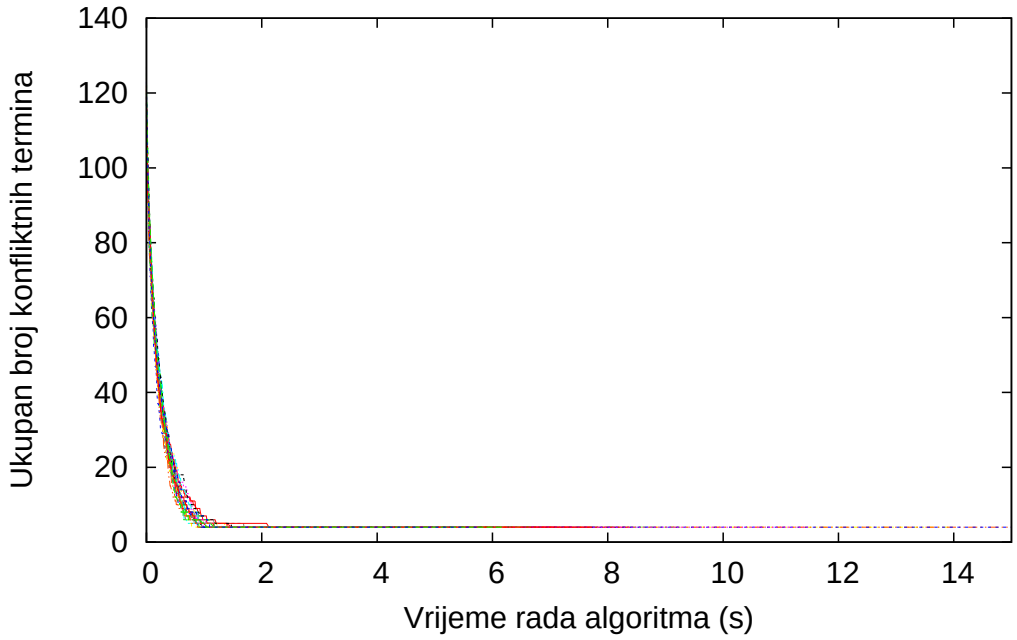
U ovom primjeru heuristička se informacija računa na sljedeći način. Za svakog se studenta pokuša načiniti smještaj u svaku moguću grupu te se za taj razmještaj izračuna

```
1  public class PSO<P> implements IAlgorithm<P> {
2
3    // Parametri i strukture podataka
4    private RandomGen rgen = new RandomGenDefault();
5    private Data<P> data;
6    private ISolutionFactory<P> solFactory;
7    private Evaluator<P> evaluator;
8    private Population<P> population;
9    private Comparator<Solution<P>> comparator;
10   private int numberOfParticles;
11   private int epsilon;
12
13   // Radni podatci
14   private Solution<P> bestSolution;
15
16   public PSO(Data<P> data, ISolutionFactory<P> solFactory,
17     Evaluator<P> evaluator, Comparator<Solution<P>> comparator,
18     int numberOfParticles, int epsilon) {
19
20     this.data = data;
21     this.solFactory = solFactory;
22     this.evaluator = evaluator;
23     this.comparator = comparator;
24     this.numberOfParticles = numberOfParticles;
25     this.epsilon = epsilon;
26     population = new Population<P>(numberOfParticles, solFactory);
27     EvoService.initialize(
28       data, population, 0, population.size()-1, rgen, evaluator
29     );
30     for(Solution<P> sol : population) {
31       sol.context = sol;
32     }
33     bestSolution = population.findBest(comparator);
34   }
```

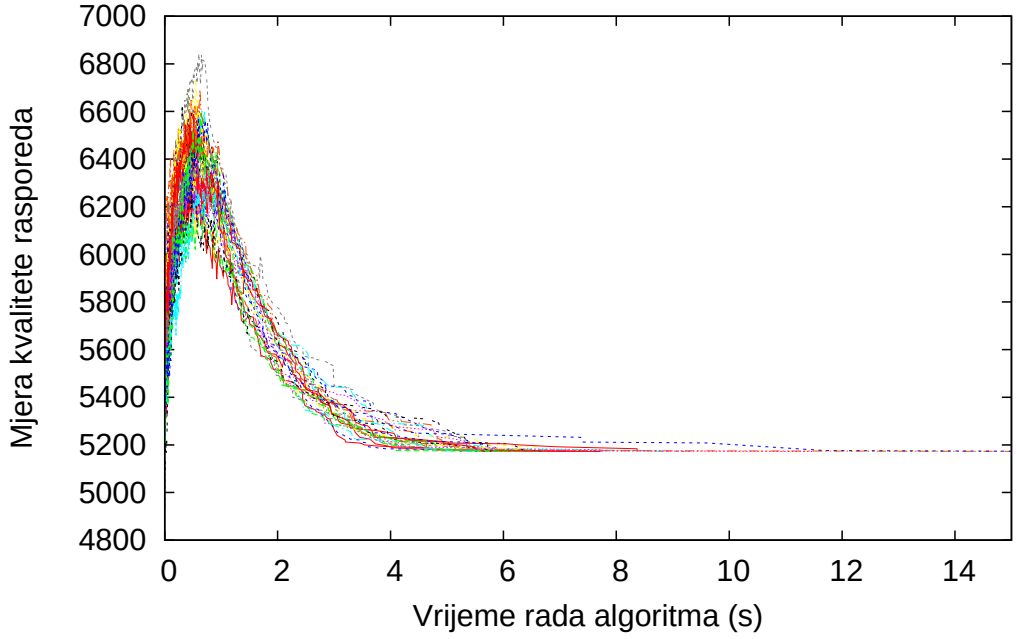
Izvorni tekst programa 5.7: Algoritam roja čestica za rješavanje problema jednostavnog raspoređivanja.

```
35    public void singleStep() {
36      for(int solIndex = 0; solIndex < numberOfParticles; solIndex++) {
37        Solution<P> currentSol = population.get(solIndex);
38        Solution<P> pbest = (Solution<P>)currentSol.context;
39
40        Solution<P> epsilonBestSol = null;
41        for(int i = -epsilon; i <= epsilon; i++) {
42          int pos = (numberOfParticles+solIndex+i) % numberOfParticles;
43          Solution<P> sol = population.get(pos);
44          Solution<P> solBest = (Solution<P>)sol.context;
45          if(epsilonBestSol==null ||
46            comparator.compare(epsilonBestSol, solBest)>0) {
47            epsilonBestSol = solBest;
48          }
49        }
50        Solution<P> newSol = solFactory.createNewSolution();
51        newSol.context = currentSol.context;
52        for(int i = 0; i < newSol.studentGroup.length; i++) {
53          if(rgen.nextDouble()<0.99) {
54            if(rgen.nextDouble()<0.10) {
55              newSol.studentGroup[i] = currentSol.studentGroup[i];
56            } else {
57              double d = rgen.nextDouble();
58              if(d<0.33) {
59                newSol.studentGroup[i] = bestSolution.studentGroup[i];
60              } else if(d < 0.66) {
61                newSol.studentGroup[i] = epsilonBestSol.studentGroup[i];
62              } else {
63                newSol.studentGroup[i] = pbest.studentGroup[i];
64              }
65            }
66          } else {
67            newSol.studentGroup[i] = rgen.nextInt(data.groups.length);
68          }
69          newSol.groupSize[newSol.studentGroup[i]]++;
70        }
71        EvoService.fixGroups(data, newSol, rgen);
72        evaluator.evaluate(newSol);
73        population.set(solIndex, newSol);
74      }
75      boolean bestUpdated = false;
76      for(int solIndex = 0; solIndex < numberOfParticles; solIndex++) {
77        Solution<P> currentSol = population.get(solIndex);
78        Solution<P> pbest = (Solution<P>)currentSol.context;
79        if(comparator.compare(currentSol, pbest)<0) {
80          currentSol.context = currentSol; pbest = currentSol;
81        }
82        if(comparator.compare(bestSolution, pbest)>0) {
83          bestSolution = pbest; bestUpdated = true;
84        }
85      }
86    }
87  }
```

Nastavak izvornog koda s prethodne stranice.

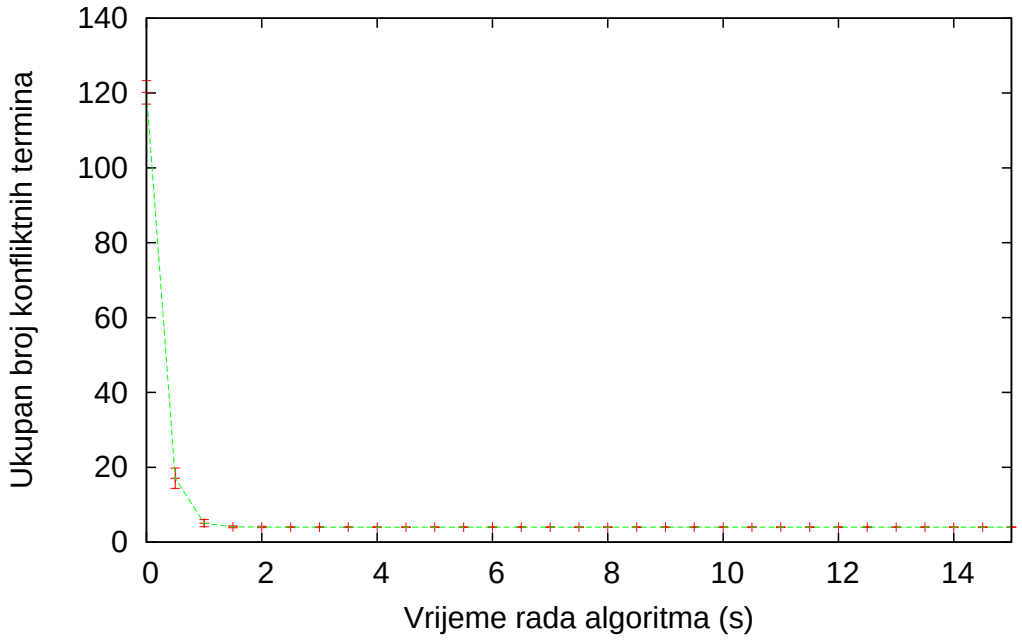


(a) Broj konflikata

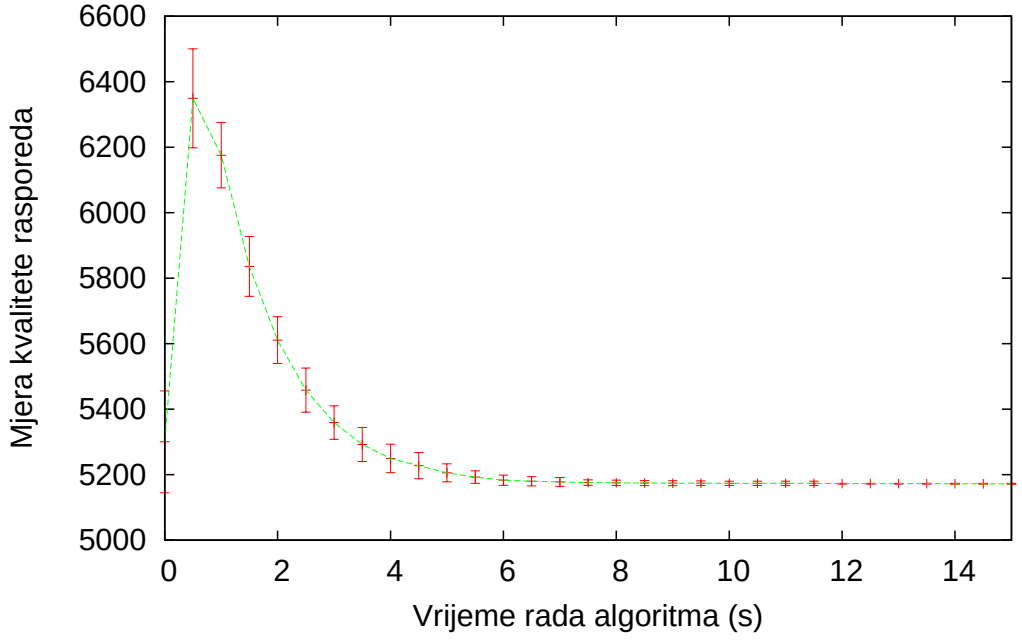


(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.3: Evolucija rješenja primjenom algoritma roja čestica za rješavanje problema izrade rasporeda.



(a) Broj konflikata



(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.4: Evolucija rješenja primjenom algoritma roja čestica za rješavanje problema izrade rasporeda – prosječni parametri.

broj kolizija te kvaliteta. Ako ima n studenata i k grupa, složenost ovog postupka je $n \times k$ s obzirom da se svaki student gleda izolirano od ostalih. Označimo s q_{max} najgoru a s q_{min} najbolju kvalitetu bilo kojeg ispitanog razmještaja. Označimo s c_{max} najveći broj konflikata bilo kojeg ispitanog razmještaja. Označimo s $c_{i,j}$ broj konfliktnih termina za slučaj da je student i smješten u grupu j te s $q_{i,j}$ mjeru kvalitete tog rasporeda za isti slučaj. Heuristička informacija smještaja studenta i u grupu j tada se računa prema izrazu:

$$c_{max} - c_{i,j} + \frac{q_{max} - q_{i,j}}{q_{max} - q_{min}}.$$

Ovako izračunate vrijednosti pamte se u polju heuristics i koriste se prilikom rada algoritma. Računanje heurističkih informacija moguće je načiniti prilikom pokretanja algoritma jer se broj studenata te definirani slijedovi termina ne mijenjaju tijekom rada algoritma.

Prilikom jedne iteracije, svaki mrav gradi rješenje na način da slučajnim redosljedom odabire studente i dodjeljuje im grupu u skladu sa slučajnim proporcionalnim pravilom koje se temelji na vrijednosti heurističkih informacija i feromonskih tragova. Tijekom izgradnje rješenja ne pazi se na prekapacitiranost slijedova. Stoga je posljednji korak u izgradnji rješenja korekcija tog rješenja kako bi se dobilo rješenje koje poštuje čvrsta ograničenja, te vrednovanje rješenja.

Nakon što svi mravi izgrade rješenje, poznato je najbolje pronađeno rješenje u toj iteraciji kao i najbolje ikada pronađeno rješenje koje se pamti nezavisno. Ako je najbolje rješenje iteracije bolje od dotadašnjeg najboljeg globalnog rješenja, ažuriraju se vrijednosti granica feromonskih tragova τ_{max} i τ_{min}. Potom se obavlja isparavanje feromonskih tragova po svih čvorova grafa, nakon čega slijedi deponiranje novih količina feromona. Nove feromone deponiraju samo dva mrava: virtualni mrav koji predstavlja najbolje ikada pronađeno rješenje te s polovičnim doprinosom najbolji mrav trenutne iteracije.

Kretanje broja konflikata te kvalitete rasporeda kroz vrijeme za 30 eksperimenata prikazano je na slici 5.5; zbirni rezultati prikazani su na slici 5.6. Rješavan je problem koji je prethodno opisan u podpoglavlju o genetskom algoritmu, a rezultati su dobiveni uz koloniju od 50 mrava te parametre $\alpha = 2$, $\beta = 2$, $\rho = 0.2$ te $a = 20$.

```
1  public class ACO<P> implements IAlgorithm<P> {
2    // Parametri i strukture podataka
3    private RandomGen rgen = new RandomGenDefault();
4    private Data<P> data;
5    private ISolutionFactory<P> solFactory;
6    private Evaluator<P> evaluator;
7    private Population<P> population;
8    private Comparator<Solution<P>> comparator;
9    private IHeuristicsProvider<P> heuristicsProvider;
10   private double alpha;
11   private double beta;
12   private double rho;
13   private double tauMax;
14   private double tauMin;
15   private double a;
16   private int numberOfAnts;
17   // Radni podatci
18   private Solution<P> bestSolution;
19   private double[][] heuristics;
20   private double[][] trails;
21
22   public ACO(Data<P> data, ISolutionFactory<P> solFactory,
23     Evaluator<P> evaluator, Comparator<Solution<P>> comparator,
24     int numberOfAnts, IHeuristicsProvider<P> heuristicsProvider,
25     double alpha, double beta, double rho, double a) {
26     this.data = data; this.solFactory = solFactory;
27     this.evaluator = evaluator; this.comparator = comparator;
28     this.heuristicsProvider = heuristicsProvider;
29     this.numberOfAnts = numberOfAnts;
30     this.alpha = alpha; this.beta = beta;
31     this.rho = rho; this.a = a;
32     population = new Population<P>(numberOfAnts, solFactory);
33     EvoService.initialize(
34       data, population, 0, population.size()-1, rgen, evaluator);
35     bestSolution = population.findBest(comparator);
36     tauMax = heuristicsProvider.getDeltaTau(bestSolution)/rho;
37     tauMin = tauMax / a;
38     heuristics = new double[data.jmbags.length][data.groups.length];
39     trails = new double[data.jmbags.length][data.groups.length];
40     for(int stIndex = 0; stIndex < data.jmbags.length; stIndex++) {
41       for(int grIndex = 0; grIndex < data.groups.length; grIndex++) {
42         heuristics[stIndex][grIndex] = heuristicsProvider.
               calculateHeuristics(stIndex, grIndex);
43         trails[stIndex][grIndex] = tauMax;
44       }
45     }
46   }
47   public void evaporate() {
48     double rho1 = 1-rho;
49     for(int stIndex = 0; stIndex < data.jmbags.length; stIndex++) {
50       for(int grIndex = 0; grIndex < data.groups.length; grIndex++) {
51         double t = trails[stIndex][grIndex]*rho1;
52         if(t<tauMin) { trails[stIndex][grIndex] = tauMin; }
53         else if(t>tauMax) { trails[stIndex][grIndex] = tauMax; }
54         else { trails[stIndex][grIndex] = t; }
55       }
56     }
57   }
```

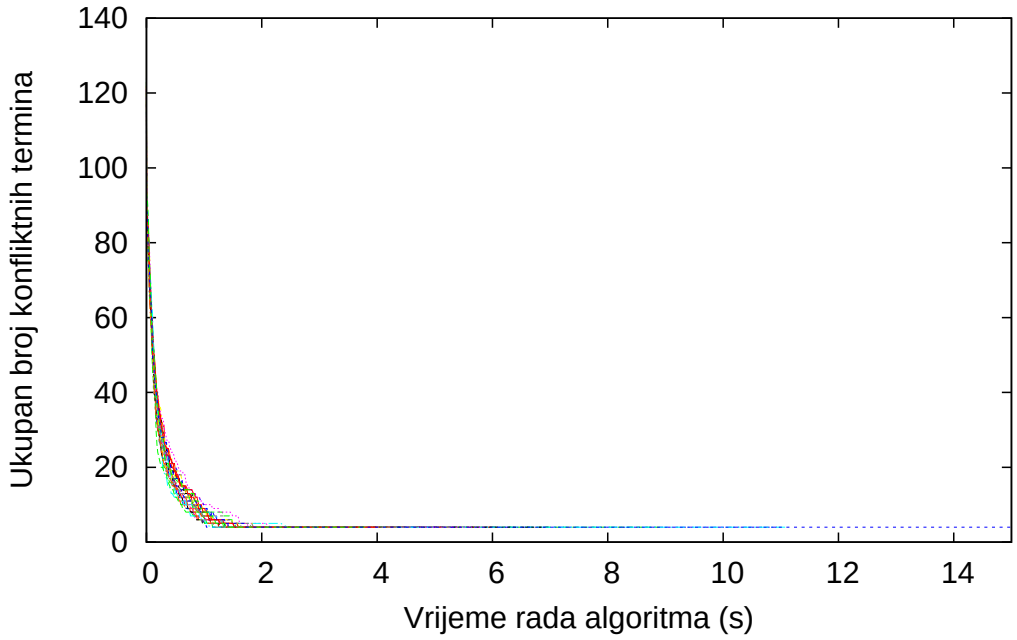
Izvorni tekst programa 5.8: Max-Min mravlji algoritam za rješavanje problema jednostavnog raspoređivanja.

```
58   public void singleStep() {
59     Solution<P> runBest = null;
60     int[] indexes = EvoService.sequence(data.jmbags.length);
61
62     for(int ant = 0; ant < numberOfAnts; ant++) {
63       EvoService.permute(indexes, rgen);
64       Solution<P> sol = solFactory.createNewSolution();
65       double[] partial = new double[sol.groupSize.length];
66       for(int i = 0; i < indexes.length; i++) {
67         int stIndex = indexes[i];
68         double sum = 0;
69         for(int grIndex = 0; grIndex < sol.groupSize.length; grIndex++) {
70           partial[grIndex] = Math.pow(trails[stIndex][grIndex], alpha)*
                 Math.pow(heuristics[stIndex][grIndex], beta);
71           sum += partial[grIndex];
72         }
73         for(int grIndex = 0; grIndex < sol.groupSize.length; grIndex++) {
74           partial[grIndex] = partial[grIndex] / sum;
75         }
76         int chosenGrIndex = 0;
77         double cumulative = 0;
78         double random = rgen.nextDouble();
79         for(int j = 0; j < sol.groupSize.length; j++) {
80           cumulative += partial[j];
81           if(random < cumulative) break;
82           chosenGrIndex++;
83         }
84         if(chosenGrIndex >= sol.groupSize.length) {
85           chosenGrIndex = sol.groupSize.length-1;
86         }
87         sol.groupSize[chosenGrIndex]++;
88         sol.studentGroup[stIndex] = chosenGrIndex;
89       }
90       EvoService.fixGroups(data, sol, rgen);
91       evaluator.evaluate(sol);
92       population.set(ant, sol);
93       if(runBest==null || comparator.compare(runBest, sol)>0) {
94         runBest = sol;
95       }
96     }
97
98     boolean updateBest = comparator.compare(runBest, bestSolution)<0;
99     if(updateBest) {
100      tauMax = heuristicsProvider.getDeltaTau(runBest);
101      tauMin = tauMax / a;
102    }
103
104    evaporate();
105    Solution<P> sol1 = updateBest ? runBest : bestSolution;
106    Solution<P> sol2 = !updateBest ? runBest : bestSolution;
107    deposit(sol1, sol2);
108
109    if(updateBest) {
110      bestSolution = runBest;
111    }
112  }
```

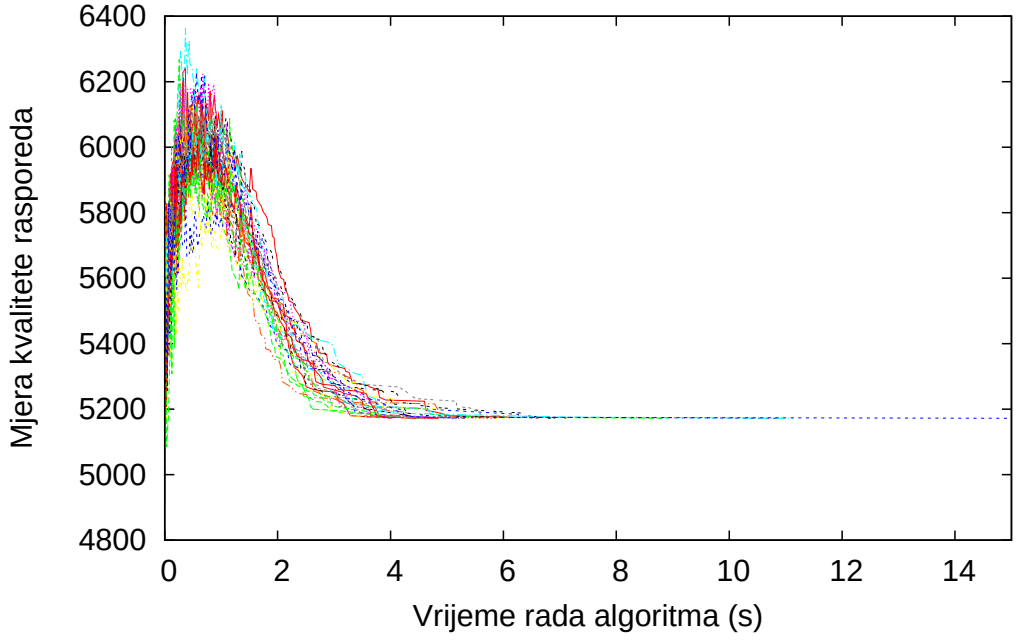
Nastavak izvornog koda s prethodne stranice.

```
113   protected void deposit(Solution<P> sol1, Solution<P> sol2) {
114
115     double dtau1 = heuristicsProvider.getDeltaTau(sol1);
116     double dtau2 = 0.5*heuristicsProvider.getDeltaTau(sol2);
117
118     for(int i = 0; i < sol1.studentGroup.length; i++) {
119
120       trails[i][sol1.studentGroup[i]] += dtau1;
121       trails[i][sol2.studentGroup[i]] += dtau2;
122
123       if(trails[i][sol1.studentGroup[i]]<tauMin)
124           trails[i][sol1.studentGroup[i]] = tauMin;
125       else if(trails[i][sol1.studentGroup[i]]>tauMax)
126           trails[i][sol1.studentGroup[i]] = tauMax;
127
128       if(trails[i][sol2.studentGroup[i]]<tauMin)
129           trails[i][sol2.studentGroup[i]] = tauMin;
130       else if(trails[i][sol2.studentGroup[i]]>tauMax)
131           trails[i][sol2.studentGroup[i]] = tauMax;
132
133     }
134   }
135 }
```

Nastavak izvornog koda s prethodne stranice.

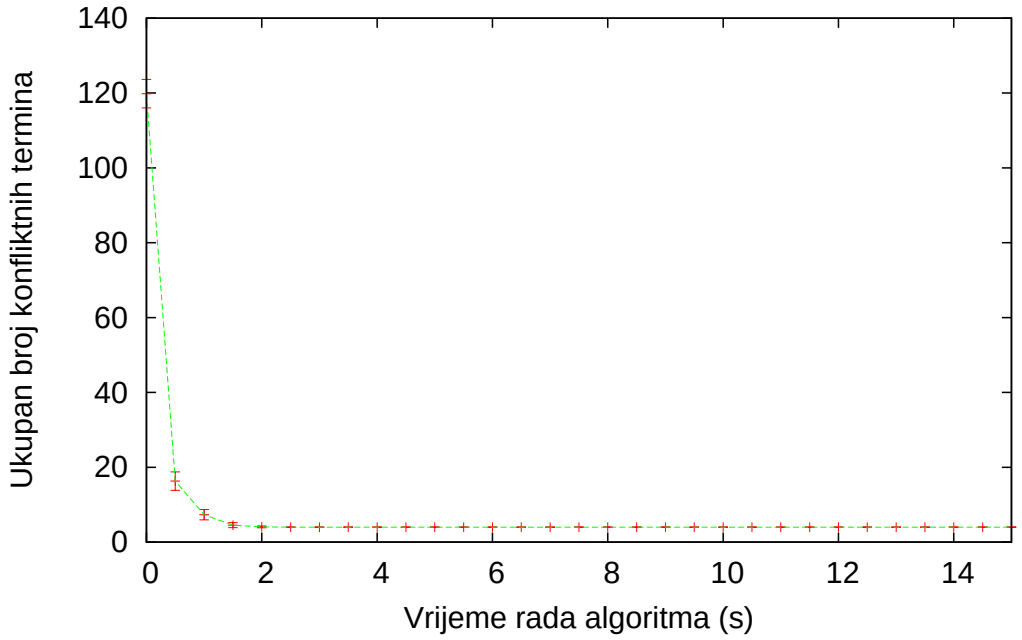


(a) Broj konflikata

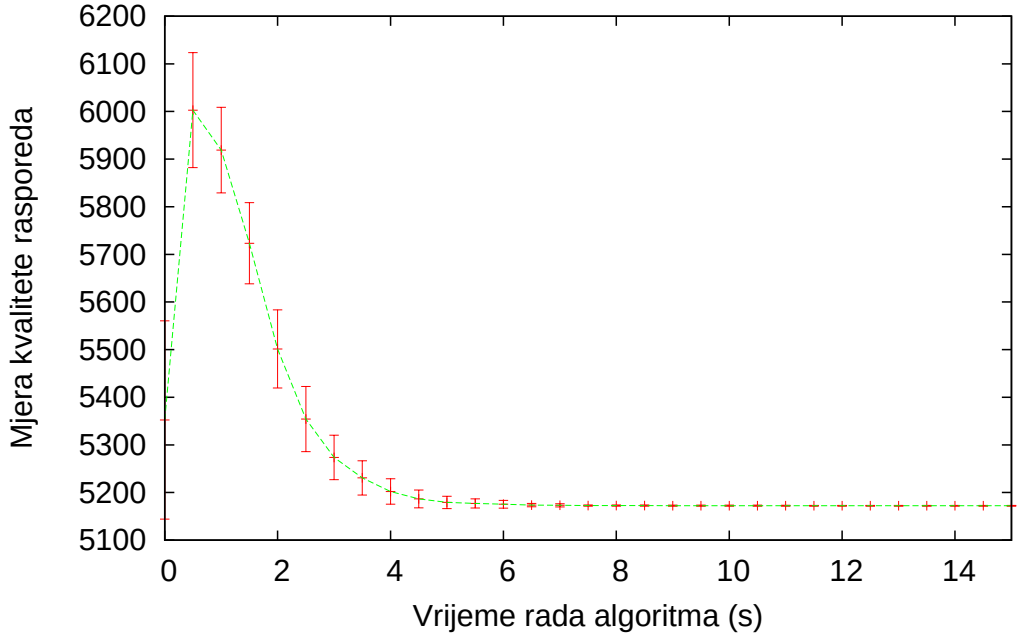


(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.5: Evolucija rješenja primjenom algoritma *Max-Min* mravlji sustav za rješavanje problema izrade rasporeda.



(a) Broj konflikata



(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.6: Evolucija rješenja primjenom algoritma *Max-Min* mravlji sustav za rješavanje problema izrade rasporeda – prosječni parametri.

5.1.5 Jednostavan imunološki algoritam

Implementaciju jednostavnog imunološkog algoritma za problem jednostavnog raspoređivanja prikazuje izvorni tekst programa 5.9. Parametri koji kontroliraju rad algoritma su popSize (veličina populacije), beta (broj klonova koji će biti načinjen za svaku jedinku populacije) te newCount (broj novih slučajno stvorenih jedinki koje će zamijeniti najgore jedinke u populaciji). Svaki se klon mutira zadanom vjerojatnošću koja je ista neovisno o kvaliteti jedinke. U populaciju klonova kopira se i čitava trenutna populacija te se potom popSize−newCount najboljih jedinki iz tako stvorene unije stare populacije i populacije klonova te newCount slučajno stvorenih jedinki prenosi u sljedeću generaciju.

Kretanje broja konflikata te kvalitete rasporeda kroz vrijeme za 30 eksperimenata prikazano je na slici 5.7; zbirni rezultati prikazani su na slici 5.8. Rješavan je problem koji je prethodno opisan u podpoglavlju o genetskom algoritmu, a rezultati su dobiveni uz populaciju od 40 jedinki, uz vjerojatnost mutacije grupe jednog studenta iznosa tri jedinične vjerojatnosti mutacije, uz $\beta = 3$ te uz newCount=2.

5.1.6 Algoritam klonske selekcije

Implementaciju algoritma klonske selekcije za problem jednostavnog raspoređivanja prikazuje izvorni kod 5.10. Radom algoritma upravljaju parametri popSize (veličina populacije), beta (parametar kloniranja), minMutProb (vjerojatnost mutiranja grupe jednog studenta u najboljem rješenju), maxMutProb (vjerojatnost mutiranja grupe jednog studenta u najgorem rješenju) te newCount (broj jedinki koje se stvaraju slučajno i ulaze u sljedeću generaciju jedinki).

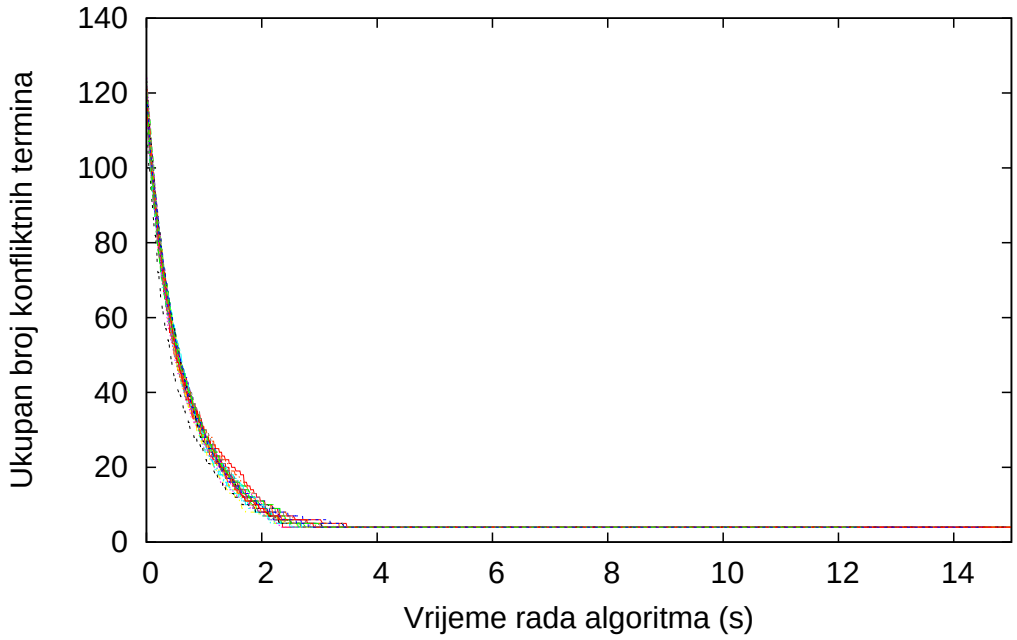
Kretanje broja konflikata te kvalitete rasporeda kroz vrijeme za 30 eksperimenata prikazano je na slici 5.9; zbirni rezultati prikazani su na slici 5.10. Rješavan je problem koji je prethodno opisan u podpoglavlju o genetskom algoritmu, a rezultati su dobiveni uz populaciju od 50 jedinki, uz minimalnu vjerojatnost mutacije grupe jednog studenta iznosa tri jedinične vjerojatnosti mutacije, uz maksimalnu vjerojatnost mutacije grupe jednog studenta iznosa trideset jediničnih vjerojatnosti mutacije, uz $\beta = 0.7$ te uz newCount=2.

```
1  public class SIA<P> implements IAlgorithm<P> {
2
3    // Parametri i strukture podataka
4    private RandomGen rgen = new RandomGenDefault();
5    private Data<P> data;
6    private ISolutionFactory<P> solFactory;
7    private Evaluator<P> evaluator;
8    private Population<P> population;
9    private double mutProb;
10   private Comparator<Solution<P>> comparator;
11   private int beta;
12   private int popSize;
13   private int newCount;
14
15   // Radni podatci
16   private Solution<P> bestSolution;
17
18   public SIA(Data<P> data, ISolutionFactory<P> solFactory,
19     Evaluator<P> evaluator,
20     Comparator<Solution<P>> comparator, int popSize,
21     int beta, double mutProb, int newCount) {
22
23     this.data = data;
24     this.solFactory = solFactory;
25     this.evaluator = evaluator;
26     this.comparator = comparator;
27     this.popSize = popSize;
28     this.beta = beta;
29     this.newCount = newCount;
30     this.mutProb = mutProb;
31
32     population = new Population<P>(popSize, solFactory);
33     EvoService.initialize(
34       data, population, 0, population.size()-1, rgen, evaluator
35     );
36
37     bestSolution = population.findBest(comparator);
38
39   }
```

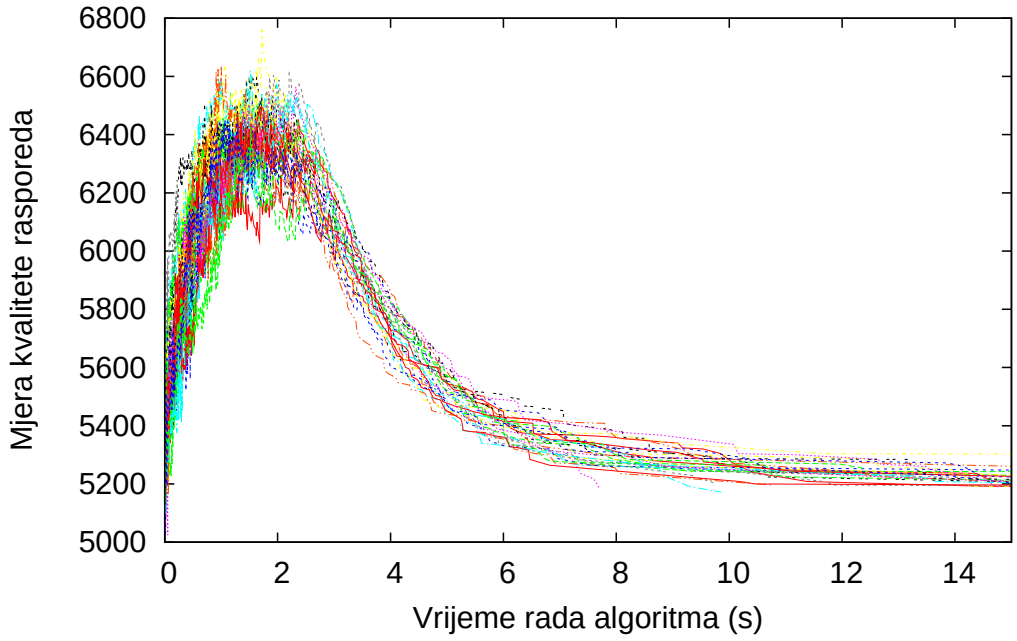
Izvorni tekst programa 5.9: Jednostavan imunološki algoritam za rješavanje problema jednostavnog raspoređivanja.

```
40   public void singleStep() {
41
42     Population<P> clones = new Population<P>(
43       (beta+1)*popSize, solFactory
44     );
45
46     int cloneIndex = 0;
47     for(int solIndex = 0; solIndex < popSize; solIndex++) {
48       Solution<P> original = population.get(solIndex);
49       for(int i = 0; i <= beta; i++) {
50         Solution<P> clone = clones.createNewSolution();
51         clone.copyFrom(original);
52         clones.set(cloneIndex, clone);
53         if(i!=0) {
54           EvoService.mutate(data, clone, rgen, mutProb);
55         }
56         evaluator.evaluate(clone);
57         cloneIndex++;
58       }
59     }
60
61     clones.sort(comparator);
62
63     int toCopy = popSize - newCount;
64     for(int i = 0; i < toCopy; i++) {
65       population.set(i, clones.get(i));
66     }
67
68     for(int i = 0; i < newCount; i++) {
69       Solution<P> sol = population.createNewSolution();
70       population.set(toCopy+i, sol);
71     }
72     EvoService.initialize(
73       data, population, toCopy, popSize-1, rgen, evaluator
74     );
75
76     boolean bestUpdated = false;
77     for(Solution<P> sol : population) {
78       if(comparator.compare(sol, bestSolution)<0) {
79         bestSolution = sol;
80         bestUpdated = true;
81       }
82     }
83
84   }
85
86 }
```

Nastavak izvornog koda s prethodne stranice.

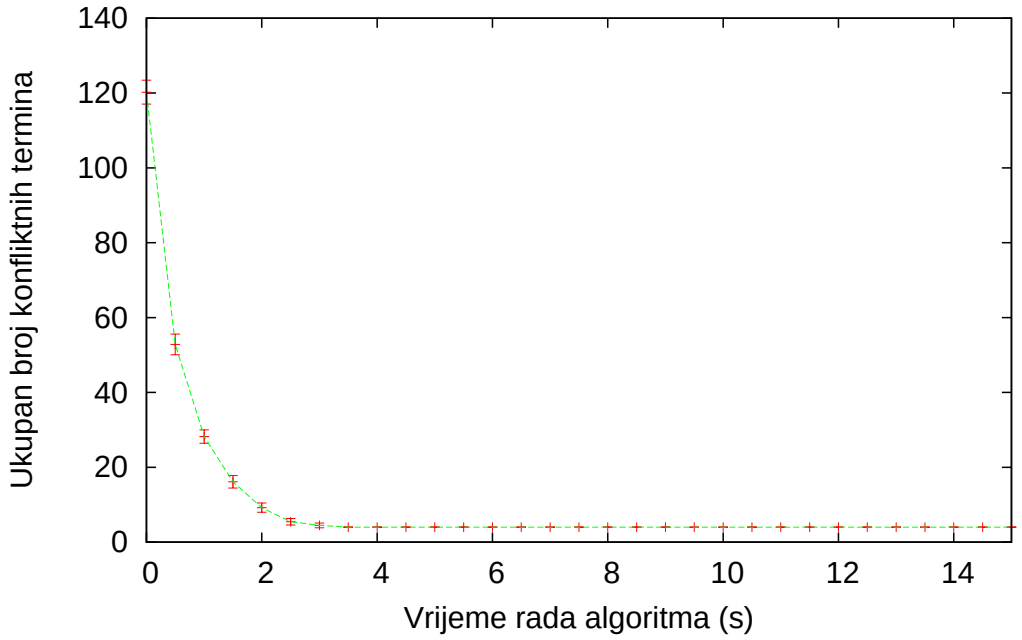


(a) Broj konflikata

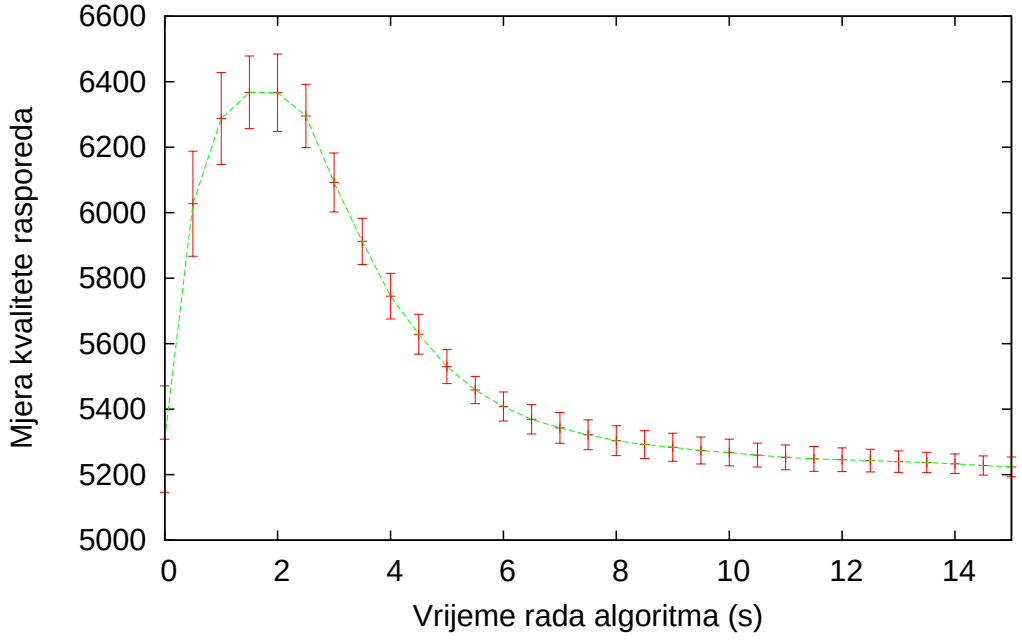


(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.7: Evolucija rješenja primjenom jednostavnog imunološkog algoritma za rješavanje problema izrade rasporeda.



(a) Broj konflikata



(b) Kazna zbog nekompaktnosti rasporeda

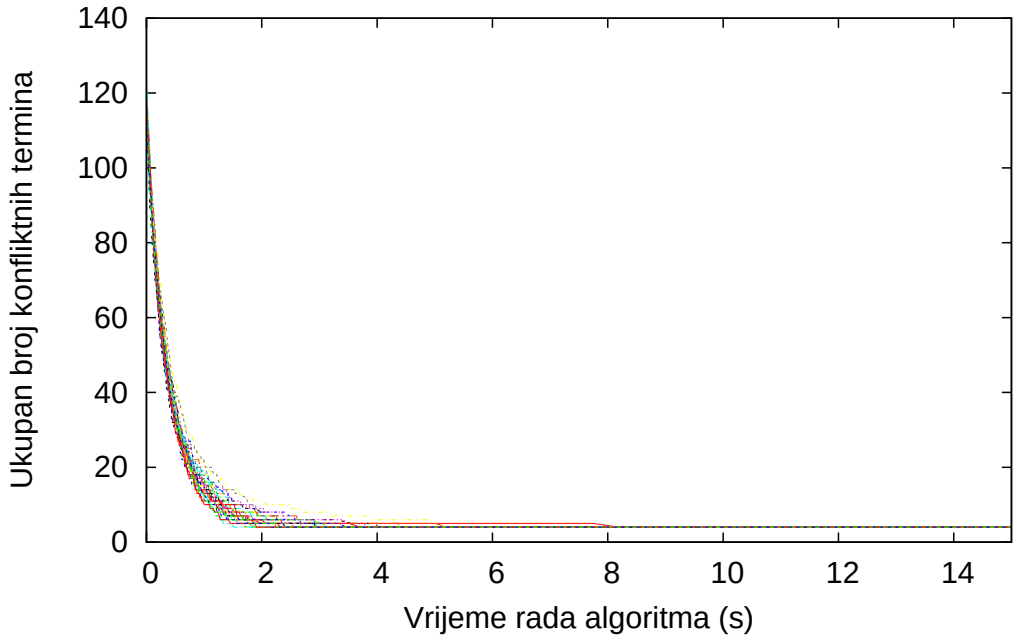
Slika 5.8: Evolucija rješenja primjenom jednostavnog imunološkog algoritma za rješavanje problema izrade rasporeda – prosječni parametri.

```
1  public class ClonALG<P> implements IAlgorithm<P> {
2
3    // Parametri i strukture podataka
4    private RandomGen rgen = new RandomGenDefault();
5    private Data<P> data;
6    private ISolutionFactory<P> solFactory;
7    private Evaluator<P> evaluator;
8    private Population<P> population;
9    private double minMutProb;
10   private double maxMutProb;
11   private Comparator<Solution<P>> comparator;
12   private double beta;
13   private int popSize;
14   private int newCount;
15
16   // Radni podatci
17   private Solution<P> bestSolution;
18   private int[] counts;
19   private int totalClones;
20
21   public ClonALG(Data<P> data, ISolutionFactory<P> solFactory,
22     Evaluator<P> evaluator,
23     Comparator<Solution<P>> comparator, int popSize,
24     double beta, double minMutProb, double maxMutProb,
25     int newCount) {
26
27     this.data = data;
28     this.solFactory = solFactory;
29     this.evaluator = evaluator;
30     this.comparator = comparator;
31     this.popSize = popSize;
32     this.beta = beta;
33     this.newCount = newCount;
34     this.minMutProb = minMutProb;
35     this.maxMutProb = maxMutProb;
36
37     population = new Population<P>(popSize, solFactory);
38     EvoService.initialize(
39       data, population, 0, population.size()-1, rgen, evaluator
40     );
41
42     bestSolution = population.findBest(comparator);
43
44     counts = new int[popSize];
45     for(int i = 0; i < popSize; i++) {
46       int n = (int)(popSize * this.beta / (i+1));
47       if(n==0) n=1;
48       counts[i] = n;
49       totalClones += n;
50     }
51
52     population.sort(comparator);
53   }
```

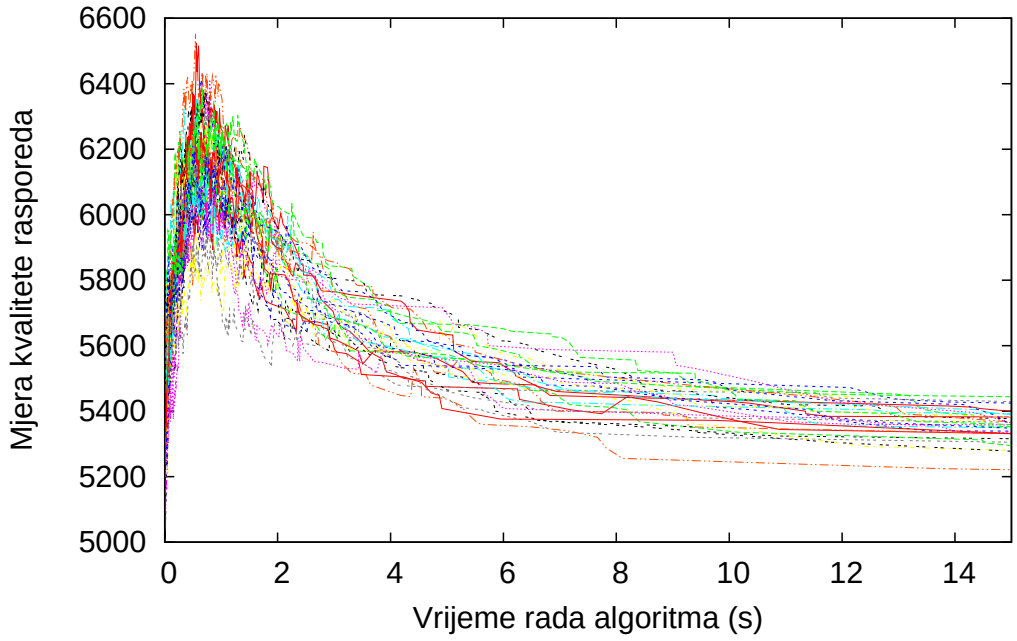
Izvorni tekst programa 5.10: Algoritam klonske selekcije za rješavanje problema jednostavnog raspoređivanja.

```
54   public void singleStep() {
55
56     Population<P> clones = new Population<P>(
57       popSize+totalClones, solFactory
58     );
59
60     int cloneIndex = 0;
61     for(int solIndex = 0; solIndex < popSize; solIndex++) {
62       Solution<P> original = population.get(solIndex);
63       double mutProb = (maxMutProb-minMutProb)*solIndex/popSize +
             minMutProb;
64       for(int i = 0; i <= counts[solIndex]; i++) {
65         Solution<P> clone = clones.createNewSolution();
66         clone.copyFrom(original);
67         clones.set(cloneIndex, clone);
68         if(i!=0) {
69           EvoService.mutate(data, clone, rgen, mutProb);
70         }
71         evaluator.evaluate(clone);
72         cloneIndex++;
73       }
74     }
75
76     clones.sort(comparator);
77
78     int toCopy = popSize - newCount;
79     for(int i = 0; i < toCopy; i++) {
80       population.set(i, clones.get(i));
81     }
82
83     for(int i = 0; i < newCount; i++) {
84       Solution<P> sol = population.createNewSolution();
85       population.set(toCopy+i, sol);
86     }
87     EvoService.initialize(data, population, toCopy, popSize-1, rgen,
         evaluator);
88
89     population.sort(comparator);
90     if(comparator.compare(population.get(0), bestSolution)<0) {
91       bestSolution = population.get(0);
92     }
93
94   }
95 }
```

Nastavak izvornog koda s prethodne stranice.

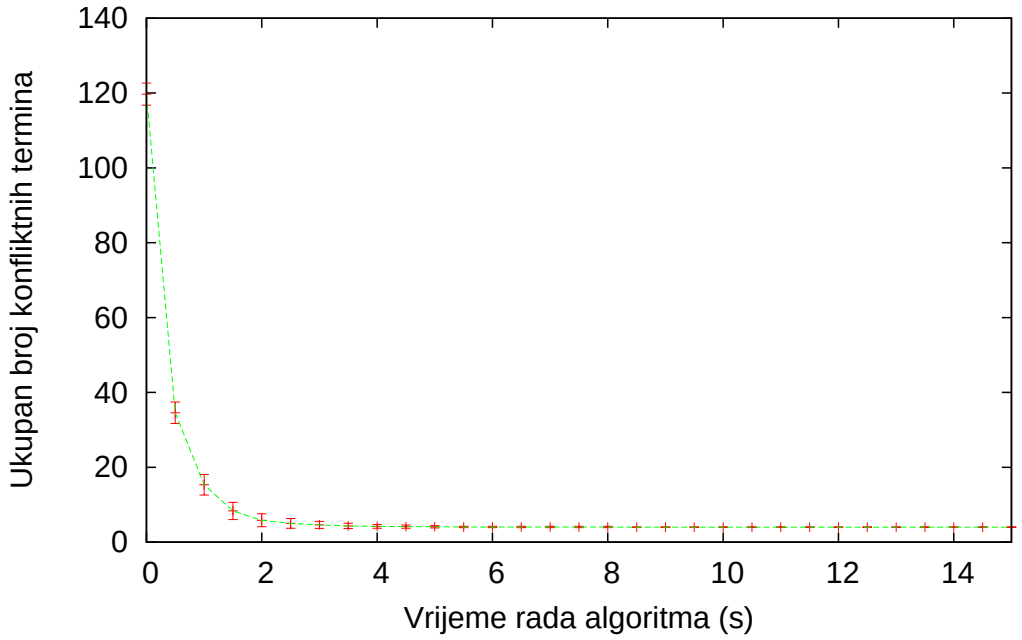


(a) Broj konflikata

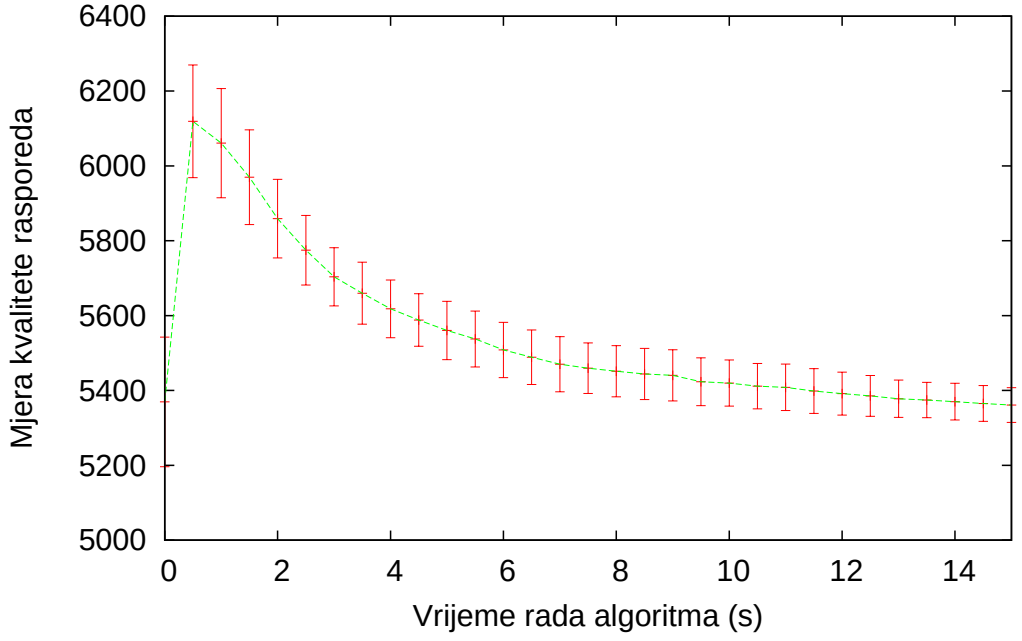


(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.9: Evolucija rješenja primjenom algoritma klonske selekcije za rješavanje problema izrade rasporeda.



(a) Broj konflikata



(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.10: Evolucija rješenja primjenom algoritma klonske selekcije za rješavanje problema izrade rasporeda – prosječni parametri.

5.1.7 Usporedba algoritama

U svrhu usporedbe rada opisanih algoritama zbirni su rezultati svakog od algoritama ponovno nacrtani na istoj slici (slika 5.11), i to za svih 30 sekundi rada algoritama (prethodne slike prikazuju samo prvih 15 sekundi postupka optimizacije kako bi se bolje vidjelo ponašanje na početku optimizacijskog procesa). Pri tome slika 5.11a za svaki od algoritama prikazuje kretanje broja konflikata u najboljem rješenju u promatranim trenutcima. Slika 5.11b prikazuje za svaki od algoritama kretanje mjere kvalitete u najboljem rješenju u promatranim trenutcima (manje je bolje).

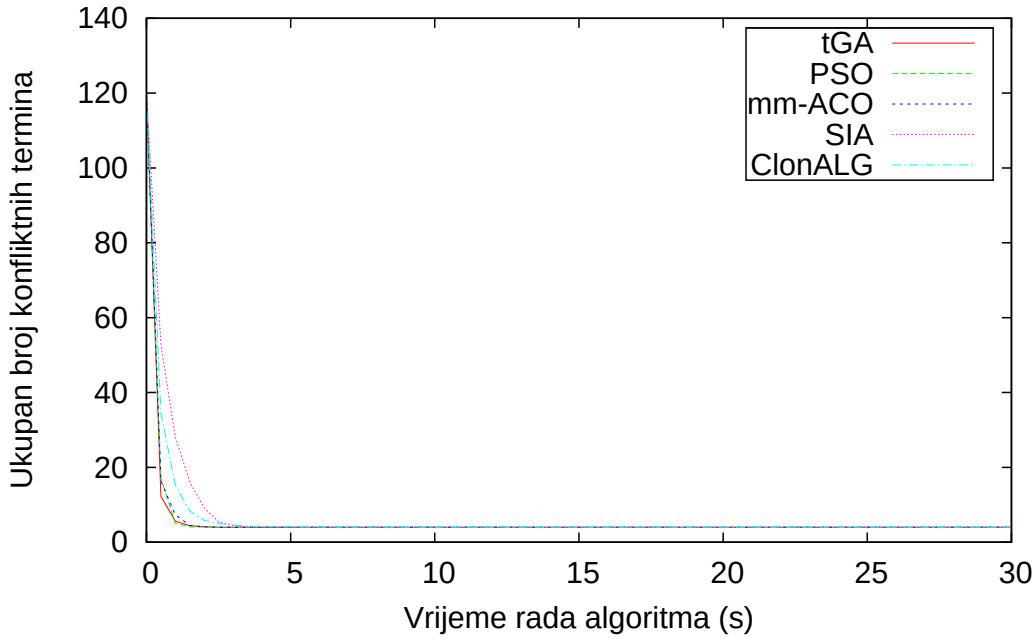
Kako se može vidjeti sa slike 5.11a, svih pet algoritama vrlo brzo uspjevaju količinu konflikata spustiti na minimalnu. Eliminacijski genetski algoritam, algoritam roja čestica te Max-Min mravlji algoritam pri tome pokazuju malu prednost u brzini kojom dolaze do tog rezultata.

Usporedi li se pak kvaliteta dobivenih rasporeda, upravo očekivano, algoritam *Max-Min* mravlji sustav tijekom čitavog optimizacijskog postupka pokazuje sposobnost pronalaska najkvalitetnijih rješenja u usporedbi s ostalim algoritmima. Ovu činjenicu možemo objasniti time da jedino algoritam *Max-Min* mravlji sustav za razliku od ostalih ima na raspolaganju dodatnu heurističku informaciju čijom uporabom može kvalitetnije voditi postupak pretraživanja. U početnom dijelu optimizacije, sljedeća dva algoritma koja postižu najbolje rezultate su algoritam klonske selekcije te algoritam roja čestica. Međutim, u kasnijem tijeku optimizacije situacija se donekle mijenja pa tako pred kraj algoritam klonske selekcija čak postiže najslabije rezultate[2]. Ovo zapažanje upućuje na mogućnost da bi zajednički rad ovih algoritama mogao postizati još bolje rezultate.

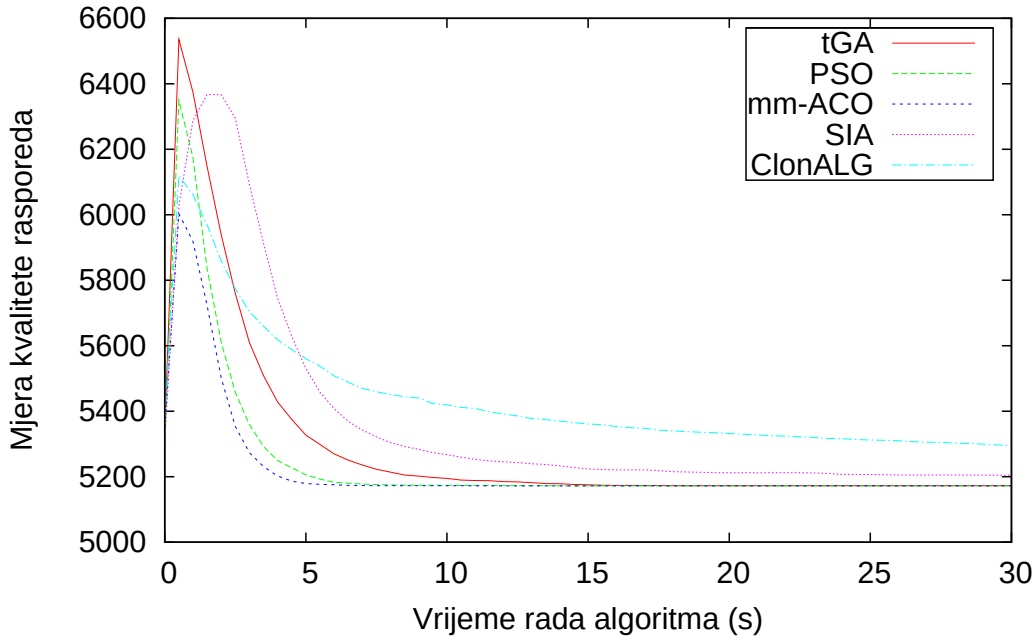
5.2 Problem izrade rasporeda obaveznih provjera znanja

Problem izrade rasporeda obaveznih provjera znanja definiran je u podpoglavlju 4.5. Najjednostavniji način prikaza rješenja je uporabom vektora duljine o gdje je o upravo broj kolegija koje treba rasporediti. Elementi vektora su pri tome skalari koji mogu poprimiti vrijednosti iz skupa $\{1, \ldots, p\}$ gdje je p broj termina koji sudjeluju u izradi rasporeda. Ovakav način predstavljanja rješenja pogodan je i za često korištene evolucijske operatore: primjerice, uniformno križanje koje možemo provoditi po parovima

[2]Ovo vrijedi uz napomenu da niti jedan od algoritama nije posebno podešavan tako da postiže najbolje moguće rezultate. Umjesto toga, korištene su "razumne" pretpostavljene vrijednosti.



(a) Broj konflikata



(b) Kazna zbog nekompaktnosti rasporeda

Slika 5.11: Usporedba evolucije rješenja za sve algoritme.

elemenata dvaju vektora, mutaciju koju možemo provoditi nad elementima jednog vektora i sl.

Međutim, iako je predloženi način predstavljanja rješenja najjednostavniji, on nije najpogodniji za vrednovanje rješenja. Naime, tijekom vrednovanja često su potrebne sljedeće informacije:

- koji su još kolegiji smješteni u isti termin kao i kolegij c_i,
- koliko je ukupno studenata smješteno u termin t_j,
- koji su kolegiji smješteni u termin koji je odmah prije termina t_j (ili odmah nakon termina t_j) i mnoge druge.

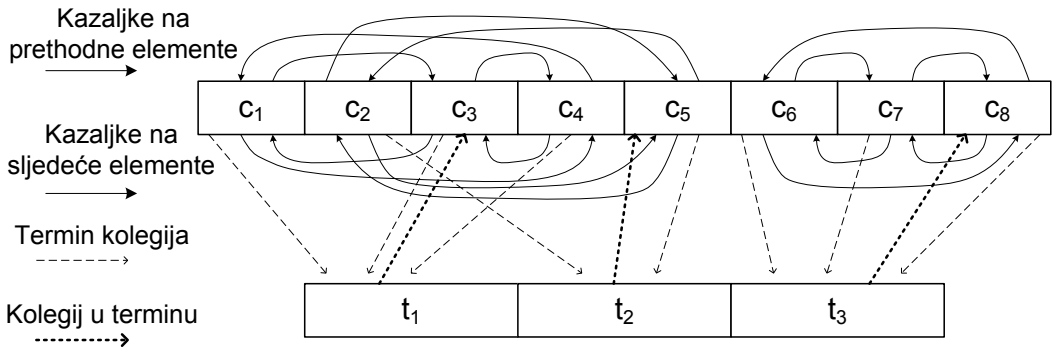
Problemu prikaza rješenja može se pristupiti i s druge strane: umjesto da ga se orijentira prema vektoru kolegija, može ga se orijenitrati prema vektoru termina. Pri tome svaki termin pamti kolekciju kolegija koji su mu pridruženi. Ovakva struktura podataka može brzo dati odgovor na pitanja koja su orijentirana na termine (poput koji su sve kolegiji u menom terminu i sl.). Međutim, ovim pristupom postaju problematični upiti i operatori koji rade s kolegijima; primjerice, ako je potrebno kolegij c_i ubaciti u termin t_j, najprije treba pretražiti sve termine kako bi se utvrdilo u koji je termin taj kolegij bio prethodno smješten, potom ga treba ukloniti i tek tada ga se može ubaciti u novi termin. Dodatno, ako se struktura podataka načini loše, postoji mogućnost da će se pri tome često koristiti memorijski podsustav što bi moglo usporiti izvođenje programa.

Stoga se kao jedno moguće rješenje nameće uporaba redundantnih struktura podataka koje će osigurati brzo izvođenje često korištenih operacija. Jedno takvo rješenje inspirirano radom [Talbi and Weinberg, 2007] te iskorišteno u radu [Čupić et al., 2009a] prikazuje izvorni tekst programa 5.11.

Ideja je sljedeća. Kromosom će paralelno voditi dvije povezane strukture podataka: vektor kolegija te vektor termina. Elementi vektora kolegija su objekti tipa KCourse, koji se mogu povezivati u dvostruko povezanu listu zahvaljujući varijablama previous i next. Te varijable nisu po tipu reference već cijeli brojevi koji imaju mogućnost pamćenja indeksa prethodnog/sljedećeg kolegija u polju svih kolegija kcourse jednog kromosoma. Vrijednost -1 tada služi kao zamjena za **null**-referencu. Osim ove dvije varijable, kolegij pamti i indeks termina u koji je smješten (varijabla term). Za pamćenje svih termina kromosom koristi polje objekata tipa KTerm. Svaki objekt tipa KTerm pamti indeks

```
1  public class Kromosom {
2    KCourse[] kcourse;
3    KTerm[] kterms;
4    int[] clusterTerms;
5    int[] eval;
6  }
7
8  public class KTerm {
9    int someCourse = -1;
10   int coursesCount = 0;
11 }
12
13 public class KCourse {
14   int previous = -1;
15   int next = -1;
16   int term = -1;
17   int index;
18 }
```

Izvorni tekst programa 5.11: Struktura rješenja za problem rasporeda provjera znanja.



Slika 5.12: Struktura podataka za problem izrade rasporeda obaveznih provjera znanja.

(varijabla someCourse) jednog od kolegija koji je smješten u taj termin (bilo kojeg) te koliko je ukupno kolegija smješteno u taj termin.

Uporabom ovakve strukture moguće je dobiti izvođenje najčešćih upita i operacija u složenosti O(1). Ovo je pojašnjeno u nastavku na primjeru koji pokazuje slika 5.12. Slika prikazuje raspored u kojem je osam kolegija smješteno u tri termina. Gornje polje je polje objekata KCourse. Na slici 5.12 pune strelice iznad kolegija predstavljaju sadržaj varijabli previous dok pune strelice ispod kolegija predstavljaju sadržaj varijabli next. Vrijednosti varijabli za polje kolegija prikazane su u tablici 5.1.

Polje na dnu slike je polje kterms. Elementi tog polja prikazani su u tablici 5.2.

Tablica 5.1: Sadržaj polja kolegija za primjer prikazan slikom 5.12.

Pozicija	previous	next	term
1	3	4	1
2	5	5	2
3	4	1	1
4	1	3	1
5	2	2	2
6	7	8	3
7	8	6	3
8	6	7	3

Tablica 5.2: Sadržaj polja termina za primjer prikazan slikom 5.12.

Pozicija	someCourse	coursesCount
1	3	3
2	5	2
3	8	3

Traži li se u koji je termin smješten kolegij s indeksom 2, to se direktno pogleda u polju kcourse na poziciji 2 (varijabla term). Traži li se koji su sve kolegiji smješteni u termin 3, dovoljno je u polju kterms na poziciji 3 pogledati varijablu someCourse – to je referenca na prvi kolegij; do ostalih se direktno dolazi praćenjem reference next odgovarajućih objekata. Operaciju mutacije jednog kolegija sada se može provesti u složenosti O(1): kolegij c_i se dohvati u polju kcourse na poziciji i, izbaci ga se iz dvostruko povezane liste, promijeni mu se termin na željeni i na kraju ga se ubaci u dvostruko povezanu listu tog novog termina (koju se direktno dohvati u polju kterms) na poziciji promatranog termina. Kako nema potrebe za bilo kakvim pretraživanjem ili iteriranjem po kolekcijama, sve ove operacije obavljaju se u konstantnom vremenu. Operaciju križanja kao i druge operacije također se mogu izvesti na sličan način, što znači da će ukupna složenost križanja i mutacije biti linearno proporcionalna broju kolegija.

```
class GlobalData {
  Course[] courses;
  Term[] terms;
  CourseCluster[] courseClusters;
  int[][][] acceptableCourseTerms;
}

public class Course {
  private String code;
  private String name;
  private int index;
  private int studentsCount;
  private double[] sharedStudents;
  private int clusterIndex = -1;
}

public class Term {
  private String time;
  private int dayIndex;
  private int withinDayIndex;
  private int capacity;
  private int hardCapacity;
  private int index;
}

public class CourseCluster {
  private int index;
  private int[] courseIndexes;
}
```

Izvorni tekst programa 5.12: Struktura podataka za pamćenje globalnih podataka.

5.2.1 Globalna struktura podataka

Informacije koje ne ovise o pojedinačnom rješenju pamte se u globalnoj strukturi podataka (izvorni tekst programa 5.12).

U globalnim podatcima pamti se polje kolegija, gdje svaki kolegij čuva informacije o broju studenata, o broju dijeljenih studenata s drugim kolegijima (varijabla sharedStudents) te indeks klastera kojem pripada. Jedan od zahtjeva koji treba podržavati je upareno razmještanje kolegija, gdje dva ili više kolegija treba razmještati zajedno u odabrani termin. Grupu kolegija koju treba razmještati na ovaj način zvat ćemo jednim klasterom kolegija; globalna struktura podataka sadrži polje svih takvih klastera (polje courseClusters) a svaki kolegij pamti indeks klastera u koji je pridružen ili -1 ako se može razmještati slobodno.

Polje dijeljenih studenata sharedStudents u primjercima razreda Course po tipu je polje decimalnih brojeva; razlog je taj što se u njega ne pohranjuje direktno broj dijeljenjih studenata sa svakim od kolegija (polje ima elemenata koliko postoji kolegija) već broj

dijeljenih studenata koji je dobiven uzimajući u obzir sve specifičnosti opisane u podpoglavlju 4.5 (primjerice, kolika je subjektivna težina tih kolegija, koliko ima studenata koji ih dijele a imaju ih kao kolegije modula, koliko je drugih situacija itd).

Razred Term predstavlja skup informacija o pojedinom terminu. Omogućava pamćenje indeksa dana kojem taj termin pripada (varijabla dayIndex), indeksa termina u tom danu (varijabla withinDayIndex), mekog i čvrstog kapaciteta termina (varijable capacity i hardCapacity) i druge.

Konačno, razred CourseCluster omogućava pamćenje klastera kolegija koje treba raspoređivati zajedno.

5.2.2 Vrednovanje rješenja

Za potrebe vrednovanja rješenja koristi se trokomponentni vektor $\vec{e} = (e_1, e_2, e_3)$. Komponente vektora opisane su u nastavku.

- Komponenta e_1 predstavlja ukupni broj konflikata u rasporedu; pri tome se pod pojmom konflikt podrazumijeva da jedan student treba u istom terminu prisustvovati na dvije provjere znanja.

- Komponenta e_2 predstavlja mjeru nekvalitete načinjenog rasporeda s obzirom na studente koji imaju druge ispite u bliskim terminima. U primjerima koji će biti prikazani u ovom podpoglavlju, ova mjera je računata kao broj studenata koji imaju ispit u nekom od susjednih termina (gleda se samo taj isti dan te sljedeći dan) pomnožen s koeficijentom razlike termina. Opći oblik korištene kazne je oblika

$$A \cdot \Delta s \cdot B, \tag{5.2}$$

  gdje je A koeficijent vezan uz razliku dana promatranih termina a B koeficijent vezan uz razliku termina unutar istog dana. Oznaka Δs označava broj dijeljenih studenata između promatranih kolegija. Primjerice, ako se promatraju kolegiji c_1 i c_2 koji su smješteni u termine t_1 i t_2 te koji dijele Δs studenata, doprinos komponenti e_2, ako su termini smješteni u isti dan bit će:

$$4 \cdot \Delta s \cdot (1 + \frac{1}{|\Delta w|}) \tag{5.3}$$

  gdje je $|\Delta w|$ razlika withinDayIndex-a tih termina. U ovom slučaju i s obzirom na

izraz (5.2) koristi se $A = 4$ i $B = 1 + \frac{1}{|\Delta w|}$. Za termine koji su smješteni u susjedne dane doprinos će biti računat kao

$$1 \cdot \Delta s \cdot 1. \tag{5.4}$$

S obzirom na izraz (5.2) koristi se $A = 1$ i $B = 1$. U oba slučaja, iznosu će se na kraju dodati još i dvostruka vrijednost broja kolizija studenata.

- Komponenta e_3 predstavlja prekapacitiranost termina, odnosno ukupni broj studenata razmješten po svim terminima koji ne stanu u zadani kapacitet termina.

5.2.3 Genetski algoritam za rješavanje problema rasporeda obaveznih provjera znanja

Za potrebe rješavanja ovog problema iskorišten je eliminacijski genetski algoritam. Ispitivanje utjecaja nekoliko različitih parametara troturnirskog eliminacijskog genetskog algoritma na konačne rezultate opisano je u radu [Čupić et al., 2009a], gdje je analiziran utjecaj triju parametara navedenih u nastavku.

1. *Način usporedbe jedinki* – razmatrano je (i) definiranje vrijednosti kazne kao težinsku sumu $k = w_1 \cdot e_1 + w_2 \cdot e_2 + w_3 \cdot e_3$ i usporedba jedinki temeljem tih suma, odnosno (ii) definiranje važnosti komponenti vektora $\vec{e}$ i deterministička usporedba komponentu po komponentu (najprije e_1 pa ako nema odluke, tada e_3 pa ako nema odluke, tada e_2). U radu su korištene težine $w_1 = w_3 = 1$ i $w_2 = 0.05$.

2. *Operator zamjene* – razmatrana su dva načina kako utvrditi koju će jedinku iz populacije zamijeniti dijete stvoreno križanjem i mutacijom. Jedna mogućnost je zamijeniti najlošijeg od triju roditelja koji su izabrani u koraku troturnirske selekcije. Druga je mogućnost čitavu populaciju najprije podijeliti u četiri razreda s obzirom na dobiveno dijete i odnos prema čvrstim ograničenjima ($e_1 + e_3$) i mekom ograničenju e_2:

   - razred 1 čine sve jedinke populacije koje imaju u odnosnu na dijete veća kršenje čvrstih ograničenja i veća kršenje mekih ograničenja,
   - razred 2 čine sve jedinke populacije koje imaju u odnosnu na dijete veća kršenje čvrstih ograničenja i manja kršenje mekih ograničenja,

- razred 3 čine sve jedinke populacije koje imaju u odnosnu na dijete manje kršenje čvrstih ograničenja i veća kršenje mekih ograničenja te
- razred 4 čine sve jedinke populacije koje imaju u odnosnu na dijete manje kršenje čvrstih ograničenja i manje kršenje mekih ograničenja. Iz populacije se potom kao jedinka koju će se zamijeniti bira neka slučajna jedinka iz razreda 1 ako on nije prazan. Ako je razred 1 prazan, onda se bira neka slučajna jedinka iz razreda 2. Ako je i razred 2 prazan, onda se uzima neka slučajna jedinka iz razreda 3 ako on nije prazan. Konačno, ako je i razred 3 prazan, tada se kao kandidat uzima neka slučajna jedinka iz razreda 4.

3. *Podešavanje razdiobe prema kojoj se generiraju slučajni brojevi* – pri čemu je ideja bila provjeriti ima li razlike između uporabe generatora s uniformnom razdiobom te uporabe generatora s podešavanom razdiobom. Pri tome bi se podešavanje generatora koji se koristi za dodjelu termina kolegiju provodilo tako da se analizira zastupljenost termina u rješenjima populacije pa se terminima koji su rjeđe zastupljeni (ili uopće nisu) poveća vjerojatnost generiranja.

Osim navedenih parametara, mijenjane su i vjerojatnosti mutacija te veličine populacija. Provedeni eksperimenti pokazali su da se najbolja rješenja doista dobivaju kada se koristi prilagodba razdiobe vjerojatnosti, mala vjerojatnost mutacije te umjerena veličina populacije. Problem koji je u tom primjeru rješavan sadrži 77 kolegija koje treba rasporediti u 30 termina u periodu od 10 dana. Genetski algoritam je pri tome imao postavljenu granicu na 4000000 iteracija (jedna iteracija je jedan odabir roditelja, križanje, mutacija te zamjena).

Kasnije provedenim modifikacijama genetskog algoritma operator križanja zamijenjen je s tri načina provođenja križanja (k_1, k_2 i $k3$). Prilikom rada algoritma operator križanja se svaki puta bira posredstvom slučajnog mehanizma, i to u 60% slučajeva operator k_1, u 20% slučajeva operator k_2 te u 20% slučajeva operator k_3.

Operator k_1 stvara dijete tako da termine za kolegije s vjerojatnošću od 80% uzima iz boljeg roditelja a s vjerojatnošću od 20% iz goreg roditelja; po vrsti izveden je kao uniformno križanje.

Operator k_2 gleda na načinjeni raspored kao na vektor koji ima onoliko elemenata koliko ima kolegija a i-ta komponenta je indeks termina dodijeljen i-tom kolegiju. Ako se s $\vec{r}_1$ označi pripadni vektor boljeg roditelja a s $\vec{r}_2$ pripadni vektor lošijeg roditelja,

dijete je definirano vektorom $\vec{r}_d = \vec{r}_1 + (\vec{r}_1 - \vec{r}_2) \cdot \vec{r}$, pri čemu je $\vec{r}$ slučajno generirani vektor s elementima između 0 i 1. Ideja iza ovog operatora je sljedeća: ako je nekom kolegiju lošije u petak a bolje mu je u srijedu, možda će mu biti još bolje u ponedjeljak ili utorak.

Operator k_3 je operator koji u potpunosti zanemaruje lošijeg roditelja. Umjesto toga, najprije se boljeg roditelja kopira u dijete i potom se pokušava određeni broj puta provesti sljedeći postupak.

1. Slučajno odaberi neki kolegij i.
2. Načini popis P svih kolegija koji s njime ne dijele studente.
3. Posredstvom slučajnog mehanizma iz te liste odaberi jedan od kolegija j.
4. Zamijeni termine kolegijima i i j.

Nakon primjene operatora križanja radi se mutacija (uz definiranu vjerojatnost svakom se kolegiju slučajno mijenja termin) te na kraju u 10% slučajeva i lokalna pretraga koja je prikazana izvornim kodom 5.13. Operator mutacije ovisno o trenutnom rješenju radi različite izmjene nad rješenjem, kako je opisano u nastavku.

- Ako rješenje ima konflikata i prekapacitiranih termina, tada će se za svaki prekapacitirani termin slučajno odabrati jedan od kolegija koji je smješten u taj termin i taj će se kolegij prebaciti u neki drugi koji ima dovoljno mjesta i gdje prebacivanjem neće nastati konflikti (ako takav termin postoji). Potom se u rješenju traže kolegiji koji imaju konflikte i jedan od kolegija koji je u konfliktu se pokušava prebaciti u neki drugi termin u kojem ima dovoljno mjesta i u kojem neće nastati konflikti.
- Ako rješenje ima konflikata ali nema prekapacitiranih termina, na prethodno opisani način se razrješavaju samo konflikti.
- Ako rješenje nema konflikata ali ima prekapacitiranih termina, na prethodno opisani način se razrješavaju samo prekapacitirani termini.
- Ako rješenje nema konflikata i nema prekapacitiranih termina, obavlja se mutacija termina svakog od kolegija u skladu sa zadanom vjerojatnošću mutacije. U

određenom postotku slučajeva također se odabiru dva termina i radi se kompletna zamjena dodijeljenih kolegija.

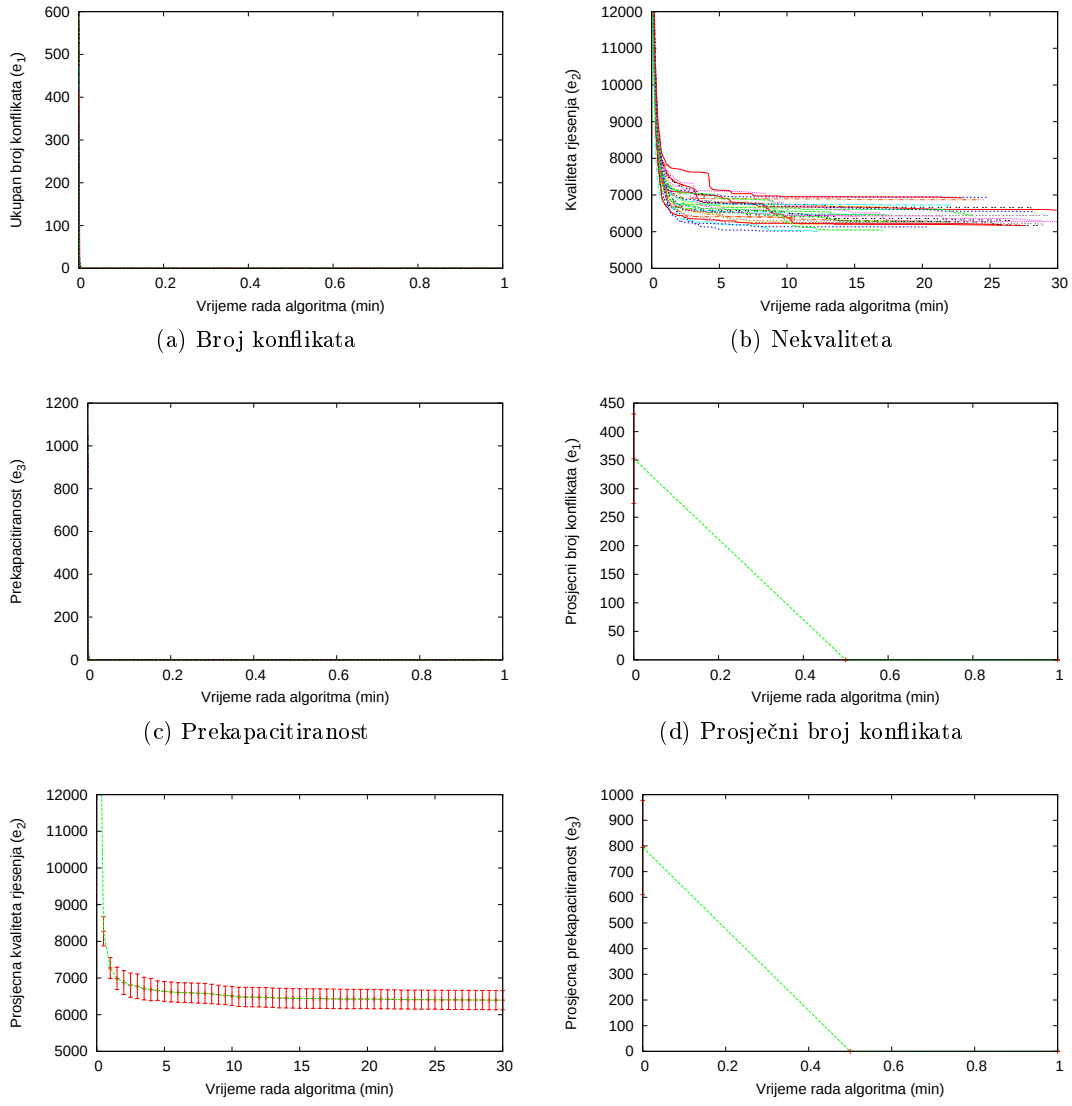
Osnovna ideja lokalne pretrage pogledati za svaki od kolegija koliko od doprinosi ukupnoj kazni te potom uporabom proporcinalne slučajne selekcije odabrati jedan kolegij koji će se pokušati popraviti (vjerojatnost odabira kolegija je to veća što je doprinos ukupnoj kazni rješenja veći). Nakon što se odabere kolegij, radi se promjena njegovog termina u sve dozvoljene termine (i popratno vrednovanje takvog rješenja što je vrlo sporo). Pronađe li se bolji termin za taj kolegij, rješenje se modificira tako da se prihvaća najbolji pronađeni termin. Ovaj se postupak potom ponavlja 10 puta.

```
Pokušaj 10 puta
  Stvori histogram kazne po kolegijima
  Koristeći histogram slučajno odaberi jedan kolegij K
  Stavi K u svaki od preostalih termina
  Ako postoji bolji termin za K od trenutnog, prihvati ga
kraj
```

Slika 5.13: Pseudokod algoritma lokalne pretrage.

Rad opisanog algoritma može se pratiti na slikama 5.14 i 5.15. Algoritam je pokrenut nad problemom koji se sastoji od razmještanja 140 kolegija u 38 termina pri čemu je definiran i jedan par kolegija koji se mora razmještati zajedno. U navedenom problemu svi termini nisu prihvatljivi za sve kolegije (primjerice, društveni kolegiji mogu ići samo u večernjim terminima). Načinjeno je 30 eksperimenata, svaki u trajanju od 30 minuta.

Slika 5.14 prikazuje rad algoritma uz uključenu lokalnu pretragu; slika 5.15 prikazuje rad algoritma uz isključenu lokalnu pretragu. Korištena je populacija od 200 jedinki, vjerojatnost pokušaja zamjene termina od 50% (u rješenju će se to pokušati najviše jednom) te vjerojatnost mutacije termina kolegija od 0.5%. Jedinke se biraju troturnirski, uspoređuju se po komponentama (poretkom $e_1 + e_3$ pa e_2) a zamjenjuje se najlošiji roditelj turnira. Na slikama 5.14a i 5.14c koje prikazuju broj konflikata te prekapacitiranost krivulje su pripijene uz ordinatu jer algoritam te vrijednosti uz uporabu lokalne pretrage razrješava vrlo brzo na početku pretrage.



(a) Broj konflikata

(b) Nekvaliteta

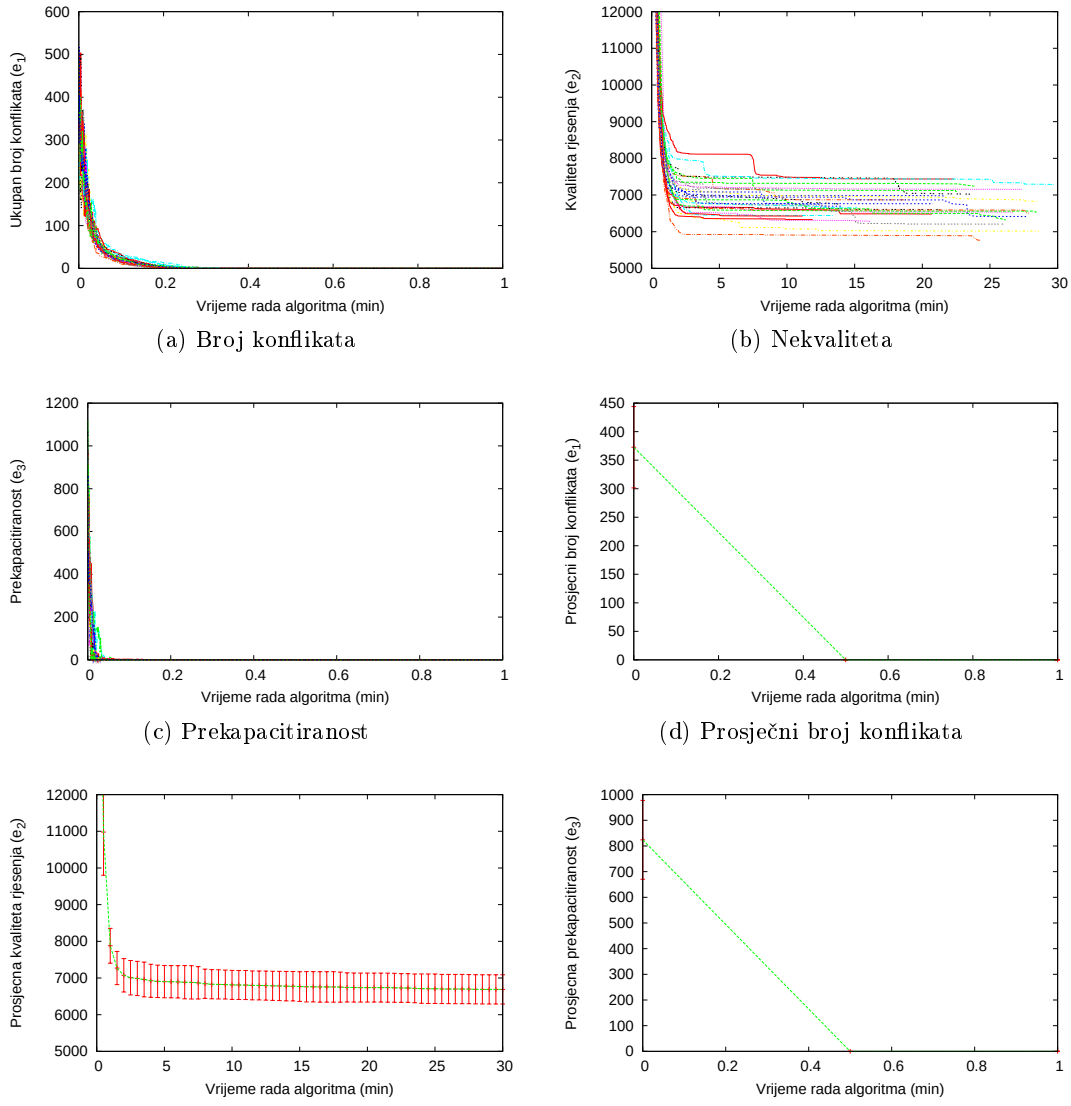
(c) Prekapacitiranost

(d) Prosječni broj konflikata

(e) Prosječna nekvaliteta

(f) Prosječna prekapacitiranost

Slika 5.14: Evolucija rješenja primjenom genetskog algoritma za rješavanje problema izrade rasporeda provjera znanja. Algoritam koristi lokalnu pretragu.



(a) Broj konflikata

(b) Nekvaliteta

(c) Prekapacitiranost

(d) Prosječni broj konflikata

(e) Prosječna nekvaliteta

(f) Prosječna prekapacitiranost

Slika 5.15: Evolucija rješenja primjenom genetskog algoritma za rješavanje problema izrade rasporeda provjera znanja. Lokalna pretraga je onemogućena.

5.2.4 Algoritam diferencijske evolucije

Algoritam diferencijske evolucije izvorno je definiran za rad nad kontinuiranim prostorom pretraživanja. Stoga je za potrebe rješavanja problema izrade rasporeda provjera znanja prikaz jedinke obogaćen dodatnim vektorom decimalnih brojeva čiji su elementi aproksimacije indeksa termina dodijeljenih dvakom od kolegija. Vektor stoga ima onoliko elemenata koliko ima kolegija. Indeks termina pojedinog kolegija dobit će se operacijom zaokruživanja te eventualnom korekcijom na najbliži dozvoljeni indeks termin.

U postupku rješavanja opisanog problema korišten je sljedeći algoritam diferencijske evolucije. Za svaki se ciljni vektor $\vec{t}$ bira bazni vektor $\vec{b}$ te dva slučajna vektora $\vec{r}_1$ i $\vec{r}_2$ (pojam vektor ovdje treba poistovijetiti sa spomenutim vektorom decimalnih brojeva pridruženim svakom rješenju). Potom se osigurava da vektor $\vec{r}_1$ predstavlja bolje rješenje od vektora $\vec{r}_2$ (ako to nije slučaj, vektori se samo zamijene). Gradi se vektor mutant $\vec{m}$. Za svaku se komponentu baznog vektora računa

$$f_i = F + 2 \cdot \Delta F \cdot rand() - \Delta F \tag{5.5}$$

i potom se postavlja

$$m_i = b_i + f_i \cdot (r_{1,i} - r_{2,i}). \tag{5.6}$$

Obavlja se križanje ciljnog vektora $\vec{t}$ i mutanta $\vec{m}$ kako bi se dobio probni vektor $\vec{p}$. S vjerojatnošću križanja C_r probni vektor $\vec{p}$ preuzima i-tu komponentu od mutanta a s vjerojatnošću $1 - C_r$ preuzima i-tu komponentu od ciljnog vektora. Nad tako dobivenim rješenjem provodi se postupak korekcija. Za kolegij c_i indeks termina se dobije zaokruživanjem i-te komponente probnog vektora $\vec{p}$. Ako je to nepostojeći termin, traži se najbliži postojeći i dozvoljeni termin, i ujedno se korigira i-ta komponenta probnog vektora $\vec{p}$. Nakon korekcije, temeljem probnog vektora generira se rješenje (popunjavaju se polja termina i kolegija objašnjena na početku ovog poglavlja), i probno rješenje se vrednuje. S definiranom vjerojatnošću nad probnim se rješenjem još pokreće i postupak lokalne pretrage koji je implementiran jednako kao i kod implementacije genetskog algoritma). Konačno, uspoređuje se dobrota probnog vektora i ciljnog vektora. Ako ciljni vektor nije bolji od probnoga, u sljedeću generaciju odlazi probni vektor; inače u sljedeću generaciju odlazi ciljni vektor. Pri tome, ako se u sljedećoj generaciji već

nalazi strukturno jednako rješenje, ono se neće dodati. Kako bi se dobio potreban broj rješenja za novu generaciju, opisani postupak će se ponavljati sve dok se ne izgenerira dovoljan broj različitih rješenja.

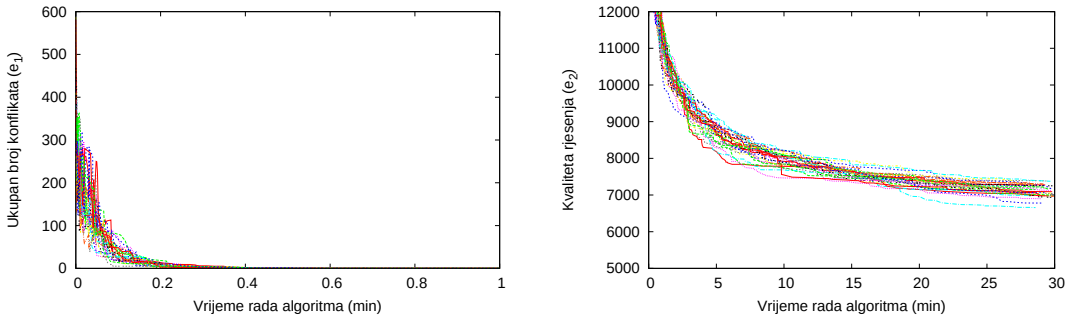
Slika 5.16 prikazuje rad algoritma na prethodno opisanom problemu raspoređivanja 140 kolegija uz uključenu lokalnu pretragu; slika 5.17 prikazuje rad algoritma na istom problemu uz isključenu lokalnu pretragu. Korištena je populacija od 200 jedinki, vjerojatnost lokalne pretrage od 1% (u slučaju da je lokalna pretraga bila uključena). Preostali parametri su redom $F = 0,4$, $\Delta F = 0,1$, $C_r = 0,01$. Načinjeno je 30 eksperimenata, svaki u trajanju od 30 minuta.

5.2.5 Max-Min algoritam mravljeg sustava

Za potrebe rješavanja problema rasporeda provjera znanja ostvaren je algoritam *Max-Min* mravlji sustav (engl. *Max-Min Ant System*). Problem je prikazan grafom sastavljenim od onoliko podgrafova koliko ima grupa predmeta koje se mogu nezavisno raspoređivati. Najveći broj grupa su pri tome grupe sastavljene od samo jednog predmeta; naime, svi nepovezani predmeti stvorit će svaki svoju grupu. Predmeti koji se moraju razmještati zajedno stvorit će samo jednu grupu.

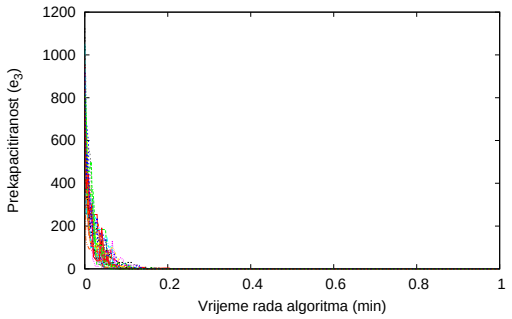
Svaki podgraf sastoji se od jednog čvora koji predstavlja jednu grupu predmeta te je lukovima povezan s čvorovima termina. Feromonski trag mravi će deponirati na bridove. Opisani graf (odnosno vrijednosti feromonskih tragova bridova) pamti se uporabom matrice. Matrica ima onoliko redaka koliko ima grupa kolegija i onoliko stupaca koliko ima termina. Feromonski trag zapisan na lokaciji (i, j) odgovara feromonskom tragu za smještanje grupe kolegija i u termin t_j.

Heuristička informacija u ovom slučaju nije statička niti unaprijed poznata. Naime, unaprijed nema nikakvih razloga da pojedini kolegiji preferiraju pojedine termine. Mravlji algoritmi pripadaju u porodicu konstruktivističkih metaheuristika koje rješenje grade element po element. Stoga se za potrebe ovog algoritma heuristička informacija gradi na zahtjev, što je pojašnjeno u sljedećem primjeru. Neka su grupe kolegija g_1, g_{17} i g_{35} već razmještene. Neka se kao sljedeći korak razmatra kamo staviti grupu g_{20} (algoritam svaku puta redosljed grupa bira slučajno). Potrebna heuristička informacija $\eta_{20,j}$ za smještaj grupe g_{20} u termin j dobiva se tako da pogleda koliko će smještanjem te grupe u taj termin biti načinjeno novih kolizija (jer su grupe g_1, g_{17} i g_{35} već raz-

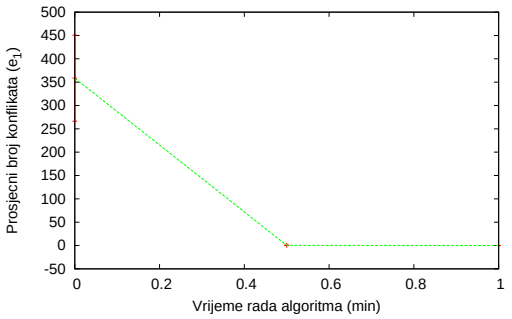


(a) Broj konflikata

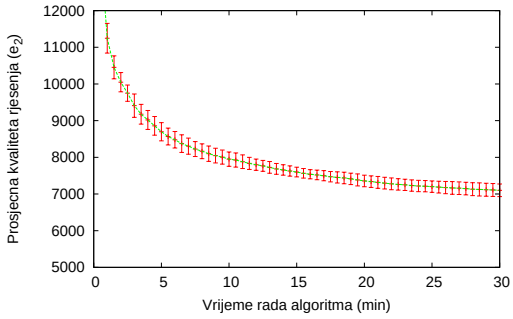
(b) Nekvaliteta



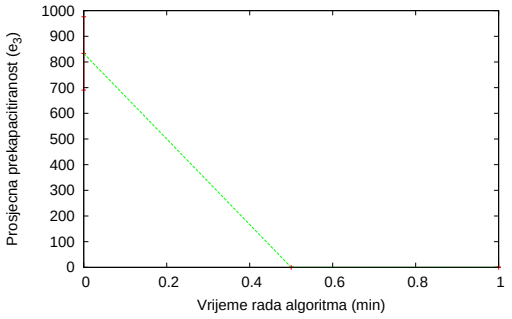
(c) Prekapacitiranost



(d) Prosječni broj konflikata

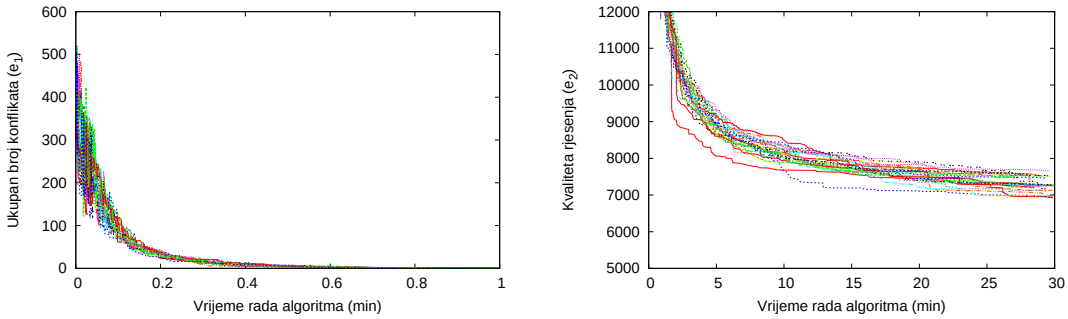


(e) Prosječna nekvaliteta



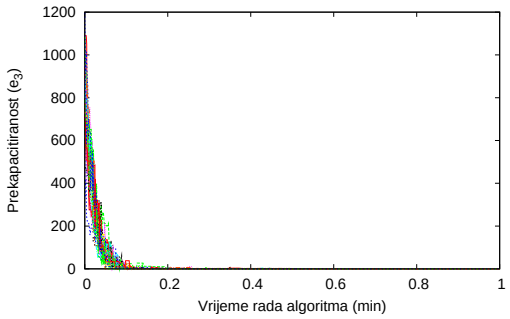
(f) Prosječna prekapacitiranost

Slika 5.16: Evolucija rješenja primjenom algoritma diferencijske evolucije za rješavanje problema izrade rasporeda provjera znanja. Algoritam koristi i lokalnu pretragu.

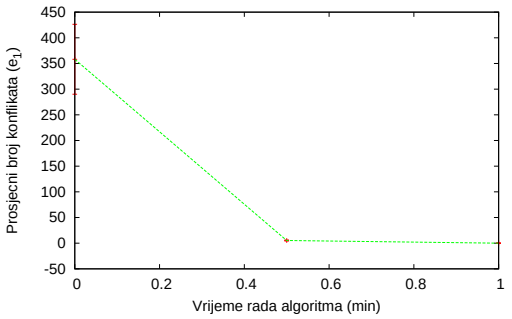


(a) Broj konflikata

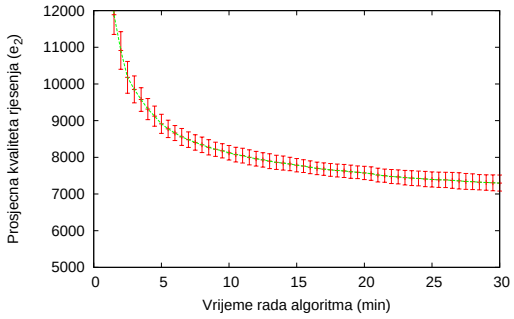
(b) Nekvaliteta



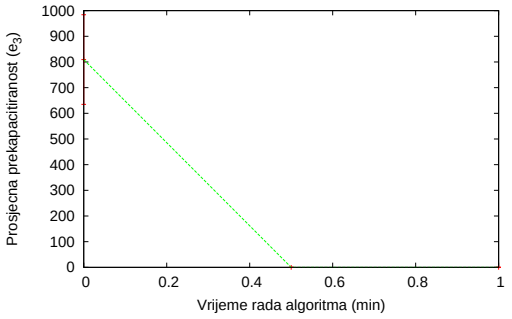
(c) Prekapacitiranost



(d) Prosječni broj konflikata



(e) Prosječna nekvaliteta



(f) Prosječna prekapacitiranost

Slika 5.17: Evolucija rješenja primjenom algoritma diferencijske evolucije za rješavanje problema izrade rasporeda provjera znanja. Lokalna pretraga je onemogućena.

mještene) – neka taj broj bude označen s k_0. Potom se gleda koliko još studenti upisani na kolegije te grupe imaju već razmještenih ispita u terminima koji su taj isti dan (ali nisu baš u terminu j) – neka taj broj bude označen s k_1. Konačno, gleda se koliko još studenti upisani na kolegije te grupe imaju već razmještenih ispita u terminima koji su sljedeći dan – neka taj broj bude označen s k_2. Vrijednost heurističke informacije tada se računa kao:

$$\eta_{20,j} = \frac{1}{1 + w_0 \cdot k_0 + w_1 \cdot k_1 + w_2 \cdot k_2}. \tag{5.7}$$

Što je težinska suma veća, situacija je lošija pa se zato kao heuristička informacija uzima recipročna vrijednost uz pomak 1 koji spriječava dijeljenje s nulom.

Temeljem ovako definirane heurističke informacije i temeljem postojećih feromonskih tragova radi se proporcionalna slučajna selekcija termina u koji se potom razmješta grupa predmeta. Postupak se ponavlja za svaku grupu sve dok raspored nije gotov. Nad tako konstruiranim rasporedom pokreće se lokalna pretraga. Nakon što svi mravi završe konstrukciju rasporeda, pronalazi se najbolje rješenje te iteracije – neka to rješenje bude označeno s I_{best}. Algoritam dodatno još prati najbolje ikada pronađeno rješenje – A_{best} te najbolje pronađeno rješenje od posljednjeg restarta algoritma – G_{best}. Ako je najbolje rješenje iteracije bolje od A_{best} ili G_{best}, ažuriraju se ta rješenja. Radi se isparavanje feromonskih tragova po svim bridovima te deponiranje nove količine feromona po bridovima koje definira A_{best}.

Kako bi se odredila količina novih feromona koje treba deponirati, definirana je skalarna funkcija $score(S)$ koja vraća kvalitetu rješenja S, prema pseudokodu 5.18.

Ideja iza opisanog načina vrednovanja je sljedeća. Prilikom inicijalizacije algoritma nasumice se stvori 20 rješenja i odabere se najbolje. Temeljem tog rješenja zapamti se koliko ono krši čvrsta ograničenja, max_cvrsta = 2∗(e[1] + e[3]), i koliko ono krši meka ograničenja, max_meka = 4∗e[2], i ti se podatci čuvaju kao početna procjena. Kako postoje dvije komponente a potreban je jedan skalar, prilikom vrednovanja rješenja gleda se koliko je rješenje bolje od zapamćenih maksimuma. Pri tome se poboljšanje u kršenju čvrstog ograničenja množi s 10 tako da je taj broj uvijek 10 ili više (jer se za čvrsta ograničenja, ako ih se krši, mjera tog kršenja prati kao cijeli pozitivan broj). Preostali interval od 0 do 10 (odnosno nešto niže) koristi se da bi se linearno preslikalo poboljšanje s obzirom na meka ograničenja.

Osnovni razlog za ovakvo definiranje funkcije *score* je nastojanje da se razlika između

```
funkcija score(S)
  cvrsta = S.eval[1] + S.eval[3]
  meka = S.eval[2]
  razlika = max_cvrsta - cvrsta
  ako je razlika < 0
    vrati 1/cvrsta + 1/meka
  kraj
  razlika *= 10
  razlika2 = max_meka - meka
  ako je razlika2 < 0
    vrati 1 + razlika + 0.01 / meka
  kraj
  vrati 1 + razlika + (8.8/max_meka)*razlika2 + 0.01 / meka
kraj
```

Slika 5.18: Pseudokod vrednovanja kvalitete rješenja za ACO.

dobrog i lošeg rješenja učini jasnija i veća, što bi algoritmu trebalo omogućiti lakši rad. Zbog toga se prilikom detekcije stagnacije algoritma, odnosno situacija da algoritam kroz zadani broj iteracija nije poboljšao G_{best} radi reinicijalizacija feromonskih tragova na njihove maksimalne vrijednosti te ažuriranje varijabli max_cvrsta i max_meka temeljem rješenja A_{best}.

Detekcija stagnacije implementirana je na sljedeći način. Zadaje se prag stagnacije N_p te multiplikator praga N_m. Početno, $N_m = 1$. Algoritam održava brojač stagnacije N_n. Svaki puta kada najbolje rješenje iteracije I_{best} nije bolje od G_{best}, brojač stagnacije se povećava za 1. Ako najbolje rješenje iteracije I_{best} bude bolje od G_{best}, brojač stagnacije se resetira na 1. Ako je najbolje rješenje iteracije I_{best} bolje i od A_{best}, multiplikator praga se resetira na 1. Ako brojač stagnacije dosegne vrijednost $N_p \cdot N_m$, radi se reinicijalizacija feromonskih tragova. Svi tragovi se postavljaju na maksimalne, multiplikator se udvostručuje $N_m \leftarrow N_m \cdot 2$, zaboravlja se G_{best} i korigiraju faktori za funkciju $score(S)$.

Tijekom rada algoritma, omjer između τ_{max} i τ_{min} određen je faktorom a, a τ_{max} se ažurira svaki puta kada se pronađe novo A_{best} na vrijednost:

$$\tau_{max} = \frac{score(A_{best})}{\rho}. \tag{5.8}$$

Slika 5.19 prikazuje rad algoritma na prethodno opisanom problemu raspoređivanja 140 kolegija uz uključenu lokalnu pretragu; slika 5.20 prikazuje rad algoritma na istom

Tablica 5.3: Rezultati algoritama: iznos čvrstog ograničenja e_1.

Algoritam	Prosjek	Median	Std. odst.	Najbolji	Najgori
GA	0.0	0.0	0.0	0.0	0.0
GA/NLS	0.0	0.0	0.0	0.0	0.0
DE	0.0	0.0	0.0	0.0	0.0
DE/NLS	0.0	0.0	0.0	0.0	0.0
ACO	0.0	0.0	0.0	0.0	0.0
ACO/NLS	0.5	0.0	2.01	0.0	10.0

problemu uz isključenu lokalnu pretragu. Korištena je populacija od 50 mrava, te parametri $\alpha = 3$, $\beta = 2$ te $a = 500$. Načinjeno je 30 eksperimenata, svaki u trajanju od 30 minuta.

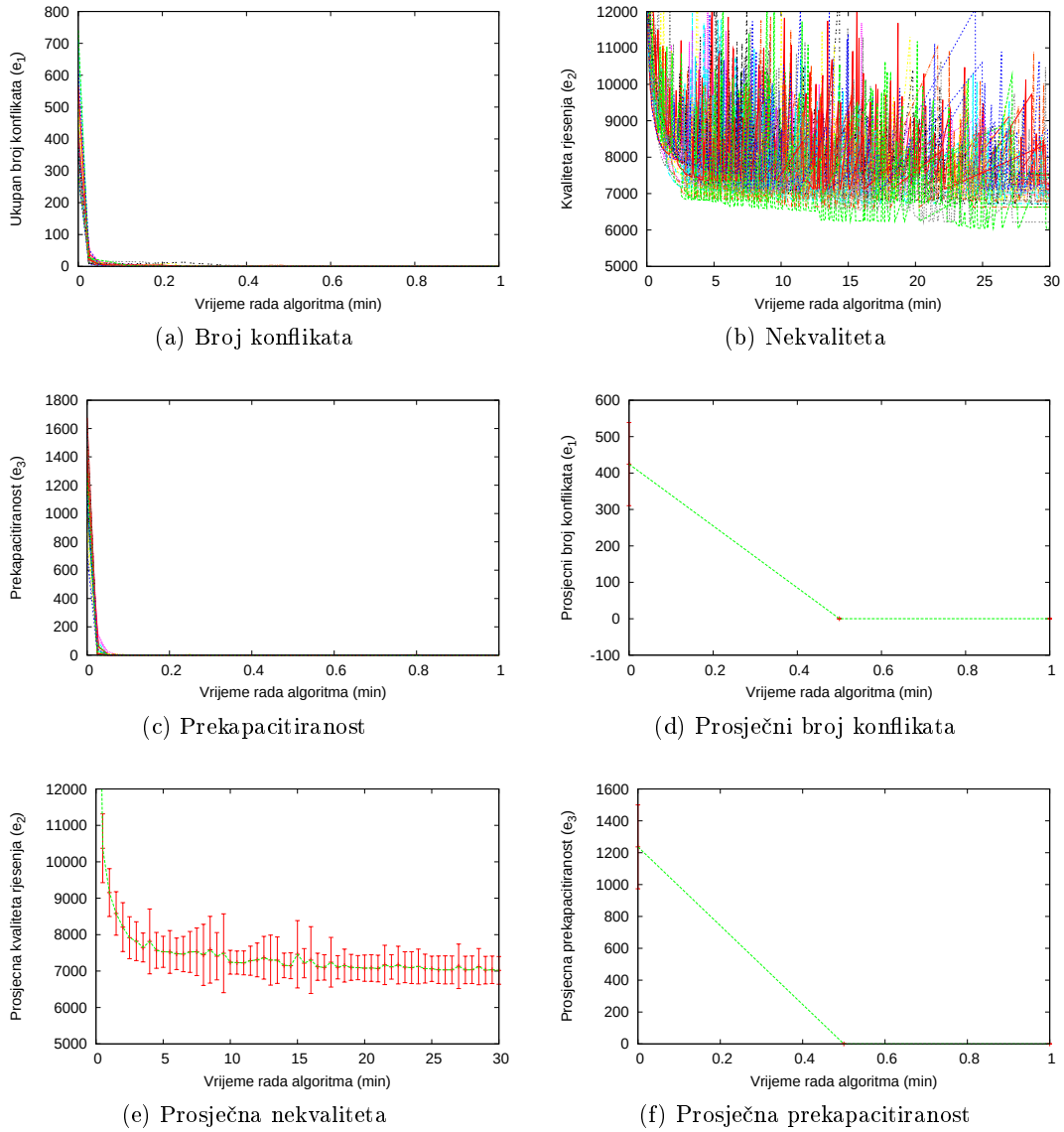
Na slikama 5.19 i 5.20 jasno se mogu vidjeti i trenutci u kojima se algoritam restarta i krene u novu epohu pretraživanja. Međutim, ono što se na slici direktno ne vidi je najbolje pronađeno rješenje koje algoritam pamti i održava (A_{best}). Na prikazanim slikama to bi rješenje odgovaralo onom koje u bilo kojem trenutku ima najmanji iznos po ordinati – naime, neovisno o trenutnom G_{best}, u trenutku prestanka rada algoritam će kao rezultat optimizacije ponuditi rješenje A_{best}. Kretanje rješenja A_{best} u ovisnosti o vremenu za slučaj s i bez lokalne pretrage prikazuju slike 5.21 i 5.22.

5.2.6 Usporedba algoritama

Temeljem načinjenih eksperimenata može se načiniti i usporedba rada triju prikazanih algoritama i to s i bez lokalne pretrage. Prije no što se načini sama usporedba, potrebno je napomenuti da algoritmi nisu niti na koji način podešavani problemu nad kojim su pokrenuti već su svi parametri postavljeni na neke "razumne" vrijednosti.

Tablica 5.3 prikazuje podatke za prvo čvrsto ograničenje: broj preostalih konflikata (e_1). Statistički podatci su dobiveni obradom najboljeg rješenja za svako pokretanje algoritma (dakle, za svaki algoritam prikupljeno je 30 rješenja). Iz podataka je jasno da sva tri algoritma uz pomoć lokalne pretrage uspješno svode količinu konflikata na 0. Bez lokalne pretrage svi su algoritmi osim Max-Min mravljeg sustava također uspješni, dok Max-Min mravlji sustav uspjeva ovaj broj spustiti na relativno nisku vrijednost, ali ne uvijek na nulu.

Tablica 5.4 prikazuje postignute rezultate s obzirom na mjeru nekvalitete rješenja (komponentu e_2). Iz podataka se vidi da je genetski algoritam ovdje u blagoj prednosti



(a) Broj konflikata

(b) Nekvaliteta

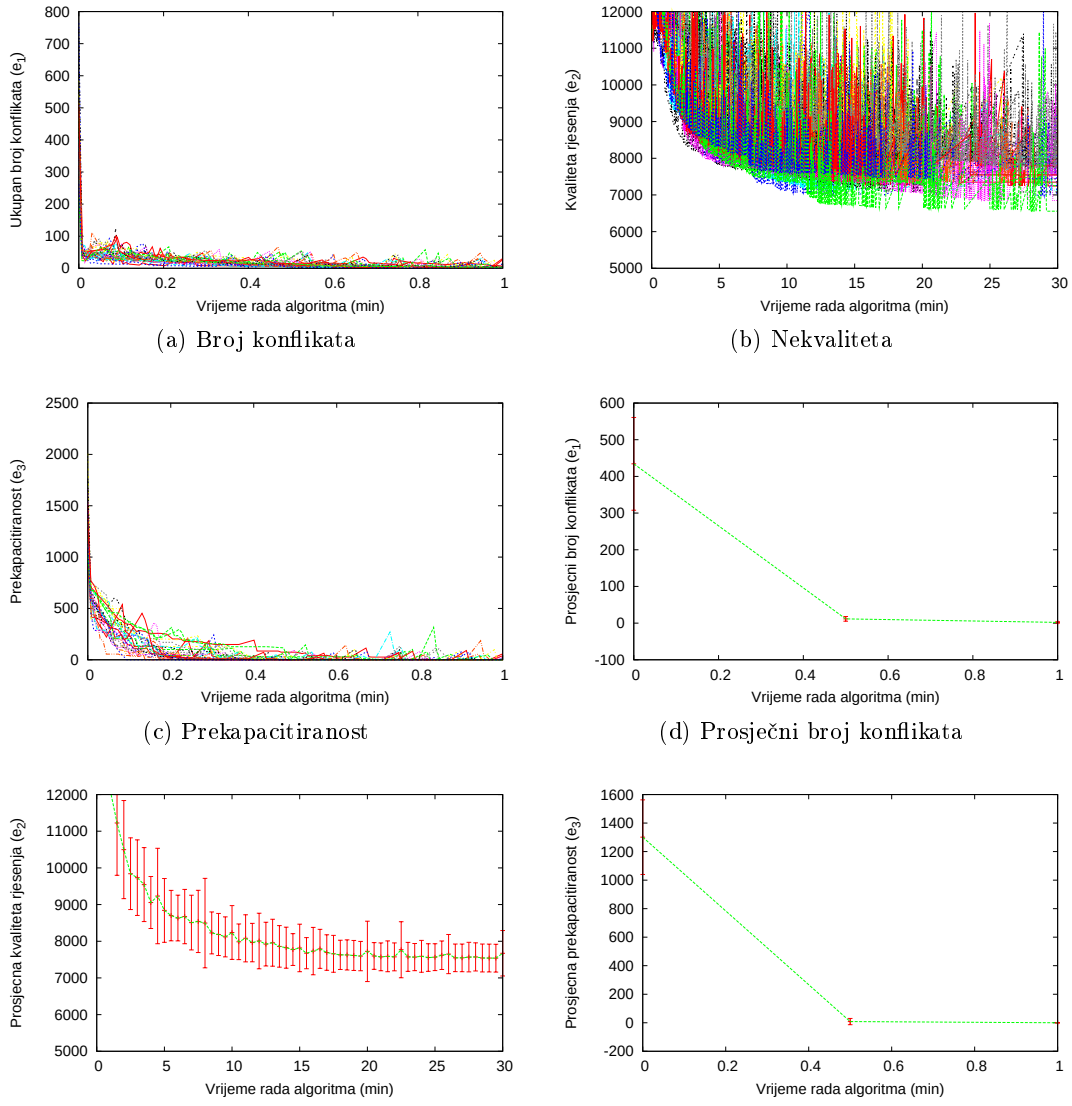
(c) Prekapacitiranost

(d) Prosječni broj konflikata

(e) Prosječna nekvaliteta

(f) Prosječna prekapacitiranost

Slika 5.19: Evolucija rješenja primjenom algoritma *Max-Min* mravlji sustav za rješavanje problema izrade rasporeda provjera znanja. Algoritam koristi i lokalnu pretragu. Prikazan je G_{best}.



(a) Broj konflikata

(b) Nekvaliteta

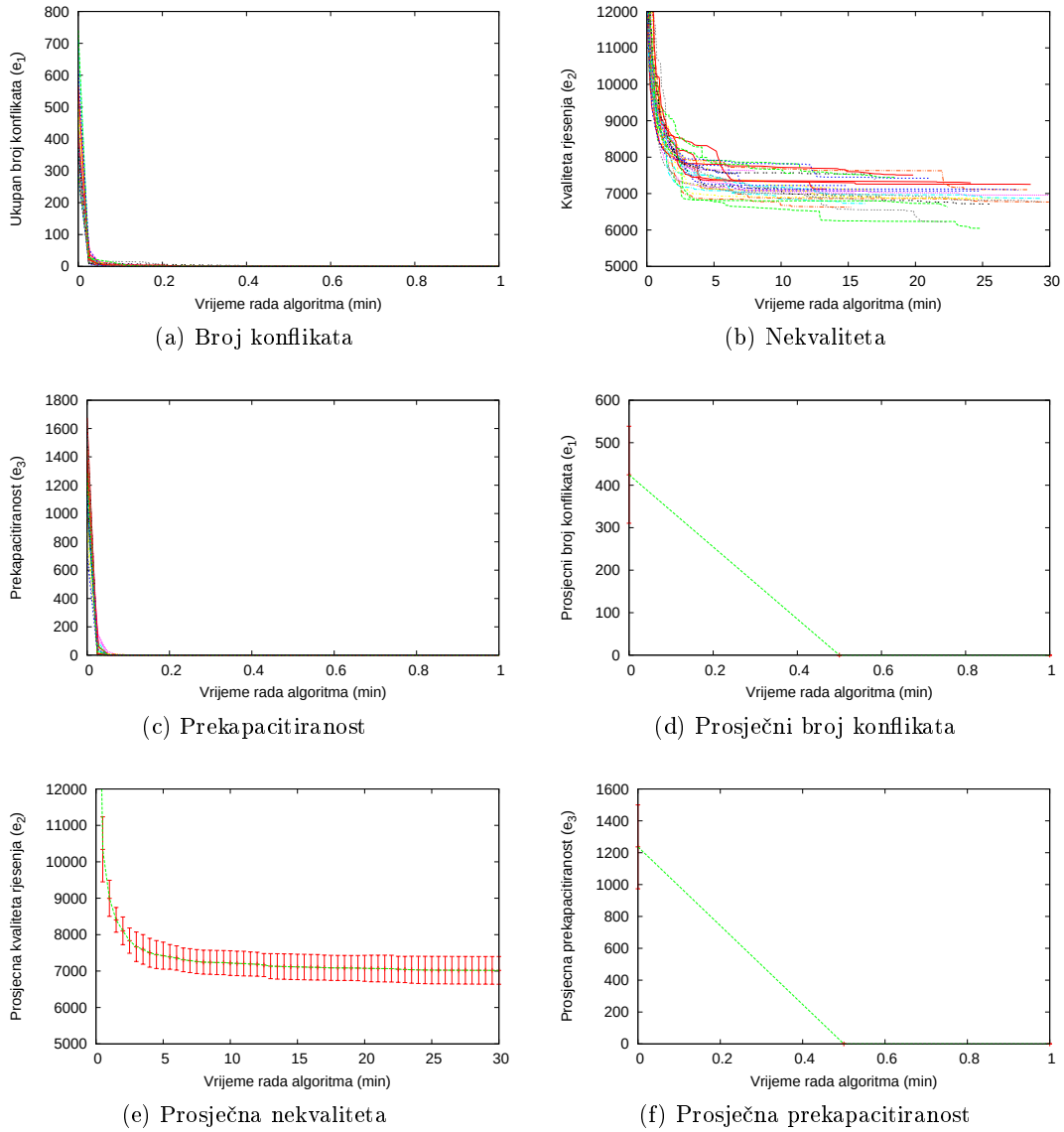
(c) Prekapacitiranost

(d) Prosječni broj konflikata

(e) Prosječna nekvaliteta

(f) Prosječna prekapacitiranost

Slika 5.20: Evolucija rješenja primjenom algoritma *Max-Min* mravlji sustav za rješavanje problema izrade rasporeda provjera znanja. Lokalna pretraga je onemogućena. Prikazan je G_{best}.



(a) Broj konflikata

(b) Nekvaliteta

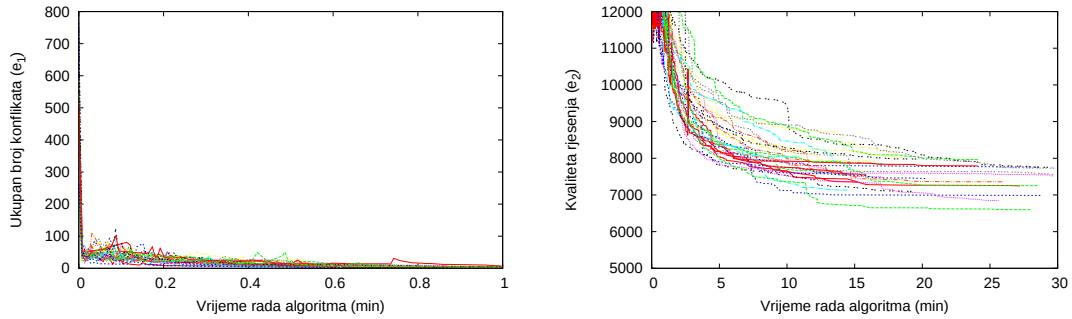
(c) Prekapacitiranost

(d) Prosječni broj konflikata

(e) Prosječna nekvaliteta

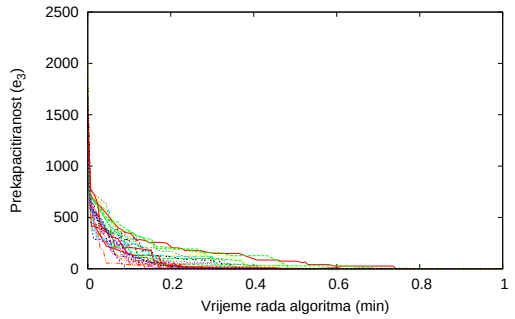
(f) Prosječna prekapacitiranost

Slika 5.21: Evolucija rješenja primjenom algoritma *Max-Min* mravlji sustav za rješavanje problema izrade rasporeda provjera znanja. Algoritam koristi i lokalnu pretragu. Prikazan je A_{best}.

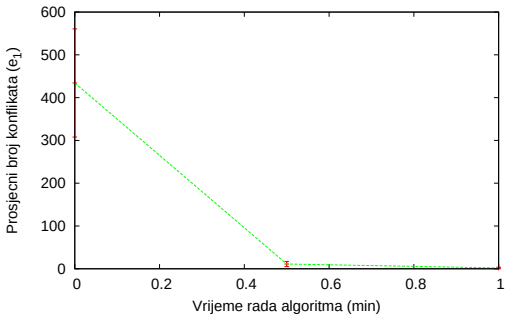


(a) Broj konflikata

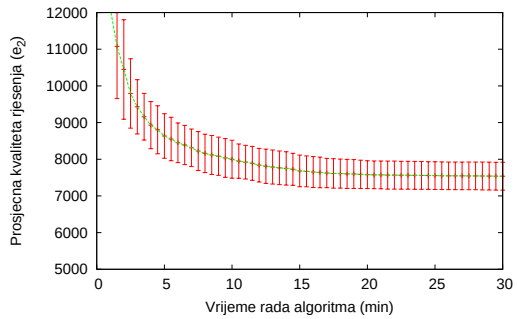
(b) Nekvaliteta



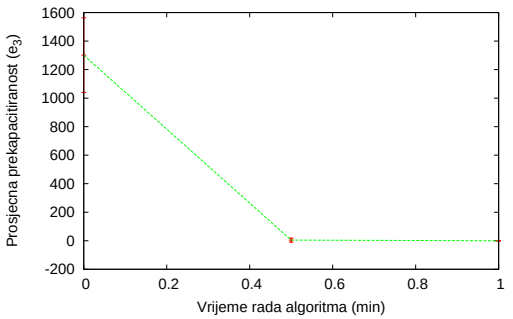
(c) Prekapacitiranost



(d) Prosječni broj konflikata



(e) Prosječna nekvaliteta



(f) Prosječna prekapacitiranost

Slika 5.22: Evolucija rješenja primjenom algoritma *Max-Min* mravlji sustav za rješavanje problema izrade rasporeda provjera znanja. Lokalna pretraga je onemogućena. Prikazan je A_{best}.

Tablica 5.4: Rezultati algoritama: iznos mekog ograničenja e_2.

Algoritam	Prosjek	Median	Std. odst.	Najbolji	Najgori
GA	6394.97	6344.0	262.42	6007.00	6932.00
GA/NLS	6688.93	6673.00	397.87	5768.00	7425.00
DE	7100.43	7092.00	172.17	6657.00	7382.00
DE/NLS	7297.93	7280.50	218.30	6913.00	7665.00
ACO	7017.80	7047.50	380.535	6044.00	7628.00
ACO/NLS	7672.77	7591.00	620.46	6556.00	9593.00

Tablica 5.5: Rezultati algoritama: iznos čvrstog ograničenja e_3.

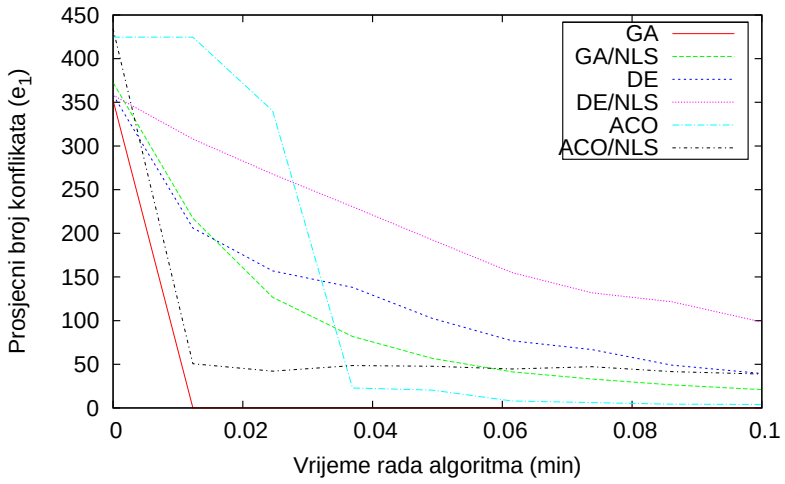
Algoritam	Prosjek	Median	Std. odst.	Najbolji	Najgori
GA	0.0	0.0	0.0	0.0	0.0
GA/NLS	0.0	0.0	0.0	0.0	0.0
DE	0.0	0.0	0.0	0.0	0.0
DE/NLS	0.0	0.0	0.0	0.0	0.0
ACO	0.0	0.0	0.0	0.0	0.0
ACO/NLS	0.63	0.0	3.47	0.0	19.0

pred algoritmom diferencijske evolucije te Max-Min mravljim sustavom.

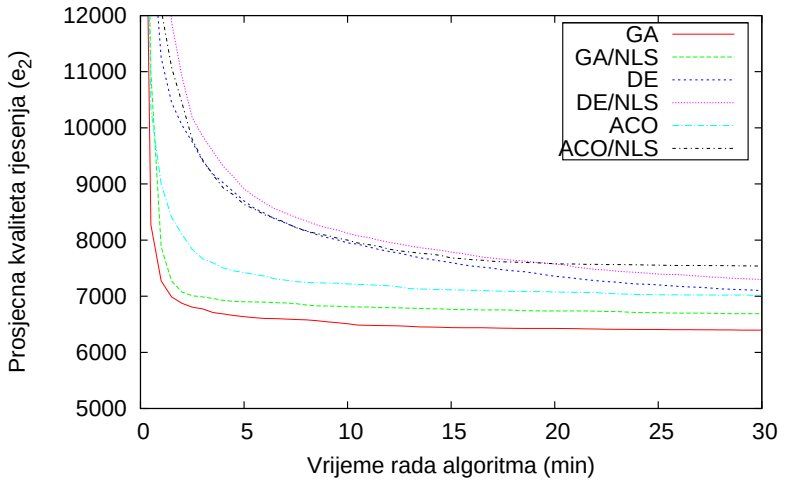
Konačno, tablica 5.5 prikazuje postignute rezultate s obzirom na količinu prekapacitiranosti termina (komponentu e_3). Sva tri algoritma uz uključenu lokalnu pretragu te genetski algoritam i algoritam diferencijske evolucije bez lokalne pretrage uspjevaju riješiti ovo ograničenje. Max-Min mravlji sustav bez lokalne pretrage ne uspije uvijek razriješiti svu prekapacitiranost, ali je svakako svede na vrlo malu.

Grafički prikaz rezultata dat je na slici 5.23. Grafovi za e_1 i e_3 su crtani samo za prvi 6 sekundi (0.1 minutu) jer svi algoritmi u tom vremenu razriješe ono što mogu i kasnije se vrijednosti više ne mijenjaju značajno. Stoga je prikazan sam početak kako bi se mogla pratiti dinamika rada algoritama. Genetski algoritam i Max-Min mravlji sustav na početku uspjevaju dosta agresivno smanjiti vrijednosti e_1 i e_3. Lokalna pretraga koja se kod algoritma Max-Min mravlji sustav dugoročno pokazuje korisnom u ovom prvom dijelu algoritmu smeta – moguće objašnjenje je pojava razlike između rješenja koje algoritam generira temeljem heurističke informacije i feromonskih tragova te rješenja koje preostane nakon lokalne pretrage; algoritmu treba određeno vrijeme dok prihvati ovako modificirano rješenje i stoga u prvom dijelu uz lokalnu pretragu algoritam radi slabije.

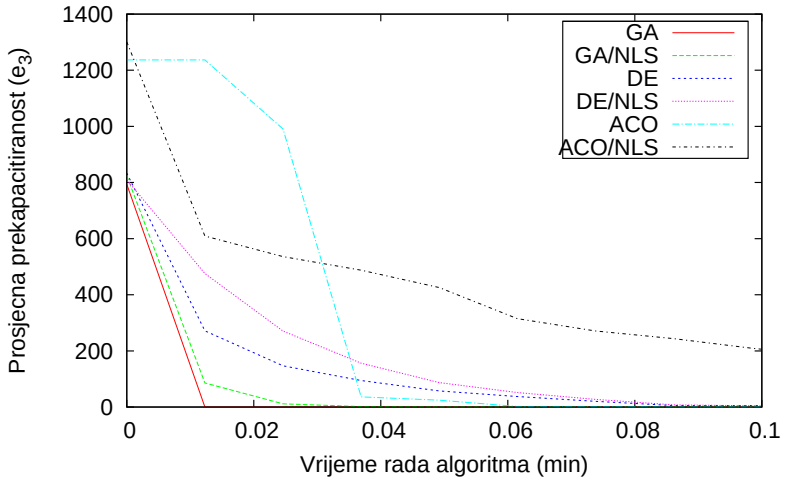
Iz ovih rezultata vidljivo je da su algoritmi evolucijskog računanja prikladni za rje-



(a) Broj konflikata



(b) Nekvaliteta



(c) Prekapacitiranost

Slika 5.23: Usporedba evolucije rješenja kod triju opisanih algoritama u izvedbama s i bez lokalne pretrage.

šavanje problema izrade rasporeda provjera znanja.

5.3 Drugi problemi raspoređivanja

S obzirom na sličnosti u pristupu rješavanja preostalih problema iz poglavlja 4, u ovoj se disertaciji evolucijski algoritmi razvijeni za njihovo rješavanja neće detaljno opisivati. Ono što je važno je istaknuti da su se algoritmi evolucijskog računanja doista pokazali prikladnima za rješavanje svih navedenih problema, o čemu svjedoči i njihova uspješna primjena na Fakultetu elektrotehnike i računarstva u posljednjih pola desetljeća.

Više detalja o evolucijskim algoritmima primijenjenim na pojedine probleme može se potražiti u literaturi navedenoj u nastavku.

- *Raspoređivanje neraspoređenih studenata po predavanjima*: više se može pogledati u radu [Čupić et al., 2010]; primijenjen je genetski algoritam.
- *Raspored laboratorijskih vježbi*: više se može pogledati u radovima [Bratković et al., 2009, Dino Matijaš et al., 2010, Čupić and Franović, 2010] gdje je opisana primjena genetskog algoritma te mravljeg algoritma na navedeni problem te u diplomskim radovima studenata [Omrčen, 2010, Matijaš, 2009, Herman, 2010, Molnar, 2011].
- *Raspored provjera znanja*: više se može pogledati u radovima [Čupić et al., 2009a, Čupić and Franović, 2010] gdje je primijenjen genetski algoritam te u radu [Komar et al., 2011] koji opisuje raspodijeljeni paralelni evolucijski algoritam.
- *Raspored prostorija za provjere znanja*: više se može pogledati u radu [Čupić et al., 2010] gdje je primijenjen genetski algoritam.
- *Raspoređivanje timova*: više se može pogledati u radu [Čupić and Franović, 2010] gdje je primijenjen algoritam klonske selekcije.
- *Vrsta jednostavnog raspoređivanja*: više se može pogledati u završnim radovima [Bobesić, 2010, Boltužić, 2010, Ćutić, 2010, Franović, 2010, Pribil, 2010] u kojima je opisana primjena algoritama evolucijskih računanja na problem izrade rasporeda nadoknada, i to redom: stohastičko difuzno pretraživanje [Bobesić, 2010], algoritam kolonije pčela [Boltužić, 2010], algoritam roja čestica [Ćutić, 2010], algoritam klonske selekcije [Franović, 2010] te genetski algoritam [Pribil, 2010].

Poglavlje 6

Paralelizacija evolucijskih algoritama

Evolucijsko računanje pokazalo se je prikladnim za rješavanje čitavog niza teških optimizacijskih problema. Pod pojmom teški optimizacijski problemi podrazumijevaju se problemi čija je složenost tolika da bi i na najmodernijim danas dostupnim superračunalima iscrpna pretraga trajala dulje no što smo spremni čekati; u problemima raspoređivanja koji su opisani u ovoj disertaciji čest je slučaj da bi iscrpna pretraga trajala dulje no što je star svemir.

Metaheuristike evolucijskog računanja pomažu da se unatoč takvoj složenosti dođe do prihvatljivih rješenja, iako ne nude nikakve garancije optimalnosti rješenja ako im se izvođenje ograniči na konačno vrijeme. Ukloni li se ova ograda, za neke od metaheuristika postoje dokazi pronalaska optimalnog rješenja. Primjerice, pogledaju li se imunološki algoritmi, to su algoritmi koji svoj rad temelje na različitim operatorima mutacije. Ako se pri tome koristi operator kod kojeg je (i) djelovanje upravljano vjerojatnostima te (ii) postoji vjerojatnost različita od nule da se u okviru jedne mutacije promijene svi geni rješenja koje se mutira, takav će algoritam, pusti li ga se da radi beskonačno dugo sigurno u jednom trenutku izgenerirati i ono optimalno rješenje. Ovisno o tome koliko smo sretni ili ne, tada postoji mogućnost da bi i postupak iscrpne pretrage prije došao do tog rješenja, jer će taj postupak sigurno završiti u konačnom vremenu (rješenja s kojima radimo su diskretna, s fiksnim brojem elemenata i fiksnim brojem izbora pa je i njihov broj konačan).

Odustane li se od zahtjeva optimalnosti, ipak je činjenica da će rješenja koja algo-

ritam pronalazi biti to bolja što je više različitih rješenja algoritam mogao istražiti. U tom smislu, algoritmima evolucijskog računanja važno je osigurati mogućnost pretraživanja što je moguće većeg područja, a to je vremenski skupo. Izvodi li se algoritam na jednoprocesorskom računalu, pomoći nema. Međutim, ima li se na raspolaganju višeprocesorsko računalo, ili više računala, paralelizacijom je moguće u ograničenom vremenu omogućiti algoritmu šire i kvalitetnije pretraživanje prostora rješenja.

Danas se paralelizacija radi iz nekoliko razloga.

1. *Paralelizacija zbog velikih količina podataka.* Postoje problemi čija su rješenja prevelika da bi stala u spremnik jednog računala. Postoje problemi kod kojih se vrednovanje radi nad velikom količinom podataka koji opet svi ne stanu u spremnik jednog računala. Paralelizacijom je ovakve probleme moguće razdijeliti na više računala i izgradnju čitavog rješenja ili vrednovanje rješenja obavljati dio po dio.

2. *Paralelizacija zbog skraćivanja vremena izvođenja.* Uz konstantan obim posla koji je potrebno načiniti, ako je dijelove algoritma moguće paralelizirati, čitav će postupak biti prije gotov. Ovo može biti interesantno ako postoje stoga vremenska ograničenja koja je potrebno zadovoljiti.

3. *Paralelizacija zbog povećanja obima poslova.* Ako se vrijeme izvođenja algoritma drži konstantnim, paralelizacija dijelova algoritma omogućit će da se u istom vremenu odradi više posla.

Primjena paralelizacije kod algoritama evolucijskog računanja osigurava sljedeće:

- ubrzavanje algoritma kako bi se prije došlo do prihvatljivih rješenja,
- povećanje kvalitete pronađenih rješenja jer algoritam može istražiti veći prostor rješenja te
- povećanje robusnosti algoritma jer će se smanjiti vjerojatnost ostanka u lošijim područjima zbog mogućnosti istraživanja većeg dijela prostora rješenja.

6.1 Vrste paralelizacije

Uobičajno je paralelizaciju promatrati na tri razine:

1. paralelizacija na razini algoritama,
2. paralelizacija na razini populacije te
3. paralelizacija na razini jedne iteracije algoritma.

Paralelizacija na razini algoritama predstavlja svaki oblik paralelizacije kod kojeg se istovremeno izvodi više primjeraka algoritama. Ova vrsta paralelizacije može biti suradna (engl. *cooperative*) i nesuradna (engl. *non-cooperative*). Nesuradna paralelizacija je paralelizacija kod koje različiti primjerci algoritama međusobno ne komuniciraju i ne dijele nikakve informacije. Primjer ovakve paralelizacije je nezavisno pokretanje više primjeraka algoritama te uzimanje odabir najboljeg rješenja među rješenjima koje su pronašli pojedini primjerci algoritama. Druga mogućnost je pokretanje više primjeraka algoritama ali s različitim parametrima, u nadi da će uz neke algoritam raditi bolje. Već i ovakav oblik nesuradne paralelizacije donosi povećanu robusnost algoritma te dobivanje boljih rješenja.

Suradna paralelizacija podrazumijeva pokretanje više primjeraka algoritama koji međusobno razmjenjuju informacije. Ovisno o vrsi algoritma, moguće je razmjenjivati različite podatke, poput dijelova populacije (često se dijeli samo najbolje rješenje), ili informacija koje utječu na rad algoritma (trenutna vjerojatnost mutacije, vrijednost feromonskih tragova kod mravljih algoritama i slično). Uvođenje suradne paralelizacije povlači porast broja parametara takvog algoritma: treba odrediti što se šalje, tko kome šalje (topologija) te u kojim se trenutcima informacije šalju. Poznati primjeri ovakve vrste paralelizacije su otočni model te celularni model često korišteni kod genetskih algoritama.

Paralelizacija na razini populacije uključuje postupke paralelizacije koji omogućavaju bržu izmjenu populacija. Primjerice, ako je potrebno izgraditi sljedeću populaciju od n jedinki, posao izgradnje se može paralelizirati na k radnika pa će vrijeme potrebno za izgradnju populacije tada biti k puta manje u odnosnu na izgradnju jedne po jedne jedinke slijedno. Ovu vrstu paralelizacije moguće je ostvariti na više načina. Jedna mogućnost je paralelizirati čitav postupak izgradnje jedinke. Kod genetskog algoritma to bi značilo da radnik obavlja selekciju, križanje, mutaciju i vrednovanje, i takvu jedinku predaje dalje. Kod mravljih algoritama to bi značilo da jedan radnik obavlja izgradnju čitavog rješenja i provodi njegovo vrednovanje. Slično je moguće ostvariti i kod algorit-

ma diferencijske evolucije i drugih. Druga je mogućnost paralelizirati samo izvođenje vrednovanja, ako je to vremenski zahtjevna operacija. U tom slučaju postoji k radnika kojima se šalju jedinke a svaki radnik obavlja vrednovanje dobivenih jedinki i vraća rezultat vrednovanja. Uz k radnika u istom će se vremenu moći vrednovati približno k-puta više jedinki nego na jednom procesoru pa će se time postići ubrzanje.

Paralelizacija na razini jedne iteracije algoritma obuhvaća postupke paralelizacije koji su spušteni na najnižu moguću razinu. Čitav algoritam obavlja se kao da je slijedni, a paralelizacija je spuštena na razinu operatora. Primjerice, moguće je implementirati paralelni operator mutacije, paralelni operator križanja, paralelno vrednovanje rješenja i slično. Razlika u odnosu na paralelizaciju na razini populacije je u tome što kod ovog pristupa nemamo k radnika kod kojih svaki u potpunosti obavlja slijedno implementiran operator (npr. k radnika koji svaki vrednuje svoju jedinku). Umjesto toga, ovdje govorimo o paralelizaciji implementacije operatora, gdje se, primjerice, paralelizira izvedba algoritma vrednovanja jedinke tako da se što prije dođe do rezultata.

Potrebno je istaknuti da suradna paralelizacija na razini algoritama nije isto što i paralelizacija na razini populacije. Naime, iako se možda ne čini očito, nije isto pokrenuti 4 algoritma od kojih svaki ima populaciju od 50 jedinki i povremeno razmjenjuju rješenja te jedan algoritam koji ima 200 jedinki i koristi četiri radnika za paralelizaciju. U oba slučaja, broj obrađenih jedinki u jedinici vremena bit će isti. Međutim, uslijed efekata izoliranog pretraživanja unutar manjih populacija te uslijed povremene razmjene rješenja prvi pristup iskazivat će drugačiju dinamiku promjena u populaciji i često će naći bolja rješenja u odnosu na jednu veliku populaciju.

6.2 Načini i cijena paralelizacije

Kako se može provesti postupak paralelizacije uvelike će ovisiti o željenom broju radnika k (a time će i vrsta radnika biti utvrđena). Ako je prihvatljiv manji broj radnika, pojam radnika tada možemo poistovijetiti s procesorom računala. Tehnologije koje nam pri tome stoje na raspolaganju su uobičajena višedretvenost gdje je algoritam ujedno i jedan proces. S obzirom da sve dretve imaju pristup svim podatcima u adresnom prostoru procesa, dijeljenje poslova moguće je obaviti uz minimalne troškove a jedinke i sve potrebne podatkovne strukture tipično se prenose preko reference (ili kazaljke) čime se stvara minimalni mogući promet na podatkovnoj sabirnici računala. Trošak koji

plaćamo kod ovog oblika paralelizacije je trošak sinkronizacijskih mehanizama kojima ćemo osigurati da su svi podatci u konzistentnom stanju. Ovakav oblik paralelizacije prikladan je za sve vrste paralelizacije i možemo ga koristiti od izvedbe paralelizacije na razini jedne iteracije algoritma pa sve do izvedbe paralelizacije na razini algoritama. Problem ovog pristupa je u malom broju radnika koje podržava. Danas su među masovno raširenim računalima uobičajena računala dvije do četiri jezgre i tek se pojavljuju računala sa šest jezgri; stoga je to upravo i granica na broj radnika koji se može koristiti u paralelizaciji. Radi li se na četiri-jezgrenom računalu, moći će se koristiti najviše četiri radnika.

Razmatraju li se druge mogućnosti paralelizacije unutar istog računala, postoji i mogućnost paralelizacije na razini više procesa. Pri tome radnici mogu biti različiti procesi koji međusobno komuniciraju uporabom mehanizama operacijskog sustava za međuprocesnu komunikaciju; to su danas uobičajeno dijeljena memorija, komunikacija cjevovodima ili komunikacija općenitijim mrežnim podsustavom. Cijena koja se plaća ovakvim pristupom viša je od uporabe radnika unutar jednog procesa. Mehanizmi međuprocesne komunikacija su sporiji a mogu zahtjevati i dodatni utrošak vremena na poslove serijalizacije, slanja i deserijalizacije jedinki te serijalizacije, slanja i deserijalizacije rezultata (ako se radi o mehanizmima poput cjevovoda ili mrežnog podsustava). Također, ovakav način paralelizacije bit će neprikladan za paralelizaciju na nižim razinama. Broj radnika i dalje je ograničen na broj procesora odnosno jezgri koje se nalaze u računalu, pa nema neke očite prednosti u odnosu na paralelizaciju unutar istog procesa.

Za veći broj radnika danas je uobičajeno posegnuti za paralelizacijom uporabom više računala. Tu se može razmišljati o specijaliziranim okruženima poput grozda (engl. *cluster*), grida ili pak danas popularnog računarstva u oblacima (engl. *cloud-computing*). Međutim, kako u najvećem broju slučajeva ovakva infrastruktura nije dostupna, najčešće se koriste računala koja su dostupna, poput računala u nekom računalnom laboratoriju u trenutcima kada se ona ne koriste i slično. Kada se govori o ovakvom načinu paralelizacije, važno je uočiti da su ovdje troškovi komunikacije te troškovi serijalizacije i deserijalizacije izuzetno veliki, što uvelike ograničava vrste paralelizacije na koje je ovakav scenarij uopće primjenjiv i to svakako nisu paralelizacije na najnižoj razini. Prednost ovog pristupa je što omogućava izrazito velik broj radnika koje je moguće upogoniti. Tako primjerice imati 80 radnika ili 160 radnika nije ništa neobično. Ovaj

način paralelizacije najčešće se izvodi direktno uporabom mrežnog podsustava (primjerice, uporabom protokola TCP) a moguća je i uporaba specijaliziranih biblioteka poput biblioteke MPI koja programeru nudi jednostavan način uporabe ali i povećava troškove komuniciranja i zahtjeva prilagodbu podataka koji se šalju.

Konačno, s današnjim razvojem osobnih računala treba spomenuti još jedan način paralelizacije koji je vezan uz jedno računalo ali nudi relativno velik broj specijaliziranih radnika – radi se o grafičkom procesoru (engl. *Graphics Processing Unit*, GPU) koji je moguće iskoristiti za dobivanje čak i preko 100 procesnih jedinica. Međutim, kako su ti procesori daleko skromniji po mogućnostima od centralnog procesora, nisu prikladni za sve vrste problema. Dodatnu cijenu koju treba uzeti u obzir je prošak prebacivanje podataka iz radne memorije računala u memoriju grafičke kartice da bi ih GPU uopće vidio te prebacivanje rezultata iz memorije grafičke kartice u radnu memoriju računala da bi ih program koji se izvršava na računalu mogao iskoristiti. Ovisno o vrsti problema koji se rješava, rezultati mogu biti ubrzanje od više stotina puta [Harding and Banzhaf, 2007], ili mogu biti još i lošiji no da se izvode samo na centralnom procesoru. Za programiranje algoritama za izvođenje na grafičkom procesoru danas se najčešće koristi CUDA od tvrtke NVidia.

Na koji način obaviti paralelizaciju, dosta je osjetljivo pitanje. Korist koja se može dobiti od paralelizacije uobičajeno se mjeri pojmom *ubrzanje*. Neka imamo slijedni algoritam koji posao obavlja u vremenu T. Neka je s $r_s = \frac{T_s}{T}$ označen postotak vremena u kojem se izvodi dio algoritma koji nije moguće paralelizirati. Za dio algoritma koji je trošio preostalo vrijeme $T_p = T - T_s$ (neka je $r_p = (1 - r_s)$) neka je obavljena paralelizacija tako da se taj posao obavlja paralelno na k radnika. Time će vrijeme potrebno za izvođenje tog posla pasti na $T_p' = \frac{T_p}{k}$, čime će za čitav posao trebati $T' = T_s + T_p'$. Ubrzanje S definira se kao omjer ova dva vremena:

$$S = \frac{T}{T'} = \frac{T}{T_s + T_p'} = \frac{T}{T \cdot r_s + \frac{T \cdot r_p}{k}} = \frac{1}{r_s + \frac{r_p}{k}} = \frac{1}{(1 - r_p) + \frac{r_p}{k}}, \tag{6.1}$$

što direktno odgovara ubrzanju definiranom Amdahlovim zakonom [Amdahl, 1967].

Koja je posljedica ovog zakona? Neka imamo generacijski genetski algoritam koji radi s populacijom od 100 jedinki. Neka je algoritam pokrenut na jednoprocesorskom računalu. Neka je izmjereno vrijeme potrebno za stvaranje nove populacije 10 ms, a od tog vremena neka se na vrednovanje jedinki troši 6 ms. Ako se pokuša paralelizacija

operatora vrednovanja, koliko se najviše može ubrzati postupak stvaranja nove generacije ako se može uzeti proizvoljno mnogo radnika? U ovom slučaju je $r_s = \frac{4}{10} = 0,4$ i $r_p = \frac{6}{10} = 0,6$. Limes izraza izvedenog za ubrzanje kada se pusti da $k \to \infty$ je

$$S|_{k\to\infty} = \frac{1}{r_s} \tag{6.2}$$

što je u ovom slučaju jednako $\frac{1}{0.4} = 2,5$. Slijedi da, što god da se radi, takvom se paralelizacijom postupak može ubrzati najviše 2,5 puta, odnosno vrijeme skratiti s 10 ms na 4 ms. Dakako, ovo je gornja granica ubrzanja koja pretpostavlja da ne postoji trošak paralelizacije (sinkronizacije, serijalizacije, deserijalizacije, komunikacijski troškovi i slično). Uvede li se trošak paralelizacije τ po svakom radniku definiran kao postotak s obzirom na vrijeme T, tada se ukupno vrijeme uz paralelizaciju na k radnika može zapisati kao $T_p'' = \frac{T_p}{k} + k \cdot (\tau \cdot T)$. Tada će ubrzanje biti:

$$S = \frac{T}{T''} = \frac{T}{T_s + T_p''} = \frac{T}{T \cdot r_s + \frac{T \cdot r_p}{k} + k \cdot (\tau \cdot T)} = \frac{1}{r_s + \frac{r_p}{k} + k \cdot \tau} = \frac{1}{(1 - r_p) + \frac{r_p}{k} + k \cdot \tau}. \tag{6.3}$$

Da bi ubrzanje bilo veće od 1, nazivnik razlomka mora biti manji od 1. Međutim, trošak paralelizacije doprinosi iznosu vrijednosti nazivnika, i sasvim je moguće paralelizacijom generirati algoritam koji radi još i sporije no što je radio bez paralelizacije. Da se ovo ne bi dogodilo, član $k \cdot \tau$ mora biti zanemariv u odnosu na $\frac{r_p}{k}$, što pak znači da se paralelizacija isplati samo ako je trošak paralelizacije bitno manji od trajanja posla koji se paralelizira. Upravo iz ovog razloga nikada se neće raditi paralelizacija operatora mutacije na način da se dijelovi jedinke protokolom TCP šaljemo na k radnika i potom prikuplja natrag rezultat – trošak paralelizacije tu će biti bitno veći od izvođenja operatora mutacije bez paralelizacije i takvom ćemo paralelizacijom usporiti a ne ubrzati algoritam.

6.3 Paralelizacija algoritama za rješavanje problema izrade rasporeda provjera znanja

Temeljem prethodnog razmatranja, algoritme za rješavanje problema izrade rasporeda moguće je paralelizirati na nekoliko načina. Kao primjer ćemo uzeti algoritme koji rješavaju problem izrade rasporeda obaveznih provjera znanja.

6.3.1 Paralelizacija algoritma vrednovanja

Kod problema izrade rasporeda obaveznih provjera znanja vrednovanje je vremenski vrlo skupa operacija. Algoritam vrednovanja prikazan je pseudokodom 6.1.

```
e = {0,0,0};
za svaki termin ti {
  za svaki kolegij ci iz ti {
    za svaki kolegij cj iz ti {
      e[1] += doprinos_za_e1(ci,cj)
      e[2] += doprinos_za_e2(ci,cj)
    }
    za svaki termin tj razlicit od ti {
      za svaki kolegij cj iz tj {
        e[2] += doprinos_za_e2(ci,cj,ti,tj)
      }
    }
  }
  e[3] += doprinos_za_e3(ti)
}
za svaki par kolegija (ci,cj) iz liste za razmicanje {
  e[2] += doprinos_blizine_za_e2(ci,cj)
}
```

Slika 6.1: Pseudokod algoritma vrednovanja rješenja za problem izrade rasporeda provjera znanja.

Algoritam najprije prolazi kroz svaki od termina te svaki od kolegija smještenih u taj termin te za te kolegije računa:

- kaznu uslijed konflikata s kolegijima koji su smješteni u taj isti termin te
- kaznu zbog blizine smještaja sa svim drugim kolegijima u svim drugim terminima.

Složenost ovog postupka je kvadratna s brojem kolegija koji se razmješta. Kako je tipičan broj kolegija s kojima je algoritam u praksi korišten oko 140, to znači da se za potrebe vrednovanja gledaju desetci tisuća parova kolegija – i to troši nezanemarivu količinu vremena. Na kraju se još prolazi kroz popis kolegija čiju blizinu smještaja u rasporedu treba dodatno kazniti i ažuriraju se kazne.

Predloženi model paralelizacije opisane metode za vrednovanje unutar jednog računala proizlazi iz činjenice da se algoritam vrednovanja sastoji od dva dijela:

- petlje koja ide po svim terminima te nakon toga

- petlje koja ide po svim parovima kolegija iz liste za razmicanje.

Stoga se umjesto slijednog izvođenja vanjske petlje predlaže paralelno izvođenje unutrašnjosti petlje; naime, s obzirom na pojedine termine, konačna kazna je aditivna, pa se može računati po dijelovima i potom na kraju pribrojiti. Temeljem ovog razmatranja definiramo strukturu EvalJob, kako prikazuje izvorni kod 6.1.

```
1  class EvalJob implements Callable<int[]> {
2    int phase;
3    int fromTermIndex;
4    int toTermIndex;
5    int[] eval = new int[3];
6    Kromosom k;
7    Configuration conf;
8
9    @Override
10   public int[] call() throws Exception {
11     if(phase==0) {
12       k.evaluatePart1(conf, eval, fromTermIndex, toTermIndex);
13     } else {
14       k.evaluatePart2(conf, eval);
15     }
16     return eval;
17   }
18 }
```

Izvorni tekst programa 6.1: Segment posla za paralelno vrednovanje.

Primjerci razreda EvalJob opisivat će dio posla koji je potrebno obaviti. Varijabla fromTermIndex pri tome čuva indeks termina od kojeg je potrebno obaviti vrednovanje, a varijabla toTermIndex čuva indeks termina do kojeg je potrebno obaviti vrednovanje. Kako bi se izbjegli troškovi sinkronizacije, svaki primjerak ovog razreda ima svoj privremeni spremnik za vrijednosti kazni rješenja (polje eval). Varijable k i conf čuvaju informaciju o rješenju koje je potrebno vrednovati te o statičkim podatcima koji se koriste prilikom vrednovanja (broj dijeljenih studenata između kolegija i slično). Pomoćna varijabla phase omogućava da se primjerak ovog razreda koristi ili za djelomično izvođenje petlje (ako je phase=0), ili da se koristi za drugi korak vrednovanja u kojem se prolazi kroz listu predmeta za razmicanje (ako je phase=1).

Koliko će posla biti odrađeno u okviru jednog primjerka razreda EvalJob ovisit će

```
BazenDretvi bd = new BazenDretvi(brojRadnika);

vrednuj(k)
  e = {0,0,0};
  List<EvalJob> poslovi = razloži(k,chunkSize);
  List<Rezultat> rezultati = bd.dodajPosloveURed(poslovi);
  za svaki r iz rezultati
    r_e = r.pričekajIzračunKazne();
    e[1] += r_e[1];
    e[2] += r_e[2];
    e[3] += r_e[3];
  kraj
  vrati e;
kraj
```

Slika 6.2: Pseudokod paralelnog algoritma vrednovanja rješenja za problem izrade rasporeda provjera znanja.

o rasponu zadanih indeksa fromTermIndex i toTermIndex. Ako rješenje sadrži n termina, posao vrednovanja moguće je razložiti na $n + 1$ primjerak razreda EvalJob, gdje svaki primjerak odradi samo jedan zadani termin i $(n + 1)$-vi kontrolu razmicanja. Druga je mogućnost definirati da svaki primjerak odrađuje nekoliko uzastopnih termina; primjerice, ako želimo da svaki primjerak odradi k termina, tada trebamo ukupno $n/k + 1$ primjerak razreda. Broj uzastopnih termina koje će odraditi svaki primjerak zvat ćemo *veličinom posla*, odnosno *chunkSize*.

Vrednovanje se tada može obaviti uporabom danas uobičajeno dostupnih bazena dretvi (engl. *thread-pools*); to su objekti koji imaju red za prihvat poslova te w dretvi koje iz reda paralelno dohvaćaju i obavljaju posao. Paralelno vrednovanje ovakvim mehanizmom prikazano je pseudokodom 6.2.

Paralelna izvedba metode za vrednovanje započinje tako da se temeljem predanog rješenja stvori odgovarajući broj poslova koje je potrebno obaviti. Ti se poslovi potom predaju u red poslova bazena dretvi kod kojeg je broj dretvi određen varijablom brojRadnika. Kao rezultat tog poziva dobiva se lista objekata koji omogućavaju dohvat rezultata predanih poslova. Tome služi upravo metoda pricekajIzracunKazne() koja će vratiti rezultat ako je on dostupan, a zablokirat će ako rezultat još nije izračunat. Jednom kada je rezultat posla poznat, sadržaj njegovog lokalnog spremnika nadodaje se u kumulativni spremnik kazne. Nakon što su prikupljeni svi rezultati, metoda vraća konačni iznos kazne.

Tablica 6.1: Utjecaj paralelizacije na vrijeme utrošeno na vrednovanje rješenja. Vremena su u milisekundama.

Veličina posla	Serijski	1	2	3	4
1	102656.67	110095.67	55605.00	38139.33	31845.00
2	102718.00	107151.67	54933.67	38995.00	31899.33
5	102593.33	105759.67	54113.00	38282.67	33770.33
10	102614.00	104998.67	55012.00	50858.33	50863.33
15	102644.67	104721.67	64184.33	64340.33	64317.00

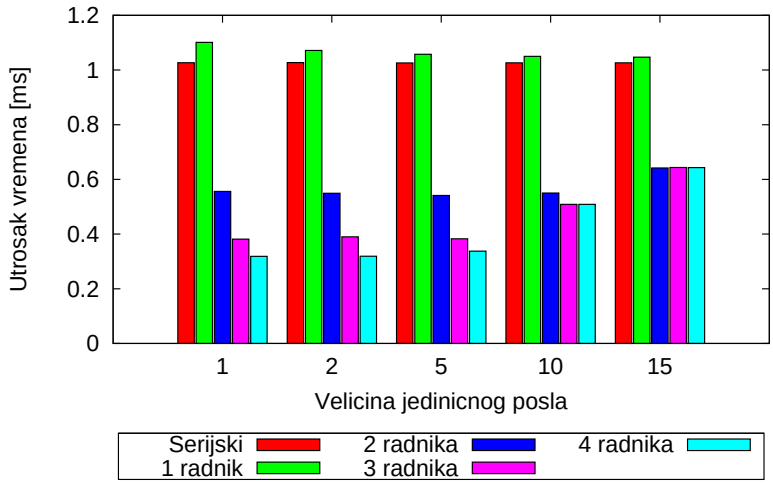
U programskom jeziku Java mogućnosti opisanog bazena dretvi standardno su dostupne uporabom razreda Executors odnosno sučeljaExecutorService. Spomenuti razred te sučelje standardni su dio Javinog paketa "java.util.concurrent" od verzije 1.5. Objekti koji su se ponašaju kao opisani razred Rezultat također su definirani sučeljem Future dostupnog u istom paketu, i upravo se takvi objekti dobivaju kao rezultat slanja poslova u red poslova čime je implementacija opisanog algoritma bitno olakšana.

Korist opisane paralelizacije istražena je na problemu vrednovanja rješenja koje se sastoji od 37 termina. Eksperimenti su provedeni na sljedećim slučajevima:

- serijski algoritam vrednovanja,
- paralelni algoritam vrednovanja uz bazen dretvi s jednim radnikom,
- paralelni algoritam vrednovanja uz bazen dretvi s dva radnika,
- paralelni algoritam vrednovanja uz bazen dretvi s tri radnika te
- paralelni algoritam vrednovanja uz bazen dretvi s četiri radnika.

Pri tome je razmotreno i pet vrijednosti za veličinu posla (tj. parametar *chunkSize*): svaki posao vrednuje samo jedan termin, svaki posao vrednuje dva uzastopna termina, svaki posao vrednuje pet uzastopnih termina, svaki posao vrednuje deset uzastopnih termina te svaki posao vrednuje petnaest uzastopnih termina. Mjereno je ukupno vrijeme vrednovanja 100000 rješenja. Dobiveni rezultati prikazani su u tablici 6.1. Na slici 6.3 prikazani su rezultati normirani na jednu jedinku.

Temeljem ovih rezultata jasno je vidljiv trošak paralelizacije: dovoljno je usporediti serijsku izvedbu algoritma vrednovanja te paralelnu izvedbu algoritma vrednovanja uz jednog radnika, čime se paralelna verzija ponaša poput serijske. Zbog dodatnih troškova oko stvaranja posla, predaje posla u red poslova, dohvata rezultata, akumuliranja



Slika 6.3: Utjecaj paralelizacije na vrijeme utrošeno na vrednovanje rješenja.

rezultata u konačnu kaznu te konačno i troškova sinkronizacije, paralelna verzija algoritma koju izvodi samo jedan radnik konzistentno je sporija u odnosu na serijsku verziju algoritma. Međutim, dodavanjem još jednog radnika može se uočiti ubrzanje od čak 1.97 puta (za slučaj veličine posla 1); dodavanjem još jednog radnika postiže se ukupno ubrzanje od 2.69 dok dodavanjem i četvrtog radnika dolazimo do ukupnog ubrzanja od 3.22 puta.

Povećavanjem veličine posla smanjuje se njihov broj a time i trošak sinkronizacije te trošak akumuliranja rezultata. Tako primjerice uz dva radnika optimalna veličina posla je 5 termina čime je zadatak vrednovanja razložen na $37/5+1 = 9$ poslova. Povećanjem veličine posla na mjeru koja ukupni broj poslova čini sumjerljivim s brojem radnika gube se prednosti paralelizacije. Primjer je vidljiv na slučaju 4 radnika i veličini posla 15; uz tu veličinu posla broj poslova bit će 4. Dva radnika će tada vrednovati po 15 termina, jedan radnik će vrednovati preostalih 7 (i nakon toga čekati) i četvrti radnik će obaviti provjeru razmicanja kolegija i nakon toga će čekati.

Uporabom opisanog pristupa moguće je ubrzati izvođenje evolucijskih algoritama. Pri tome će stvarna ubrzanja biti manja od ubrzanja dobivenih samo paralelizacijom metode vrednovanja jer će algoritam dio vremena trošiti na serijski dio posla u kojem se obavljaju drugi evolucijski operatori poput križanja i mutacije kod genetskog algoritma. Međutim, kako je složenost tih operatora linearna s brojem kolegija a složenost vrednovanja kvadratna, ubrzanja neće biti zanemariva.

Opisanom metodom paralelizacije operatora vrednovanja ujedno je postignuto i ubrzanje operatora lokalne pretrage – naime, operator lokalne pretrage generira niz kandidata i svakog od njih vrednuje kako bi utvrdio je li došlo do poboljšanja; stoga ubrzavanje vrednovanja ima posljedice i na rad drugih operatora.

Opisani pristup paralelizacije ima još jedno dobro svojstvo: kako vrednovanje koriste svi algoritmi evolucijskog računanja, ubrzavanjem operatora vrednovanja automatski je ubrzano izvođenje svih algoritama evolucijskog računanja koji pri tome niti na koji način nisu mijenjani, odnosno koji mogu biti napisani prema osnovnom serijskom pseudokodu.

6.3.2 Paralelizacija na razini populacije

U prethodnom odjeljku pokazano je kako je moguće paralelizirati operator vrednovanja (tj. evaluacije), što može dovesti do ubrzanja. Kako bi se postigla veća ubrzanja, a u skladu s Amdahlovim zakonom, potrebno je što je moguće više smanjiti dio posla koji se izvodi serijski. Jedan način za postizanje ovog cilja je dalje paralelizirati i sve preostale operatore, pri čemu će se ipak malo po malo akumulirati troškovi paralelizacije i rekonstrukcije konačnih rješenja. osim toga, kako će neki od operatora ionako biti relativno niske složenosti, povećava se rizik da će troškovi paralelizacije premašiti eventualni dobitak paralelnog izvođenja te da će u konačnici postupak trajati dulje.

Stoga se umjesto paralelizacije svakog pojedinkog operatora predlaže drugi pristup koji je upravo prirodan za populacijske algoritme evolucijskog računanja. Radi se o paralelizaciji na razini populacije, gdje će jedan posao uključivati čitav slijed primjene evolucijskih operatora počev od selekcije roditelja pa do stvaranja i vrednovanja rješenja. Time će zasigurno ukupno trajanje takvog posla bitno nadmašivati troškove sinkronizacije i druge troškove paralelizacije pa se očekuju bolji iznosi ubrzanja.

Sva razmatranja u nastavku provedena su uz pretpostavku da je jednom stvoreno rješenje (jedinka, kromosom, čestica, vektor, antitijelo – ovisno o algoritmu koji se razmatra) nepromijenjivo u spremniku računala (engl. *immutable*); svaka modifikacija nad rješenjem zapravo će u spremniku stvoriti novo rješenje koje ima navedenu modifikaciju. Populacija će tada biti niz referenci na rješenja.

6.3.2.1 Paralelizacija eliminacijskog genetskog algoritma

Za potrebe paralelizacije promotrimo izvedbu eliminacijskog genetskog algoritma prikazanu pseudokodom 6.4.

```
Jedinka[] populacija;
Jedinka najbolja;

korak()
  ponavljaj za i = 1 do vel_pop
    indeksi=slucajno_odaberi_tri_indeksa(vel_pop);
    sortiraj(populacija, indeksi);
    dijete = krizaj_i_mutiraj(
      populacija[indeksi[0]],
      populacija[indeksi[1]]
    );
    vrednuj(dijete);
    lokalno_pretrazi(dijete);
    ako je bolje(dijete, najbolja) tada
      najbolja = dijete;
    kraj
    populacija[indeksi[2]] = dijete;
  kraj
kraj

Jedinka optimiraj()
  populacija = stvoriPocetnuPopulaciju(vel_pop);
  dok nije uvjet zaustavljanja ispunjen
    korak();
  kraj
  vrati najbolja;
kraj
```

Slika 6.4: Pseudokod slijedne izvedbe eliminacijskog genetskog algoritma.

Prikazana verzija algoritma razlikuje se od osnovnog kanonskog oblika eliminacijskog genetskog algoritma jer kao minimalnu količinu posla obavlja upravo onoliko stvaranja djece kolika je veličina populacije, čime algoritam postaje pseudo-generacijski. Taj je zadatak definiran u metodi korak(). Metoda optimiraj() potom ponavlja ovaj osnovni korak sve dok se ne zadovolji uvjet zaustavljanja. Korištena metoda sortiraj sortira predano polje indeksa prema dobroti jedinki čiji su to indeksi u populaciji, tako da na prvo mjesto stavi indeks najbolje jedinke. Križanjem se stvara nova jedinka koja se potom još mutira i lokalno pretražuje; nakon ubacivanja u populaciju jedinka postaje

nepromjenjiva. Polje populacija koje se ovdje koristi predstavlja polje referenci na jedinke.

Pristup paralelizaciji ove izvedbe algoritma koji se predlaže je sljedeći: paralelizira se cijela metoda korak() na način da se definira vel_pop poslova koji se ubace u red poslova bazena dretvi koji će potom te poslove obavljati paralelno uporabom raspoloživog broja radnika. U tom slučaju populacija postaje dijeljeni resurs iz kojeg radnici čitaju i zapisuju jedinke. Stoga je pristup populaciji potrebno zaštititi sinkronizacijskim mehanizmom. Na ovaj način paralelizirana verzija prikazana je pseudokodom 6.5.

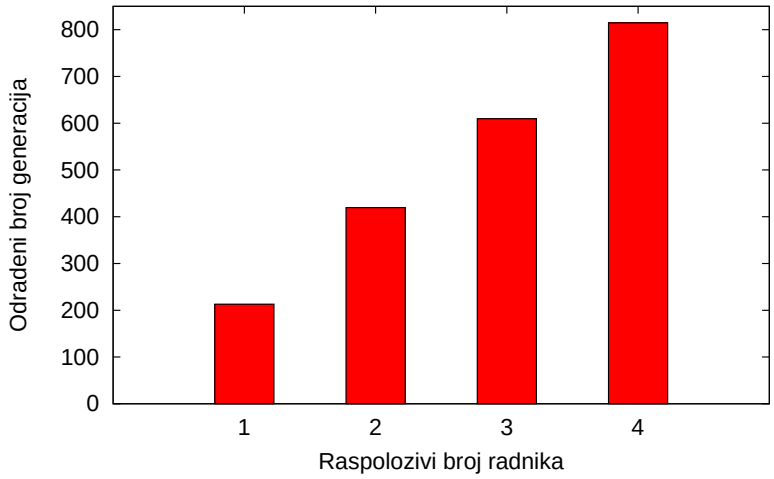
Elementarni posao koji radnici obavljaju paralelno opisan je metodom posao(). U okviru tog posla slučajno se odabiru tri različita indeksa te se odgovarajuće jedinke dohvaćaju iz populacije. Radi se sortiranje i dvije bolje jedinke stvaraju novo dijete. Stvoreno dijete potom se ubacuje u populaciju. Pri tome su svi korišteni operatori implementirani na uobičajen slijedni način.

Tijekom čitavog posla radnik prolazi kroz samo dvije sinkronizacijske točke: metodu dohvati_jedinke i metodu ubaci_jedinku koje obje traju izuzetno kratko. Metoda dohvati_jedinke populaciju će zaključati kako bi iskopirala reference na jedinke čiji su indeksi predani kao argument. Kako se jednike nakon ubacivanja u populaciju više ne mijenjaju, nema potrebe za dodatnim zaključavanjem jedinki. Metoda ubaci_jedinku napisana je uvažavajući činjenicu da je nakon dohvata reference najgore jedinke neki drugi radnik na to mjesto mogao staviti bolju jedinku. Stoga metoda prima tri argumenta: poziciju jedinke, očitanu staru referencu na jedinku te referencu na novu jedinku. Nakon zaključavanja populacije, najprije se provjerava je li populacija promijenjena na način da je na zadanoj poziciji nova jedinka koja je bolja od jedinke koju želimo postaviti na to mjesto. Ako je to slučaj, ubacivanje se preskače. U suprotnom, nova se jedinka ubacuje u populaciju i po potrebi se ažurira referenca na najbolje pronađeno rješenje.

Odgovor na pitanje je li potrebno provjeravati da se ne ažuriranjem ne izgubi potencijalno bolje rješenje nije jednoznačan. Moguće je pronaći i argumente za provođenje takvih provjera kao i argumente protiv provođenja takvih provjera. U konačnici, potrebno je odabrati jedan od pristupa. Radi li se ova provjera, povezana kazna je vrlo mala jer se usporedba svodi na usporedbu najviše tri para cijelih brojeva. Na pitanje je li potrebno čitanje referenci i pisanje referenci obavljati pod sinkronizacijskim mehanizmima odgovor je jasan: to je svakako potrebno jer čitanje i pisanje referenci nije atomarna operacija.

```
Jedinka[] populacija;
Jedinka najbolja;
BazenDretvi bd;

korak()
  ubaci vel_pop poslova u red poslova
  pričekaj da svi poslovi budu odrađeni
kraj

posao()
    indeksi=slucajno_odaberi_tri_indeksa(vel_pop);
    {j1,j2,j3} = dohvati_jedinke(indeksi);
    sortiraj({j1,j2,j3}, indeksi);
    dijete = krizaj_i_mutiraj(j1,j2);
    vrednuj(dijete);
    lokalno_pretrazi(dijete);
    ubaci_jedinku(j3,i3,dijete);
kraj

jedinke dohvati_jedinke(indeksi)
  zakljucaj_populaciju();
  jedinke = preuzmi_iz_populacije(indeksi);
  otkljucaj_populaciju();
  vrati jedinke;
kraj

ubaci_jedinku(stara_jedinka, indeks_jedinke, nova_jedinka)
  zakljucaj_populaciju();
  ako stara_jedinka != populacija[indeks_jedinke] &&
    populacija[indeks_jedinke] je bolja od nova_jedinka tada
      otkljucaj_populaciju();
      povratak;
  kraj
  populacija[indeks_jedinke] = nova_jedinka;
  ako je bolje(nova_jedinka, najbolja) tada
    najbolja = nova_jedinka;
  kraj
  otkljucaj_populaciju();
kraj

Jedinka optimiraj()
  populacija = stvoriPocetnuPopulaciju(vel_pop);
  dok nije uvjet zaustavljanja ispunjen
    korak();
  kraj
  vrati najbolja;
kraj
```

Slika 6.5: Pseudokod paralelne izvedbe eliminacijskog genetskog algoritma.



Slika 6.6: Utjecaj paralelizacije na odrađeni broj generacija kod genetskog algoritma.

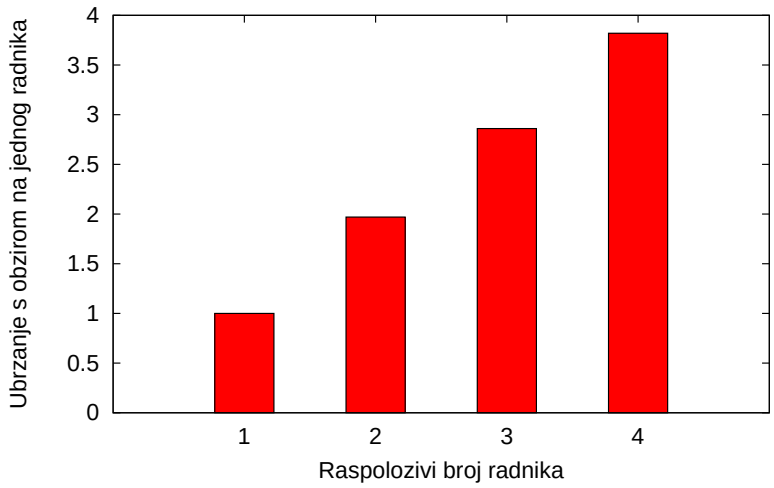
Kako bi se provjerila ostvarena ubrzanja, ovaj pristup paralelizacije primijenjen je na eliminacijski genetski algoritam koji rješava problem izrade rasporeda provjera znanja. Potom su načinjeni eksperimenti u kojima se je mjerilo koliko generacija uspjeva odraditi takav genetski algoritam uz fiksirano vrijeme izvođenja od 20 sekundi a uz različit broj radnika. Eksperiment je rađen s populacijom od 48 jedinki. Pojam generacija u ovom kontekstu se odnosi na stvaranje upravo 48 novih jedinki. Ispitano je ponašanje algoritma za jednog, dva, tri i četiri radnika. Rezultati su prikazani u tablici 6.2 te na slikama 6.6 i 6.7.

Iz rezultata je vidljivo da opisan način paralelizacije može bitno ubrzati problem izrade rasporeda te omogućiti da se u istom vremenu dođe do kvalitetnijih rješenja.

Paralelizacija generacijskog genetskog algoritma izvodi se na sličan način, pri čemu posao obuhvaća sve operacije od selekcije do vrednovanja. Kako bi se smanjio broj sinkronizacija, pronalazak najbolje jedinke moguće je načiniti kada svi radnici završe

Tablica 6.2: Utjecaj paralelizacije na odrađeni broj generacija kod genetskog algoritma primijenjenog na problem izrade rasporeda provjera znanja uz fiksno vrijeme izvođenja.

Broj radnika	Broj odrađenih generacija	Ubrzanje
1	213.0	1.00
2	419.5	1.97
3	609.6	2.86
4	814.6	3.82



Slika 6.7: Postignuto ubrzanje uslijed paralelizacije kod genetskog algoritma.

u dijelu koda koji se obavlja serijski. Kako se kod generacijskog genetskog algoritma iz jedne populacije stvara nova druga, stara populacija više nije dijeljeni resurs koji se istovremeno čita i modificira pa je uporaba sinkronizacijskih mehanizama kod čitanja jedinki populacije također nepotrebna čime je ovaj oblik genetskog algoritma još prikladniji za ovakav oblik paralelizacije.

6.3.2.2 Paralelizacija algoritma Max-Min mravlji sustav

Paralelizacija algoritma *Max-Min* mravlji sustav provest će se temeljem pseudokoda 6.8.

Slično kao kod paralelizacije eliminacijskog genetskog algoritma, paralelizacija se primijenjuje na metodu korak(). Ta se metoda sastoji od dva dijela: najprije se prolazi kroz petlju u kojoj svaki mrav stvara svoj prijedlog rješenja optimizacijskog problema. Nakon što su svi mravi stvorili prijedlog rješenja, pronalazi se najbolje rješenje u toj iteraciji. Po potrebi se ažuriraju najbolje ikada pronađeno rješenje te najbolje rješenje trenutne epohe (epoha je period od zadnjeg resetiranja feromonskih tragova). Potom slijedi isparavanje feromonskih tragova, deponiranje novih feromona te po potrebi resetiranje feromonskih tragova ako je detektirana stagnacija.

U metodi korak() stoga postoje dva kandidata za paralelizaciju:

- paralelizacija posla pojedinih mrava (prva petlja) te
- paralelizacija drugog dijela metode (isparavanje, deponiranje novih feromona).

```
Jedinka[] populacija;
Jedinka najboljaUIteraciji;
Jedinka najboljaIkada;
Jedinka najboljaUEpohi;
brojacStagnacije = 0;
limitStagnacije = 4;
faktor = 1;

korak()
  brojacStagnacije++;
  ponavljaj za i = 1 do broj_mrava
    Jedinka r = stvori_rjesenje();
    vrednuj(r);
    lokalno_pretrazi(r);
    populacija[i] = r;
    ako je bolje(r, najboljaUIteraciji) tada
      najboljaUIteraciji = r;
    kraj
  kraj
  ako bolje(najboljaUIteraciji, najboljaIkada) tada
      najboljaIkada = najboljaUIteraciji;
      faktor = 1;
  kraj
  ako bolje(najboljaUIteraciji, najboljaUEpohi) tada
      najboljaUEpohi = najboljaUIteraciji;
      brojacStagnacije=0;
  kraj
  azuriraj taumax, taumin
  ispari feromonske tragove
  deponiraj nove feromone
  ako je brojacStagnacije>faktor*limitStagnacije tada
    resetiraj feromonske tragove;
    najboljaUEpohi = null;
    faktor = faktor*2;
  kraj
kraj

Jedinka optimiraj()
  populacija = stvoriPocetnuPopulaciju(vel_pop);
  dok nije uvjet zaustavljanja ispunjen
    korak();
  kraj
  vrati najboljaIkada;
kraj
```

Slika 6.8: Pseudokod slijedne izvedbe algoritma Max-Min mravlji sustav.

S obzirom na složenost i trajanje izvođenja posla pojedinih mrava pažnja će se posvetiti upravo paralelizaciji tog dijela algoritma. Kako su mjerenja pokazala, utjecaj ovog drugog dijela posla koji se izvodi slijedno doista nije značajan.

Temeljem ovog razmatranja, predložena je paralelna izvedba algoritma prikazana pseudokodom 6.9. Metoda optimiraj jednaka je metodi optimiraj prikazanoj u pseudokodu 6.8.

Posao mrava koji se obavlja paralelno za pojedine mrave opisan je metodom posao() i sastoji se od konstrukcije rješenja, vrednovanja rješenja, obavljanja lokalne pretrage te ažuriranja najboljeg rješenja iteracije. Konstrukcija rješenja kod problema izrade rasporeda provjera znanja je zahtjevan posao jer se heurističke informacije grade istovremeno s izgradnjom rješenja. Vrednovanje rješenja te lokalna pretraga vremenski su zahtjevne iz već objašnjenih razloga. Posljedica ovoga je vrlo veliki omjer vremena potrošenog na posao jednog mrava te vremena utrošenog na provođenje drugog koraka algoritma koje uključuje isparavanje i deponiranje novih feromona.

Iz tog razloga očekuje se da će paralelizacija postizati vrlo visoka ubrzanja.

Kako bi se provjerila opravdanost ovih zapažanja, ovaj pristup paralelizacije primijenjen je na algoritam Max-Min mravlji sustav koji rješava problem izrade rasporeda provjera znanja. Načinjeni su eksperimenti u kojima se je mjerilo koliko generacija uspjeva odraditi takav mravlji algoritam uz fiksirano vrijeme izvođenja od 20 sekundi a uz različit broj radnika. Pojam jedna generacija ovdje se odnosi na čitav ciklus od toga da svaki mrav kolonije stvori primjerak rješenja pa do završnog deponiranja feromonskih tragova. Eksperiment je rađen s populacijom od 48 mrava. Ispitano je ponašanje algoritma za jednog, dva, tri i četiri radnika. Rezultati su prikazani u tablici 6.3 te na slikama 6.10 i 6.11.

Iz rezultata je vidljivo da opisan način paralelizacije može bitno ubrzati problem izrade rasporeda te omogućiti da se u istom vremenu dođe do kvalitetnijih rješenja.

6.3.2.3 Paralelizacije drugih algoritama

Temeljem prethodno opisanih načela moguće je paralelizirati i druge algoritme evolucijskog računanja.

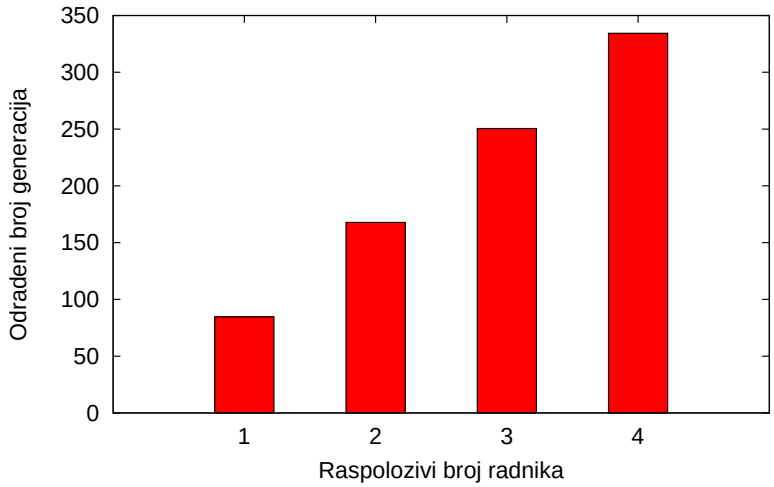
Algoritam roja čestica, bilo s globalnim susjedstvom ili s lokalnim susjedstvom tipični je primjer generacijskog algoritma. Stoga se paralelno može obavljati posao koji

```
Jedinka[] populacija;
Jedinka najboljaUIteraciji;
Jedinka najboljaIkada;
Jedinka najboljaUEpohi;
brojacStagnacije = 0;
limitStagnacije = 4;
faktor = 1;

posao()
  Jedinka r = stvori_rjesenje();
  vrednuj(r);
  lokalno_pretrazi(r);
  populacija[i] = r;
  azuriraj_najbolju(r);
kraj

azuriraj_najbolju(r)
  udi_u_kriticni_odsjecak
  ako je bolje(r, najboljaUIteraciji) tada
    najboljaUIteraciji = r;
  kraj
  izadi_iz_kriticnog_odsjecka
kraj

korak()
  brojacStagnacije++;
  ubaci broj_mrava poslova u red poslova
  pričekaj da svi poslovi budu odrađeni
  ako bolje(najboljaUIteraciji, najboljaIkada) tada
      najboljaIkada = najboljaUIteraciji;
      faktor = 1;
  kraj
  ako bolje(najboljaUIteraciji, najboljaUEpohi) tada
      najboljaUEpohi = najboljaUIteraciji;
      brojacStagnacije=0;
  kraj
  azuriraj taumax, taumin
  ispari feromonske tragove
  deponiraj nove feromone
  ako je brojacStagnacije>faktor*limitStagnacije tada
    resetiraj feromonske tragove;
    najboljaUEpohi = null;
    faktor = faktor*2;
  kraj
kraj
```

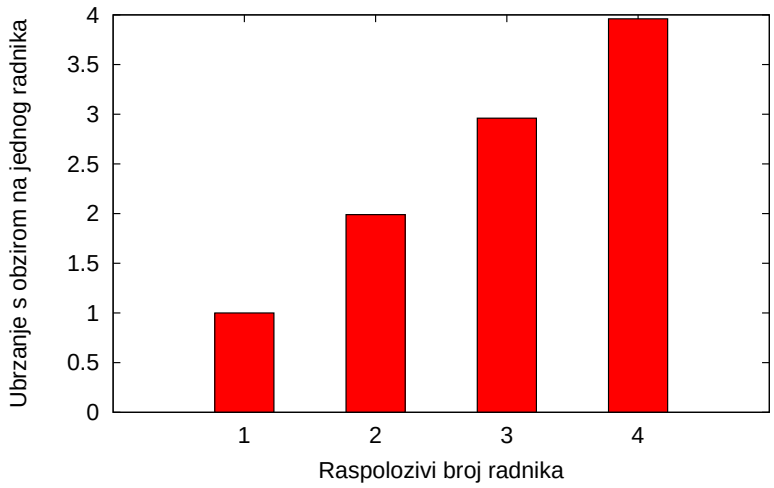
Slika 6.9: Pseudokod paralelne izvedbe algoritma Max-Min mravlji sustav.

Tablica 6.3: Utjecaj paralelizacije na odrađeni broj generacija kod algoritma Max-Min mravlji sustav primijenjenog na problem izrade rasporeda provjera znanja uz fiksno vrijeme izvođenja.

Broj radnika	Broj odrađenih generacija	Ubrzanje
1	84.5	1.00
2	167.8	1.99
3	250.5	2.96
4	334.3	3.96



Slika 6.10: Utjecaj paralelizacije na odrađeni broj generacija kod algoritma Max-Min mravlji sustav.



Slika 6.11: Postignuto ubrzanje uslijed paralelizacije kod algoritma Max-Min mravlji sustav.

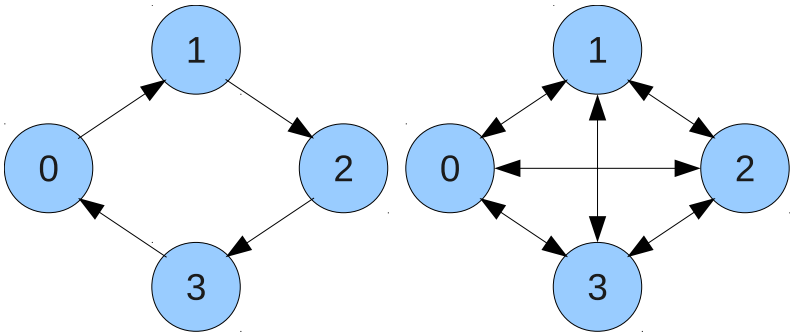
uključuje pronalazak najbolje jedinke lokalnog susjedstva (što se radi za svaku česticu ako algoritam koristi lokalno susjedstvo), izračun nove brzine, izračun nove pozicije te eventualna lokalna pretraga. Nakon što se temeljem postojeće populacije čestica stvori nova populacija čestica, nova populacija postaje trenutna. Ovakav pristup omogućava da se čitanje i stvaranje novih čestica u potpunosti radi bez sinkronizacije. Sinkronizacijski su mehanizmi potrebni samo kako bi se utvrdilo jesu li svi poslovi stvaranja nove populacije završili. Tada se u slijednom dijelu koda može potražiti eventualno novo najbolje rješenje i potom krenuti u novi krug.

Imunološki algoritmi poput jednostavnog imunološkog algoritma te algoritma klonske selekcije također su populacijski algoritmi pa je princip paralelizacije jednak. Razliku u odnosu na prethodne algoritme čini potreba za sortiranjem velikog broja jedinki što može biti također izvor usporenja rada (je li to značajan trošak ili ne ovisit će o vremenskoj zahtjevnosti izvođenja kloniranja, hipermutacije te vrednovanja). Najjednostavniji pristup koristit će paralelizaciju na razini posla koji će se sastojati od stvaranja, mutiranja i vrednovanja populacije hiperklonova te eventualne lokalne pretrage, dok će se sortiranje i konačni odabir jedinki za novu populaciju raditi u slijednom dijelu koda. Ako se pokaže da je upravo sortiranje usporedive vremenske zahtjevnosti kao i predhodni posao, može se razmisliti i o uporabi paralelnih algoritama sortiranja.

Konačno, algoritam diferencijske evolucije također je primjer generacijskog algoritma kod kojeg se paralelizacija provodi na prethodno opisani način. Temeljem postojeće populacije stvara se nova populacija pri čemu se čitav posao stvaranja kompletne nove jedinke može raditi paralelno; u slijednom dijelu koda tada se samo ažurira trenutno najbolje rješenje i provjerava uvijet zaustavljanja.

6.3.3 Paralelizacija na razini algoritama

Osim do sada prikazanih načina paralelizacije algoritama evolucijskog računanja moguće je posegnuti za još jednim oblikom koji zahtjeva minimalnu uporabu sinkronizacijskih mehanizama – radi se o paralelizaciji na razini algoritama. Kod ovog oblika paralelizacije zapravo se više ne bi smjelo govoriti o paralelizaciji slijednog algoritma. Naime, svi prethodno opisani primjeri paralelizacije svodili su se na pokušaj da se slijedni algoritam na neki način ubrza kako bi u konačnici u istom vremenu mogao proći kroz veći broj generacija i time doći do kvalitetnijeg rješenja. Paralelizacija o kojoj se ovdje govori



(a) Prsten (b) Potpuno povezana

Slika 6.12: Primjeri topologija razmjene informacija između algoritama.

mijenja dinamiku zbivanja u populaciji i algoritam koji tako nastaje ponaša se drugačije od paraleliziranog algoritma (paraleliziranog na prethodno opisane načine).

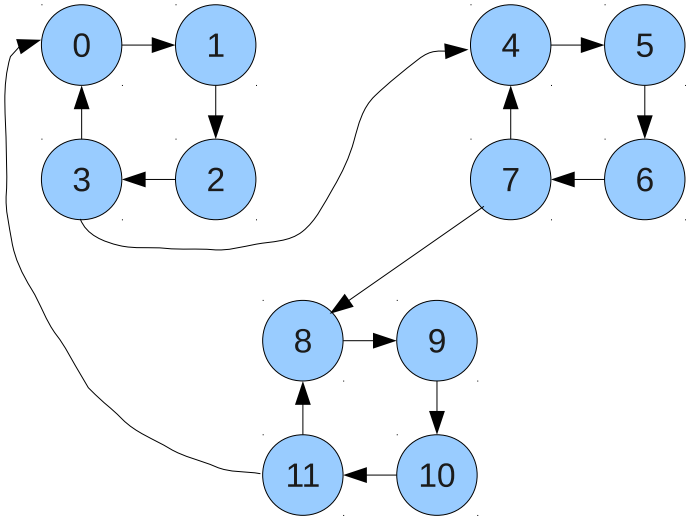
Primjerice, kada se govori o paralelizaciji na razini genetskih algoritama, tada se podrazumijeva uporaba više primjeraka genetskih algoritama koji rade paralelno i koji u određenim trenutcima razmjenjuju informacije (što razmjenjuju, kada razmjenjuju, s kime razmjenjuju i kako koriste te informacije parametri su ovog oblika paralelizacije). U literaturi je ovakav model paralelizacije poznat pod nazivom *otočni model*. Dva primjera ovakve organizacije prikazana su na slici 6.12.

Slika 6.12a prikazuje topologiju *prstena* u kojoj svaki od algoritama ima samo jednog susjeda i informacije šalje isključivo njemu. Definirano je i susjedstvo između zadnjeg algoritma i prvog kako bi se osiguralo konstantno kruženje informacija. Komunikacija u prstenima uobičajeno je jednosmjerna. Na slici 6.12a algoritam 0 šalje informacije algoritmu 1, algoritam 1 algoritmu 2, algoritam 2 algoritmu 3 te algoritam 3 algoritmu 0.

Slika 6.12b prikazuje *potpuno-povezanu* topologiju u kojoj je svatko susjed svakome. U toj topologiji svaki od algoritama informacije razmjenjuje sa svim drugim algoritmima što ovisno o problemu koji se rješava može predstavljati značajan komunikacijski trošak.

Osim navedenih topologija postoji još niz drugih poput hijerarhijske strukture (primjerice n-arno stablo gdje protok informacija ide od listova prema korijenu), slučajno stvorene topologije gdje se susjedstvo definira posredstvom slučajnog mehanizma, topologije slične prstenu ali kod kojih svaka jedinka može primati/slati informacije nekolicini najbližih susjeda i slično.

Za ostvarivanje ovakvog algoritma nije nužna paralelizacija. Naime, ovakav se sustav



Slika 6.13: Topologija ugniježđenih prstena.

može emulirati i na jednom procesoru, pri čemu se svaki od algoritama izvodi određeno vrijeme, i potom se obavi razmjena poruka; nakon toga opet se svaki od algoritama izvodi neko vrijeme i opet se obavi razmjena poruka. Razlog uvođenja ovakvih topologija proizlazi iz različitog ponašanja jednog algoritma s populacijom od $4n$ jedinki i četiri algoritma svaki s populacijom od n jedinki koji međusobno povremeno razmjenjuju rješenja. Naime, od topologije poput *prstena* očekuje se da će se zbog izolacije algoritama u svakom od njih omogućiti lokalno istraživanje drugog dijela prostora pretraživanja pa će se tek kroz periodičku razmjenu informacija na kraju doći do konačnog rješenja. Jedan algoritam s velikom populacijom, ako koristi agresivan selekcijski pritisak može unatoč velikoj populaciji relativno brzo konvergirati ka nekom rješenju koje ne mora nužno biti dobro.

Ako je na raspolaganju više radnika, ovakav se algoritam može prirodno paralelizirati tako da se definira onoliko poslova koliko ima algoritama. Primjerice, na dvoprocesorskom računalu jedan procesor može izvoditi algoritam 0 pa algoritam 1 dok drugi procesor može izvoditi algoritam 2 pa algoritam 3. Kada se izvede zadani broj iteracija, slijedni dio koda obavlja razmjenu informacija i postupak se ponavlja. Kako bi procesori bili dobro iskorišteni, broj iteracija treba biti tako podešen da izvođenje svakog od algoritama traje poprilici jednako dugo.

Dio eksperimenata opisanih u nastavku rađen je s topologijom ugniježđenih prstena koja je prikazana na slici 6.13. Ukupan broj primjeraka algoritama je 12. Definirana su

tri manja prstena (0,1,2,3), (4,5,6,7) i (8,9,10,11). Potom su oni povezani u veći prsten i to na način da zadnji algoritam jednog prstena informacije šalje prvom algoritmu sljedećeg prstena. Tako su dodane veze (3,4), (7,8) i (11,0). Unutar malih prstena informacije se razmjenjuju češće u odnosu na razmjenu kroz veći prsten. U eksperimentima koji su provedeni korišten je omjer 4:1 što znači da su informacije slane kroz veliki prsten tek nakon što obavljen jedan čitav ciklus razmjene unutar manjih prstena.

Je li ovakav pristup uspješan u rješavanju problema raspoređivanja ispitano je na problemu razmještaja 560 studenta u jednu od 6 mogućih grupa pri čemu je svaka grupa imala 2 termina predavanja i kapacitet od 95 studenata. Zadatak je rasporediti studente tako da nemaju konflikata s postojećim zauzećima i da imaju što je moguće bolju kvalitetu ukupnog rasporeda.

Eksperimenti su načinjeni s 12 turnirskih genetskih algoritama, s 12 algoritama Max-Min mravlji sustav, s 12 algoritama roja čestica te s 12 jednostavnih imunoloških algoritama. Za svaki od njih rezultati su prikazani na slikama 6.14, 6.15, 6.16 i 6.17. Svi eksperimenti provedeni su na jednoprocesorskom računalu. Broj iteracija svakog algoritma bio je tako podešen da izvođenje algoritma traje 125 ms. Time je simulacija jedne epohe trajala $12 \cdot 125ms = 1,5s$ nakon čega je slijedila razmjena informacija. Svaku četvrtu epohu informacije su proslijeđene i kroz veliki prsten. U svim eksperimentima kao informacija se je slalo najbolje rješenje koje algoritam ima. Svi algoritmi koristili su populacije od samo 15 jedinki; međutim, kako je sudjelovalo 12 primjeraka algoritama, ukupni broj jedinki zapravo je bio $12 \cdot 15 = 180$. Ugradnja dobivenih rješenja riješena je kako slijedi.

- U turnirskom genetskom algoritmu dobiveno rješenje zamijenilo je neku slučajno odabranu jedinku; izuzetak je najbolje rješenje koje je bilo zaštićeno od izbacivanja kako bi se osigurao elitizam.

- U algoritmu *Max-Min* mravlji sustav dobiveno rješenje, ako je bilo bolje od trenutno najboljeg rješenja algoritma, prihvaćeno je kao najbolje; u suprotnom, rješenje je ignorirano. Nije rađeno ažuriranje feromonskih tragova.

- U algoritmu roja čestica dobiveno rješenje zamijenilo je nasumično odabranu česticu. Pri tome, ako je to rješenje bolje od najboljeg rješenja iz memorije te čestice, ono je zapamćeno u memoriji čestice; ako nije, prethodno zapamćeno najbolje rje-

šenje nije mijenjano čime ugradnja ovog rješenja ima efekt dislociranja čestice na tu lokaciju.

- U jednostavnom imunološkom algoritmu rješenje je ubačeno u populaciju na slučajnu lokaciju pri čemu je najbolje rješenje algoritma bilo zaštićeno od izbacivanja ako je bilo bolje od pristiglog rješenja.
- U algoritmu klonske selekcije rješenje je ubačeno u populaciju ako nije bilo gore od najgoreg trenutnog rješenja populacije (naime, algoritam klonske selekcije održava sortiranu populaciju pa je ovo jednostavno provjeriti).

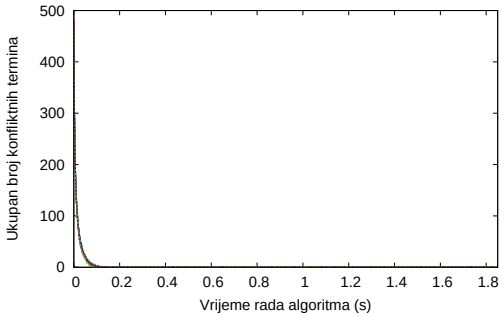
Za svaki algoritam smješten u prikazanu topologiju eksperiment je ponovljen 30 puta u trajanju od 22 minute, što odgovara vremenu od 1 minute i 50 sekundi za svaki primjerak algoritma. Iz prikazanoga je vidljivo da su svi algoritmi uspješno riješili problem. Broj konflikata koji nije moguće razriješiti je 0, i to je upravo broj konflikata do kojeg je svaki od algoritama uspio doći.

Rezultati su prikazani na slikama 6.14, 6.15, 6.16 i 6.17, i to redom za genetski algoritam, algoritam roja čestica, algoritam Max-Min mravlji sustav te jednostavan imunološki algoritam.

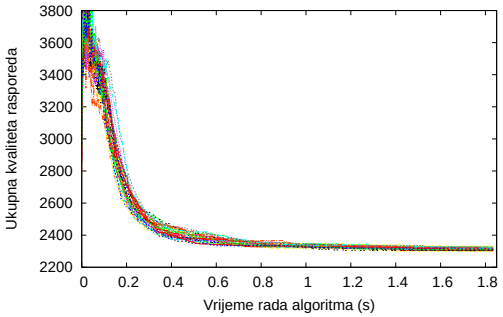
Ovaj model paralelizacije prikladan je i za izvođenje na više računala. Primjerice, ako je na raspolaganju 12 računala, svako računalo može čitavo vrijeme izvoditi jedan primjerak algoritma a razmjena informacija može biti ostvarena komunikacijom preko mreže. U takvom scenariju uobičajeno je odustati od zahtjeva sinkronosti izvođenja. Umjesto toga, algoritmi rade čitavo vrijeme, periodički šalju rješenja susjedima definiranim toplogijom, povremeno provjeravaju ima li novih rješenja i ako ima, ugrađuju ih. Ovakav pristup paralelizaciji iskorišten je u sljedećem odjeljku.

6.4 Hibridni paralelni evolucijski algoritam

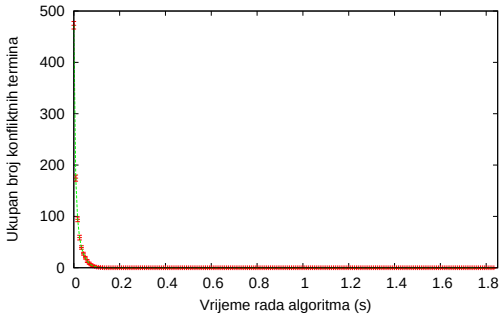
U ovom poglavlju do sada je prikazano nekoliko načina paralelizacije koji su ispitani s evolucijskim algoritmima računanja primijenjenim na problem raspoređivanja nastavnih obaveza. Od prikazanih načina paralelizacije posebno je interesantan pristup izložen u podpoglavlju 6.3.3 – otočni model. Naime, takav pristup izvorno je razvijen kako bi populaciju jednog algoritma podijelio u više manjih populacija koje bi se kroz određene vremenske periode nezavisno evoluirale i koje bi potom razmjenjivale informacije



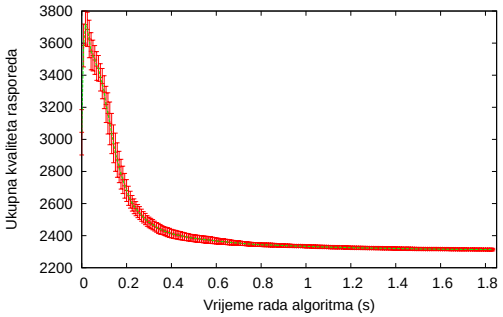
(a) Konflikti



(b) Kvaliteta

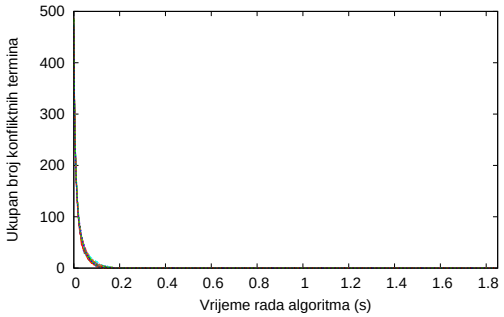


(c) Prosjek – konflikti

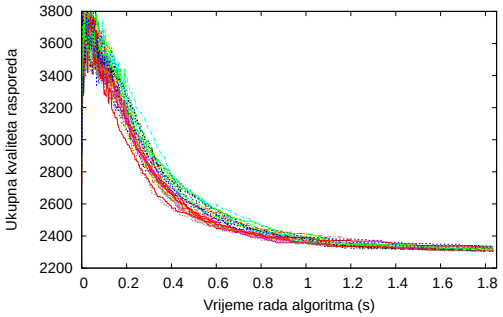


(d) Prosjek – kvaliteta

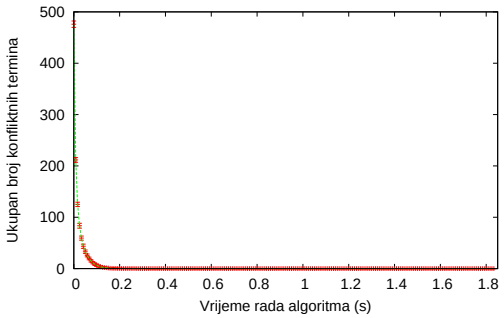
Slika 6.14: Rezultati pokretanja topologije s 12 genetskih algoritama.



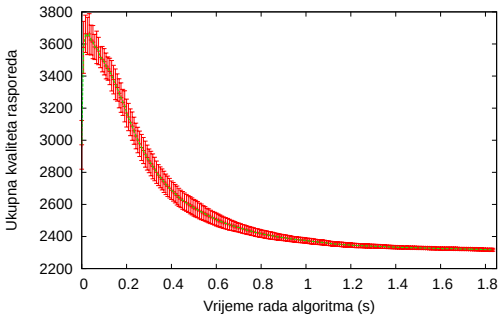
(a) Konflikti



(b) Kvaliteta

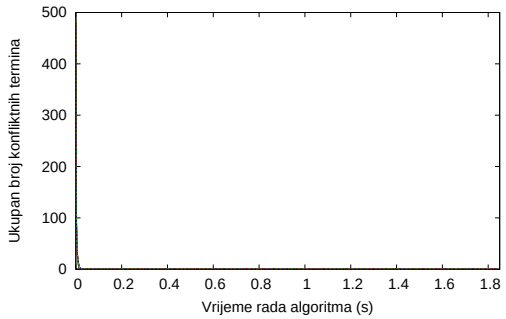


(c) Prosjek – konflikti

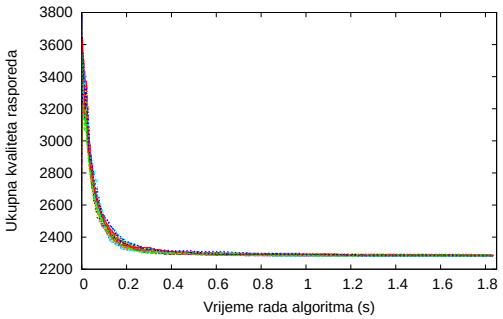


(d) Prosjek – kvaliteta

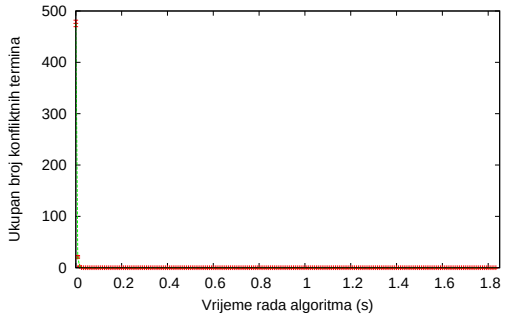
Slika 6.15: Rezultati pokretanja topologije s 12 algoritama roja čestica.



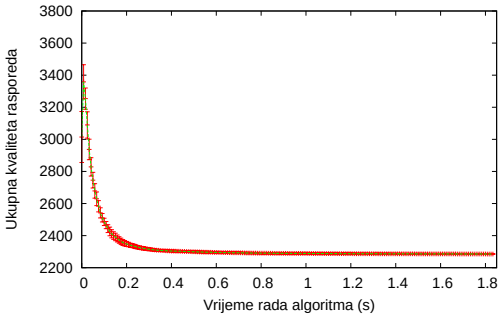
(a) Konflikti



(b) Kvaliteta

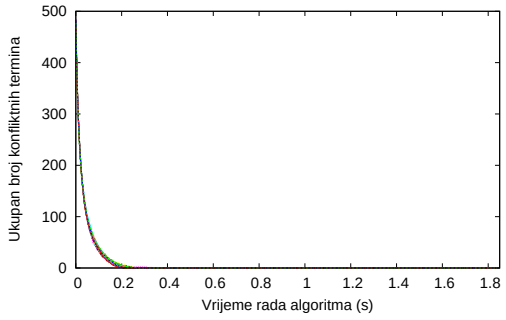


(c) Prosjek – konflikti

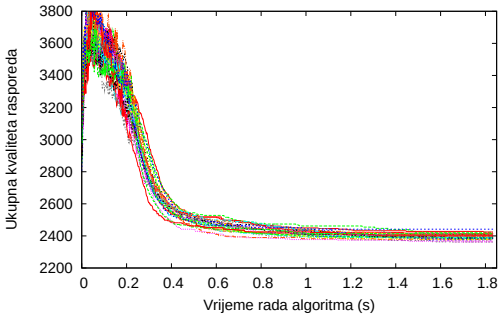


(d) Prosjek – kvaliteta

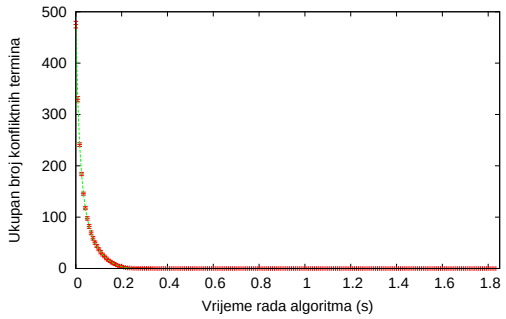
Slika 6.16: Rezultati pokretanja topologije s 12 algoritama Max-Min mravlji sustav.



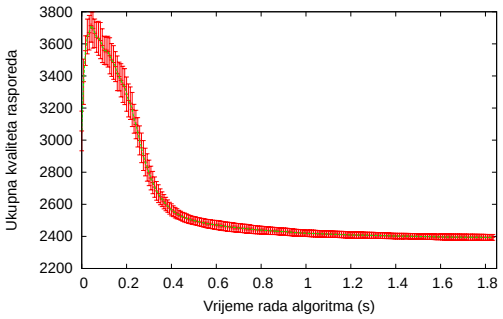
(a) Konflikti



(b) Kvaliteta



(c) Prosjek – konflikti



(d) Prosjek – kvaliteta

Slika 6.17: Rezultati pokretanja topologije s 12 algoritama jednostavnog imunološkog sustava.

(najčešće u obliku boljih ili čak samo najboljih rješenja). Ovakav pristup ne zahtjeva nužno paralelizam; kako je objašnjeno u podpoglavlju 6.3.3, čitav se sustav može izvoditi na jednom procesoru. Međutim, takav pristup nudi mogućnost jeftine paralelizacije. *Jeftine* u smislu da se svakom primjerku algoritma može dodijeliti zaseban procesor što obuhvaća i situaciju u kojoj se koriste procesori više računala pri čemu se komunikacija ostvaruje putem lokalne mreže. Time je moguće dobiti uistinu paralelno izvođenje algoritama pri čemu se bilo kakvi troškovi sinkronizacije mogu svesti na zanemarivi minimum. Potencijalni problem koji postoji kod ovakvog modela raspodijeljenog izvođenja je propusnost komunikacijskog kanala; ovisno o broju poruka koje se šalju te njihovoj veličini, komunikacijski kanal može postati usko grlo.

Prednosti koje se postižu ovako raspodijeljenim načinom izvođenja su sljedeće:

- moguće je raditi s većim populacijama,
- moguće je pretražiti veći prostor stanja te
- moguće je doći do kvalitetnijih rješenja.

Kako bi se ovakav model paralelizacije mogao ispitati, razvijen je sustav za raspodijeljeno izvođenje algoritama evolucijskog računanja [Komar et al., 2011, Komar, 2011, Grbić, 2011] uporabom programskog jezika Java. Razvijeni sustav preuzima brigu o tehničkim detaljima vezanim uz distribuciju algoritama, o definiciji topologije za razmjenu informacija, o trenutcima u kojima će se razmjenjivati informacije te o prijenosu tih informacija. Stoga se korisnik sustava može posvetiti razvoju algoritama koji trebaju prema sustavu za distribuciju trebaju ponuditi propisano sučelje. Najvažnije metode tog sučelja su:

- *dohvatiRješenje()* čiji je zadatak dohvatiti jedno ili više rješenja iz trenutne populacije koja će se dalje proslijediti drugim algoritmima kako je to definirano topologijom te
- *ugradiRjesenje(Skup<Rjesenje> r)* čiji je zadatak na neki način iskoristiti dobivena rješenja.

Algoritmi pri tome nemaju nikakvog utjecaja na trenutak kada će te metode biti pozvane – ti se detalji definiraju na razini sustava za raspodijeljeno izvođenje prilikom definiranja topologije razmjene informacija.

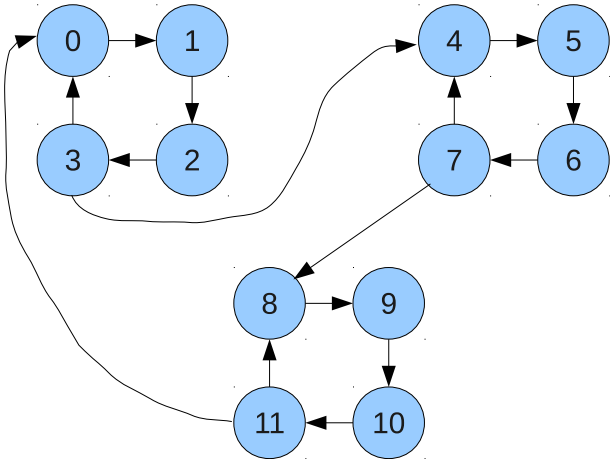
6.4.1 Raznolikost algoritama

Tijekom eksperimentiranja s različitim algoritmima evolucijskog računanja zaključeno je sljedeće:

- algoritmi evolucijskog računanja prikladni su za rješavanje problema raspoređivanja nastavnih obaveza te
- svaki od algoritama evolucijskog računanja prostor pretražuje na drugačiji način i iskazuje drugačiju dinamiku populacije.

Primjeri dinamike kojom se pronalaze najbolja rješenja prikazani su na primjerima dvije vrste rasporeda u poglavlju 5 na slikama 5.11 i 5.23. Temeljem uočenog ponašanja poslavljena je hipoteza da će se hibridizacijom algoritma za rješavanje problema odnosno suradnjom više različitih algoritama evolucijskog računanja postići usporedivi ili čak bolji rezultati no da se radi samo s jednom vrstom algoritma. Naime, svaki od algoritama evolucijskog računanja koristi drugačije definirane operatore, neki koriste spremnik za pohranu dobrih rješenja, neki koriste stalnu a neki promjenjivu vjerojatnost izmjene jedinki te konačno različiti algoritmi ispituju okoline različitih rješenja različitim intenzitetima.

Primjerice, mravlji algoritmi zahvaljujući uporabi heurističke informacije relativno brzo mogu locirati isplative prostore pretraživanja. Djelotvornost kojom će ovo postići ovisit će, dakako, o kvaliteti heurističke informacije. Genetski algoritam je primjer algoritma koji sav prostor pretražuje podjednako i daje "blagu" prednost boljim rješenjima prilikom izvođenja pretrage. Imunološki algoritmi, a posebice algoritam klonske selekcije, ponašaju se kao algoritmi koji stavljaju velik naglasak na fino pretraživanje okolice najboljih rješenja a grublje pretraživanje okolice lošijih jedinki. Zahvaljujući operatoru hiperkloniranja ovi algoritmi stvaraju velik broj klonova dobrih rješenja koja potom fino pretražuju i mali broj klonova loših rješenja koja modificiraju u većoj mjeri. Algoritam roja čestica zahvaljujući definiranom lokalnom susjedstvu i pamćenju dobrih rješenja pretražuje prostor u okolini dobrih rješenja; uslijed lokalno definiranog susjedstva propagacija eventualno pronađenih lokalnih optimuma je usporena čime algoritam iskazuje dobro ponašanje. Konačno, algoritam diferencijske evolucije ima ugrađeno automatsko podešavanje obima promjena koje se rade zahvaljujući činjenici da se gleda razlika između dviju jedinki populacije. Kako je populacija na početku raširena na velikom



Slika 6.18: Topologija ugniježđenih prstena.

prostoru pretraživanja, modifikacije će također biti velike; kada počne konvergencija, populacija će se sažeti, razlike između jedinki će biti manje i algoritam će raditi finije pretraživanje.

Poligon za ispitivanje postavljene hipoteze je prethodno opisani otočni model i to topologija prikazana na slici 6.18.

U svrhu ispitivanja različitih kombinacija algoritama uz obakvu topologiju načinjeni su eksperimenti za dva primjera koji su opisani u nastavku.

6.4.2 Ispitivanje na problemu jednostavnog raspoređivanja

Problem raspoređivanja neka je sljedeći: potrebno je rasporediti 561 studenata u 6 grupa; svaka grupa ima pridružen slijed od od dva predavanja koja moraju pohađati, svako u trajanju od 2 sata. Kapacitet svake od grupa je 95 studenata. Raspored treba načiniti tako da se što je moguće bolje uklopi u postojeći raspored studenata. U ovu svrhu korištene su prethodno opisane implementacije sljedećih algoritama: genetski algoritam, algoritam *Max-Min* mravlji sustav, algoritam roja čestica te jednostavan imunološki algoritam. Korištena je topologija prikazana na slici 6.18 koja se sastoji od 12 radnika (označenih brojevima od 0 do 11) pri čemu svaki radnik izvodi jedan primjerak algoritma koji mu se dodijeli. Ispitano je 15 slučajeva kombinacija algoritama, prema tablici 6.4. Algoritmi su se pridijeljivali u čvorove redoslijedom kojim su navedeni u odgovarajućem retku tablice, uz ponavljanje po potrebi. Primjerice, konfiguracija koja odgovara retku 7 u tablici predstavlja topologiju kod koje je algoritam roja čestica

Tablica 6.4: Algoritmi korišteni u primjeru 1 u pojedinim konfiguracijama i njihov poredak.

Broj	Algoritmi
1	SIA
2	ACO
3	ACO, SIA
4	PSO
5	PSO, SIA
6	PSO, ACO
7	PSO, ACO, SIA
8	GA
9	GA, SIA
10	GA, ACO
11	GA, ACO, SIA
12	GA, PSO
13	GA, PSO, SIA
14	GA, PSO, ACO
15	GA, PSO, ACO, SIA

dodijeljen radnicima 0, 3, 6 i 9, algoritam *Max-Min* mravlji sustav radnicima 1, 4, 7 i 10 te jednostavan imunološki algoritam radnicima 2, 5, 8 i 11. Ovakav način ispitivanja osmišljen je kako bi se mogli usporediti slučajevi gdje se koristi samo jedna vrsta algoritma te slučajevi gdje se koristi više vrsta algoritama. Svaki pojedini algoritam radio je s vlastitom populacijom od samo 15 jedinki, čime je ukupna veličina populacije bila $15 \times 12 = 180$ jedinki.

Opisani sustav algoritama pokretao se je na jednom računalu pri čemu je svaki od algoritama smještenih u pojedine čvorove topologije simuliran slijedno u trajanju od približno 125 milisekundi. Na kraju svakog ciklusa (nakon odsimuliranih svih 12 algoritama) izvršena je migracija rješenja u skladu s topologijom. Pri tome su migracije unutar malih prstena rađene svaki puta a migracije između malih prstena svaki četvrti puta. Ovo je ponavljano do isteka 22. minute, čime je svaki od algoritama radio približno 1 minutu i 50 sekundi. Kako je vrijeme potrebno za izvođenje jednog ciklusa 125 ms $\times 12 = 1,5$ s, kroz vrijeme od 22 minute dogodilo se je ukupno 880 migracija od čega njih 660 samo unutar malih prstena, a 220 i unutar malih prstena i između prstena.

Kako bi se prikupili podatci za izračun prosječnih vrijednosti, za svaku je konfiguraciju obavljeno 30 pokretanja. Dobiveni rezultati prikazani su u tablici 6.5. Svaka od

ispitanih konfiguracija pokazala se je uspješnom u pronalasku rasporeda u kojem niti jedan od studenata nema konflikata s postojećim obavezama. Stoga su srednja vrijednost, medijan, standardno odstupanje, minimalna i maksimalna vrijednost sve jednake 0, pa te vrijednosti nisu prikazivane u zasebnoj tablici.

Sve su se konfiguracije također pokazale uspješne u minimizaciji kazne zbog nekompaktnosti. Najbolje je rješenje pri tome postigla konfiguracija koja koristi 12 primjeraka algoritma *Max-Min* mravlji sustav, što je i očekivano. Naime, kako su kod ovog problema termini nepromjenjivi, algoritmu *Max-Min* mravlji sustav stavljena je na raspolaganje vrlo kvalitetna heuristička informacija zahvaljujući kojoj je ovaj algoritam u velikoj prednosti pred drugim algoritmima koji ne koriste heurističku informaciju. Zahvaljujući tome, svaka konfiguracija koja je kao jedan od algoritama imala i *Max-Min* mravlji sustav postigla je bolje rezultate u odnosu na konfiguracije koje nisu imale taj algoritam.

Ako se iz razmatranja izuzmu konfiguracije koje su imale *Max-Min* mravlji sustav a to su konfiguracije 2, 3, 6, 7, 10, 11, 14 i 15 te ako se usporede samo algoritmi koji pretragu rade bez dostupne heurističke informacije, sljedeća najbolja kombinacija je kombinacija 12 koja se sastoji od genetskog algoritma i algoritma roja čestica. Nešto lošiji rezultati dobiveni su za konfiguracije koje se sastoje samo od genetskog algoritma, samo od algoritma roja čestica te za konfiguraciju koja se sastoji od genetskog algoritma, algoritma roja čestica te jednostavnog imunološkog algoritma.

6.4.3 Ispitivanje na problemu izrade rasporeda provjera znanja

Problem izrade rasporeda provjera znanja opisan je u podpoglavlju 4.5. Za razliku od prethodnog problema, kod rasporeda provjera znanja teško je definirati kvalitetnu heuristiku. Naime, kako je pretpostavka da u terminima provjera znanja studenti nemaju drugih obaveza, unaprijed je teško definirati koji je termin pogodniji za koji kolegij. Nekvaliteta rasporeda proizlazit će jedino iz međusobnog odnosa više kolegija a ne iz apsolutnog termina u koji je pojedini kolegij smješten. Stoga se kod ove vrste problema ne očekuje da će mravlji algoritmi biti u prednosti pred ostalima što su provedeni eksperimenti i potvrdili.

U svrhu ispitivanja algoritama evolucijskog računanja na ovom problemu načinjeno je ukupno 30 eksperimenata. Algoritmi koji su ispitivani su generacijski genetski

Tablica 6.5: Ukupna kazna zbog nekompaktnosti za primjer 1 i za svaku od konfiguracija.

Top.	Sred. vr.	Medijan	Std. ods.	Min	Max
1	2394.00	2391.50	18.24	2362.00	2442.00
2	**2285.30**	**2284.50**	**2.67**	**2282.00**	**2291.00**
3	2289.37	2288.50	4.92	2282.00	2302.00
4	2317.20	2316.50	9.15	2304.00	2337.00
5	2325.07	2325.50	12.41	2300.00	2353.00
6	2287.30	2287.00	3.54	2283.00	2295.00
7	2292.13	2292.00	5.36	2284.00	2310.00
8	2313.20	2312.50	7.94	2298.00	2332.00
9	2326.00	2326.00	12.46	2305.00	2354.00
10	2286.90	2287.00	3.60	2282.00	2296.00
11	2292.40	2292.00	5.37	2283.00	2301.00
12	**2311.60**	**2310.50**	**9.32**	**2297.00**	**2330.00**
13	2318.97	2317.50	9.19	2305.00	2339.00
14	2290.40	2290.00	3.95	2284.00	2301.00
15	2291.77	2291.50	6.10	2284.00	2305.00

algoritam, algoritam *Max-Min* mravlji sustav, algoritam harmonijske pretrage te jednostavan imunološki algoritam. Jedna serija eksperimenata načinjena je bez uporabe lokalne pretrage a druga serija načinjena je uz uporabu lokalne pretrage. Svaki je eksperiment ponavljan 30 puta kako bi se ublažio utjecaj slučajnosti na dobivene rezultate. Verzije algoritama koje su korištene u ovom eksperimentu su verzije koje u populaciju ne puštaju rješenja koja ne zadovoljavaju tvrda ograničenja. Stoga sva rješenja koja algoritmi daju zadovoljavaju tvrda ograničenja i jedino se razlikuju po ukupnoj kazni zbog nekvalitete rasporeda.

Eksperimenti su izvedeni uporabom razvijenog sustava za raspodijeljeno izvođenje evolucijskih algoritama opisanog u [Komar et al., 2011, Komar, 2011, Grbić, 2011]. Korištena je topologija prikazana na slici 6.18. Svaki je algoritam izvođen na zasebnoj jezgri procesora pri čemu su eksperimenti obavljeni na ukupno tri četverojegrena procesora uz komunikaciju između računala preko Ethernet lokalne mreže. Algoritmi korišteni u pojedinim konfiguracijama navedeni su u tablici 6.6. Razmjena rješenja unutar malih prstena obavljala se je svake sekunde. Razmjena rješenja između malih prstena obavljala se je svakih četiri sekunde.

U tablici 6.7 prikazani su dobiveni rezultati nakon 15 minuta izvođenja algoritama za slučaj isključene lokalne pretrage, dok su u tablici 6.8 prikazani dobiveni rezultati

Tablica 6.6: Algoritmi korišteni u primjeru 2 u pojedinim konfiguracijama.

Broj	Algoritmi
1	GA
2	ACO
3	HS
4	SIA
5	ACO, SIA
6	GA, SIA
7	HS, SIA
8	GA, ACO
9	HS, ACO
10	GA, HS
11	GA, ACO, SIA
12	HS, ACO, SIA
13	HS, GA, SIA
14	HS, GA, ACO
15	GA, ACO, HS, SIA

nakon 15 minuta izvođenja algoritama za slučaj s uključenom lokalnom pretragom.

Algoritmi su uspoređeni temeljem dva kriterija. Prvi kriterij je postignuta srednja vrijednost najboljeg rješenja; kako se radi o kazni, bolje je ono rješenje koje ima manji iznos kazne. Drugi kriterij bio je iznos standardnog odstupanja; što je iznos standardnog odstupanja manji, smatramo da je algoritam bolji. Tablica 6.9 prikazuje za svaku od konfiguracija algoritama bez lokalne pretrage rang temeljem iznosa kazne (stupac *Rang 1*), rang temeljem iznosa standardnog odstupanja (stupac *Rang 2*), prosječni rang izračunat kao srednja vrijednost prethodna dva stupca te ukupni rang temeljem rangiranja prosječnih rangova. Tablica 6.10 prikazuje podatke za konfiguracije algoritama s lokalnom pretragom.

Analizom podataka iz tablice 6.9 vidimo da u slučaju isključene lokalne pretrage algoritam Max-Min mravlji sustav i algoritam harmonijske pretrage postižu rezultate koji su pri dnu postignutih rezultata. Genetski algoritam, iako ima najmanji iznos kazne rješenja ima veliko standardno odstupanje. Jednostavni imunološki algoritam ima rang 4 i po kvaliteti rješenja i po standardnom odstupanju. Gledano po združenom kriteriju, najbolje su konfiguracija koja ima samo jednostavni imunološki algoritam te konfiguracija triju algoritama: algoritma harmonijske pretrage, genetskog algoritma te jednostavnog imunološkog algoritma. Pri tome ova kombinacija algoritama ima rješenje manje kazne ali zato ima veće standardno odstupanje. Gledano prema ukupnom rangu

Tablica 6.7: Ukupna kazna rasporeda ispita za svaku od 15 konfiguracija algoritama; slučaj bez lokalne pretrage.

Top.	Sred. vr.	Medijan	Std. ods.	Min	Max
1	8360.93	8268.50	336.56	7773.65	9268.66
2	13945.27	13900.25	673.03	11502.16	15241.14
3	9027.19	8956.44	356.34	8229.81	9915.08
4	**8528.38**	**8544.13**	**282.04**	**8019.02**	**8988.56**
5	8693.05	8684.23	307.23	8188.19	9393.65
6	8564.78	8607.68	313.79	7989.16	9187.45
7	8530.44	8503.94	327.94	7971.45	9438.60
8	8566.29	8496.49	263.80	8073.88	9182.13
9	9145.58	9178.81	277.39	8683.03	9765.71
10	8532.97	8512.40	282.07	8033.37	9224.59
11	8521.31	8539.44	287.54	7746.39	9121.68
12	8564.38	8468.03	373.60	7894.99	9281.56
13	**8500.56**	**8513.43**	**286.45**	**7869.61**	**9073.73**
14	8638.65	8679.27	260.35	8138.26	9141.11
15	8635.20	8598.97	336.31	8084.07	9534.84

Tablica 6.8: Ukupna kazna rasporeda ispita za svaku od 15 konfiguracija algoritama; slučaj s lokalnom pretragom.

Top.	Sred. vr.	Medijan	Std. ods.	Min	Max
16	8647.30	8616.41	281.91	8074.22	9137.97
17	7229.79	7244.57	199.29	6876.89	7652.16
18	7082.34	6926.31	487.06	6705.13	8371.19
19	8684.83	8669.47	297.03	8074.87	9263.97
20	7334.49	7325.82	193.48	6923.82	7857.88
21	8714.65	8792.09	278.91	8181.44	9140.44
22	6991.13	6997.94	154.26	6679.22	7397.97
23	7348.36	7318.02	165.55	6898.88	7713.64
24	**6984.27**	**6980.09**	**141.33**	**6632.96**	**7302.03**
25	7157.22	7137.48	182.64	6782.06	7525.67
26	7458.63	7447.73	201.25	7083.65	7899.91
27	**6984.45**	**6981.43**	**141.25**	**6782.86**	**7323.54**
28	7146.31	7113.99	206.37	6791.71	7649.45
29	7070.11	7055.32	162.70	6774.52	7388.59
30	7050.49	7018.05	187.93	6713.52	7454.53

Tablica 6.9: Poredak konfiguracija algoritama; slučaj bez lokalne pretrage i trajanjem eksperimenta od 15 minuta.

Konf.	Algoritmi	Rang 1	Rang 2	Prosj. rang	Ukupni rang
1	GA	1	12	6.5	7
2	ACO	15	15	15	15
3	HS	13	13	13	14
4	**SIA**	**4**	**4**	**4**	**1**
5	ACO, SIA	12	8	10	11
6	GA, SIA	8	9	8.5	9
7	HS, SIA	5	10	7.5	8
8	GA, ACO	9	2	5.5	4
9	HS, ACO	14	3	8.5	9
10	GA, HS	6	5	5.5	4
11	GA, ACO, SIA	3	7	5	3
12	HS, ACO, SIA	7	14	10.5	12
13	**HS, GA, SIA**	**2**	**6**	**4**	**1**
14	HS, GA, ACO	11	1	6	6
15	GA, ACO, HS, SIA	10	11	10.5	12

dalje opet slijedi nekoliko konfiguracija koje sadrže kombinacije algoritama.

Analizom podataka iz tablice 6.10 možemo uočiti da se situacija mijenja. Genetski algoritam te jednostavan imunološki algoritam su sada među najslabijima, a srednje rezultate postižu algoritam harmonijske pretrage te algoritam *Max-Min* mravlji sustav. Pogleda li se poredak samo prema stupcu *Rang 1* ili samo prema stupcu *Rang 2*, vrh poretka drže kombinacije algoritama. Gledano prema ukupnom rangu, gotovo prvih 50% mjesta također pripadaju konfiguracijama koje sadrže kombinacije algoritama.

Tablice 6.11 i 6.12 prikazuju rangove algoritama s i bez lokalne pretrage uz istu topologiju i pokrenute nad istim problemom ali uz trajanje eksperimenta od 30 minuta. Za slučaj bez lokalne pretrage genetski algoritam i dalje pronalazi najkvalitetnije rješenje ali uz veliko standardno odstupanje; druga po kvaliteti je kombinacija genetskog algoritma, algoritma Max-Min mravljeg sustava te jednostavnog imunološkog algoritma koje ujedno postiže i treće po redu standardno odstupanje čime je ta kombinacija prema ukupnom rangu najbolja. U slučaju korištenja lokalne pretrage, algoritam harmonijske pretrage nakon 30 minuta se je ipak pokazao kao algoritam koji na ovom problemu uspjeva pronaći najbolje rješenje uz najmanje standardno odstupanje. Nakon njega slijede kombinacije algoritama koje opet uključuju ovaj algoritam.

Tablica 6.10: Poredak konfiguracija algoritama; slučaj s lokalnom pretragom i trajanjem eksperimenta od 15 minuta.

Konf.	Algoritmi	Rang 1	Rang 2	Prosj. rang	Ukupni rang
16	GA	13	13	13	13
17	ACO	9	9	9	8
18	HS	6	15	10.5	11
19	SIA	14	14	14	15
20	ACO, SIA	10	8	9	8
21	GA, SIA	15	12	13.5	14
22	HS, SIA	3	3	3	3
23	GA, ACO	11	5	8	7
24	**HS, ACO**	**1**	**2**	**1.5**	**1**
25	GA, HS	8	6	7	6
26	GA, ACO, SIA	12	10	11	12
27	**HS, ACO, SIA**	**2**	**1**	**1.5**	**1**
28	HS, GA, SIA	7	11	9	8
29	HS, GA, ACO	5	4	4.5	4
30	GA, ACO, HS, SIA	4	7	5.5	5

Tablica 6.11: Poredak konfiguracija algoritama; slučaj bez lokalne pretrage i trajanjem eksperimenta od 30 minuta.

Konf.	Algoritmi	Rang 1	Rang 2	Prosj. rang	Ukupni rang
1	GA	1	12	6.5	6
2	ACO	15	15	15	15
3	HS	13	14	13.5	14
4	SIA	7	1	4	2
5	ACO, SIA	12	11	11.5	13
6	GA, SIA	5	9	7	8
7	HS, SIA	6	5	5.5	5
8	GA, ACO	8	7	7.5	9
9	HS, ACO	14	8	11	11
10	GA, HS	4	4	4	2
11	**GA, ACO, SIA**	**2**	**3**	**2.5**	**1**
12	HS, ACO, SIA	9	13	11	11
13	HS, GA, SIA	3	6	4.5	4
14	HS, GA, ACO	11	2	6.5	6
15	GA, ACO, HS, SIA	10	10	10	10

Tablica 6.12: Poredak konfiguracija algoritama; slučaj s lokalnom pretragom i trajanjem eksperimenta od 30 minuta.

Konf.	Algoritmi	Rang 1	Rang 2	Prosj. rang	Ukupni rang
16	GA	14	14	14	13
17	ACO	9	12	10.5	11
18	**HS**	**1**	**1**	**1**	**1**
19	SIA	13	15	14	13
20	ACO, SIA	11	10	10.5	11
21	GA, SIA	15	13	14	13
22	HS, SIA	4	4	4	4
23	GA, ACO	10	7	8.5	8
24	**HS, ACO**	**2**	**2**	**2**	**2**
25	GA, HS	8	8	8	7
26	GA, ACO, SIA	12	6	9	9
27	HS, ACO, SIA	3	3	3	3
28	HS, GA, SIA	7	11	9	9
29	HS, GA, ACO	6	5	5.5	5
30	GA, ACO, HS, SIA	5	9	7	6

6.4.4 Korist od kombinacije algoritama

Prilikom rješavanja problema raspoređivanja, posebice onih koje i raspodijeljeni evolucijski algoritam riješava vrlo sporo, na kvalitetu dobivenih rješenja utjeće mnoštvo parametara. Problemi koji se pri tome javljaju su:

- nepoznavanje optimalnog rješenja čime ne postoji mogućnost procjenje stvarne udaljenosti između trenutno pronađenog rješenja i optimalnog rješenja,
- pitanje algoritma koji će u prosjeku davati dobra rješenja i njegovih parametara,
- pitanje vremena koje će algoritmu biti stavljeno na raspolaganje za pronalazak dobrog rješenja.

Eksperimenti provedeni u okviru ovog poglavlja pokazuju da kombinacije algoritama mogu postići jednake ili bolje rezultate no što ih postižu pojedinačni algoritmi. Ova tvrdnja pri tome ne vrijedi sasvim općenito. Odabirom proizvoljne kombinaciju algoritama moguće je dobiti i algoritam koji nad promatranim problemom postiže loše rezultate, što pokazuju provedeni eksperimenti. Međutim, prikladnom kombinacijom algoritama moguće je dobiti algoritam koji daje vrlo dobre rezultate. Također, ako neki pojedinačni algoritam postiže dobre rezultate, kombinacije u koje je on uključen često

će također postići dobre rezultate. Tako je, primjerice, u eksperimentima provedenim u okviru podpoglavlja 6.4.2 algoritam *Max-Min* mravlji sustav davao odlične rezultate zahvaljujući kvalitetnoj heurističkoj informaciji koju je jedini imao; međutim, i sve kombinacije algoritama u koje je on bio uključen također su davale odlične rezultate. S druge strane, prvi sljedeći najbolji algoritam koji nije uključivao spomenuti mravlji algoritam opet je bila kombinacija izvedena od genetskog algoritma i algoritma roja čestica.

Eksperimenti provedeni u okviru podpoglavlja 6.4.3 također prikazuju ponašanje algoritama bez lokalne pretrage odnosno sa lokalnom pretragom uz ograničenje vremena od 15 minuta te uz ograničenje vremena od 30 minuta. Kako se radi o 12 algoritama koji rade u paraleli, to je ekvivalent eksperimentima od 3 sata odnosno od 6 sati pokrenutih na jednom računalu. I ti eksperimenti pokazuju da kombinacije algoritama mogu davati vrlo kvalitetna rješenja, te da istovremeno mogu poslužiti kao mehanizam za stabilizaciju rada algoritma pri čemu pod pojmom stabilizacija smatramo smanjivanje standardnog odstupanja dobivenih rješenja. Dakako, treba voditi računa o tome da su ovi rezultati dobiveni uz fiksirane migracijske parametre čiji bi utjecaj dalje trebalo istražiti.

Poglavlje 7

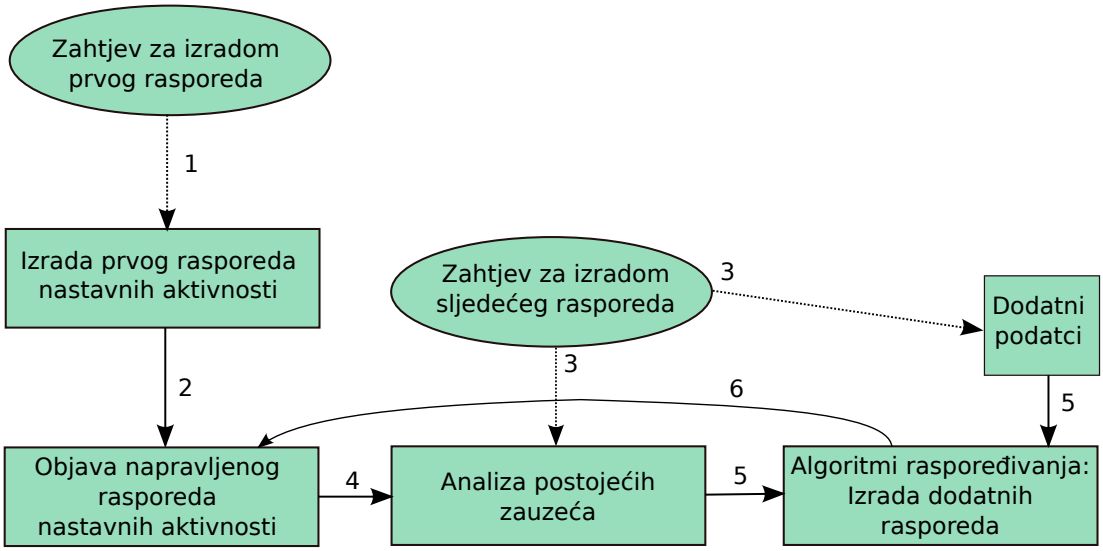
Model sustava za objavu rasporeda

U prethodnim poglavljima ove disertacije naglasak je stavljen na načine rješavanja različitih problema raspoređivanja nastavnih aktivnosti. Pri tome nisu razmatrana dva važna problema:

- *priprema podataka* temeljem kojih će se krenuti u rješavanje problema raspoređivanja te
- *objava rezultata* zahvaljujući kojoj sudionici nastavnog procesa dobivaju informaciju o nastavnim aktivnostima te zahvaljujući kojoj je prethodni korak uopće moguć.

Izrada prvog rasporeda nastavnih aktivnosti uvijek kreće od pretpostavke da su sudionici nastavnog procesa uvijek slobodni. Stoga se prilikom izrade takvog rasporeda pažnja posvećuje isključivo kvaliteti načinjenog rasporeda. Međutim, jednom kada je prvi raspored načinjen, svako sljedeće raspoređivanje mora uzeti u obzir postojeća zauzeća osoba koje se raspoređuju kako bi se izbjegle kolizije s tim zauzećima. Tijek poslova prikazan na slici 7.1 ovo jasno ilustrira. Svaki novi posao raspoređivanja započinje analizom postojećih zauzeća nakon čega slijedi izrada dodatnog rasporeda temeljem podataka o zauzećima te temeljem dodatnih podataka koji pobliže definiraju problem raspoređivanja. Nakon što je raspored načinjen, posljednji korak je njegova objava čime se zatvara ciklus raspoređivanja.

U okviru posljednjeg poglavlja ove disertacije stoga je opisan model podataka koji omogućava izradu programskog sustava temeljem kojeg se osigurava podrška za izvođenje opisanih ciklusa raspoređivanja.



Slika 7.1: Ciklus izrade rasporeda nastavnih aktivnosti

7.1 Model sustava za upravljanja kolegijima

Svi problemi raspoređivanja koji su upisani u ovoj disertaciji pripadaju u razred problema raspoređivanja nastavnih obaveza. Stoga je u nastavku opisan model jednostavnog sustava za upravljanje kolegijima (engl. *Course Management System, CMS*) koji je dovoljno općenit da omogućava primjenu u osnovnim školama, srednjim školama te visokoškolskim ustanovama. Treba odmah istaknuti i da je fokus razmatranja podrška za rješavanje problema raspoređivanja pa definirani model neće uključivati sve segmente jednog klasičnog sustava za upravljanje kolegijima. Međutim, dodavanje takvih dijelova modelom nije niti na koji način onemogućeno.

Za definiranje modela korišteni su koncepti objektno-orijentirane paradigme. Stoga je predloženi model definiran na razini razreda i njihovih međusobnih veza. Za pohranu podataka modela (engl. *data persistence*) u tom je slučaju moguće iskoristiti neku od objektnih baza podataka ili, ono što je danas puno češći pristup, iskoristiti neki od podsustava za preslikavanje objekata u relacije i potom pohranu u provjerene relacijske baze podataka.

Predloženi model prikazan je na slici 7.2 uporabom dijagrama razreda jezika UML [Fowler and Scott, 2003]. Za raspoređivanje je centralan apstraktni razred *ApstraktniDogađaj* koji predstavlja općeniti događaj; događaj ima vrijeme početka, trajanje i

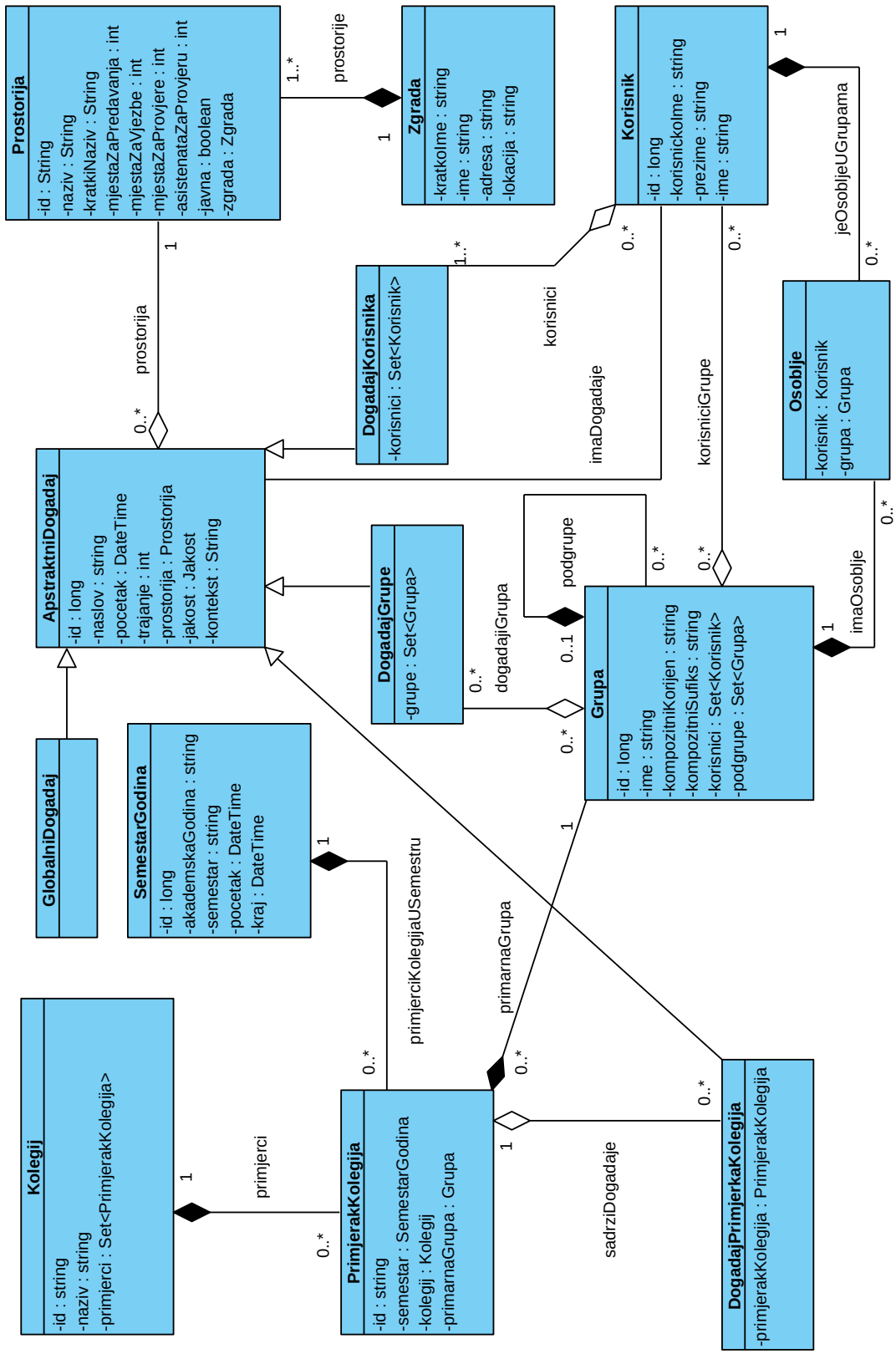
lokaciju na kojoj se održava. Iz tog događaja izvedene su specijalizacije poput događaja kolegija, događaja grupe i slično. Svi ostali razredi modela u određenoj mjeri služe za organizaciju korisnika prema različitim kriterijima te potporu upravljanju događajima.

Razred *SemestarGodina* predstavlja jedan semestar u određenoj akademskoj godini. Razred definira svojstvo *akademskaGodina* te svojstvo *semestar* čime je semestar akademske godine jednoznačno određen. U visokoškolskim ustanovama godina uobičajeno ima više semestara pa će za istu akademsku godinu postojati više ovakvih objekata koji će se razlikovati po vrijednosti svojstva *semestar*. U srednjoškolskim i osnovnoškolskim ustanovama uobičajeno je da postoji samo godina, pa će u svakoj akademskoj godini postojati samo jedan ovakav objekt. Svojstvo *id* služi kao primarni ključ i omogućava efikasnu pohranu relacija u relacijsku bazu podataka, kako bi se izbjegla uporaba kompozitnog ključa (*akademskaGodina,semestar*). Uređeni par (*akademskaGodina,semestar*) u sustavu mora biti jedinstven. Svojstva *početak* i *kraj* omogućavaju pamćenje datuma i vremena početka i kraja semestra.

Razred *Kolegij* predstavlja kolegij. Kolegij ima svoju šifru (svojstvo *id*) te naziv. Šifra se pri tome može koristiti i kao primarni ključ.

Kolegij se može nuditi u jednom ili u više semestara akademske godine, a barem jednom u akademskoj godini. Jedno izvođenje kolegija modelirano je razredom *PrimjerakKolegija*. U tom razredu svojstvo *id* predstavlja primarni ključ. Svojstva *semestar* i *kolegij* povezuju primjerak kolegija sa semestrom u kojem se nudi te s kolegijem. Uređeni par (*semestar*, *kolegij*) u sustavu mora biti jedinstven što se u relacijskoj bazi podataka može osigurati odgovarajućim složenim indeksom. Primjerci razreda *Kolegij* mogu održavati kolekciju svih pridruženih primjeraka razreda *PrimjerakKolegija*.

Korisnik je modeliran razredom *Korisnik*. ovaj razred omogućava pamćenje primarnog ključa korisnika, korisničko ime, te ime i prezime korisnika. Sigurna autentifikacija te autorizacija korisnika ovdje se neće razmatrati jer se mogu rješavati na različite načine. Podrška za najjednostavniju moguću autentifikaciju je uključiti svojstvo *zaporka* u kojem bi se čuvala korisnikova zaporka (ili njezin sažetak što je danas uobičajena praksa) dok se autorizacija može riješiti uvođenjem razreda *Uloga* koji bi predstavljao moguće uloge u sustavu (poput uloge *student* te uloge *administrator*). Svakom bi se korisniku tada mogao dodijeliti skup odgovarajućih uloga. Moguća su dakako i druga rješenja.



Slika 7.2: UML dijagram objektnog modela podataka

Zgrada je modelirana razredom *Zgrada* koji omogućava pamćenje kratkog te punog imena zgrade, adresa na kojoj se zgrada nalazi te pamćenje geografskih koordinata zgrade.

Prostorija u zgradi modelirana je razredom *Prostorija*. Za svaku se prostoriju pamte se sljedeći podatci: primarni ključ, puni naziv prostorije, kratki naziv prostorije, broj raspoloživih mjesta kada se prostorija koristi za predavanja, broj raspoloživih mjesta kada se prostorija koristi za laboratorijske vježbe, broj raspoloživih mjesta kada se prostorija koristi za provođenje ispita, broj djelatnika potrebnih za provođenje ispita u toj prostoriji, zastavica koja govori je li prostorija javna te veza prema zgradi u kojoj se prostorija nalazi.

Razred *Grupa* predstavlja skup korisnika, a može sadržavati i druge grupe pri čemu tako definirana struktura mora biti stablasta (odnosno bez ciklusa). Razred *Grupa* omogućava pamćenje primarnog ključa, naziva grupe, skup korisnika koji pripadaju toj grupi te skupa grupa koje su djeca te grupe. Kako bi se osigurao brzi dohvat čitavog podstabla grupa iz relacijske baze podataka, grupi su dodana još dva svojstva: *kompozitniKorijen* te *kompozitniSufiks*. Uporaba ovih svojstava opisana je u nastavku. Prostor primarnih ključeva koji se koristi za različite istaknute razrede (poput razreda *PrimjerakKolegija*) treba biti disjunktan. Svim grupama koje pripadaju nekom primjerku razreda *PrimjerakKolegija* vrijednost svojstva *kompozitniKorijen* treba postaviti na vrijednost primarnog ključa tog primjerka razreda *PrimjerakKolegija*. Za vrijednosti svojstva *kompozitniSufiks* koje je po tipu znakovni niz predlaže se uporaba hijerarhijskih numeričkih identifikatora, primjerice oblika `0/1/1` izuzev za primarnu grupu primjerka kolegija čije je svojstvo *kompozitniSufiks* postavljeno na prazan znakovni niz. Direktna djeca[1] primarne grupe tada će imati svojstvo *kompozitniSufiks* postavljeno na znakovne nizove `0`, `1`, `2`, itd. Ove se grupe mogu iskoristiti za definiranje različitih podjela korisnika na razini čitavog kolegija. Tako grupa čije je svojstvo *kompozitniSufiks* postavljeno na `0` može predstavljati grupu korisnika za predavanja, grupa čije je svojstvo *kompozitniSufiks* postavljeno na `1` može predstavljati grupu korisnika za laboratorijske vježbe, itd. Podgrupe grupe za predavanja tada će imati svojstvo *kompozitniSufiks* postavljeno na `0/0`, `0/1`, `0/2`, itd., i tek će te grupe doista sadržavati i korisnike. Koristi li se ovakav način definiranja grupa, važno je istaknuti sljedeće.

[1] Grupe mogu sadržavati podgrupe koje mogu sadržavati podgrupe itd. Pojam "direktna" odnosi se na prvu razinu podgrupa.

- Dubine pojedinih grana stabla ne moraju biti jednake. Primjerice, *grupa za predavanja* ima još samo jednu razinu grupa ispod sebe: stvarne grupe za predavanja koje sadrže korisnike. *Grupa za laboratorijske vježbe* ima još dvije razine grupa ispod sebe. Prvu razinu čine *grupa za 1. laboratorijsku vježbu*, *grupa za 2. laboratorijsku vježbu* i slično dok drugu razinu čine grupe koje odgovaraju pojedinim terminima svake od laboratorijskih vježbi i tek te grupe sadrže korisnike.

- Struktura grupa čini višestruke poglede na korisnike. Korisnik istovremeno može biti i u nekoj od grupa za predavanja, i u nekoj od grupa za 1. laboratorijsku vježbu, i u nekoj od grupa za 2. laboratorijsku vježbu, itd.

Ovako definirana struktura grupa može se koristiti i za autorizaciju korisnika. Primjerice, jedna od direktnih podgrupa primarne grupe kolegija može biti grupa za definiranje prava pristupa značajkama kolegija. Podgrupe te grupe mogu biti grupe koje sadrže nastavnike nositelje, izvođače predavanja, asistente organizatore te asistente. Temeljem analize kojim od tih grupa korisnik pripada sustav korisniku može ponuditi informacije i akcije za koje korisnik ima prava pristupa.

Uvođenje ovakvog dvostrukog praćenja svake od grupa (uporabom primarnih ključeva te uporabom para *compositeRoot* i *compositePart*) omogućava se efikasnost rada direktno kroz objektni model kao i efikasnost dohvata čitavog podstabla u relacijskim bazama podataka. Primjerice, kako bismo dohvatili sve grupe za predavanja, dovoljno je uporabom operatora `LIKE` u SQL upitu zatražiti sve grupe čiji je *compositePart* `LIKE '0/%'`. Sve grupe za sve laboratorijske vježbe dobit će se upitom oblika `LIKE '1/%'`, dok će se sve grupe za prvu laboratorijsku vježbu dobiti upitom `LIKE '0/0/%'`.

7.1.1 Modeliranje događaja

Osnovni model događaja definiran je apstraktnim razredom *ApstraktniDogađaj*. Ovaj razred omogućava pohranu svih osnovnih informacija nekog događaja a to su naziv događaja, datum i vrijeme početka događaja, trajanje događaja te prostorija u kojoj se događaj odvija. Razred *ApstraktniDogađaj* definira i dodatno svojstvo *jakost* čija je namjena omogućiti definiranje važnosti događaja prilikom provođenja postupaka raspoređivanja. Primjerice, događaj može predstavljati termin predavanja i u kontekstu

analize zauzeća korisnika u svrhu pripreme podataka za novo raspoređivanje takav događaj mora biti uzet uobzir. No, s druge strane, događaj može biti neobavezan izlet ili privatni događaj korisnika (kava s prijateljima) koji se prilikom izrade novih rasporeda po potrebi može i zanemariti. Konačno, razred *ApstraktniDogađaj* definira i dodatno svojstvo *kontekst* koje može poslužiti sustavu za pohranu dodatnih informacija temeljem kojih bi korisnicima mogao ponuditi "pametnu" navigaciju kroz kalendar u kojem su prikazani događaji (primjerice, klik na događaj koji je predavanje vodi korisnika na stranicu dotičnog kolegija i slično). Razred *ApstraktniDogađaj* je apstraktan jer donosi samo dio semantike – definirane su informacije o mjestu, vremenu, trajanju i važnosti događaja, ali nisu i informacije o korisnicima na koje se ovaj događaj odnosi.

Stoga su iz razreda *ApstraktniDogađaj* izvedena četiri nova razreda. Razred *GlobalniDogađaj* predstavlja specijalizaciju razreda *ApstraktniDogađaj* i dodatno definira njegovu semantiku – događaji predstavljeni primjercima razreda *GlobalniDogađaj* odnose se na sve korisnike sustava. Primjeri su proslava Dana Škole, proslava Dana Fakulteta i slično. Razred *DogađajPrimjerkaKolegija* definira dodatno svojstvo *primjerakKolegija* kojim događaj povezuje s primjerkom kolegija; time se događaj odnosi na sve korisnike koji su upisani na taj primjerak kolegija ili koji pripadaju u osoblje kolegija. Primjer uporabe ovakvih događaja je objava termina obaveznih provjera znanja. Razred *DogađajGrupe* definira dodatno svojstvo skup *grupe* kojim događaj povezuje sa svim grupama navedenim u zadanom skupu; time se događaj odnosi na sve korisnike koji su smješteni u bilo koju od tih grupa. Primjeri uporabe ovakvih događaja ima najviše: izvođenje predavanja, termini laboratorijskih vježbi, termini provjera znanja nakon razmještanja korisnika u dvorane i slično. Konačno, razred *DogađajKorisnika* predstavlja specijalizaciju koja uvodi zaseban skup korisnika (svojstvo *korisnici*) na koje se taj događaj odnosi.

Razred *Osoblje* omogućava dodjelu nastavnog osoblja grupama. Primjerice, uporabom primjerka razreda *Osoblje* nastavnik se povezuje uz grupu u kojoj izvodi predavanja (i time automatski dobiva i sve događaje te grupe), asistent se povezuje s grupama u kojima izvodi laboratorijske vježbe (i time automatski dobiva i sve događaje tih grupa) i slično.

7.1.2 Objava rasporeda

Postupak objave rasporeda u ovakvom se modelu svodi na stvaranje primjeraka različitih razreda koji opisuju događaje. Primjerice, objava rasporeda provjera znanja provodi se stvaranjem primjeraka razreda *DogađajPrimjerkaKolegija*. Objava rasporeda predavanja (poznata pod nazivom *satnica*) provodi se stvaranjem/ažuriranjem grupa za predavanja te stvaranjem/povezivanjem s primjericima razreda *DogađajGrupe*. Na sličan se način objavljuju i druge vrste rasporeda.

7.1.3 Analiza zauzeća korisnika

Temeljem definiranog modela, analiza zauzeća nekog korisnika provodi se u četiri koraka.

1. Dohvati sve događaje primjerke razreda *GlobalniDogađaj*.
2. Dohvati sve događaje primjerke razreda *DogađajPrimjerkaKolegija* za sve kolegije na kojima je korisnik registriran.
3. Dohvati sve događaje primjerke razreda *DogađajGrupe* koji se odnose na bilo koju od grupa u koje je korisnik smješten ili koji se odnose na bilo koju od grupa za koju postoji primjerak razreda *Osoblje* koji povezuje tog korisnika i tu grupu.
4. Dohvati sve događaje primjerke razreda *DogađajKorisnika* koji u skupu korisnika imaju promatranog korisnika.

Analiza zauzeća za više korisnika može se provesti uzastopnim ponavljanjem opisanog postupka (po jednom za svakom korisnika).

7.2 Ispitivanje sustava temeljenog na definiranom modelu

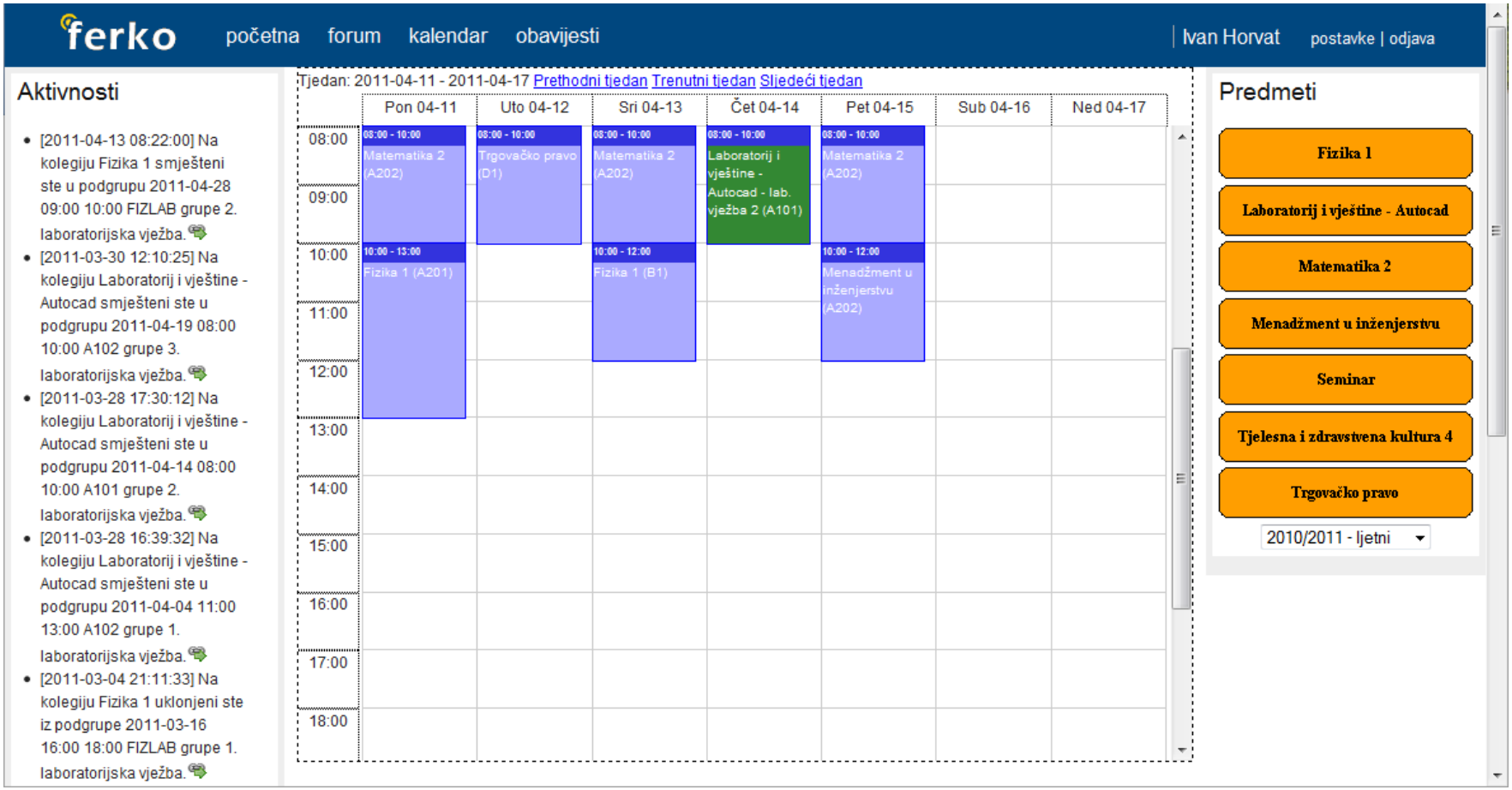
Model podataka koji je opisan u ovom poglavlju dopunjen je i implementiran u okviru web-orijentiranog sustava za upravljanje kolegijima *Ferko* koji je razvijen na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Sustav *Ferko* pušten je u pogon s početkom zimskog semestra akademske godine 2008./2009. te se je u protekle tri godine pokazao vrlo stabilnim i uspješnim sustavom čija je centralna komponenta upravo kalendar događaja.

Oko kalendara događaja izgrađena je čitava infrastruktura koja omogućava upravljanje kolegijima, organizaciju i provođenje ispita, organizaciju i provođenje laboratorijskih vježbi, organizaciju i provođenje različitih vrsta nadoknada te još niz drugih usko povezanih mogućnosti. Svakom je korisniku na raspolaganje stavljena i živa poveznica (engl. *link*) na njegov kalendar u *iCal* formatu čime su je kalendar događaja korisnicima dostupan i kroz *Google Calendar*, druge organizatore vremena te konačno i na danas uobičajenim pametnim mobilnim uređajima.

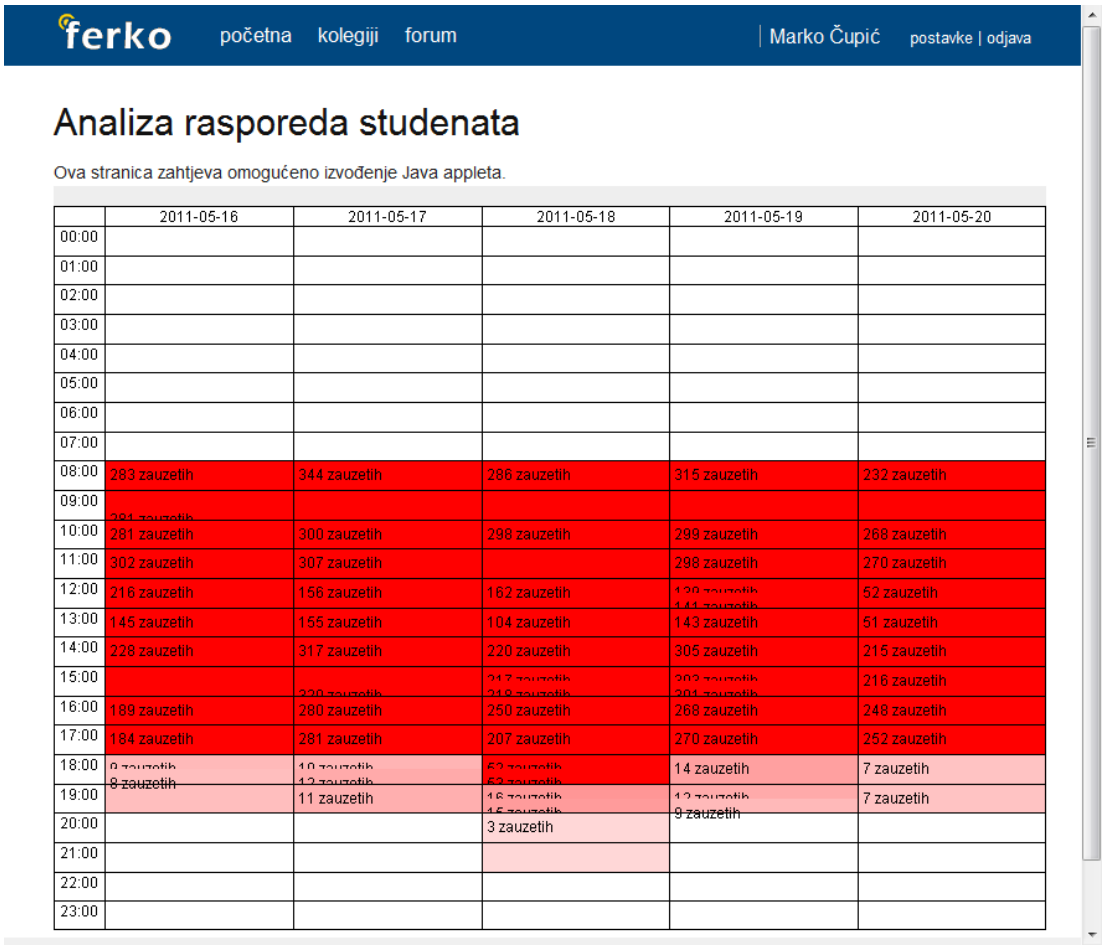
Primjer kalendara prikazanog u sustavu *Ferko* prikazan je na slici 7.3. U tom prikazu, plavom bojom su prikazani termini koji odgovaraju predavanjima. Klikom miša na neki od tih termina korisnik se vodi na stranicu dotičnog kolegija. Zelenom bojom prikazani su termini laboratorijskih vježbi. Klik miša na takve termine korisnika vodi na web stranicu odgovarajuće laboratorijske vježbe. Osim prikazanih, sustav crvenom bojom prikazuje termine provjera znanja, sivom bojom termine prezentacija seminarskih radova itd. Brzo dekodiranje vrste događaja te slaganje odgovarajuće poveznice moguće je bez ikakvih dodatnih analiza upravo zahvaljujući prethodno spomenutom kontekstu koji je pohranjen uz svaki događaj (svojstvo *context* razreda *ApstraktniDogađaj*).

Svi djelatnici u sustavu *Ferko* imaju mogućnost obavljanja analize zauzeća studenata; primjer takve analize prikazan je na slici 7.4 gdje je provedena analiza zauzeća na kolegiju s 640 studenata. Različite nijanse crvene boje prikazuju termine u kojima je više ili manje studenata zauzeto. Bijelom bojom su prikazani slobodni termini. Ostvaren je interaktivan prikaz koji korisniku omogućava da klikom na termin koji ga zanima dobije i točan popis studenata koji su u tom terminu zauzeti. Broj tih studenata prikazan je za svaki termin i u grafičkom pregledniku. Ovakve analize obično se rade za manji broj studenata kada je potrebno dogovoriti termin zajedničkog sastanka (primjeri su sastanci mentora s diplomandima, sastanci mentora sa studentima za seminar i slično). Analize se za veći broj studenata rade kada je potrebno organizirati nadokande, dodatna predavanja i slično. Tada se umjesto grafičkog preglednika pokreće dohvat rezultata analize u tekstovnom obliku koji potom služi kao ulaz u neki od daljnjih algoritama raspoređivanja.

Osim analize zauzeća korisnika, u sustavu *Ferko* implementirana je i analiza zauzeća svih prostorija koja se korisnicima nudi u dva oblika: kao grafički prikaz kalendara (slika 7.5) ili kao živa poveznica na *iCal* format zauzeća te prostorije koji je moguće koristiti u



Slika 7.3: Naslovnica sustava *Ferko* s prikazanim kalendarom.

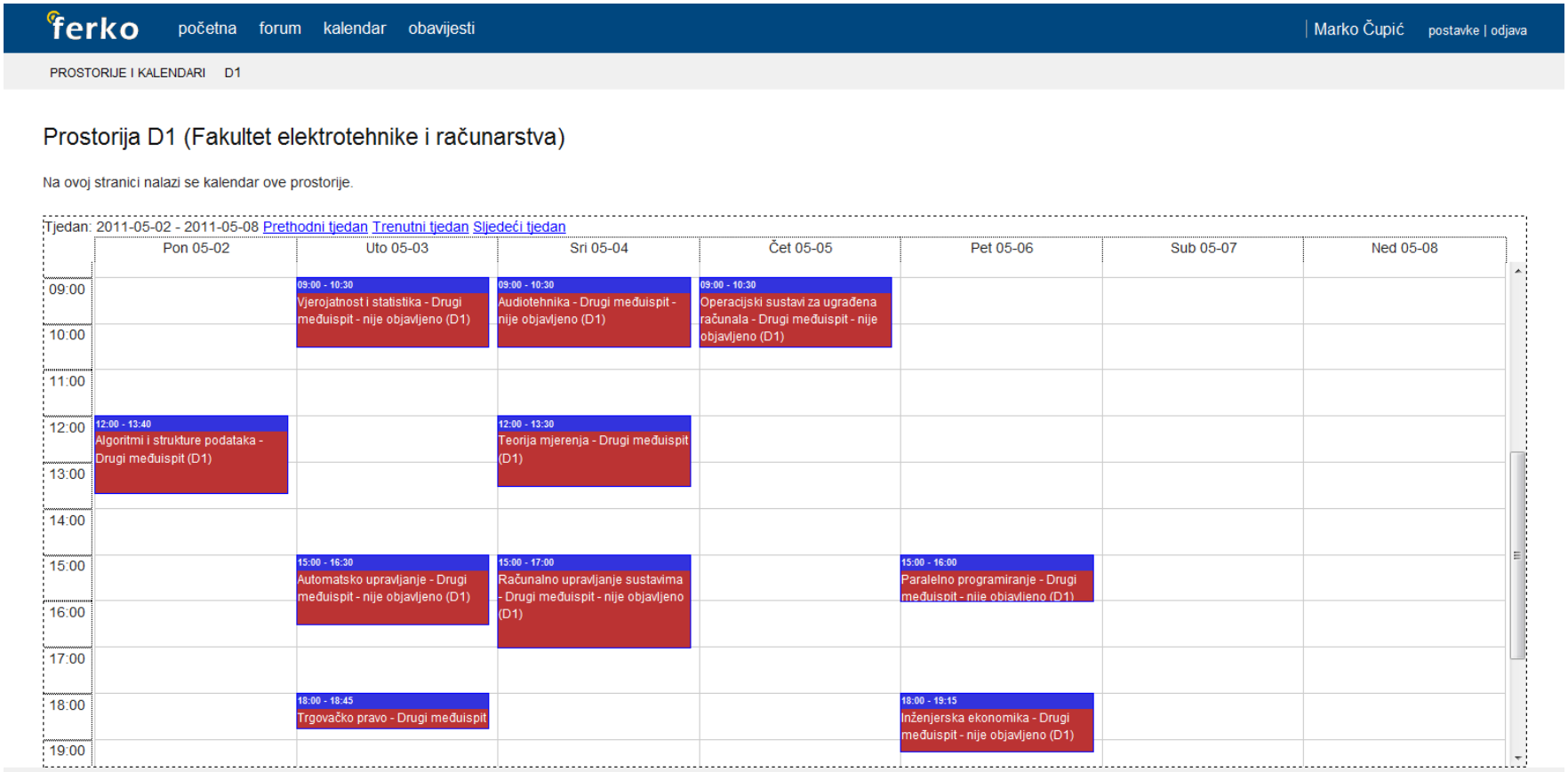


Slika 7.4: Analiza zauzeća korisnika u sustavu *Ferko*.

svim modernim organizatorima vremena. U slučaju grafičkog prikaza sustav *Ferko* nudi i dodatne informacije, gdje je to moguće. Primjerice, na slici 7.5 su prikazana zauzeća dvorane D1 u tjednu od 2. svibnja 2011. do 5. svibnja 2011. što je tjedan namijenjen provođenju drugog međuispita. Svi prikazani termini su termini u kojima je ta dvorana rezervirana upravo za provođenje drugog međuispita što je naznačeno i crvenom bojom termina. Osim te informacije, sustav je za svaki prikazani ispit provjerio i je li raspored po studentima na tim kolegijima načinjen i objavljen, ili još nije. Tako se vidi da je u trenutku pristupanja ovom kalendaru kolegij Algoritmi i strukture podataka raspored načinio i objavio, dok kolegij Vjerojatnost i statistika to još nije učinio.

Ostale mogućnosti sustava *Ferko* poput burze grupa, korisničkih stranica kolegija (engl. *wiki-pages*) te niz drugih ovdje dalje nisu opisane jer nisu u direktnoj vezi s fokusom ove disertacije (raspoređivanje nastavnih obaveza).

Sustav *Ferko* implementiran je uporabom programskog jezika Java i standardnih tehnologija koje ta platforma nudi. Za pohranu podataka moguće je koristiti bilo koju od uobičajeno dostupnih relacijskih baza podataka. Produkcijska verzija sustava *Ferko* koja se koristi na Fakultetu elektrotehnike i računarstva koristi relacijsku bazu podataka *MySql*. Pri tome se u izvornom tekstu programa nigdje direktno ne komunicira s bazom uporabom jezika SQL niti se radi s relacijama. Umjesto toga koristi se danas standardna tehnologija JPA pri čemu je JPA kratica od *Java Persistence API* te jezik JPA-QL (odnosno *JPA Query Language*). Tehnologija JPA definira sučelje temeljem kojeg se objekti mogu zapisivati u relacijsku bazu podataka tako da programer sustava čitavo vrijeme radi s objektima a pomoćna biblioteka koja je napisana u skladu s normom JPA obavlja preslikavanje tih objekata u relacijsku bazu podataka odnosno prilikom čitanja reverzni postupak: sastavljanje objekata temeljem podataka zapisanih u relacijskoj bazi podataka. Kao implementacija tehnologije JPA korištena je biblioteka *Hibernate*. Jezik JPA-QL koji je korišten za komunikaciju s bazom pripada u jezike više razine u odnosu na jezik SQL te omogućava postavljanje upita na razini objekata. Zadatak odabrane implementacije tehnologije JPA tada je ujedno i razrješavanje ovakvih upita u odgovarajuće upite jezika SQL.



Slika 7.5: Naslovnica sustava *Ferko* s prikazanim kalendarom studenta Ivana Horvata.

7.3 Pregled ostvarenih programskih rješenja i algoritama

U okviru ove disertacije opisana je uporaba algoritama evolucijskog računanja na probleme raspoređivanja nastavnih obaveza. Osim ostvarenih algoritama izgrađen je i niz programskih sustava koji se temelje na definiranom modelu podataka ili koji služe za potporu ili izvođenje algoritama raspoređivanja. Stoga je u nastavku naveden sažeti popis ostvarenoga.

- Ostvaren je programski sustav *Ferko*. Razvoj sustava *Ferko* započeo je tijekom ljeta 2008. godine. Prva verzija sustava *Ferko* puštena je u pogon početkom zimskog semestra akademske godine 2008./2009. i od tada se sustav kontinuirano koristi i nadograđuje. Danas je ovo jedan od temeljnih informatičkih sustava Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu. U razvoju i izgradnji ovog sustava sudjelovao je i niz studenata (podatci su dostupni na mrežnom sjedištu sustava).

- Ostvarena je web-aplikacija putem koje se unose zahtjevi za izradu rasporeda laboratorijskih vježbi. Aplikacija korisnicima omogućava detaljnu specifikaciju zahtjeva koje je potrebno uvažiti prilikom izrade rasporeda laboratorijskih vježbi. Aplikacija je potom integrirana u sustav *Ferko*. Putem ove aplikacije asistenti svih kolegija Fakulteta elektrotehnike i računarstva koji imaju laboratorijske vježbe na početku semestra unose svoje zahtjeve. Potom se temeljem tih zahtjeva provodi izrada rasporeda laboratorijskih vježbi. Web-aplikacija za unos zahtjeva za laboratorijskim vježbama u uporabi je od zimskog semestra akademske godine 2008./2009. Ovu web-aplikaciju su u okviru seminarskih radova implementirali studenti Mihej Komar i Toni Pivčević.

- Načinjen je sustav za izradu rasporeda laboratorijskih vježbi. U okviru tog sustava riješeno je povezivanje s aplikacijom za unos zahtjeva za laboratorijskim vježbama, riješeno je prikupljanje podataka o slobodnim terminima dvorana preko sustava FERWeb i riješen je dohvat postojećih zauzeća studenata analizom kalendara sustava *Ferko*. Osmišljena su i implementirana dva algoritma za rješavanje problema izrade rasporeda laboratorijskih vježbi temeljena na evolucijskom računanju. Prvi algoritam temeljen je na genetskom algoritmu. U njegovoj izradi još su sudjelovali i studenti Zlatko Bratković, Tomislav Herman i Vjera Omrčen te doc.dr.sc.

Domagoj Jakobović. Drugi algoritam temeljen je na algoritmu *Max-Min* mravlji sustav a u njegovoj izradi još su sudjelovali i studenti Goran Molnar, Vatroslav Dino Matijaš te prof.dr.sc. Bojana Dalbelo Bašić. O oba rješenja objavljeni su radovi na međunarodnim znanstvenim skupovima. Prva primjena ovog sustava na razini Fakulteta elektrotehnike i računarstva bila je u ljetnom semestru akademske godine 2007./2008. i od tada se sustav neprekidno koristi.

- U okviru sustava *Ferko* načinjen je modul koji djelatnicima omogućava automatiziranu izradu rasporeda za neke jednostavnije slučajeve koji su ograničeni na studente jednog kolegija (tipičan primjer je organiziranje nadoknada nastavnih aktivnosti i slično). U okviru predmeta Projekt te u okviru završnih radova implementaciju su načinili studenti Mislav Bobesić, Filip Boltužić, Denis Ćutić, Tin Franović i Siniša Pribil uz dodatnu pomoć Ivana Ferdelje, dipl.ing.

- Načinjene su prototipne verzije algoritama za rješavanje problema izrade rasporeda provjera znanja. Uporabom ovih algoritama po prvi puta je u ljetnom semestru akademske godine 2008./2009. načinjen raspored svih obaveznih provjera znanja: 1. međuispita, 2. međuispita, završnog ispita te ponovljenih provjera. Od zimskog semestra akademske godine 2009./2010. a u okviru diplomskih radova studenata Miheja Komara te Đorđa Grbića započet je rad na daljnjem ispitivanju algoritama evolucijskog računanja primijenjenih na problem izrade rasporeda provjera znanja te na izgradnji okvira za paralelizaciju tih algoritama. Uporabom tako izgrađenog sustava rasporedi se izrađuju i objavljuju posljednjih nekoliko semestara. Ostvarena je i web-aplikacija koja nakon svakog semestra studentima omogućava glasanje o težini provedenih provjera znanja na svakom od kolegija. Ti se podatci potom koriste pri generiranju rasporeda provjera znanja za sljedeću godinu kako bi se teži kolegiji više razmaknuli i eventualno isprepleli s lakšim kolegijima.

- Programski je ostvaren algoritam za izradu rasporeda prostorija za provjere znanja koja se temelji na genetskom algoritmu. Sustav *Ferko* dodatno je proširen kako bi se djelatnici svakog od kolegija koji ima provjere znanja prije dodjele dvorana mogli izjasniti o broju studenata koji su spremni staviti u svaku od raspoloživih dvorana te o broju djelatnika koji će rasporediti u svaku od raspoloživih dvorana. Temeljem tih podataka i generiranog rasporeda provjera znanja genetskim se

algoritmom rješava problem izrade rasporeda prostorija za provjere znanja kojem je cilj minimizirati potrošnju djelatnika te osigurati da svi kolegiji dobiju dovoljan broj dvorana kako bi se ispiti mogli provesti. Proteklih nekoliko semestara uporabom ovog algoritma provodi se dodjela dvorana za sve kolegije na Fakultetu elektrotehnike i računarstva. Jednom načinjeni raspored objavljuje se u sustavu *Ferko* tako da djelatnici svakog od kolegija trebaju još samo odlučiti kako žele rasporediti studente (abecedno ili slučajno) te objaviti taj raspored u kalendarima studenata.

- Ostvareni su algoritmi temeljeni na algoritmima evolucijskog računanja za rješavanje problema jednostavnog raspoređivanja koji osim razrješavanja konflikata vode računa i o kvaliteti generiranih rasporeda. Ovi se algoritmi koriste za rješavanje niza povremenih problema raspoređivanja koji se javljaju svaki semestar.

- Ostvareni su algoritmi temeljeni na algoritmima evolucijskog računanja koji omogućavaju raspoređivanje naknadno upisanih studenata po grupama za predavanja pazeći pri tome na postojeća ograničenja kapaciteta grupa. Ispitana je uporaba genetskog algoritma te algoritma klonske selekcije na ovome problemu. U posljednjih nekoliko semestara ovi se algoritmi koriste u koordinaciji sa studentskom službom za provođenje upisa na Fakultetu elektrotehnike i računarstva.

- Sustav *Ferko* proširen je s dodatnim modulom koji je na raspolaganju djalatnicima studentske službe a koji omogućava da se ažurira upis studenta na jedan ili više predmeta u *Ferku* te da se potom studentu dodijele odgovarajuće grupe za predavanja pazeći da time student može pohađati sva predavanja te da su zadovoljena ograničenja na kapacitete grupa. Sustav operateru nudi više alternativnih rješenja (ako takva postoje). Ako nije moguće načiniti dodjelu grupa bez kršenja ograničenja kapaciteta, sustav će operateru ponuditi izbor najboljih mogućih rješenja.

- Ostvaren je algoritam temeljen na genetskom algoritmu za uklanjanje konflikata u satnici za predavanja na razini studenata. Taj je algoritam u ljetnom semestru akademske godine 2010./2011. iskorišten za korekciju dodijeljenih grupa studentima koji su imali jednu ili više kolizija u rasporedu predavanja. Naknadnom analizom preostalih kolizija praktički se je došlo do skupa studenata koji imaju

upisanu kombinaciju kolegija kod koje konflikte nije moguće razriješiti i koji bi u načelu trebali ispisati jedan od konfliktnih predmeta.

- Uporabom algoritma klonske selekcije ostvaren je algoritam za rješavanje problema raspoređivanja timova. Uporabom ovog algoritma svake se akademske godine obavlja raspoređivanje predaje projekata u tri ciklusa na jednom od kolegija Fakulteta elektrotehnike i računarstva.

- Kao dokaz primjenjivosti evolucijskih algoritama i na problem izrade prezentacijskih grupa ostvaren je algoritm koji se temelji na višekriterijskoj optimizaciji uporabom genetskog algoritma. Konkretno, ostvaren je algoritam NSGA-II koji istovremeno pokušava optimirati kvalitetu raspodjele rangova unutar studentskih makrogrupa te količinu vremena u kojem su studenti svake makrogrupe raspoloživi za prezentacije.

Poglavlje 8

Zaključak

Raspoređivanje nastavnih aktivnosti težak je kombinatorički optimizacijski problem s kojim su obrazovne ustanove suočene gotovo svakodnevno. U znanstvenoj literaturi uobičajeno se razmatraja samo manji podskup ovakvih problema, pri tome najčešće samo problem izrade rasporeda predavanja na sveučilištu te problem izrade rasporeda ispita. Rjeđe se nailazi još i na spomen problema izrade rasporeda predavanja za osnovne škole.

U praksi postoji čitav niz različitih problema raspoređivanja nastavnih aktivnosti o kojima nema dostupne znanstvene literature. Dodatno, postupci raspoređivanja nisu međusobno nezavisni već rezultat jednog postupka utjeće na rezultate sljedećeg. Zajedničko svim tim problemima raspoređivanja je da postoje u gotovo svakoj obrazovnoj instituciji u određenoj mjeri, nije ih moguće rješavati ručno i dobiti kvalitetno rješenje (a često i uopće nekakvo rješenje) te naposlijetku, rješenja su nužno potrebna.

Stoga je u okviru prvog dijela ove disertacije načinjen opsežan pregled niza problema raspoređivanja nastavnih aktivnosti koji je rezultirao definiranjem formalnih modela osam čestih problema raspoređivanja nastavnih obaveza i još jednog pridruženog podproblema.

Sljedeće što je ispitano u okviru ove disertacije je primjenjivost algoritama evolucijskog računanja na svaki od opisanih problema. Naime, suočeni s tolikim brojem različitih problema raspoređivanja, jasno je da se ne isplati pisati specijalizirane postupke za rješavanje svakog od njih. Jedan od razloga je i to što ovaj broj niti približno nije gotov i uvijek su moguće varijacije i podvrste problema koje specijalizirani postupci neće moći rješavati. Stoga je jedna od postavljenih hipoteza bila da su algoritmi

evolucijskog računanja primjenjivi na sve opisane probleme raspoređivanja te da, unatoč *No-Free-Lunch* teoremu, daju dovoljno dobra rješenja koja se mogu primjenjivati u svakodnevnom životu. U okviru provedenog ispitivanja za sve su opisane probleme raspoređivanja ostvareni su algoritmi evolucijskog računanja temeljem kojih je provedeno ispitivanje. Implementacije za slučaj problema jednostavnog raspoređivanja te problema izrade rasporeda provjera znanja opisane su i u okviru ove disertacije, a o primjenjivosti opisanih algoritama najbolje svjedoči njihova svakodnevna uporaba.

Algoritmi evolucijskog računanja nemaju nikakvih jamstava da će u kraćem ograničenom vremenu pronaći dovoljno dobro rješenje da bi ono bilo primjenjivo. Kako bi se povećala vjerojatnost pronalaska takvih rješenja nužno je osigurati da algoritmi ipak pretraže dovoljno velik prostor rješenja. Ako je vrijeme unutar kojeg je pretragu moguće raditi ograničeno, tada se povećanje prostora koji će biti pretražen postiže paralelizacijom na više procesora ili čak izgradnjom raspodijeljenog algoritma koji koristi više računala. U okviru ove disertacije opisane su mogućnosti paralelizacije algoritama evolucijskog računanja s osvrtom na prethodno definirane probleme raspoređivanja.

Temeljem provedenih istraživanja uočeno je i da različiti algoritmi evolucijskog računanja pretraživanje obavljaju na različit način koristeći različite operatore. Stoga je postavljena hipoteza da će hibridni algoritam koji pretragu radi uporabom više različitih algoritama evolucijskog računanja raditi barem jednako dobro kao i izolirani algoritmi evolucijskog računanja, a da će često davati i bolje rezultate. U okviru ove disertacije provedeni su eksperimenti na dva različita problema raspoređivanja koji su potvrdili postavljenu hipotezu.

Konačno, kako bi se pružila potpora za provođenje procesa raspoređivanja definiran je model podataka koji s jedne strane služi za pohranu rezultata algoritama raspoređivanja (odnosno objavu načinjenih rasporeda) a s druge strane omogućava pribavljanje podataka potrebnih za sljedeće postupke raspoređivanja (analiza zauzeća korisnika). U okviru ove disertacije definiran je osnovni model podataka koji omogućava objavu rezultata te analizu zauzeća. Ostvarivost ovog modela isprobana je i u praksi; temeljem opisanog modela i njegovog proširenja izgrađen je web-temeljeni sustav za upravljanje kolegijima *Ferko* koji se u trenutku pisanja ove disertacije uspješno koristi već pune tri godine (odnosno šest semestara) od strane nekoliko stotina djelatnika i više tisuća studenata za organizaciju svakodnevnih nastavnih aktivnosti.

U okviru ove disertacije ostvareni su sljedeći znanstveni doprinosi:

1. definiran je formalni model specifičnih optimizacijskih problema raspoređivanja s primjenom na organizaciju nastavnih procesa na fakultetu,
2. razrađen je prijedlog i ostvareni su algoritmi za rješavanje prikazanih problema raspoređivanja nastavnih obaveza temeljeni na evolucijskom računanju,
3. razrađen je prijedlog i ostvareni su paralelni algoritmi za rješavanje problema raspoređivanja nastavnih aktivnosti temeljeni na problemski specifičnoj paralelizaciji kao i pristupi temeljeni na otočnom modelu i uporabi raznorodnih evolucijskih algoritama te
4. definirana je arhitektura i ostvaren je prototip sustava koji nudi potporu za objavu načinjenih rasporeda te analizu zauzeća korisnika koja je potrebna za sljedeće cikluse raspoređivanja.

Kako bi se provjerila uporabivost i svojstva algoritama evolucijskog računanja na rješavanje problema raspoređivanja nastavnih obaveza te uporabivost predloženog modela podataka, izgrađen je web-programski sustav te je ostvaren niz algoritama evolucijskog računanja za rješavanje problema raspoređivanja nastavnih obaveza. Konkretno, ostvareno je sljedeće:

- ostvaren je web-programski sustav *Ferko* koji omogućava objavu načinjenih rasporeda i omogućava analizu zauzeća korisnika,
- ostvarena je web-aplikacija putem koje se unose zahtjevi za izradu rasporeda laboratorijskih vježbi temeljem kojih se potom generira raspored laboratorijskih vježbi,
- načinjen je sustav za izradu rasporeda laboratorijskih vježbi uporabom genetskog algoritma te uporabom algoritma Max-Min mravlji sustav,
- u okviru sustava *Ferko* načinjen je modul koji djelatnicima omogućava automatiziranu izradu rasporeda za jednostavnije slučajeve problema raspoređivanja,
- načinjeno je više algoritama za rješavanje problema izrade rasporeda provjera znanja koji se temelje na algoritmima evolucijskog računanja te raspodijeljeni sustav za izvođenje algoritama evolucijskog računanja,

- programski je ostvaren algoritam za izradu rasporeda prostorija za provjere znanja koja se temelji na genetskom algoritmu,
- ostvareni su algoritmi temeljeni na algoritmima evolucijskog računanja za rješavanje problema jednostavnog raspoređivanja koji osim razrješavanja konflikata vode računa i o kvaliteti generiranih rasporeda,
- ostvareni su algoritmi temeljeni na algoritmima evolucijskog računanja koji omogućavaju raspoređivanje naknadno upisanih studenata po grupama za predavanja pazeći pri tome na postojeća ograničenja kapaciteta grupa,
- sustav *Ferko* proširen je s dodatnim modulom koji je na raspolaganju djalatnicima studentske službe a koji omogućava da se ažurira upis studenta na jedan ili više predmeta u *Ferku* te da se potom studentu dodijele odgovarajuće grupe za predavanja pazeći da time student može pohađati sva predavanja te da su zadovoljena ograničenja na kapacitete grupa,
- ostvaren je algoritam temeljen na genetskom algoritmu za uklanjanje konflikata u satnici za predavanja na razini studenata,
- uporabom algoritma klonske selekcije ostvaren je algoritam za rješavanje problema raspoređivanja timova te je
- kao dokaz primjenjivosti evolucijskih algoritama i na problem izrade prezentacijskih grupa ostvaren algoritam koji riješava taj problem i koji se temelji na višekriterijskoj optimizaciji.

Temeljem provedenih eksperimenata opisanih u ovoj disertaciji može se zaključiti da su algoritmi zasnovani na evolucijskom računanju prikladni za rješavanje prikazanih problema raspoređivanja nastavnih aktivnosti. Nedostatak populacijskih algoritama evolucijskog računanja pri tome je relativno visok trošak izvođenja koji za slijedne algoritme znači dugo vrijeme izvođenja prije no što se postignu zadovoljavajuća rješenja. Ovaj se problem može ublažiti paralelizacijom algoritama na više procesora unutar istog računala ili raspodijeljenim izvođenjem na više računala. Također, kako pokazuju provedeni eksperimenti, uporaba hibridnog algoritma evolucijskog računanja može postizati kvalitetne rezultate uz malo standardno odstupanje.

Unatoč tome, ova disertacija ostavlja otvorenim i niz pitanja koja zahtjevaju daljnja istraživanja. Samo neka od takvih su istražiti koji su razlozi uspješne ili neuspješne suradnje različitih metaheurističkih algoritama, istražiti utjecaj parametara migracije kod raspodijeljenog hibridnog algoritma evolucijskog računanja na kvalitetu i stabilnost dobivenih rješenja, utvrditi prikladnost različitih topologija kod ovakvog algoritma i slično.

Literatura

[Affenzeller et al., 2009] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic algorithms and Genetic programming. Modern Concepts and Practical Applications.* CRC Press, Boca Raton, USA, 2009.

[Al-Betar et al., 2010] M. Al-Betar, A. Khader, and I. Liao. A harmony search with multi-pitch adjusting rate for the university course timetabling. In Z. Geem, editor, *Recent Advances In Harmony Search Algorithm*, volume 270 of *Studies in Computational Intelligence*, pages 147–161. Springer Berlin / Heidelberg, 2010. URL http://dx.doi.org/10.1007/978-3-642-04317-8_13. 10.1007/978-3-642-04317-8_13.

[Aladag et al., 2009] C. H. Aladag, G. Hocaoglu, and M. A. Basaran. The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem. *Expert Systems with Applications*, 36(10):12349–12356, 2009. ISSN 0957-4174. doi: DOI:10.1016/j.eswa.2009.04.051. URL http://www.sciencedirect.com/science/article/B6V03-4W7RYBY-1/2/4413c4b8c46627f7c89bf3058a30ff6d.

[Alba and Dorronsoro, 2008] E. Alba and B. Dorronsoro. *Cellular Genetic Algorithms*, volume 42 of *Series Operations Research/Computer Science Interfaces Series*. Springer, 2008.

[Alvarez-Valdes et al., 2002] R. Alvarez-Valdes, E. Crespo, and J. M. Tamarit. Design and implementation of a course scheduling system using tabu search. *European Journal of Operational Research*, 137(3):512–523, 2002. ISSN 0377-2217. doi: DOI: 10.1016/S0377-2217(01)00091-1. URL http://www.sciencedirect.com/science/article/B6VCT-44SHF9C-4/2/ffc575b8713110a537eeecc22c7b342b.

[Amdahl, 1967] G. Amdahl. Validity of the single processor approach to achieving

large-scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485. ACM Press, 1967.

[Asmuni et al., 2007] H. Asmuni, E. Burke, J. Garibaldi, and B. McCollum. A novel fuzzy approach to evaluate the quality of examination timetabling. In E. Burke and H. RudovA!, editors, *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 327–346. Springer Berlin / Heidelberg, 2007. URL http://dx.doi.org/10.1007/978-3-540-77345-0_21. 10.1007/978-3-540-77345-0_21.

[Azimi, 2004] Z. Azimi. Comparison of metaheuristic algorithms for examination timetabling problem. *Journal of Applied Mathematics and Computing*, 16: 337–354, 2004. ISSN 1598-5865. URL http://dx.doi.org/10.1007/BF02936173. 10.1007/BF02936173.

[Back et al., 1997] T. Back, U. Hammel, and H.-P. Schwefel. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, 1997.

[Bader-El-Den et al., 2009] M. Bader-El-Den, R. Poli, and S. Fatima. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1:205–219, 2009. ISSN 1865-9284. URL http://dx.doi.org/10.1007/s12293-009-0022-y. 10.1007/s12293-009-0022-y.

[Banks et al., 2007] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization. part i: background and development. *Natural Computing*, 6:467–484, 2007. ISSN 1567-7818. URL http://dx.doi.org/10.1007/s11047-007-9049-5. 10.1007/s11047-007-9049-5.

[Banks et al., 2008] A. Banks, J. Vincent, and C. Anyakoha. A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*, 7:109–124, 2008. ISSN 1567-7818. URL http://dx.doi.org/10.1007/s11047-007-9050-z. 10.1007/s11047-007-9050-z.

[Beligiannis et al., 2008] G. N. Beligiannis, C. N. Moschopoulos, G. P. Kaperonis, and S. D. Likothanassis. Applying evolutionary computation to the sc-

hool timetabling problem: The greek case. *Computers and Operations Research*, 35(4):1265–1280, 2008. ISSN 0305-0548. doi: DOI:10.1016/j.cor.2006.08.010. URL http://www.sciencedirect.com/science/article/B6VC5-4M340GJ-1/2/80b53518bc1b8df77673a3193e232062.

[Bobesić, 2010] M. Bobesić. Rješavanje problema izrade rasporeda nadoknada primjenom stohastičkog difuznog pretraživanja, Lipanj 2010. Završni rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[Boltužić, 2010] F. Boltužić. Primjena algoritma kolonije pčela na kombinatoričke probleme, Lipanj 2010. Završni rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[Bonabeau et al., 1997a] E. Bonabeau, A. Sobkowski, G. Theraulaz, and J. L. Deneuborg. Adaptive task allocation inspired by amodel of division of labor in social insects. In D. Lundha, B. Olsson, and A. Narayanan, editors, *Bio-Computation and Emergent Computing*, pages 36–45, Singapore, 1997a. World Scientific Publishing.

[Bonabeau et al., 1997b] E. Bonabeau, G. Theraulaz, J. L. Denebourg, S. Aron, and S. Camazine. Self-organization in social insects. *Tree*, 12(5):188–193, 1997b.

[Bonačić et al., 2009] I. Bonačić, T. Herman, T. Krznar, E. Mangić, G. Molnar, and M. Čupić. Optical character recognition of seven–segment display digits using neural networks. In D. Čišić, v. Hutinski, M. Baranović, M. Mauher, and V. Dragšić, editors, *Proceedings of MIPRO 2009 – 32st International Convention on Information and Communication Technology, Electronics and Microelectronics*, volume V Student papers, pages 323–328. DENONA, zagreb, 2009.

[Box, 1957] G. E. P. Box. Evolutionary operation: A method for increasing industrial productivity. *Appl. Statistics*, 6(2):81–101, 1957.

[Brailsford et al., 1999] S. C. Brailsford, C. N. Potts, and B. M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, 119(3):557–581, 1999. ISSN 0377-2217. doi: DOI:10.1016/S0377-2217(98)00364-6. URL http://www.sciencedirect.com/science/article/B6VCT-3XK6T2T-1/2/b28d5d1c3f636b791e109275ba50cc0f.

[Bratković et al., 2009] Z. Bratković, T. Herman, V. Omrčen, M. Čupić, and D. Jakobović. University course timetabling with genetic algorithm: A laboratory excercises case study. In C. Cotta and P. Cowling, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 5482 of *Lecture Notes in Computer Science*, pages 240–251. Springer Berlin / Heidelberg, 2009. URL http://dx.doi.org/10.1007/978-3-642-01009-5_21. 10.1007/978-3-642-01009-5_21.

[Brélaz, 1979] D. Brélaz. New methods to color the vertices of a graph. *Commun. ACM*, 22:251–256, April 1979. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/359094.359101. URL http://doi.acm.org/10.1145/359094.359101.

[Bremermann, 1962] H. J. Bremermann. Optimization through evolution and recombination. *Self-Organizing Systems*, 1962.

[Bullnheimer et al., 1999] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank-based version of the ant system: A computational study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.

[Burke and Carter, 1997] E. Burke and M. Carter, editors. *Practice and Theory of Automated Timetabling II: Second International Conference, PATAT'97, Toronto, Canada*, volume 1408 of *Lecture Notes in Computer Science*, August 1997. Springer.

[Burke et al., 1994] E. Burke, D. Elliman, and R. Weare. A genetic algorithm based university timetabling system. In *East-West Conference on Computer Technologies in Education, Crimea, Ukraine*, pages 35–40, 1994.

[Burke et al., 1997] E. Burke, K. Jackson, J. H. Kingston, and R. Weare. Automated university timetabling: The state of the art. *Computer Journal*, 40(9):565–571, 1997.

[Burke et al., 2003a] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: an emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Meta-Heuristics*, pages 457–474. Kluwer, 2003a.

[Burke and Causmaecker, 2002] E. K. Burke and P. D. Causmaecker, editors. *Practice and Theory of Automated Timetabling IV: 4th International Conference, PATAT 2002, Gent, Belgium*, volume 2740 of *Lecture Notes in Computer Science*, August 2002. Springer.

[Burke and Erben, 2000] E. K. Burke and W. Erben, editors. *Practice and Theory of Automated Timetabling III: Third International Conference, PATAT 2000, Konstanz, Germany*, volume 2079 of *Lecture Notes in Computer Science*, August 2000. Springer.

[Burke and Newall, 1999] E. K. Burke and J. P. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.

[Burke and Petrovic, 2002] E. K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266 – 280, 2002. ISSN 0377-2217. doi: DOI:10.1016/S0377-2217(02)00069-3. URL http://www.sciencedirect.com/science/article/B6VCT-45SS31C-2/2/3ce08a88e882a25b5d4e804216641a73.

[Burke and Ross, 1997] E. K. Burke and P. Ross, editors. *Practice and Theory of Automated Timetabling: First International Conference, PATAT'95, Edinburgh, U.K.*, volume 1153 of *Lecture Notes in Computer Science*, September 1997. Springer.

[Burke and Rudová, 2006] E. K. Burke and H. Rudová, editors. *Practice and Theory of Automated Timetabling VI: 6th International Conference, PATAT 2006, Brno, Czech Republic*, volume 3867 of *Lecture Notes in Computer Science*, September 2006. Springer.

[Burke and Trick, 2004] E. K. Burke and M. A. Trick, editors. *Practice and Theory of Automated Timetabling V: 5th International Conference, PATAT 2004, Pittsburgh, PA, USA*, volume 3616 of *Lecture Notes in Computer Science*, August 2004. Springer.

[Burke et al., 2003b] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9:451–470, December 2003b. ISSN 1381-1231. doi: 10.1023/B:HEUR.0000012446.94732.b6. URL http://portal.acm.org/citation.cfm?id=965845.965864.

[Burke et al., 2007] E. K. Burke, B. Mccollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.

[Burnet, 1957] F. M. Burnet. A modification of jerne's theory of antibody production using the concept of clonal selection. *Australian Journal of Science*, 20:67–69, 1957.

[Burnet, 1959] F. M. Burnet. *The clonal selection theory of acquired immunity*. Vanderbilt University Press, Nashville, Tennessee, U.S.A., 1959.

[Burnet, 1978] F. M. Burnet. Clonal selection and after. *Theoretical Immunology*, pages 63–85, 1978.

[Camazine et al., 2001] S. Camazine, J. L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, editors. *Self-organization in Biological Systems*. Princeton University Press, Princeton, NJ, 2001.

[Carter, 1983] M. Carter. A decomposition algorithm for practical timetabling problems. Technical report, Department of Industrial Engineering, University of Toronto, 1983. Paper 83-06.

[Carter and Laporte, 1996] M. Carter and G. Laporte. Recent developments in practical examination timetabling. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 3–21. Springer Berlin / Heidelberg, 1996. URL http://dx.doi.org/10.1007/3-540-61794-9_49. 10.1007/3-540-61794-9_49.

[Carter and Laporte, 1998] M. Carter and G. Laporte. Recent developments in practical examination timetabling. In E. Burke and M. Carter, editors, *Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*, pages 3–19. Springer Berlin / Heidelberg, 1998. URL http://dx.doi.org/.

[Carter, 1986] M. W. Carter. A survey of practical applications of examination timetabling algorithms. *Oper. Res.*, 34:193–202, March 1986. ISSN 0030-364X. doi: 10.1287/opre.34.2.193. URL http://portal.acm.org/citation.cfm?id=11117.11118.

[Carter, 2001] M. W. Carter. A comprehensive course timetabling and student scheduling system at the university of waterloo. In *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, PATAT '00, pages 64–84, London, UK, 2001. Springer-Verlag. ISBN 3-540-42421-0. URL http://portal.acm.org/citation.cfm?id=646431.692902.

[Carter et al., 1994] M. W. Carter, G. Laporte, and J. W. Chinneck. A general examination scheduling system. *Interfaces*, 24(3):109–120, 1994. ISSN 00922102.

[Chiarandini et al., 2006] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9:403–432, 2006. ISSN 1094-6136. URL http://dx.doi.org/10.1007/s10951-006-8495-8. 10.1007/s10951-006-8495-8.

[Clerc, 1999] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proc. 1999 Congress on Evolutionary Computation, Washington, DC*, pages 1951–1957, Piscataway, NJ: IEEE Service Center, 1999.

[Colorni et al., 1991] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *Proceedings of European conference on artificial life*, pages 134–142, Paris, France, 1991.

[Colorni et al., 1992] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 134–142, Cambridge, MA, 1992. MIT Press.

[Colorni et al., 1998] A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for high school timetabling. *Computational Optimization and Applications*, 9:275–298, 1998. ISSN 0926-6003. URL http://dx.doi.org/10.1023/A:1018354324992. 10.1023/A:1018354324992.

[Cordon et al., 2002] O. Cordon, F. Herrera, and T. Stützle. A review on the ant colony optimization metaheuristic: Basis, models and new trends. *Mathware and Soft Computing*, 9:141–175, 2002.

[Corne et al., 1993] D. Corne, H.-L. Fang, and C. Mellish. Solving the modular exam scheduling problem with genetic algorithms. In *Proceedings of the 6th international conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 370–373. Gordon & Breach Science Publishers, 1993. ISBN 2-88124-604-4. URL http://portal.acm.org/citation.cfm?id=1114022.1114080.

[Cutello and Nicosia, 2002a] V. Cutello and G. Nicosia. An immunological approach to combinatorial optimization problems. In *Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence, Seville, Spain*, pages 361–370, 2002a.

[Cutello and Nicosia, 2002b] V. Cutello and G. Nicosia. Multiple learning using immune algorithms. In *Proceedings of 4th International Conference on Recent Advances in Soft Computing, RASC 2002, Nottingham, UK*, pages 102–107, 2002b.

[Cutello and Nicosia, 2005] V. Cutello and G. Nicosia. The clonal selection principle for in silico and in vitro computing. In L. N. de Castro and F. J. V. Zuben, editors, *Recent Developments in Biologically Inspired Computing*, chapter IV, pages 104–146. Idea Group Publishing, Hershey, London, Melbourne, Singapore, 2005.

[Cutello et al., 2003] V. Cutello, G. Nicosia, and M. Pavone. A hybrid immune algorithm with information gain for the graph coloring problem. In *GECCO '03, Chicago, IL, USA*, volume 2723, pages 171–182. Springer, 2003.

[Cutello et al., 2005] V. Cutello, G. Narzisi, G. Nicosia, and M. Pavone. Clonal selection algorithms: A comparative case study using effective mutation potentials. In *ICARIS 2005*, volume 3627, pages 13–28. Springer, 2005.

[Cutello et al., 2007] V. Cutello, G. Nicosia, P. S. Oliveto, and M. Romeo. On the convergence of immune algorithms. In *The First IEEE Symp. on Foundations of Computational Intelligence, FOCI 2007*, pages 409–415, Honolulu, Hawaii, USA, 2007. IEEE Press.

[Ćutić, 2010] D. Ćutić. Primjena algoritma roja čestica na problem izrade rasporeda nadoknada, Lipanj 2010. Završni rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[Darwin, 1859] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* John Murray, 1st edition, 1859.

[Dasgupta and Niño, 2009] D. Dasgupta and L. F. Niño. *Immunological Computation. Theory and Applications.* CRC Press, Boca Raton, USA, 2009.

[de Castro and Timmis, 2002] L. N. de Castro and J. Timmis. *Artificial Immune Systems: A new computational intelligence approach.* Springer-Verlag, Great Britain, 2002.

[de Castro and Zuben, 2000] L. N. de Castro and F. J. V. Zuben. The clonal selection algorithm with engineering applications. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00), Workshop on Artificial Immune Systems and Their Applications*, pages 36–37, Las Vegas, Nevada, USA, 2000.

[de Castro and Zuben, 2002] L. N. de Castro and F. J. V. Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation*, 6:239–251, June 2002.

[de Castro and Zuben, 1999] L. N. de Castro and F. J. V. Zuben. Artificial immune systems - part i: Basic theory and applications. Technical report, Department of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil, December 1999. TR DCA 01/99.

[de Werra, 1985] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985. ISSN 0377-2217. doi: DOI:10.1016/0377-2217(85)90167-5. URL http://www.sciencedirect.com/science/article/B6VCT-48NBKHD-32Y/2/87ecd333ea5bd503f592389b1ad7218e.

[Deb, 2009] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, West Sussex, United Kingdom, 2009.

[Denebourg et al., 1990] J. L. Denebourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behaviour*, 3:159–168, 1990.

[Denzinger et al., 1997] J. Denzinger, M. Fuchs, and M. Fuchs. High performance atp systems by combining several ai methods. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '97)*, pages 102–107, 1997.

[Di Gaspero and Schaerf, 2001] L. Di Gaspero and A. Schaerf. Tabu search techniques for examination timetabling. In E. Burke and W. Erben, editors, *Practice and Theory of Automated Timetabling III*, volume 2079 of *Lecture Notes in Computer Science*, pages 104–117. Springer Berlin / Heidelberg, 2001. URL http://dx.doi.org/10.1007/3-540-44629-X_7. 10.1007/3-540-44629-X_7.

[Dino Matijaš et al., 2010] V. Dino Matijaš, G. Molnar, M. Čupić, D. Jakobović, and B. Dalbelo Bašić. University course timetabling using aco: A case study on laboratory

exercises. In R. Setchi, I. Jordanov, R. Howlett, and L. Jain, editors, *Knowledge-Based and Intelligent Information and Engineering Systems*, volume 6276 of *Lecture Notes in Computer Science*, pages 100–110. Springer Berlin / Heidelberg, 2010. URL http://dx.doi.org/10.1007/978-3-642-15387-7_14. 10.1007/978-3-642-15387-7_14.

[Dorigo and Stützle, 2004] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[Dorigo et al., 1991] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. Technical report, Dipartimento di Elettronica, Politecnico di Milano, Milano, 1991. Technical report 91-016.

[Dorigo et al., 1996] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 26(1):29–41, 1996.

[Dowsland and Thompson, 2005] K. Dowsland and J. Thompson. Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society*, 56:426—-438, 2005.

[Duong and Lam, 2004] T.-A. Duong and K.-H. Lam. Combining constraint programming and simulated annealing on university exam timetabling, 2004.

[Eberhart and Kennedy, 1995] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, Japan, 1995. Piscataway, NJ: IEEE Service Center.

[Eberhart and Shi, 2001] R. C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proc. Congress on Evolutionary Computation 2001, Seoul, Korea*, pages 81–86, Piscataway, NJ: IEEE Service Center, 2001.

[Eberhart et al., 1996] R. C. Eberhart, P. K. Simpson, and R. W. Dobbins. *Computational Intelligence PC Tools*. Academic Press Professional, Boston, MA, 1996.

[Eley, 2007] M. Eley. Ant algorithms for the exam timetabling problem. In *Proceedings of the 6th international conference on Practice and theory of automated timetabling*

VI, PATAT'06, pages 364–382, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-77344-4, 978-3-540-77344-3. URL http://portal.acm.org/citation.cfm?id=1782534.1782564.

[Erben and Keppler, 1996] W. Erben and J. Keppler. A genetic algorithm solving a weekly course-timetabling problem. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 198–211. Springer Berlin / Heidelberg, 1996. URL http://dx.doi.org/10.1007/3-540-61794-9_60. 10.1007/3-540-61794-9_60.

[Even et al., 1976] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computation*, 5(4):691–703, 1976.

[Feoktistov, 2006] V. Feoktistov. *Differential Evolution: In Search of Solutions*, volume 5 of *Optimization and Its Applications*. Springer, New York, Inc. Secaucus, NJ, USA, 2006.

[Fogel, 1962] L. J. Fogel. Autonomous automata. *Ind. Res.*, 4:14–19, 1962.

[Fogel et al., 1966] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.

[Fowler and Scott, 2003] M. Fowler and K. Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2003. ISBN 0-321-19368-7.

[Franović, 2010] T. Franović. Rješavanje problema izrade rasporeda nadoknada primjenom algoritma klonske selekcije, Lipanj 2010. Završni rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[Friedberg, 1958] R. M. Friedberg. A learning machine: Part i. *IBM J.*, 2(1):2–13, 1958.

[Friedberg et al., 1959] R. M. Friedberg, B. Dunham, and J. H. North. A learning machine: Part ii. *IBM J.*, 3(7):282–287, 1959.

[Garey and Johnson, 1979] M. Garey and D. Johnson. *Computers and Intractability, A guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.

[Geem, 2010] Z. W. Geem, editor. *Recent Advances in Harmony Search Algorithm*, volume 270 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, 1st edition, 2010.

[Glavinić et al., 2008] V. Glavinić, M. Čupić, and S. Groš. Studtest – a platform supporting complex and interactive knowledge assessment. *International Journal of Emerging Technologies in Learning (iJET)*, 3:33–39, 2008.

[Glover and Kochenberger, 2003] F. Glover and G. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Boston, MA, 2003.

[Goss et al., 1989] S. Goss, S. Aron, J.-L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76:579–581, 1989.

[Grbić, 2011] D. Grbić. Utjecaj parametara algoritama evolucijskog računanja na kvalitetu rješenja za problem rasporeda međuispita, 2011. Diplomski rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu, očekivani mjesec obrane: lipanj.

[Haddad et al., 2006] O. Haddad, A. Afshar, and M. Mariño. Honey-bees mating optimization (hbmo) algorithm: A new heuristic approach for water resources optimization. *Water Resources Management*, 20:661–680, 2006. ISSN 0920-4741. URL http://dx.doi.org/10.1007/s11269-005-9001-3. 10.1007/s11269-005-9001-3.

[Haken, 1983] H. Haken. *Synergetics*. Springer-Verlag, Berlin, 1983.

[Harding and Banzhaf, 2007] S. Harding and W. Banzhaf. Fast genetic programming on GPUs. In M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, editors, *Proceedings of the 10th European Conference on Genetic Programming*, volume 4445 of *Lecture Notes in Computer Science*, pages 90–101, Valencia, Spain, 2007. Springer. ISBN 3-540-71602-5. doi: doi:10.1007/978-3-540-71605-1_9.

[Herman, 2010] T. Herman. Rješavanje problema raspoređivanja u visokoškolskim ustanovama evolucijskim algoritmom, 2010. Diplomski rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[Holland, 1962] J. H. Holland. Outline for a logical theory of adaptive systems. *J. Assoc. Comput. Mach.*, 3:297–314, 1962.

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

[Johnson and McGeoch, 1997] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John-Wiley and Sons, Ltd., 1997.

[Junginger, 1986] W. Junginger. Timetabling in germany: A survey. *Interfaces*, 16(4): 66–74, 1986.

[Karaboga and Basturk, 2007] D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39:459–471, 2007. ISSN 0925-5001. URL http://dx.doi.org/10.1007/s10898-007-9149-x. 10.1007/s10898-007-9149-x.

[Karaboga and Basturk, 2008] D. Karaboga and B. Basturk. On the performance of artificial bee colony (abc) algorithm. *Applied Soft Computing*, 8(1):687–697, 2008. ISSN 1568-4946. doi: DOI:10.1016/j.asoc.2007.05.007. URL http://www.sciencedirect.com/science/article/B6W86-4NWCGRR-G/2/422ccff5df9d32a5bf8517068ca2a094.

[Kendall and Li, 2008] G. Kendall and J. Li. Combining examinations to accelerate timetable construction. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008), August 2008.*, 2008.

[Kennedy and Eberhart, 1995] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, NJ, 1995.

[Kennedy and Eberhart, 1997] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *In Proceedings of the IEEE 1997 International Conference on Systems, Man and Cybernetics*, pages 4104–4109, 1997.

[Kennedy et al., San Francisco, USA] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm intelligence*. Morgan Kaufman, 2001, San Francisco, USA.

[Komar, 2011] M. Komar. Izgradnja platforme za raspodijeljene algoritme evolucijskog računanja, 2011. Diplomski rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu, očekivani mjesec obrane: lipanj.

[Komar et al., 2011] M. Komar, D. Grbić, and M. Čupić. Solving exam timetabling using distributed evolutionary computation. In *Proceedings of the 33st International Conference on Information Technology Interfaces*, pages ?–?, 2011. prihvaćeno za objavu.

[Kostuch, 2005] P. Kostuch. The university course timetabling problem with a three-phase approach. In E. Burke and M. Trick, editors, *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 109–125. Springer Berlin / Heidelberg, 2005. URL http://dx.doi.org/10.1007/11593577_7. 10.1007/11593577_7.

[Koza, 1992] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Langdon and Poli, 2010] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer, Berlin, Germany, 2010.

[Lee and Geem, 2004] K. S. Lee and Z. W. Geem. A new structural optimization method based on the harmony search algorithm. *Computers & Structures*, 82(9-10):781–798, 2004. ISSN 0045-7949. doi: DOI:10.1016/j.compstruc.2004.01.002. URL http://www.sciencedirect.com/science/article/B6V28-4BWMP8Y-2/2/cd0c48f22f516cc7df98796923cbe99a.

[Lee and Geem, 2005] K. S. Lee and Z. W. Geem. A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 194(36-38):3902–3933, 2005. ISSN 0045-7825. doi: DOI:10.1016/j.cma.2004.09.007. URL http://www.sciencedirect.com/science/article/B6V29-4DW3937-2/2/0b447356f1eea2414d356f14b137582b.

[Lewis, 2008] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 30:167–190, 2008. ISSN 0171-6468. URL http://dx.doi.org/10.1007/s00291-007-0097-0. 10.1007/s00291-007-0097-0.

[Lobo et al., 2000] F. G. Lobo, D. E. Goldberg, and M. Pelikan. Time complexity of genetic algorithms on exponentially scaled problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 151–158. Morgan-Kaufmann, 2000.

[Lourenço et al., 2003] H. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. S. Hillier, F. Glover, and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 320–353. Springer New York, 2003. ISBN 978-0-306-48056-0. URL http://dx.doi.org/10.1007/0-306-48056-5_11. 10.1007/0-306-48056-5_11.

[Matijaš, 2009] V.-D. Matijaš. Primjena algoritma mravlje kolonije na problem rasporeda laboratorijskih vježbi, 2009. Diplomski rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[McCollum, 2006] B. McCollum. University timetabling: Bridging the gap between research and practice. In *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling*, pages 15–35. Springer, 2006.

[Michalewicz and Fogel, 2004] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, Berlin, Germany, 2nd edition, 2004.

[Molnar, 2011] G. Molnar. Paralelni algoritam mravlje kolonije za izradu rasporeda laboratorijskih vježbi, 2011. Diplomski rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[Montes de Oca et al., 2009] M. A. Montes de Oca, T. Stützle, M. Birattari, and D. M. Frankenstein's pso: A composite particle swarm optimization algorithm. *IEEE Trans. on Evolutionary Computation*, 13(5):1120–1132, 2009.

[Nakrani and Tovey, 2004] S. Nakrani and C. Tovey. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior*, 12(3–4):223–240, 2004.

[Nau et al., 2004] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558608567.

[Nicolis and Prigogine, 1977] G. Nicolis and I. Prigogine. *Self-Organisation in Non-Equilibrium Systems*. John Wiley and Sons, New York, 1977.

[Ochoa et al., 2009] G. Ochoa, R. Qu, and E. K. Burke. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the 11th*

Annual conference on Genetic and evolutionary computation, GECCO '09, pages 341–348, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-325-9. doi: http://doi.acm.org/10.1145/1569901.1569949. URL http://doi.acm.org/10.1145/1569901.1569949.

[Omrčen, 2010] V. Omrčen. Primjena hibridnog evolucijskog algoritma na problem raspoređivanja u visokoškolskim ustanovama, 2010. Diplomski rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[P. and E., 2000] C. P. and S. E. Neighborhood structures for personnel scheduling: A summit meeting scheduling problem. In B. E.K. and E. W., editors, *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling, Constance, Germany*, page 277, August 2000.

[Pasteels et al., 1987] J. M. Pasteels, J.-L. Deneubourg, and S. Goss. Self-organization mechanisms in ant societies (i): Trail recruitment to newly discovered food sources. *Experientia Supplementum*, 54:155–175, 1987.

[Paunović, 2009] V. Paunović. Razvoj sustava za vođenje nastave s velikim brojem sudionika. http://www.fer.hr/_download/repository/Kvalifikacijski-Vlatka_Paunovic.pdf, 2009. Rad za kvalifikacijski doktorski ispit.

[Petrovic et al., 2005] S. Petrovic, V. Patel, and Y. Yang. Examination timetabling with fuzzy constraints. In E. Burke and M. Trick, editors, *Practice and Theory of Automated Timetabling V*, volume 3616 of *Lecture Notes in Computer Science*, pages 313–333. Springer Berlin / Heidelberg, 2005. URL http://dx.doi.org/10.1007/11593577_18. 10.1007/11593577_18.

[Pillay and Banzhaf, 2010] N. Pillay and W. Banzhaf. An informed genetic algorithm for the examination timetabling problem. *Applied Soft Computing*, 10(2):457–467, 2010. ISSN 1568-4946. doi: DOI:10.1016/j.asoc.2009.08.011. URL http://www.sciencedirect.com/science/article/B6W86-4X01P6P-3/2/8ebe4deb04c6f4ba451e6d71b3c6d7f9.

[Pinedo, 2008] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008. ISBN 0387789340, 9780387789347.

[Pribil, 2010] S. Pribil. Rješavanje problema izrade rasporeda nadoknada primjenom genetskog algoritma, Lipanj 2010. Završni rad, Fakultet elektrotehnike i računarstava Sveučilišta u Zagrebu.

[Price, 1994] K. Price. Genetic annealing. *Dr. Dobb's Journal*, 19(10):127–132, 1994.

[Price and Storn, 1997] K. Price and R. Storn. Differential evolution: A simple evolution strategy for fast optimization. *Dr. Dobb's Journal of Software Tools*, 22(4):18–24, 1997.

[Price, 1997] K. V. Price. Differential evolution vs. the functions of the 2nd iceo. In *Proc. of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pages 153–157, Indianapolis, IN, USA, April 1997.

[Price, 1999] K. V. Price. *New Ideas in Optimization*, chapter Differential Evolution. McGraw-Hill, London, 1999.

[Price et al., 2005] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution A Practical Approach to Global Optimization*. Natural Computing Series. Springer-Verlag, Berlin, Germany, 2005. URL http://www.springer.com/west/home/computer/foundations?SGWID=4-156-22-32104365-0\&teaserId=68063\&CENTER_ID=69103.

[Qu and Burke, 2007] R. Qu and E. K. Burke. Adaptive decomposition and construction for examination timetabling problems. In *Proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications (MISTA 2007)*, pages 418–425, 2007.

[Ranson and Ahmadi, 2007] D. Ranson and S. Ahmadi. An extensible modelling framework for timetabling problems. In *Proceedings of the 6th international conference on Practice and theory of automated timetabling VI*, PATAT'06, pages 383–393, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-77344-4, 978-3-540-77344-3. URL http://portal.acm.org/citation.cfm?id=1782534.1782565.

[Rechenberg, 1965] I. Rechenberg. Cybernetic solution path of an experimental problem. *Royal Aircraft Establishment, Library translation No. 1122*, 1965.

[Rechenberg, 1973] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[Reis and Oliveira, 2000] L. P. Reis and E. Oliveira. A language for specifying complete timetabling problems. In *PATAT2000 Proceedings, August 2000. Burke and Erben (Eds*, pages 322–341. Springer, 2000.

[Reynolds, 1987] C. W. Reynolds. Flocks, herds and schools: a distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[Rossi-Doria et al., 2003] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. In *Practice and Theory of AutomatedTimetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 329–351. Springer Berlin / Heidelberg, 2003. URL http://dx.doi.org/10.1007/978-3-540-45157-0_22. 10.1007/978-3-540-45157-0_22.

[Santiago-Mozos et al., 2005] R. Santiago-Mozos, S. Salcedo-Sanz, M. DePrado-Cumplido, and C. Bousoño Calzón. A two-phase heuristic evolutionary algorithm for personalizing course timetables: a case study in a spanish university. *Comput. Oper. Res.*, 32:1761–1776, July 2005. ISSN 0305-0548. doi: 10.1016/j.cor.2003.11.030. URL http://portal.acm.org/citation.cfm?id=1077840.1077846.

[Schaerf, 1999] A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13:87–127, 1999. ISSN 0269-2821. URL http://dx.doi.org/10.1023/A:1006576209967. 10.1023/A:1006576209967.

[Schwefel, 1968] H. P. Schwefel. Projekt mhd-staustrahlrohr: Experimentelle optimierung einer zweiphasendüse, teil i. Technical Report 35, AEG Forschungsinstitut, October 1968.

[Schwefel, 1975] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität Berlin, 1975.

[Shiau, 2011] D.-F. Shiau. A hybrid particle swarm optimization for a university course scheduling problem with flexible preferences. *Expert Systems with Applications*, 38(1):235–248, 2011. ISSN 0957-4174. doi: DOI:10.1016/j.eswa.2010.06.

051. URL http://www.sciencedirect.com/science/article/B6V03-50G6W69-7/2/db3a63b38e0ef809fb19c91df3ddd0b2.

[Sivanandam and Deppa, 2010] S. Sivanandam and S. Deppa. *Introduction to Genetic Algorithms*. Springer, Berlin, Germany, 2010.

[Socha et al., 2002] K. Socha, J. Knowles, and M. Sampels. A max-min ant system for the university course timetabling problem. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 63–77. Springer Berlin / Heidelberg, 2002. URL http://dx.doi.org/10.1007/3-540-45724-0_1. 10.1007/3-540-45724-0_1.

[Socha et al., 2003] K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In S. Cagnoni, C. Johnson, J. Cardalda, E. Marchiori, D. Corne, J.-A. Meyer, J. Gottlieb, M. Middendorf, A. Guillot, G. Raidl, and E. Hart, editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 334–345. Springer Berlin / Heidelberg, 2003. URL http://dx.doi.org/10.1007/3-540-36605-9_31. 10.1007/3-540-36605-9_31.

[Song and Gu, 2004] M. Song and G. Gu. Research on particle swarm optimization: A reviews. In *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, pages 2236–2241, 2004.

[Storn and Price, 1995] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute, Berkeley, CA, 1995. Technical Report TR-95-012.

[Storn and Price, 1996] R. Storn and K. Price. Minimizing the real functions of the icec'96 contest by differential evolution. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 842–844, 1996.

[Storn and Price, 1997] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[Stützle and Hoos, 2000] T. Stützle and H. Hoos. Max-min ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[Stützle and Hoos, 1997] T. Stützle and H. Hoos. The max-min ant system and local search for the traveling salesman problem. In T. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evoultionary Computation (ICEC'97)*, pages 309–314, Piscataway, NJ, 1997. IEEE Press.

[Stützle and Hoos, 1999] T. Stützle and H. Hoos. Max-min ant system and local search for combinatorial optimization problem. In S. Voss, S. Martello, I. Osmann, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trend sin Local Search Paradigms for optimization*, pages 137–154. Kluwer Academic Publishers, Dordrecht, Netherlands, 1999.

[Talbi, 2009] E.-G. Talbi. *Metaheuristics. From Desing to Implementation.* Wiley, New Jersey, USA, 2009.

[Talbi and Weinberg, 2007] E.-G. Talbi and B. Weinberg. Breaking the search space symmetry in partitioning problems: An application to the graph coloring problem. *Theoretical Computer Science (TCS)*, 378(1):78–86, 2007.

[Tripathy, 1992] A. Tripathy. Computerised decision aid for timetabling – a case analysis. *Discrete Applied Mathematics*, 35(3):313 – 323, 1992. ISSN 0166-218X. doi: DOI:10.1016/0166-218X(92)90253-7. URL http://www.sciencedirect.com/science/article/B6TYW-45W39ST-1P/2/690e982ba4fe95ca32be7f4272977e96.

[van den Broek et al., 2007] J. van den Broek, C. Hurkens, and G. Woeginger. Timetabling problems at the tu eindhoven. In E. Burke and H. Rudová, editors, *Practice and Theory of Automated Timetabling VI*, volume 3867 of *Lecture Notes in Computer Science*, pages 210–227. Springer Berlin / Heidelberg, 2007. URL http://dx.doi.org/10.1007/978-3-540-77345-0_14. 10.1007/978-3-540-77345-0_14.

[Wang et al., 2010] S. Wang, M. Bussieck, M. Guignard, A. Meeraus, and F. O'Brien. Term-end exam scheduling at united states military academy/west point. *Journal of Scheduling*, 13:375–391, 2010. ISSN 1094-6136. URL http://dx.doi.org/10.1007/s10951-009-0153-5. 10.1007/s10951-009-0153-5.

[White, 2001] G. White. Constrained satisfaction, not so constrained satisfaction and the timetabling problem. In E. Burke and M. Carter, editors, *The Practice and Theory of Automatic Timetabling: Selected Papers from the Third International Conference, Konstanz, 2000, Lecture Notes in Computer Science*, volume 2079, pages 32–47. Springer, 2001.

[White and Chan, 1979] G. White and P. Chan. Towards the construction of optimal examination timetables. *INFOR*, 17:219–229, 1979.

[Wilke et al., 2002] P. Wilke, M. Gröbner, and N. Oster. A hybrid genetic algorithm for school timetabling. In *Proceedings of the 15th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, AI '02, pages 455–464, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-00197-2. URL http://portal.acm.org/citation.cfm?id=646079.676216.

[Wolpert and Macready, 1995] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical report, Santa Fe Institute, Sante Fe, NM, USA, 1995. Technical Report SFI-TR-95-02-010.

[Wolpert and Macready, 1997] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1): 67–82, 1997.

[Wren, 1996] A. Wren. Scheduling, timetabling and rostering – a special relationship? In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 46–75. Springer Berlin / Heidelberg, 1996. URL http://dx.doi.org/10.1007/3-540-61794-9_51. 10.1007/3-540-61794-9_51.

[Yang, 2005] X.-S. Yang. Engineering optimizations via nature-inspired virtual bee algorithms. In J. Mira and J. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562 of *Lecture Notes in Computer Science*, pages 317–323. Springer Berlin / Heidelberg, 2005. URL http://dx.doi.org/10.1007/11499305_33. 10.1007/11499305_33.

[Yang, 2008] X.-S. Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, Frome, United Kingdom, 2008.

[Zervoudakis and Stamatopoulos, 2000] K. Zervoudakis and P. Stamatopoulos. A generic object-oriented constraint-based model for university course timetabling. In *The Practice and Theory of Automated Timetabling: Selected Papers from the Third International Conference, 28 – 47. Lecture Notes in Computer Science*, pages 28–47. Springer-Verlag, 2000.

[Zhang et al., 2010] D. Zhang, Y. Liu, R. M'Hallah, and S. C. Leung. A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European Journal of Operational Research*, 203(3):550–558, 2010. ISSN 0377-2217. doi: DOI:10.1016/j.ejor.2009.09.014. URL http://www.sciencedirect.com/science/article/B6VCT-4X8CCY3-1/2/07288928d26a27bcb595bf4f016afcdd.

[Čupić, 2008] M. Čupić. Web-oriented educational system for digital circuits modeling and simulation. In M. Čičin Šain, I. Turčić Prstačić, I. Sluganović, and I. Uroda, editors, *Proceedings of MIPRO 2008 – 31st International Convention on Information and Communication Technology, Electronics and Microelectronics*, volume IV Computers in Education, pages 124–129. DENONA, zagreb, 2008.

[Čupić, 2010] M. Čupić. A case study: using multiple-choice tests at university level courses – preparation and supporting infrastructure. *International Journal of Intelligent Defence Support Systems (IJIDSS)*, 3(1/2):90–100, 2010. doi: DOI: 10.1504/IJIDSS.2010.033679.

[Čupić, 2009] M. Čupić. Automatizacija izrade i obrade pisanih provjera znanja, 2009. Projekt informacijske tehnologije Ministarstva znanosti, obrazovanja i športa Republike Hrvatske, voditelj: Marko Čupić, broj projekta: 2008-078.

[Čupić and Franović, 2010] M. Čupić and T. Franović. Ferko project – intelligent support for course management at faculty level. In M. Čičin Šain, I. Uroda, I. Turčić Prstačić, and I. Sluganović, editors, *Proceedings of MIPRO 2010 – 33st International Convention on Information and Communication Technology, Electronics and Microelectronics*, volume IV Computers in Education, pages 65–70. DENONA, zagreb, 2010.

[Čupić and Mihajlović, 2010] M. Čupić and v. Mihajlović. Computer-based knowledge, self-assessment and training. *International journal of engineering education*, 26(1): 111–125, 2010.

[Čupić et al., 2009a] M. Čupić, M. Golub, and D. Jakobović. Exam timetabling using genetic algorithm. In V. Luzar-Stiffler, I. Jarec, and Z. Bekić, editors, *Proceedings of the 31st International Conference on Information Technology Interfaces*, pages 357–362, 2009a.

[Čupić et al., 2009b] M. Čupić, J. Šnajder, and B. Dalbelo Bašić. Post-test analysis of automatically generated multiple choice exams: a case study. In M. E. Auer, editor, *Proceedings of ICL 2009, Vienna: International Association of Online Engineering*, page 1, 2009b.

[Čupić et al., 2010] M. Čupić, M. Golub, and D. Jakobović. Applying ai-techniques as help for faculty administration – a case study. In *Central European Conference on Information and Intelligent Systems 2010*, pages 163–170, 2010.

[Šnajder et al., 2008] J. Šnajder, M. Čupić, B. Dalbelo Bašić, and S. Petrović. Enthusiast: An authoring tool for automatic generation of paper-and-pencil multiple-choice tests. In M. E. Auer, editor, *Proceedings of ICL 2008, Villach*, page 1, 2008.

Životopis

Marko Čupić rođen je 09. lipnja 1979. godine u Zagrebu. Diplomirao je 2002. godine na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu, smjer računarstvo, s temom "Inteligentno pretraživanje informacijskog prostora informacijske infrastrukture". Magistrirao je 2006. godine također na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu, smjer jezgra računarstva, s temom "Model sustava e-ispitivanja s potporom za inteligentno upravljanje provjerom znanja". Godine 2002. nagrađen je Rektorovom nagradom Sveučilišta u Zagrebu kao suautor studentskog rada "Web-orijentirani simulator modela neuronskih mreža s primjerom raspoznavanja novčanica i generatorom uzoraka". Od siječnja 2003. godine zaposlen je na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave Fakulteta elektrotehnike i računarstva kao znanstveni novak na projektu "Semantički Web kao omogućavatelj informacijske infrastrukture", a od 2007. godine na projektu "Univerzalna posrednička platforma za sustave e-učenja". Bio je uključen u nastavne aktivnosti Zavoda na predmetima Digitalna elektronika; Projektiranje korisničkih sučelja i interaktivnih sustava; Otvoreni uredski sustavi; Mreže računala; Digitalna logika; Interaktivna računalna grafika; Umjetna inteligencija; i Napredni operacijski sustavi, a asistirao je u vođenju 7 diplomskih i završnih radova. Predložio je i vodi dvije vještine: "Osnove programskog jezika Java" te "Rješavanje optimizacijskih problema algoritmima evolucijskog računanja u Javi". Voditelj je projekta primjene informacijske tehnologije "Automatizacija izrade i obrade pisanih provjera znanja" Ministarstva znanosti, obrazovanja i športa Republike Hrvatske.

Njegovi istraživački interesi obuhvaćaju područja evolucijskog računanja, neizrazite logike, neuroračunarstva, e-učenja i e-ispitivanja te oblikovanja i programiranja složenih sustava. Sam ili u suautorstvu je objavio 16 radova na međunarodnim znanstvenim skupovima i tri rada u časopisima s međunarodnom recenzijom, od koji jedan u časopisima indeksiranima u bazama CC i SCI Expanded. Autor je "Zbirka riješenih zadataka iz

Digitalne elektronike i Digitalne logike". Član je strukovnih udruga ACM, IEEE, AAAI. Govori engleski jezik.

Biography

Marko Čupić was born on June 09, 1979 in Zagreb, Croatia. He received his B.Sc. in computing from the University of Zagreb, Faculty of Electrical Engineering and Computing in 2002 (thesis title: “Intelligent search of information space of information infrastructure”) and M.Sc. in computer science from the same university in 2006 (thesis title: “Model of e-examination system with support for intelligent assessment management”). In 2002 he was awarded the University of Zagreb Rector Award as the co-author of the best student paper. From January 2003. he is employed at the Department of Electronics, Microelectronics, Computer and Intelligent Systems at the Faculty of Electrical Engineering and Computing, at first as a scientific novice on the project “Semantic Web as Information Infrastructure Enabler”, and from 2007 on the project “Universal Middleware Platform for e-Learning Systems”. He was involved in Department’s educational activities within the courses on Digital electronic; Design of user interfaces and interactive systems; Open office system; Computer networks; Digital Logic; Interactive Computer Graphics; Artificial Intelligence and Advanced operating systems. He also assisted in supervising 7 bachelor’s theses. He introduced two skills and gives lectures: “Introduction to Java programming language” and “Solving optimization problems using evolutionary computation algorithms in Java”. He is the leader of project of information technology application entitled “Automation of creation and processing writen knowledge assessments” of Ministry of science, education and sport of Republic Croatia. His research interests include evolutionary computation, fuzzy logic, neurocomputing, e-learning, e-assessment, and design and implementation of complex software systems. He has co-authored 16 conference papers and three journal papers, one of which in journals indexed in Current Contents and SCI Expanded. He is also an author of "Digital electronic and Digital logic - collection of solved problems". He is a member of the ACM, IEEE and AAAI. He is fluent in English.