SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 226

Izgradnja platforme za raspodijeljene algoritme evolucijskog računanja

Mihej Komar

Zagreb, lipanj 2011.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 21. veljače 2011.

DIPLOMSKI ZADATAK br. 226

Pristupnik: **Mihej Komar**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Izgradnja platforme za raspodijeljene algoritme evolucijskog računanja**

Opis zadatka:

Evolucijsko računanje danas je izuzetno rasprostranjena tehnika rješavanja optimizacijskih problema. Ovisno o veličini i složenosti problema, jedan slijedni algoritam često ne može dati zadovoljavajuće rješenje u ograničenom vremenu. Stoga je u okviru ovog diplomskog rada potrebno izraditi programsko okruženje u programskom jeziku Java koje omogućuje olakšano pisanje i izvođenje raspodijeljenih algoritama evolucijskog računanja. Programsko okruženje treba podržavati izradu različitih vrsta evolucijskih algoritama, uobičajene migracijske parametre (poput vremena migracije, topologija migracije i sl.), automatsku razmjenu rješenja između algoritama te prikupljanje statističkih podataka. Ispitati razvijeni programski sustav na nekoliko različitih optimizacijskih problema (poput izrade rasporeda međuispita i sličnih). Opisati razvijene algoritme. Statistički obraditi i usporediti rezultate dobivene za više različitih topologija i drugih parametara.

Zadatak uručen pristupniku: 25. veljače 2011.
Rok za predaju rada: 10. lipnja 2011.

Mentor:

Doc.dr. sc. Marin Golub

Djelovođa:

Doc.dr.sc. Domagoj Jakobović

Predsjednik odbora za diplomski rad profila:

Prof.dr.sc. Siniša Srbljić

Zahvaljujem mentorima doc. dr. sc. Marinu Golubu i mr. sc. Marku Čupiću na suradnji u izradi diplomskog rada. Posebno zahvaljujem mr. sc. Marku Čupiću na vođenju, pomoći i podršci.

Sadržaj

1. Uvod

Računala su danas alat kojim se mogu rješavati vrlo kompleksni problemi, a prije ih je bilo gotovo nemoguće riješiti. Primjere takvih problema pronalazimo u raznim optimizacijskim problemima. Optimizacijski problemi mogu se rješavati koristeći determističke ili stohastičke algoritme. Praktični problemi često ne mogu biti riješeni determističkim algoritmima jer determinističko pretraživanje prostora rješenja često ima preveliku algoritamsku složenost. Primjerice, rješavanje problema trgovačkog putnika pretražujući sve moguće kombinacije moguće je riješiti do manjeg broja gradova, ali za veći broj gradova algoritam postaje vremenski i prostorno prezahtjevan. Za takve probleme često nije potrebno pronaći optimalno rješenje, već dovoljno dobro rješenje. U tom slučaju mogu se koristiti algoritmi evolucijskog računanja.

Algoritmi evolucijskog računanja su podskup stohastičkih algoritama koji pomoću populacije rješenja postupno pronalaze sve kvalitetnija rješenja, ali ne garantiraju pronalazak optimalnog rješenja. Većina algoritama evolucijskog računanja inspirirana je prirodom, npr. evolucijom, mravljom kolonijom, imunološkim sustavom ili glazbom. Pogodni su za paralelizaciju, a njihovom paralelizacijom moguće je ubrzati izvršavanje, ali i povećati kvalitetu dobivenih rješenja.

Cilj ovog rada je izgradnja platforme za raspodijeljene algoritme evolucijskog računanja. Glavna motivacija za izgradnju platforme je potreba za olakšavanjem paralelizacije slijednih algoritama evolucijskog računanja koji rješavaju razne optimizacijske probleme. Automatska razmjena rješenja između algoritama, konfiguriranje migracijskih parametara i prikupljanje statističkih podataka treba biti ostvareno preko platforme te time olakšati izgradnju novih modula. Platforma treba biti razvijena u programskom jeziku Java kako bi bila neovisna o operacijskom sustavu.

Za platformu je potrebno razviti module za rješavanje problema izrade rasporeda međuispita i ponovljenih ispita specifičnih za Fakultet elektrotehnike i računarstva. Taj problem pogodan je za demonstraciju i vrednovanje platforme

jer rješavanje tog problema determinističkim algoritmima ima vrlo veliku složenost. Primjerice, postoji 40^{140} različitih rasporeda za problem s 140 predmeta i 40 termina. Determinističkom pretragom optimalno rješenje ne može biti pronađeno u razumnom vremenu.

U poglavlju 2 opisani su problemi pogodni za evolucijsko računanje i način rada tih algoritama. Navedeno je i opisano nekoliko algoritama osnovnih kategorija evolucijskog računanja. Postupci paralelizacije evolucijskog računanja, vrste migracijskih parametara i neke tehnologije koje olakšavaju paralelizaciju objašnjeni su u poglavlju 3. U poglavlju 4 opisana je razvijena platforma za raspodijeljene algoritme evolucijskog računanja, a u poglavlju 5 dva modula za platformu koji uspješno rješavaju probleme izrade rasporeda ispita i ponovljenih ispita. Provedeno je eksperimentalno vrednovanje platforme, a rezultati su navedeni u poglavlju 6. Zaključci ovog rada dani su u poglavlju 7.

2. Evolucijsko računanje

Naglim razvojem računala omogućeno je rješavanje sve kompleksnijih problema. Razvijaju se sve bolji i sve brži algoritmi koji se mogu koristiti za raznovrsne praktične probleme. Najčešći problemi koji se rješavaju pomoću računala su složeni optimizacijski problemi.

Algoritmi za rješavanje problema mogu se podijeliti na deterministčke i stohastičke algoritme. Deterministički algoritmi s jednakim ulaznim podacima izvršavaju se uvijek na isti način i daju jednake rezultate. Stohastički algoritmi barem u jednom dijelu računanja koriste slučajnost. Iz toga slijedi kako isti stohastički algoritam s istim ulaznim podacima najčešće daje različite rezultate.

Primjeri deterministčkih algoritama su: pretraživanje u širinu (engl. *breadth-first search*), pretraživanje u dubinu (engl. *depth-first search*), A*, pretraživanje usponom na vrh (engl. *hill-climbing*) i drugi. Dio tih algoritama (npr. pretraživanje u širinu i A*) potpuni[1] su i optimalni[2] pa time pogodni za pronalaženje optimalnog rješenja za određeni problem. Ipak, postoje slučajevi kada se deterministički algoritmi ne mogu koristiti. Neki od razloga su velika vremenska ili prostorna složenost algoritma i složena, nekonzistentna ili promjenjiva procjena kvalitete rješenja. Problemi koji ne mogu biti brzo i učinkovito riješeni preko deterministčkih algoritama vrlo su česti. Za takve probleme najčešće nije potrebno pronaći optimalno rješenje već je dovoljno pronaći prihvatljivo rješenje. U takvim slučajevima stohastički algoritmi mogu pokazati puno bolja svojstva od deterministčkih.

Podskup stohastičkih algoritama su heuristički algoritmi. Heuristički algoritmi dijele se na heuristike specifičnih problema i metaheuristike. Heuristike specifičnih problema su usko specijalizirani algoritmi za rješavanje nekih pro-

[1] Algoritam je potpun (engl. *complete*) ako i samo ako u konačnom vremenu pronalazi rješenje kada takvo rješenje postoji.

[2] Algoritam je optimalan (engl. *optimal*) ako i samo ako u konačnom vremenu pronalazi optimalno rješenje.

blema, a pod metaheurističke algoritme spadaju algoritmi za rješavanje šireg skupa optimizacijskih problema. Evolucijsko računanje je podskup metaheurističkih algoritama koji koriste populaciju rješenja problema. Na početku izvršavanja algoritma nastoji se pretražiti rješenja iz cijelog prostora rješenja. Tako algoritam pronalazi u kojem dijelu prostora postoje kvalitetnija rješenja. U svakoj iteraciji mijenjaju se rješenja unutar populacije pomoću operatora evolucijskog računanja. Operatori evolucijskog računanja izvedeni su tako da njihovom primjenom rješenja postupno konvergiraju prema kvalitetnijima. Time algoritam pretražuje sve bolje i uže dijelove prostora rješenja. U završnim fazama algoritma obavlja se fina pretraga oko najboljih pronađenih rješenja.

Evolucijsko računanje, zajedno s neuronskim mrežama, neizrazitom logikom i sličnim granama, spada u meko računarstvo (engl. *soft computing*). Za evolucijsko računanje potrebna je velika računalna moć te se zbog toga intenzivnije počela razvijati tek pojavom jeftinijih i moćnijih računala.

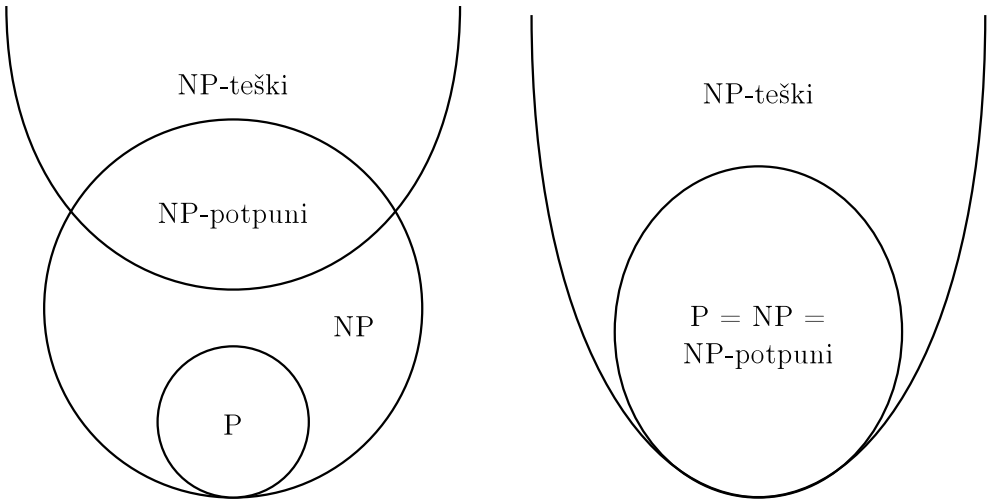
2.1. Problemi pogodni za evolucijsko računanje

Teorem *No free lunch* je teorem koji dokazuje kako ne postoji univerzalni algoritam koji se može uspješno primijeniti za sve optimizacijske probleme (Wolpert i Macready, 1997). Kao što će biti opisano u poglavlju 2.1.1, neki problemi ne mogu biti riješeni koristeći potpunu pretragu cijelog prostora rješenja. Takvi problemi najčešće se rješavaju pomoću stohastičkih algoritama. Primjerice, nasumičnim pretraživanjem mogu se rješavati svi optimizacijski problemi, ali dobivena rješenja vrlo često neće biti zadovoljavajuće kvalitete. Heuristike specifičnih problema mogu izrazito dobro rješavati manju skupinu problema, ali su loše za druge probleme. Algoritmi evolucijskog računanja mogu biti vrlo dobri za široki skup problema ako su parametri dobro podešeni.

2.1.1. Složenost optimizacijskih problema

Razred složenosti je skup problema odluke koje je moguće riješiti koristeći određenu količinu računskih resursa. Skup $\mathcal{P}$ je skup svih problema odluke koji pomoću deterministčkog Turingovog stroja mogu biti riješeni u polinomijalnoj složenosti. Skup $\mathcal{NP}$ je skup problema odluke koji mogu biti riješeni pomoću nedeterminističkog Turingovog stroja u polinomijalnoj složenosti, a rješenje može biti provjereno determinističkim Turingovim strojem u polinomijalnoj složenosti.

Poznato je da vrijedi $\mathcal{P} \subseteq \mathcal{NP}$, ali do danas nije potvrđeno ili opovrgnuto vrijedi li $\mathcal{NP} \subseteq \mathcal{P}$, tj. $\mathcal{P} = \mathcal{NP}$. Skup $\mathcal{NP}$-teških problema čine svi problemi odluke na koje se u polinomijalnom vremenu može svesti bilo koji $\mathcal{NP}$ problem. Skup $\mathcal{NP}$-potpunih problema definiran je kao presjek skupova $\mathcal{NP}$ i $\mathcal{NP}$-teških problema. Odnos između skupova vizualiziran je na slici 2.1.



(a) Složenosti problema za $\mathcal{P} \neq \mathcal{NP}$ **(b)** Složenosti problema za $\mathcal{P} = \mathcal{NP}$

Slika 2.1: Prikaz skupova složenosti problema odluke

Neki od najpoznatijih $\mathcal{NP}$-potpunih problema su: problem zadovoljavanja istinitosti (engl. *satisfiability problem*), problem trgovačkog putnika (engl. *travelling salesman problem*), problem bojanja grafova (engl. *graph coloring problem*) i drugi praktični problemi. S obzirom na to da takvi problemi do danas ne mogu biti riješeni polinomijalnom složenošću, rješavanje nekih $\mathcal{NP}$-potpunih problema potpunim i optimalnim algoritmom nije moguće. Ipak, često nije potrebno pronaći globalno najbolje rješenje, već je dovoljno pronaći dovoljno dobro rješenje. U tu svrhu upotrebljavaju se razni heuristički algoritmi poput algoritama evolucijskog računanja.

2.1.2. Predstavljanje problema na računalu

Kako bi problem mogao biti riješen pomoću algoritama evolucijskog računanja, potrebno je na računalu predstaviti rješenje problema. Predstavljanje jednog rješenja problema zove se genotip, a svi genotipovi nalaze se u skupu koji nazivamo genomom, genotipskim prostorom (engl. *genotype space*) ili prostorom pretraživanja (engl. *search space*). Gen je najmanji nedjeljivi dio genotipa. Svakom is-

pravnom genotipu pridružuje se skup njegovih svojstava, a to se zove fenotip. Svi fenotipovi nalaze se u fenotipskom prostoru (engl. *phenotype space*) ili prostorom rješenja (engl. *solution space*).

Ovakav izbor imena inspiriran je genetikom. Gen je osnovna jedinica nasljeđa i dio je molekule DNK koja definira određenu nasljednu osobinu. Svi geni nekog organizma zovu se genotip i čine nasljeđe organizma. Fenotip je skup svih fizioloških i morfoloških osobina organizma i definiran je genotipom jedinke.

Genotip

Način na koji će genotip biti predstavljen na računalu najviše ovisi o samom problemu. Kod odabira metode predstavljanja genotipa potrebno je obratiti pažnju na mogućnosti stvaranja ilegalnog genotipa i problem simetričnosti. Ilegalni genotip je onaj genotip za koji se ne može odrediti pripadajući fenotip. Problem simetričnosti je pojava kada različiti genotipovi prikazuju isto rješenje. Ako kod prikaza postoji problem simetričnosti, onda može doći do neefikasnosti algoritama i kompliciranije izrade evolucijskih operatora. Kvaliteta prikaza genotipa procjenjuje se i prema pravilu kauzalnosti, tj. kako promjene u genotipu utječu na fenotip. Male varijacije genotipa trebale bi uzrokovati male promjene u fenotipu. Genotip se može nalaziti u kontinuiranom ili diskretnom prostoru. Ako je genotip u diskretnom prostoru, prostor može, ali i ne mora imati semantiku broja. Prostor ima semantiku broja ako se mogu odrediti prethodnik ili sljedbenik određene točke u prostoru. Prostor koji nema semantiku broja nema poredak, a sve točke određene su kategorijama.

Genotip može biti predstavljen:

- binarnim nizom,
- brojevima s pomičnim decimalnim zarezom,
- permutacijskim prikazom ili
- nekim drugim strukturama.

Genotip predstavljen binarnim nizom najjednostavniji je način prikaza genotipa i prikazan je na slici 2.2. Omogućava upotrebu postojećih, jednostavnih i efikasnih operatora evolucijskog računanja. Za ovakvu vrstu genotipa obično je potrebno izraditi funkciju kodiranja rješenja u binarni niz i njegovo dekodiranje. Za razne probleme ovo je kompliciranа i vrlo skupa funkcija, a prikaz problema u binarnom nizu je težak postupak. Lošim ostvarenjem kodiranja i dekodiranja mogući su ilegalni genotipovi.

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

Slika 2.2: Genotip predstavljen binarnim nizom

Najjednostavniji način korištenja binarnog genotipa je upotreba binarnog kodiranja. Primjerice, ako se u genotipu trebaju prikazati brojevi od 0 do 15, tada će broj 7 biti prikazan kao 0111, a broj 8 kao 1000. Primjećuje se kako ovakav genotip ne poštuje pravilo kauzalnosti jer Hammingova udaljenost između susjednih genotipova nije 1. Kada se binarni niz kreira korištenjem Grayevog koda, tada susjedni genotipovi imaju minimalnu Hammingovu udaljenost.

Ako je potrebno optimirati funkciju čija je domena vektor realnih brojeva (kontinuirani prostor), bolji prikaz od binarnog niza je niz brojeva s pomičnim decimalnim zarezom. Nije potrebno obavljati kodiranje i dekodiranje, ali je potrebno osigurati zadržavanje rješenja unutar zadanog intervala. Na slici 2.3 je primjer genoma predstavljen brojevima s pomičnim zarezom.

42,567643	0,987812	-334,456521	15,995391

Slika 2.3: Genotip predstavljen brojevima s pomičnim zarezom

Kod kombinatoričkih problema često se upotrebljava permutacijski niz. Primjer genoma predstavljenog permutacijskim nizom nalazi se na slici 2.4. Obično je predstavljen nizom cijelih brojeva, a operatori moraju biti izvedeni tako da se mijenja isključivo redoslijed brojeva, ali ne i njihove vrijednosti. Tipični primjeri problema koji koriste permutacijski niz su problem trgovačkog putnika i problem dodjeljivanja (engl. *assignment problem*).

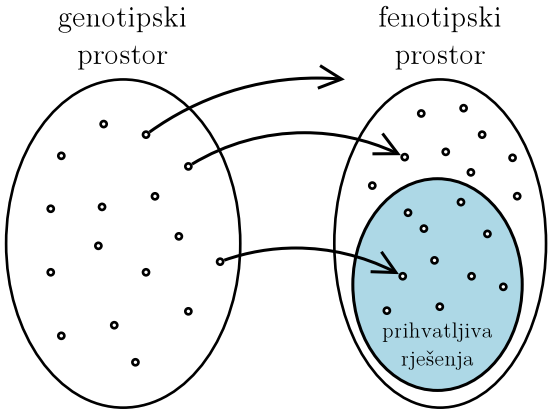
10	12	5	2	11	3	6	4	1	8	9	7

Slika 2.4: Genotip predstavljen permutacijskim nizom

Ako problem ne može biti efikasno riješen upotrebom prethodno navedenih načina prikaza genotipa, potrebno je kreirati specifičnu strukturu. Pritom je potrebno izraditi i evolucijske operatore za takvu strukturu. Struktura mora biti napravljena tako da nad njom evolucijski operatori efikasno rade i pritom zadrže predviđena svojstva. Prednost ovakvog pristupa je potencijalna brzina i efikasnost operatora, a nedostatak je nemogućnost ponovne upotrebe evolucijskih operatora na drugim problemima.

Fenotip

Fenotip je skup svojstava rješenja određen genotipom. Za fenotip se definira skup čvrstih i mekih ograničenja. Čvrsta ograničenja (engl. *hard constraints*) moraju biti ispunjena, a ako nisu ispunjena, rješenje je neprihvatljivo. Ograničenja koja definiraju kvalitetu rješenja i najčešće ne mogu biti potpuno ispunjena su meka ograničenja (engl. *soft constraints*). Broj prekoračenih mekih ograničenja potrebno je svesti na što manji broj. Na slici 2.5 prikazano je preslikavanje iz genotipskog prostora u fenotipski prostor. Može se primijetiti kako ilegalni genotipovi ne mogu biti preslikani u fenotipski prostor. Također, podskup fenotipskog prostora je skup prihvatljivih rješenja, tj. rješenja koja imaju ispunjena čvrsta ograničenja.



Slika 2.5: Preslikavanje iz genotipskog u fenotipski prostor

Funkcija cilja (engl. *objective function*) je funkcija koja se koristi u optimizaciji. Funkcija vrednovanja procjenjuje kvalitetu rješenja i računa se iz funkcije cilja. Ako je cilj optimizacije određivanje maksimuma onda se govori o funkciji prikladnosti ili dobrote (engl. *fitness function*), a ako je cilj određivanje minimuma onda je to funkcija kazne (engl. *penalty function*). Kod problema gdje se točna kvaliteta rješenja sporo računa ili je neprecizna, funkciju vrednovanja treba aproksimirati.

S obzirom na vrstu optimizacije postoje:

- jednokriterijske optimizacije i
- višekriterijske optimizacije.

Jednokriterijska optimizacija (engl. *single-objective optimization*) rješenja uspoređuje prema samo jednom kriteriju. Bilo koja dva rezultata jednokriterijskih funkcija mogu se usporediti, tj. jedno rješenje može biti bolje, lošije ili jednako

nekom drugom rješenju. U ovom radu obrađuju se isključivo primjeri jednokriterijske optimizacije.

Puno problema ne možemo ocjenjivati prema samo jednom kriteriju već prema više njih. Ako se procjene kvalitete mogu težinskom sumom izraziti preko jednog skalara, onda se problem svodi na jednokriterijsku optimizaciju. Kada to nije moguće, tada je potrebno koristiti višekriterijsku optimizaciju (engl. *multi-objective optimization*). Rezultat njene funkcije vrednovanja je višedimenzionalni vektor. Svaka dimenzija vektora predstavlja procjenu kvalitete prema drugom kriteriju. Kažemo da jedan rezultat dominira nad drugim rezultatom ako i samo ako prvo rješenje nije lošije od drugog rješenja prema bilo kojem kriteriju i prvo rješenje je strogo bolje od drugog rješenja prema barem jednom kriteriju. Rezultat višekriterijske optimizacije nije jedno najbolje rješenje, već je to skup rješenja koja međusobno ne dominiraju. Na kraju pretraživanja korisnik može odlučiti koje je rješenje najprikladnije kao rješenje problema.

2.2. Operatori evolucijskog računanja

U ovom poglavlju opisuju se najčešće korištena ostvarenja operatora koji se koriste u algoritmima evolucijskog računanja. Operatori se dijele na operatore koji koriste informacije iz genotipskog prostora i operatore koji koriste vrijednost funkcije vrednovanja. Svaki operator koji koristi informacije iz genotipskog prostora ovisi o načinu prikaza genotipa. Prilikom upotrebe nekog drugog genotipskog prostora potrebno je izraditi specifične implementacije operatora sa sličnim svojstvima kao opisani operatori.

2.2.1. Mutacija

Mutacija (engl. *mutation*) je operator koji se koristi u većini algoritama evolucijskog računanja. Određenom vjerojatnošću mijenja genotip postojećeg rješenja i time stvara novo rješenje. Novo rješenje može imati potpuno nove dijelove rješenja koje ne sadrži ni jedno drugo rješenje u populaciji. Time se povećava raznolikost rješenja u populaciji, vjerojatnost pretraživanja novih dijelova prostora rješenja (engl. *exploration*) i smanjuje vjerojatnost stagnacije u lokalnim optimumima. Pomoću vjerojatnosti mutacije određuje se razina utjecaja na izmjenu rješenja.

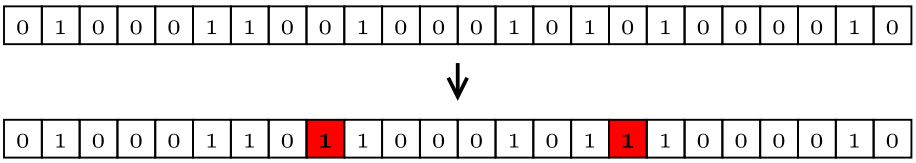
Vjerojatnost mutacije određuje se prema načinu ostvarenja operatora mutacije, vrsti problema, vrsti algoritma i broju genoma. Najčešće korištene vrijed-

nosti mutacije su između 0,01% i 5%. Niska vjerojatnost mutacije neće dovoljno pridonijeti raznolikosti rješenja, što može dovesti do prerane konvergencije. Prevelike vrijednosti mutacije pretraživanje svode na nasumičnu pretragu i znatno smanjuju konvergenciju prema boljim rješenjima.

Vjerojatnost mutacije moguće je dinamički mijenjati ovisno o fazi u kojoj se evolucijski algoritam nalazi. Na početku izvršavanja algoritma potrebno je široko pretraživanje prostora rješenja pa se koriste visoke vrijednosti mutacije. Kako algoritam konvergira prema boljim rješenjima, smanjuje se vrijednost mutacije i time omogućava finije pretraživanje užeg dijela prostora rješenja. Stopa mutacije može se adaptivno mijenjati prema stanju trenutačne populacije. Primjerice, povećavanje mutacije kod pojave stagnacije može dovesti do bijega iz lokalnog optimuma.

Mutacija binarnog genotipa

Najjednostavnija metoda mutiranja binarnog genotipa je nasumični odabir jednog bita i postavljanje njegove komplementarne vrijednosti. Prednost ovakve metode je velika brzina izvršavanja operacije mutacije, a nedostatak je nemogućnost kontroliranja broja mutiranih bitova preko vjerojatnosti mutiranja.



Slika 2.6: Primjer mutacije binarnog genotipa

Navedeni nedostaci prethodno opisane metode rješavaju se preko mutacija na razini bita. Svaki bit se mijenja određenom vjerojatnošću. Prilikom mutacije bita na mjesto starog bita postavlja se njegova komplementarna vrijednost. Primjer ovakve mutacije prikazan je na slici 2.6. Nakon mutacije izmijenjeni su 9. i 17. bit rješenja. Problem kod ovakvog operatora mutacije je složenost algoritma $O(n)$ s obzirom na broj bitova koji predstavljaju rješenje. Mutacija se može ubrzati tako da se pomoću binomne razdiobe $X \sim \mathcal{B}(n, p)$ odredi broj bitova za mutaciju. Vjerojatnost za k mutacija može se odrediti preko:

$$p_k = P(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{2.1}$$

gdje je n broj bitova, a p_k vjerojatnost k mutacija. Nakon što je određen k, potrebno je nasumično odabrati k pozicija u genotipu i izmijeniti ih.

Mutacija brojeva s pomičnim zarezom

Kod genotipa predstavljenog pomičnim zarezom operator mutacije najčešće se ostvaruje preko normalne razdiobe. Nova vrijednost rješenja dobiva se preko:

$$X \sim \mathcal{N}(\mu, \sigma^2) \tag{2.2}$$

gdje se vjerojatnost mutacije kontrolira preko standardne devijacije σ, staro rješenje predstavljeno je s μ, a novo rješenje s X. Ako je vrijednost X manja od najmanje dopuštene vrijednosti rješenja, potrebno je postaviti rješenje na najmanju dopuštenu vrijednost. Isti postupak treba napraviti i za najveću dopuštenu vrijednost.

Mutacija permutacijskog genotipa

Kod permutacijskog genotipa postoji puno različitih metoda mutacija, a neki od njih su: (a) zamjena, (b) obrtanje i (c) posmak. Mutacija zamjene nasumično odabire dva mjesta u genotipu i zamjenjuje njihove vrijednosti. Mutacija obrtanjem odabire dva mjesta u genotipu te okreće sve vrijednosti između odabranih mjesta. Mutacija posmakom odabire niz uzastopnih elemenata i translatira ih za određen broj mjesta. Nakon obavljenih mutacija potrebno je rotirati cijeli niz sve dok na prvom mjestu niza bude prvi element jer se na taj način rješava problem simetričnosti.

2.2.2. Križanje

Operator križanja (engl. *crossover*) ili rekombinacije koristi se za stvaranje novih rješenja djece $D1$ i $D2$, koja sadrže dijelove genotipa postojeća dva rješenja roditelja $R1$ i $R2$. Križanjem se obavlja fino pretraživanje prostora rješenja u kojemu se populacija nalazi, pritom koristeći informacije postojećih rješenja (engl. *exploitation*). Ovim operatorom smanjuje se raznolikost populacije. Iz populacije rješenja biraju se roditelji preko operatora odabira roditelja, koji je opisan u poglavlju 2.2.3. Iako operator križanja prvenstveno djeluje u genotipskom prostoru, može koristiti i vrijednosti funkcije vrednovanja.

Križanje binarnog genotipa

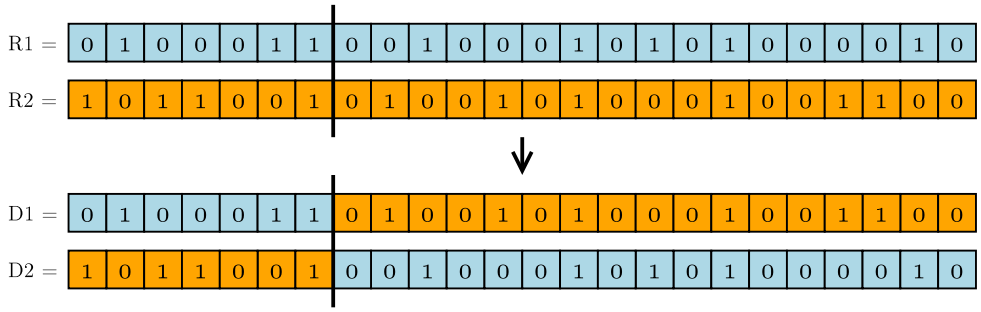
Ako su rješenja predstavljena u obliku binarnog genotipa, mogu se koristiti sljedeće metode križanja:

– križanje s jednom točkom prekida,

– križanje s t točaka prekida i

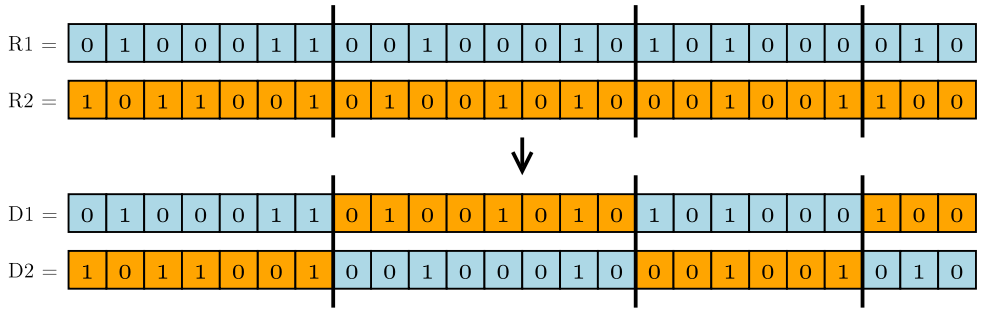
– uniformno križanje.

Križanje s jednom točkom prekida (SPX, 1-PX) je najjednostavnije križanje. Nasumičnim odabirom određuje se pozicija na kojoj će se podijeliti niz bitova. Nakon podjele, prvi dio rješenja $R1$ i drugi dio rješenja $R2$ spajaju se i čine dijete $D1$. Preostali dijelovi kromosoma čine $D2$. Primjer križanja s točkom prekida nakon 7. bita prikazan je na slici 2.7.



Slika 2.7: Primjer križanja genotipske jedinke s jednom točkom prekida

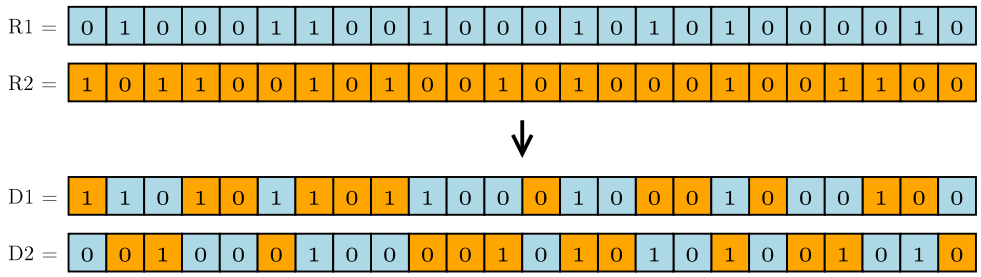
Križanje s t-točaka prekida (MPX, k-PX) slično je križanju s jednom točkom prekida, ali je razlika u broju prekida. Križanje s t-točaka ima t nasumično odabranih točaka prekida. Križanje s jednom točkom prekida i križanje s dvije točke prekida (TPX, 2-PX) specijalni su slučajevi križanja s t-točaka prekida. Primjer ovakvog križanja može se vidjeti na slici 2.8.



Slika 2.8: Primjer križanja genotipske jedinke s tri točke prekida

Uniformno križanje (UX) je križanje s velikim brojem točaka prekida. Između svaka dva bita roditelja slučajno se određuje hoće li to biti točka prekida. Svako dijete na taj način dobije oko 50% bitova svakog roditelja. Primjer uniformnog križanja prikazan je na slici 2.9.

Prethodno opisane metode križanja stvaraju po dva djeteta. Često se koristi varijanta križanja gdje operator stvara samo jedno dijete.



Slika 2.9: Primjer uniformnog križanja genotipske jedinke

Križanje brojeva s pomičnim zarezom

Kod križanja brojeva s pomičnim zarezom koriste se vrijednosti roditelja $R1$ i $R2$. Ako vrijednosti roditelja označimo s $rješenje_{R1}$ i $rješenje_{R2}$, križanjem se može dobiti dijete $rješenje_D$ preko:

$$rješenje_D = \gamma \cdot rješenje_{R1} + (1 - \gamma) \cdot rješenje_{R2} \tag{2.3}$$

Broj $\gamma \in [0, 1]$ je koeficijent kojim se određuje hoće li dijete biti sličnije roditelju $R1$ ili roditelju $R2$. Broj γ može biti:

- konstanta ($\gamma = 0,5$) i time će dijete biti aritmetička sredina roditelja,
- nasumično odabran preko jednolike ili neke druge razdiobe i
- dobiven nekom drugom formulom koja koristi vrijednosti funkcije vrednovanja roditelja.

Prva dva načina koriste informacije iz genotipskog prostora, a treća metoda koristi i vrijednosti funkcije vrednovanja. Time se ostvaruje dodatan selekcijski pritisak. Primjer formule kojom se računa γ u trećoj metodi je:

$$\gamma = \frac{vrednujRješenje(R2)}{vrednujRješenje(R1) + vrednujRješenje(R2)} \tag{2.4}$$

gdje su $vrednujRješenje(R1)$ i $vrednujRješenje(R2)$ vrijednosti funkcije vrednovanja roditelja.

Križanje permutacijskog genotipa

Križanje permutacijskog genotipa nešto je složeniji postupak od prethodno navedenih križanja jer je kompliciranije zadržati legalnost rješenja, a pritom zadržati

svojstva križanja. Ipak, postoji puno metoda za križanje permutacijskog genotipa gdje se zadržavaju svojstva oba roditelja. Neki od njih su: rubno križanje (engl. *edge crossover*), kružno križanje (engl. *cycle crossover*), križanje u poretku (engl. *order crossover*), križanje u jednoj točki s djelomičnim očuvanjem (engl. *one-point partial preservation crossover*) i drugi (Prokopec, 2009). Ovdje će biti opisano rubno križanje.

Rubno križanje prvo kreira tablicu susjedstva koju čine svi parovi susjednih elemenata oba roditelja. Na prvo mjesto djeteta postavlja se prvi element. Zatim se u svakom koraku odabiru svi parovi koji sadrže trenutačni element, a drugi element para nije već postavljen u djetetu. Susjedstvo se ugrađuje u dijete, ali prioritet imaju ona susjedstva koja se nalaze u oba roditelja. Ako ne postoji susjedstvo koje bi se moglo ugraditi u dijete, sljedeći element bira se nasumično.

Na ovaj način nastoji se što je moguće više očuvati informacije o susjedstvu iz oba roditelja. Od navedenih operatora križanja permutacijskog genotipa, ovaj operator je jedan od sporijih.

2.2.3. Odabir roditelja

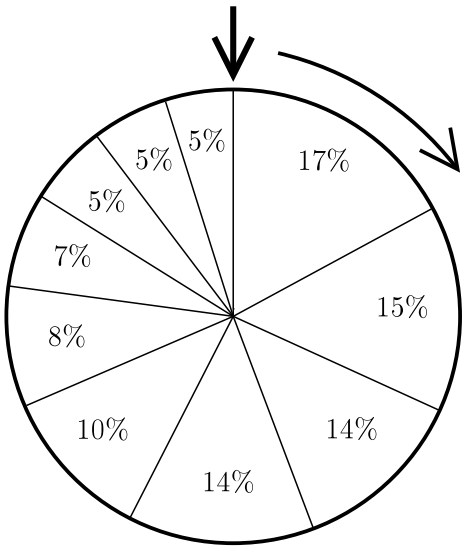
Operator odabira roditelja (engl. *selection*) ima funkciju odabira rješenja iz populacije. U ovom operatoru može se ostvariti selekcijski pritisak tako da kvalitetnija rješenja imaju veću vjerojatnost odabira nego manje kvalitetna rješenja. Time se sužava prostor pretraživanja i dovodi do konvergencije prema povoljnijim područjima prostora pretraživanja.

Proporcionalni odabir

Jedna od najčešćih ostvarenja odabira roditelja je proporcionalni odabir (engl. *roulette wheel seleection*). Svakoj jedinki pridružuje se vjerojatnost odabira prema:

$$p_k = \frac{vrednujRješenje(k)}{\sum_{i=1}^{n} vrednujRješenje(i)}, \quad k \in \{1, 2, \ldots, n\} \tag{2.5}$$

gdje je p_k vjerojatnost odabira k-tog rješenja, a $vrednujRješenje(k)$ je vrijednost funkcije vrednovanja. Na slici 2.10 nalazi se grafički prikaz kako se može vizualizirati odabir roditelja. Sva rješenja postavljena su na kolo, a površina koju zauzima pojedino rješenje je proporcionalno kvaliteti tog rješenja. Jedan roditelj odabire se tako da se zavrti kolo i pogleda na kojem rješenju je kolo zaustavljeno.



Slika 2.10: Primjer proporcionalnog odabira

Nedostaci ovakvog pristupa pojavljuju se ako postoje velike razlike između vrijednosti funkcija vrednovanja rješenja. Primjerice, ako se u ranoj fazi pronađe jedno rješenje s vrlo visokom vrijednošću funkcije vrednovanja, onda će to rješenje biti jako favorizirano u odnosu na ostala te time značajno smanjiti raznolikost populacije. Također, u kasnijim fazama vrijednosti funkcije vrednovanja mogu biti međusobno vrlo slične pa će selekcijski pritisak biti znatno slabiji. Jedan način za rješavanje problema je normalizacija vrijednosti funkcije vrednovanja preko linearnog skaliranja i to tako da omjer između najboljeg i najlošijeg rješenja bude pogodniji za proporcionalni odabir. Drugi način je rangiranje rješenja od najboljeg prema najgorem i računanjem vjerojatnosti na unaprijed određen način bez obzira na konkretnu vrijednost funkcije vrednovanja. Primjerice, mogu se postavljati vjerojatnosti kao da su vrijednosti funkcije vrednovanja redom n, $(n-1)$, $(n-2)$, $\ldots$, 2 i 1, gdje je n broj rješenja u populaciji, a rješenja sortirana od najboljeg do najgoreg. Takav odabir zove se odabir rangiranjem (engl. *ranking selection*).

Stohastičko univerzalno uzorkovanje

Stohastičko univerzalno uzorkovanje (engl. *stohastic universal sampling*) slično je proporcionalnom odabiru i omogućava odabir proizvoljnog broja rješenja pomoću generiranja samo jednog slučajnog broja (Baker, 1987). Prvo se kreira sličan vjerojatnosni kotač kao u proporcionalnom odabiru, gdje 0 označava početak kotača, tj. početak površine prvog rješenja, a 1 označava puni krug. Nasumično

se bira broj u intervalu:

$$\lambda \in \left[0, \frac{1}{m}\right] \tag{2.6}$$

gdje je m broj rješenja koji je potrebno odabrati. Formulom:

$$\lambda + \frac{i}{m}, \quad i = \{0, 1, \ldots, (m-1)\} \tag{2.7}$$

određuju se pozicije odabranih rješenja na kotaču.

Turnirski odabir

Osim prethodno navedenih odabira često se koristi i k-turnirski odabir. Nasumično se bira skup od k rješenja iz populacije, a najbolje rješenje čini roditeljem. Prednost ovakve metode je veća brzina od proporcionalnog odabira i mogućnost kontroliranja selekcijskog pritiska varijablom k. Često se koriste 2-turnirski i 3-turnirski odabiri. U nekim algoritmima koristi se i varijanta gdje se nasumično odaberu 3 rješenja. Dva najbolja rješenja čine roditelje, a novo rješenje nakon iteracije nadomješta treće, najmanje kvalitetno rješenje.

2.2.4. Ugrađivanje rješenja u populaciju

Dio algoritama evolucijskog računanja treba na neki način ugrađivati skup novih rješenja u postojeću populaciju. Neka je λ broj rješenja koje treba ugraditi u populaciju, a n veličina populacije. Potrebno je odrediti rješenja koja će ostati u populaciji i rješenja koja će biti zamijenjena novim rješenjima. Ako je $\lambda = 1$ tada se algoritam naziva eliminacijski algoritam (engl. *steady-state algorithm*), a ako je $\lambda = n$ onda je to generacijski algoritam (engl. *generational algorithm*).

Odabir rješenja za zamjenu, tj. odabir preživjelih jedinki, obavlja se prema vrijednosti funkcije vrednovanja ili prema starosti rješenja. Odabir rješenja za zamjenu prema vrijednosti funkcije vrednovanja najčešće se upotrebljava i zamjenjuje λ najgorih rješenja novim rješenjima. Metoda zamjene prema starosti rješenja zamjenjuje najstarija rješenja u populaciji novim rješenjima.

Bitna stvar kod dodavanja novih rješenja je problem duplikata. U populaciju je moguće dodati rješenje koje već postoji u toj populaciji. Takva pojava smanjuje raznolikost rješenja u populaciji i može dovesti do prerane konvergencije. U operator ugrađivanja rješenja u populaciju trebalo bi ugraditi provjeru postoji li već to rješenje u populaciji. Ako postoji, onda se rješenje ne ugrađuje u populaciju.

Druga bitna stvar kod izmjena generacija populacija je elitizam. Elitizam je strategija u kojoj se nastoji da najbolje rješenje uvijek bude odabrano za sljedeću

generaciju. Time je osigurano da se najbolje pronađeno rješenje uvijek nalazi u populaciji i na kraju izvršavanja algoritma lako izdvoji kao krajnje rješenje problema.

2.2.5. Uvjet zaustavljanja

Svi algoritmi evolucijskog računanja rade u iteracijama i potrebno je u nekom trenutku zaustaviti njihovo izvršavanje. Uvjeti zaustavljanja algoritma mogu biti:

- broj iteracija algoritma,
- vremensko trajanje izvršavanja algoritma,
- ciljana vrijednost funkcije vrednovanja,
- stagnacija algoritma ili
- kombinacija navedenih uvjeta zaustavljanja.

Najjednostavniji uvjeti zaustavljanja su dostizanje određenog broja iteracija algoritma ili maksimalnog vremenskog trajanja algoritma. Ako se zna ciljana kvaliteta rješenja, algoritam se može zaustaviti nakon njenog dostizanja. Ipak, algoritmi evolucijskog računanja nemaju jamstvo dostizanja zadane kvalitete rješenja. Složeniji uvjet zaustavljanja algoritma je detekcija stagnacije. Stagnacija se može prepoznati na više načina, npr. po velikoj sličnosti rješenja unutar populacije ili po većem broju generacija bez pronalaska novog najboljeg rješenja. Na praktičnim problemima najčešće se koristi kombinacija više navedenih uvjeta zaustavljanja algoritma.

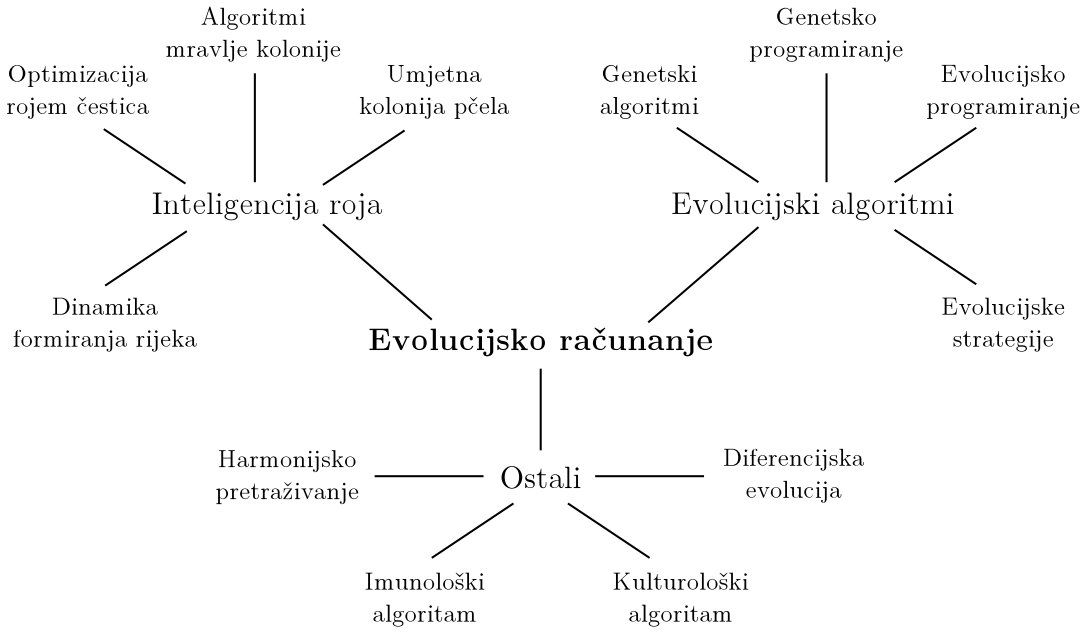
2.3. Algoritmi evolucijskog računanja

Algoritmi evolucijskog računanja dijele se u tri osnovne kategorije:

- evolucijske algoritme,
- inteligenciju roja i
- ostale.

Na slici 2.11 prikazane su kategorije algoritama evolucijskog računanja, a za svaku kategoriju navedeno je nekoliko pripadajućih algoritama.

Charles Darwin autor je prve moderne teorije o evoluciji živih bića. Pokušao je objasniti raznolikost vrsta u raznim dijelovima svijeta i postupne promjene



Slika 2.11: Podjela algoritama evolucijskog računanja

uočene u fosilnom svijetu. Zaključio je kako se kroz duga vremenska razdoblja razne biljne i životinjske vrste mijenjaju ili nestaju, ali nastaju i nove vrste. Uzrok takvim promjenama je prirodni odabir, tj. očuvanje boljih i prilagodljivijih vrsta u borbi za opstanak. Osnovne postavke Darwinove teorije mogu se svesti na sljedeće: populacije ne mogu neograničeno rasti, jedinke neke vrste pokazuju varijacije, u prosjeku se producira više potomaka nego što je potrebno za nadomještanje roditelja, postoji nadmetanje za preživljavanjem i preživljavaju najbolje prilagođene varijacije.

Evolucijski algoritmi (engl. *evolutionary algorithms*) su evolucijom nadahnuti postupci optimiranja, učenja i modeliranja računarskih problema. Osnovna zamisao evolucijskih algoritama je postupno prilagođavanje populacije trenutačnih rješenja prema sve poželjnijim i kvalitetnijim rješenjima. Evolucijski algoritmi dijele se na: (a) genetske algoritme, (b) genetsko programiranje, (c) evolucijsko programiranje i (d) evolucijske strategije. Genetski algoritmi najpopularnija su i najčešće korištena tehnika evolucijskih algoritama. Koriste se za rješavanje optimizacijskih problema. U genetskom programiranju jedinke predstavljaju računalne programe, a funkcija vrednovanja određuje kvalitetu rješavanja određenog problema. U evolucijskom programiranju jedinke su također računalni programi, ali operatori ne mijenjaju strukturu programa, već parametre. Evolucijske strategije slične su genetskim algoritmima, ali jedinke čine vektori realnih brojeva, a

najčešće koriste samopodešavajuće parametre mutacije. U poglavlju 2.3.1 bit će detaljnije opisani genetski algoritmi i njegove najčešće varijacije.

Inteligencija roja (engl. *swarm intelligence*) spada pod izranjajuću inteligenciju (engl. *emergent intelligence*) i temelji se na činjenici da veliki broj jednostavnih nezavisnih elemenata u zajednici može iskazivati inteligentno ponašanje. Primjere inteligencije roja možemo vidjeti u prirodi, npr. kolonijama mrava ili pčela, jatima ptica ili kapljicama vode. Ovakav sustav je jednostavan i decentraliziran, ali istodobno fleksibilan, robustan i samoorganizirajući. U poglavlju 2.3.2 bit će detaljnije opisan algoritam mravlje kolonije, najčešće korišten algoritam inteligencije roja.

Pod ostale algoritme evolucijskog računanja svrstavamo one algoritme koji imaju svojstva algoritama evolucijskog računanja, ali ih ne možemo svrstati pod evolucijske algoritme ili inteligenciju roja. Primjeri algoritama koji spadaju u tu kategoriju su imunološki algoritam i algoritam harmonijskog pretraživanja. Imunološki algoritam detaljno je opisan u poglavlju 2.3.3, a algoritam harmonijskog pretraživanja u poglavlju 2.3.4.

2.3.1. Genetski algoritmi

Genetski algoritmi su najpopularnija podskupina evolucijskih algoritama. Svako rješenje predstavljeno je jednom jedinkom, a populacija rješenja čini simuliranu populaciju jedinki. U generacijskom genetskom algoritmu svaka iteracija simulira jednu generaciju populacije. Populacija sljedeće generacije kreira se iz populacije trenutačne generacije pomoću operatora odabira, križanja i mutacije. Operator odabira predstavlja biranje roditelja djeteta, križanjem se dobiva dijete, a operatorom mutacije simuliraju se mutacije evolucije.

Simuliranje evolucije započelo je sredinom 1950-ih godina (Barricelli et al., 1954), a primjena evolucije na rješavanje optimizacijskih problema započela je 1960-ih godina (Bremermann, 1962). Primarni operator prvih programskih ostvarenja genetskog algoritma je operator mutacije. Do sredine 1980-ih godina genetski algoritmi nisu bili često upotrebljavani za rješavanje praktičnih problema, već krajem 1980-ih započinje njihova sve šira primjena u industriji.

Pseudokod generacijskog genetskog algoritma prikazan je u algoritmu 1. Varijabla *veličinaPopulacije* definira veličinu populacije, funkcija *kreirajPopulaciju* stvara početnu populaciju rješenja, a funkcija *vrednujRješenja* je funkcija vrednovanja nad primljenim rješenjima. Uobičajene veličine populacije su između 20

i 500. Obično se nastoji stvoriti elitistički algoritam pa se u početku iteracije populaciji P' dodaje najbolje rješenje iz trenutačne populacije. U ovom primjeru stvaraju se po dva djeteta iz para roditelja, ali vrlo često koristi se varijanta gdje se stvara samo jedno dijete.

Algoritam 1 Generacijski genetski algoritam

```
populacija ← kreirajPopulaciju(veličinaPopulacije)
vrednujRješenja(populacija)
ponavljaj dok ¬gotov
    P' ← ∅
    ponavljaj dok |P'| < veličinaPopulacije
        {R1, R2} ← odaberiRoditelje(populacija)
        {D1, D2} ← križaj(R1, R2)
        mutiraj(D1)
        mutiraj(D2)
        ugradiUPopulaciju(P', D1)
        ugradiUPopulaciju(P', D2)
    kraj
    populacija ← P'
    vrednujRješenja(P)
kraj
```

Algoritam 2 Eliminacijski genetski algoritam

```
populacija ← kreirajPopulaciju(veličinaPopulacije)
vrednujRješenja(populacija)
ponavljaj dok ¬gotov
    {R1, R2} ← odaberiRoditelje(populacija)
    D ← križaj(R1, R2)
    mutiraj(D)
    vrednujRješenje(D)
    ugradiUPopulaciju(populacija, D)
kraj
```

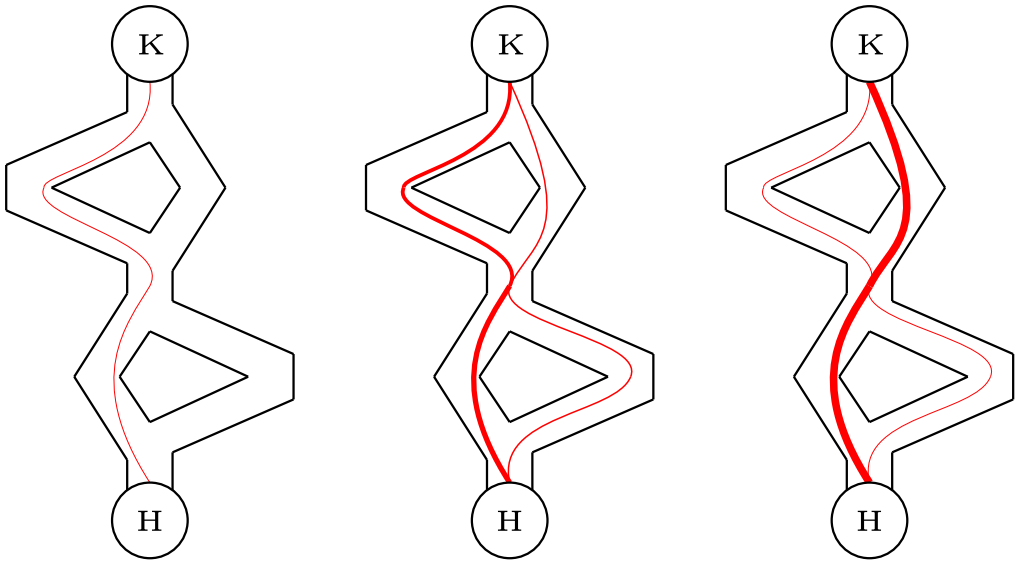
Primjer eliminacijskog genetskog algoritma prikazan je u algoritmu 2. Može se primijetiti da ova varijanta genetskog algoritma mijenja samo jedno rješenje iz populacije po iteraciji.

Postoji veliki broj naprednijih verzija genetskih algoritama (Affenzeller et al., 2009) u kojima se nastoji odgoditi stagnacija i povećati kvaliteta ili brzina algoritma. Neke od njih su RAPGA (engl. *Relevant Alleles Preserving Genetic Algorithm*), SEGA (engl. *SEgregative Genetic Algorithm*) i SASEGASA (engl. *Self-Adaptive SEgregative Genetic Algorithm with Simulated annealing Aspects*). Jedna od osnovnih promjena koje uvodi RAPGA je promjenjiva veličina populacije. U početku rada algoritma populacija je vrlo velika, a kako algoritam napreduje, veličina populacije se smanjuje. To omogućava široku pretragu prostora rješenja u početnim stadijima algoritma, a u kasnijim stadijima omogućava brzu pretragu oko najboljih rješenja. Algoritam SEGA populaciju dijeli na više izoliranih podpopulacija koje postupno spaja u jednu veliku populaciju, a algoritam SASEGASA dodatno poboljšava algoritam SEGA uvođenjem adaptivnog selekcijskog pritiska i bolju kontrolu nad spajanjem populacija.

2.3.2. Algoritmi mravlje kolonije

Algoritam mravlje kolonije je tehnika rješavanja optimizacijskih problema koji se mogu preslikati u problem pretraživanja puteva kroz graf (Dorigo i Stützle, 2004). Inspiriran je ponašanjem mrava prilikom dobavljanja hrane. Primijećeno je kako mravi vrlo često pronalaze najkraći put između izvora hrane i svoje kolonije. Na slici 2.12 opisan je primjer kako mravi pronalaze najkraći put. Slovom K označena je kolonija, a slovom H izvor hrane. Na slici 2.12a prikazan je trenutak nakon što je jedan mrav otkrio izvor hrane. Pritom je ostavio feromonski trag koji je na slici označen crvenom crtom. Nakon što je pronašao izvor hrane, istim putem se vratio u koloniju i ostavio feromonski trag. Ostali mravi počinju pratiti taj put, ali neki mravi otkrivaju nove puteve kao što se može primijetiti na slici 2.12b. Neki od novih puteva nisu kraći, ali drugi jesu. S obzirom na to da se kraći put brže prolazi, na kraćim putevima feromonski trag brže se obnavlja i time postaje sve jači. Nakon nekog vremena upravo će najkraći put biti najjače označen feromonskim tragom, a na dužim putevima trag će biti sve slabiji jer postupno isparava, kao što je prikazano na slici 2.12c.

Ponašanje mrava možemo modelirati težinskim grafom. Jedan vrh predstavlja koloniju, a neki drugi vrh izvor hrane. Ostali vrhovi povezani su bridovima koji predstavljaju sve moguće puteve kojima mrav može proći. Bridovi su različitih težina ovisno o duljini tog puta. Svakom bridu pridružen je i realni broj τ_{ij} koji predstavlja količinu feromonskog traga na bridu iz vrha i u vrh j. Najjednostavniji



(a) Nakon prvog prolaza **(b)** Nakon malo prolaza **(c)** Nakon puno prolaza

Slika 2.12: Feromonski tragovi prema broju prolazaka mrava

algoritam koji imitira ponašanje mrava je algoritam S-ACO. Vjerojatnost odabira brida iz vrha i u vrh j računa se prema formuli:

$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij}^\alpha}{\sum\limits_{l \in \mathcal{N}_i^k} \tau_{il}^\alpha}, & \text{ako } j \in \mathcal{N}_i^k; \\ 0, & \text{ako } j \notin \mathcal{N}_i^k. \end{cases} \tag{2.8}$$

gdje $\mathcal{N}_i^k$ označava skup vrhova u koje je moguće prijeći u k-tom koraku iz vrha i, a α je odabrana konstantna vrijednost. Koristeći izračunate vjerojatnosti mravi putuju grafom sve dok ne dođu do izvora hrane. Zatim se za svaki brid kojim je mrav prošao ažurira feromonski trag prema formuli:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau^k \tag{2.9}$$

gdje je $\Delta\tau^k$ broj proporcionalan kvaliteti pronađenog rješenja. Nakon prolaska svih mrava u k-tom koraku, simulira se isparavanje feromonskih tragova kao što je prikazano formulom:

$$\tau_{ij} \leftarrow \tau_{ij} \cdot (1 - \rho) \tag{2.10}$$

gdje se parametrom ρ kontrolira stopa isparavanja.

Mravlji sustav $\mathcal{MAX}$–$\mathcal{MIN}$

Svi algoritmi mravlje kolonije temelje se na istom principu, ali se razlikuju u računanju vjerojatnosti odabira pojedinih bridova, načinu i vremenu ažuriranja

i isparavanja feromonskih tragova te drugim operatorima. Jedna od najuspješnijih varijanti je mravlji sustav $\mathcal{MAX}$–$\mathcal{MIN}$ (engl. $\mathcal{MAX}$–$\mathcal{MIN}$ *Ant System* – $\mathcal{MMAS}$). Algoritam jako favorizira najboljeg mrava tako da samo najbolji put trenutačne iteracije ili najbolji put ikada pronađen ažurira feromonske tragove. Zbog intenzivnog rasta feromonskih tragova u samo manjem podskupu bridova, uvodi se maksimalna i minimalna vrijednost feromonskih tragova τ_{max} i τ_{min}. Početne vrijednosti feromonskih tragova u svim bridovima je τ_{max}, a stopa isparavanja je vrlo niska, obično oko 2%. Također, kada dođe do stagnacije, feromonski tragovi u svim bridovima postavljaju se na početnu vrijednost.

U algoritmu 3 može se vidjeti pseudokod algoritma $\mathcal{MM}$AS. Broj a je konstanta koja određuje omjer između τ_{max} i τ_{min}. Može se primijetiti kako se vrijednosti varijabli τ_{max} i τ_{min} mijenjaju svaki puta kad se pronađe novi najbolji put. Stagnacija se detektira ako u određenom broju iteracija ne bude pronađeno novo najbolje rješenje.

2.3.3. Imunološki algoritmi

Algoritam umjetnog imunološkog sustava je metaheuristički algoritam za rješavanje optimizacijskih problema inspiriran imunološkim sustavom živih organizama. Imunološki sustav je sustav bioloških struktura i procesa unutar živog organizma koji ga brani od bolesti identificiranjem i uništavanjem stranih mikroorganizama. Najbitnije stanice u imunološkom sustavu su limfociti. Postoji više vrsta limfocita, a ključni su B-limfociti i T-limfociti. B-limfociti su stanice specijalizirane za prepoznavanje oznake na površini stanica (antigena) i sazrijevanje plazma stanica koje izlučuju protutijela. T-limfociti pomažu B-limfocitima u proizvodnji protutijela te direktno identificiraju i ubijaju strane mikroorganizme. Razlikuju se prirođeni i stečeni imunitet. Stečeni imunitet dalje se dijeli na prirodno stečeni i umjetno stečeni. Prirodno stečeni imunitet može biti pasivan (npr. antitijela majke dobivena majčinim mlijekom) i aktivan (antitijela koja imunološki sustav proizvede i koja ostaju u tijelu). Umjetno stečeni imunitet može se steći pasivno (unosom gotovih antitijela) ili aktivno (cijepljenjem). S algoritamske strane posebno je zanimljiv prirodno stečeni aktivni imunitet jer se u kontaktu s uzročnicima bolesti razvija specifična imunost. Taj proces traje danima, tjednima ili mjesecima. Kada u tijelo uđe strani mikroorganizam za koji još nije stečen imunitet, procesima kloniranja i mutacije postupno se stvara sve veći broj B-limfocita koji efikasno uništavaju strani mikroorganizam.

Algoritam 3 Algoritam $\mathcal{MM}$AS

$\tau_{min} \leftarrow \frac{\tau_{max}}{a}$
za svaki i, j
  $\tau_{ij} \leftarrow \tau_{max}$
kraj
ponavljaj dok $\neg gotov$
  ako *stagnira* **onda**
    za svaki i, j
      $\tau_{ij} \leftarrow \tau_{max}$
    kraj
  kraj
  za $i = 1$ **do** n
    pokreniMrava()
  kraj
  $\tau_{max} \leftarrow \frac{1}{\rho \cdot vrednujRješenje(najboljiPutIkad)}$
  $\tau_{min} \leftarrow \frac{\tau_{max}}{a}$
  za svaki i, j
    $\tau_{ij} \leftarrow \max\Big((1-\rho) \cdot \tau_{ij}, \tau_{min}\Big)$
  kraj
  za $(i, j) \in najboljiPutIkad$
    $\tau_{ij} \leftarrow \min\Big(\tau_{ij} + \frac{1}{vrednujRješenje(najboljiPutIkad)}, \tau_{max}\Big)$
  kraj
kraj

Jednostavni imunološki algoritam

Jednostavni imunološki algoritam (engl. *Simple Immune Algorithm – SIA*) je algoritam kojim se mogu rješavati klasifikacijski i optimizacijski problemi (Cutello i Nicosia, 2002). Osnovna zamisao je slična genetskom algoritmu bez operatora križanja. Antitijelo je rješenje problema koji se rješava, a antigen je sam problem. Što je antitijelo bolje prilagođeno antigenu, to je afinitet veći. Postoji populacija antitijela nad kojom se izvršavaju operatori mutacije i kloniranja te time postupno povećava afinitet prema antigenu.

Algoritam 4 Jednostavni imunološki algoritam

```
antitijela ← kreirajPopulaciju(veličinaPopulacije)
vrednujRješenja(antitijela)
ponavljaj dok ¬gotov
  P' ← ∅
  za svaki antitijelo ∈ antitijela
    za i = 1 do brojKlonova
      klon ← antitijelo
      mutiraj(klon)
      ugradiUPopulaciju(P', klon)
    kraj
  kraj
  antitijela ← odaberiNajbolje(P', veličinaPopulacije)
  vrednujRješenja(antitijela)
kraj
```

U algoritmu 4 napisan je pseudokod jednostavnog imunološkog algoritma. Operator $odaberiNajbolje(P, n)$ odabire n najboljih antitijela iz populacije antitijela P. Stopa mutacije obično je puno veća nego kod genetskih algoritama te se zbog toga naziva hipermutacijom. Broj klonova je konstantan i definira koliko će se puta klonirati svako antitijelo iz populacije.

Druge varijante imunološkog algoritma, poput algoritma CLONALG (de Castro i Von Zuben, 2002), mijenjaju broj klonova i stopu mutacije ovisno o afinitetu pojedinog antitijela. Broj klonova je proporcionalan, a stopa mutacije je obrnuto proporcionalna afinitetu antitijela.

2.3.4. Harmonijsko pretraživanje

Algoritam harmonijskog pretraživanja (engl. *harmony search algorithm*) je metaheuristički i optimizacijski algoritam inspiriran glazbom (Geem, 2009). Inspiriran je glazbenicima koji nastoje ostvariti savršenu harmoniju glazbe, tj. stvara analogiju između potrage za savršenom harmonijom i potragom za najboljim rješenjem. Postignuta melodija glazbe predstavlja jedno rješenje optimizacijskog problema. Kako bi analogija bila ispravna i upotrebljiva, potrebno je pojednostaviti i opisati improvizaciju kod iskusnih glazbenika. Kada glazbenik improvizira, može svirati: (a) postojeću melodiju (melodiju koju zna otprije), (b) izmijenjenu postojeću melodiju i (c) potpuno novu melodiju. Prva opcija naziva se harmonijska memorija (engl. *harmony memory*), druga je podešavanje nota (engl. *pitch adjustment*), a treća je stvaranje slučajnosti (engl. *randomization*).

Uloga harmonijske memorije je mogućnost kombiniranja poznatih i kvalitetnih melodija s novim melodijama. Parametar $r_{prihvati} \in [0, 1]$ naziva se stupanj prihvaćanja harmonijske memorije. Prevelika vrijednost tog parametra onemogućuje kvalitetnu pretragu prostora rješenja, a premala vrijednost uzrokuje prenisku razinu konvergencije. Obično se upotrebljava vrijednost između 0,7 i 0,95.

Podešavanje nota uzima neku postojeću melodiju i mijenja neke njezine dijelove. Razina podešavanja nota kontrolira se pomoću parametara b_{raspon} i r_{ra}. Parametar b_{raspon} definira najveći raspon izmjene nota i može se poistovjetiti sa stopom mutacije u operatoru mutiranja. Parametar $r_{ra} \in [0, 1]$ određuje s kojom vjerojatnošću će postojeća melodija biti izmijenjena. Uobičajene vrijednosti parametra r_{ra} su između 0,1 i 0,5.

Stvaranje slučajnosti je treća opcija i pridonosi raznovrsnosti melodija. Omogućava šire pretraživanje prostora rješenja od podešavanje nota, smanjuje vjerojatnost zaustavljanja na lokalnom optimumu i povećava vjerojatnost pronalaska globalnog optimuma.

Pseudokod harmonijskog pretraživanja nalazi se na slici 5.

Algoritam 5 Harmonijsko pretraživanje

memorija ← *kreirajPopulaciju*(*veličinaMemorije*)
vrednujRješenja(*memorija*)
ponavljaj dok ¬*gotov*
  ako *slučajanBroj*([0, 1]) < $r_{prihvati}$ **onda**
    novaMelodija ← *odaberi*(*memorija*)
    ako *slučajanBroj*([0, 1]) < r_{ra} **onda**
      novaMelodija ← *mutiraj*(*novaMelodija*, b_{raspon})
    kraj
  inače
    novaMelodija ← *kreirajNovuMelodiju*()
  kraj
  vrednujRješenje(*novaMelodija*)
  ako *boljaOdNajgoreUMemoriji*(*novaMelodija*, *memorija*) **onda**
    ugradiUPopulaciju(*novaMelodija*, *memorija*)
  kraj
kraj

3. Raspodijeljeni algoritmi

Ideja paralelizacije evolucijskih algoritama postoji od 1960-ih godina kada je paralelizacija bila predložena kao način povećanja raznolikosti rješenja i odgađanja stagnacije (Bossert, 1967). Ipak, nedostatak računalne moći odgodio je efikasna programska ostvarenja sve do 1980-ih godina. U to vrijeme najpopularnija metoda paralelizacije evolucijskog računanja bila je krupnozrnata paralelizacija. Prva ostvarenja sitnozrnate paralelizacije napravljena su krajem 1980-ih godina (Gorges-Schleuter, 1989).

U zadnjih nekoliko godina paralelizacija postaje još bitnija. Procesorska moć pojedinačne jezgre raste sve sporije, ali se povećava broj jezgri unutar procesorskih jedinica. Zbog navedenog, moderni algoritmi moraju biti što je moguće više paralelizirani.

Algoritmi evolucijskog računanja jako su pogodni za paralelizaciju i mogu ostvariti brojna pozitivna svojstva (Alba i Tomassini, 2002). Uzrok jednostavnosti paralelizacije je u modularnosti većine algoritama, jednostavnosti razmjene i ugrađivanja rješenja te mogućnosti paralelne obrade rješenja unutar populacije. Prednosti paralelizacije evolucijskog računanja mogu biti: (a) ubrzavanje izvršavanja, (b) bolja kvaliteta dobivenih rješenja, (c) jednostavna integracija različitih algoritama i (d) dobivanje skupa različitih rješenja za isti problem. Ostvarene prednosti paralelizacije jako ovise o problemu koji se rješava, korištenim algoritmima, načinu paralelizacije i odabranim parametrima.

3.1. Vrste paralelizacija

Jedan od faktora bitnih pri odabiru modela paralelizacije evolucijskog računanja je arhitektura računala na kojima će se evolucijsko računanje izvršavati. Prema programskim instrukcijama i memoriji postoje sljedeće arhitekture računala: SISD, SIMD, MISD i MIMD (Hwang, 1993). Arhitektura SISD (engl. *Single Instruction, Single Data*) predstavlja standardni Von Neummannov model

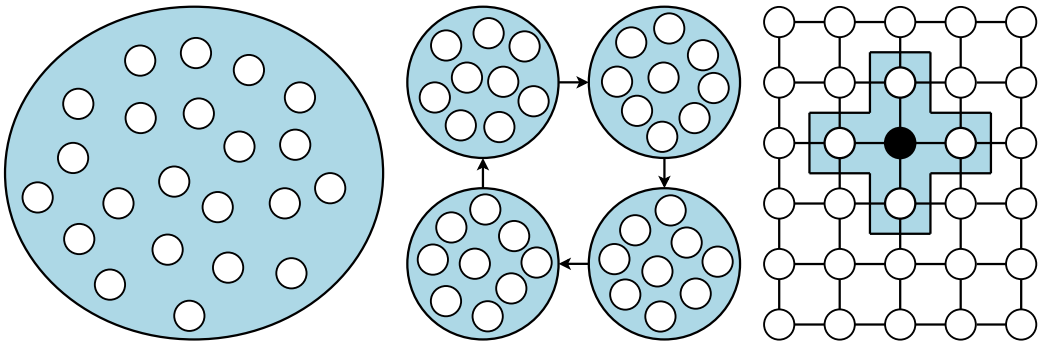
računala koji ima jedan procesor i jednu memoriju, tj. u jednom trenutku jedna programska instrukcija obrađuje jedan podatak. Računala s jednojezgrenim procesorom spadaju u takvu arhitekturu. Arhitektura SIMD (engl. *Single Instruction, Multiple Data*) označava arhitekturu gdje jedna instrukcija sinkrono obrađuje više podataka. Ovakva arhitektura računala je dobra ako problem može biti vektoriziran i omogućava paralelnu obradu velikih količina podataka istim instrukcijama. Osim usko specijaliziranih superračunala, ovakvu arhitekturu koriste moderni grafički procesori. U zadnjih nekoliko godina trend izvršavanja programa na grafičkim procesorima poznat kao GPGPU (engl. *General-Purpose computing on Graphics Processing Units*) sve više popularizira tu arhitekturu. Arhitektura MISD (engl. *Multiple Instruction, Single Data*) izvršava različite operacije nad istim podatkom, no rijetko se koristi. Jedan od primjera upotrebe su usko specijalizirani sustavi za obradu signala. Računala s arhitekturom MIMD (engl. *Multiple Instruction, Multiple Data*) imaju više procesora koji rade asinkrono i nezavisno. Mogu imati zajedničku memoriju (engl. *shared memory*) ili raspodijeljenu memoriju (engl. *distributed memory*). Kod zajedničke memorije procesori dijele istu memoriju, ali je čitanje i pisanje ekskluzivno. Raspodijeljena memorija označava model gdje svaki procesor ima vlastitu memoriju, a komunikacija se odvija preko poruka.

Modeli paralelizacije jednokriterijskih algoritama evolucijskog računanja su:

- jednopopulacijski model,
- krupnozrnati model,
- sitnozrnati model i
- hibridni paralelni evolucijski algoritam.

Modeli paralelizacije prikazani su na slici 3.1 gdje su populacije prikazane kružićima. Skupovi kandidata operatora odabira obojani su plavom bojom. Može se primijetiti kako kod sitnozrnatog modela kandidati odabira čine samo susjedna rješenja.

Osim navedenih modela, postoji trivijalni paralelni evolucijski algoritam koji predstavlja paralelno i potpuno nezavisno izvršavanje više slijednih algoritama na više procesorskih jezgri ili računala. Ovakav model koristi se za dobivanje skupa različitih krajnjih rješenja i brže dobivanje statistike slijednih algoritama.



(a) Jednopopulacijski model **(b)** Krupnozrnati model **(c)** Sitnozrnati model

Slika 3.1: Modeli paralelizacije jednokriterijskih algoritama evolucijskog računanja

3.1.1. Jednopopulacijska paralelizacija

Jednopopulacijska paralelizacija služi za ubrzavanje slijednih algoritama. Rezultati dobiveni jednopopulacijskom paralelizacijom ne razlikuju se od rezultata dobivenih slijednim algoritmom. Nastoje se paralelizirati one operacije koje zahtijevaju najviše vremena.

Jednopopulacijska paralelizacija ostvaruje se tako da jedno računalo predstavlja voditelja, a sva ostala računala predstavljaju radnike. Primjerice, paralelizacija generacijskog genetskog algoritma može biti izvedena na sljedeći način. Operatori vrednovanja, križanja i mutacije izvode se na radnicima jer su u praktičnim problemima to obično skupe operacije. Na voditelju je pohranjena trenutačna populacija i obavlja se operator odabira. U svakoj iteraciji radnicima se šalju po dva odabrana rješenja. Svaki radnik ih križa, mutira i evaluira te tako kreirano dijete šalje voditelju. Voditelj ponavlja postupak sve dok ne popuni novu populaciju. Opisani model idealan je za izvršavanje na arhitekturi MIMD sa zajedničkom memorijom.

Ovako ostvarena paralelizacija može rezultirati znatno bržim izvršavanjem algoritma, ali ne pridonosi kvaliteti dobivenih rješenja. Zbog toga se koristi s algoritmima ili problemima koji nemaju koristi od drugih vrsta paralelizacije. Potrebno je napomenuti kako ovakva paralelizacija ne pridonosi brzini izvršavanja ako su troškovi komunikacije i sinkronizacije veći od troškova izvršavanja operatora.

3.1.2. Krupnozrnata paralelizacija

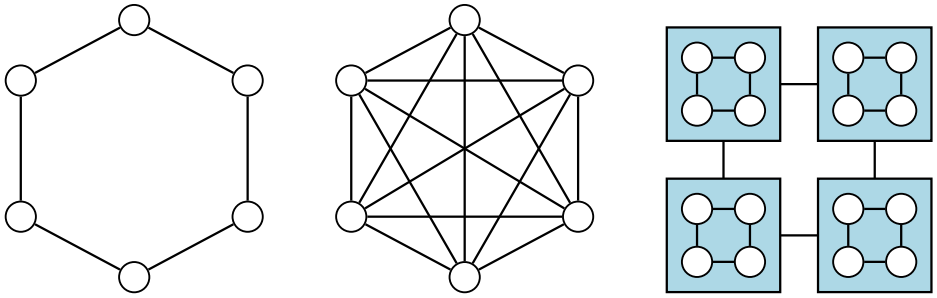
Kod krupnozrnate paralelizacije (engl. *coarse-grained*) sva rješenja nalaze se u više populacija. Svaka populacija naziva se otokom pa je drugi naziv ovog modela otočni model (engl. *island model*). Nad svakom populacijom radi jedan algoritam. Otoci povremeno razmjenjuju rješenja i na taj način nastoji se u svaku populaciju ugraditi dijelove dobrih rješenja.

Krupnozrnata paralelizacija izvodi se najbolje i implementira se najjednostavnije na arhitekturi MIMD te je zbog toga najpopularnija metoda paralelizacije. Svaka populacija može se izvoditi na jednom procesoru, a ako je model memorije raspodijeljeni model, onda se migracija obavlja preko poruka. Na taj način omogućeno je potpuno nezavisno izvođenje algoritama, a trošak komunikacije ne utječe znatno na performanse cijelog sustava.

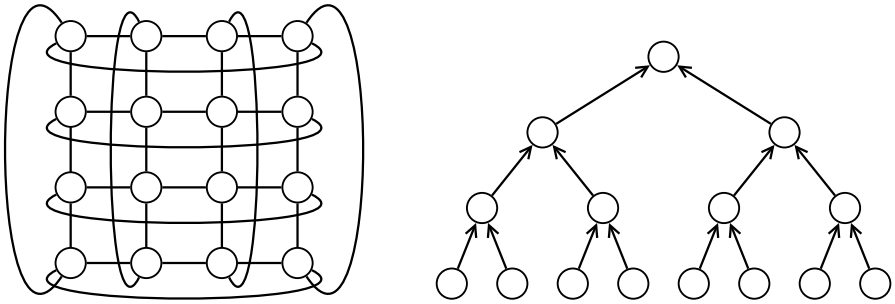
Parametri migracije

Parametri migracije definiraju na koji način će migracija biti obavljena. Pod parametre migracije spadaju: (a) topologija migracije, (b) učestalost migracije, (c) broj migrirajućih rješenja, (d) odabir rješenja za migraciju i (e) ugrađivanje primljenih rješenja.

Postoji puno različitih topologija migracije, a neke od njih prikazane su na slici 3.2. Topologije su prikazane kao grafovi gdje svaki čvor predstavlja populaciju, a bridovi prikazuju čvorove koji međusobno komuniciraju. Ciklička topologija predstavljena je cikličkim grafom kroz koji se komunicira u jednom ili u oba smjera. Potpuna topologija prikazuje potpuni graf pa sve populacije šalju rješenja svim drugim populacijama. Ovakva topologija može biti dobra za manji broj čvorova, ali za veći broj čvorova komunikacijska mreža može biti preopterećena. Otočna topologija inspirirana je migracijama u prirodi. Populacije su podijeljene u više otoka (označeni su plavom bojom). Populacije unutar otoka češće razmjenjuju rješenja pa su rješenja unutar tih populacija slična i imaju svojstva jedne velike populacije. Kao što se odvija u prirodi, migracija između otoka odvija se puno rjeđe. Očekuje se kako će svaki otok pretraživati u jednom području prostora rješenja, a migracijama se nastoji kombinirati dobre dijelove rješenja iz različitih dijelova prostora rješenja. Toroidalna topologija predstavlja 2-dimenzionalnu mrežu populacija. Dodatno, populacije prvog i zadnjeg reda, kao i populacije prvog i zadnjeg stupca povezane su i također razmjenjuju rješenja. Stablasta topologija dijeli se u više razina. Najniža razina predstavlja izolirane populacije



(a) Ciklička topologija **(b)** Potpuno povezana **(c)** Otočna topologija



(d) Toroidalna topologija **(e)** Stablasta topologija

Slika 3.2: Vrste topologija

jer ne primaju rješenja drugih populacija, ali šalju svoja rješenja populacijama viših razina. Populacija najviše razine ne šalje svoja rješenja niti jednoj drugoj populaciji, a posredno prima i kombinira rješenja svih algoritama. Ovime se postiže nezavisnost rada dijela algoritama i pretraživanje različitih dijelova prostora rješenja. Osim opisanih topologija, često se upotrebljava i slučajna topologija. Kada neka populacija treba migrirati rješenje, nasumičnim odabirom bira odredište rješenja.

Učestalost migracije definira kako često će skupina rješenja iz neke populacije migrirati u neku drugu populaciju. Kada je migracija prečesta, tada može doći do uranjene stagnacije. Gubi se svojstvo da svaka populacija pretražuje određeni dio prostora rješenja i cijeli sustav poprima svojstva jedne velike populacije. Kada je migracija prerijetka, tada se onemogućuje suradnja između populacija. Moguće je da razlika između rješenja populacija bude prevelika i dijelovi primljenog rješenja neće moći biti ispravno ugrađeni u sljedeće generacije populacija. Potrebno je odabrati način na koji se mjeri učestalost migracije. Može biti odabran broj obavljenih iteracija, vremenski interval ili broj novokreiranih rješenja. Broj obavljenih iteracija nije dobar ako se upotrebljava više različitih vrsta algoritama jer vrijeme potrebno za obavljanje jedne iteracije jednog algoritma može

biti znatno drugačije od vremena potrebnog za obavljanje iteracije nekog drugog algoritma. Vremenski interval rješava spomenuti problem, ali ovisi o brzini računala. Broj novokreiranih rješenja nastoji smanjiti probleme oba načina tako da razmak između migracija definira brojem novih rješenja koje stvori algoritam.

Broj migrirajućih rješenja definira koliko će rješenja iz pojedine populacije biti odabrano za migraciju. Obično se za broj migrirajućih rješenja odabire manji broj jedinki. Ako se odabire veći broj jedinki, onda se mogu početi pojavljivati problemi slični kao kod previsoke učestalosti migracije.

Rješenja odabrana za migraciju mogu utjecati na brzinu konvergencije i vrijeme stagnacije. Najčešće metode odabira rješenja su: odabir uvijek najboljeg rješenja, slučajni odabir ili odabir kvalitetnijih rješenja s većom vjerojatnošću. Odabir uvijek najboljeg rješenja pridonosi bržoj konvergenciji, ali druge metode mogu odgoditi pojavu stagnacije.

S obzirom na to da su algoritmi evolucijskog računanja populacijski algoritmi, ugrađivanje primljenih jedinki može biti izvedeno na jednostavan način. Primjerice, na početku svake iteracije neka postojeća rješenja iz populacije mogu biti zamijenjena s primljenim rješenjima. Ugradnja u populaciju ostvaruje se upotrebom operatora opisanom u poglavlju 2.2.4.

Konfiguracije algoritama

Kada se paralelno izvršava više odvojenih algoritama, moguće je za sve algoritme koristiti jednaku konfiguraciju, ali određene prednosti mogu biti postignute ako se koriste različite konfiguracije. Potencijalne prednosti su: odgođena stagnacija, kvalitetnija konačna rješenja i robusnost postupka optimizacije. To je moguće jer svaki algoritam ima određene prednosti i nedostatke. Primjerice, neki algoritmi mogu biti više orijentirani na pretraživanje novih dijelova prostora rješenja, a drugi pokušavaju poboljšati postojeća rješenja. Ovime je moguće nedostatke nekih algoritama ispraviti prednostima drugih algoritama. Robusnost postupka optimiranja ostvaruje se zbog više različitih vrsta algoritama pa se povećava vjerojatnost izvršavanja algoritma s boljim parametrima. Pritom algoritmi s lošijim parametrima ne bi trebali smanjiti kvalitetu rada boljih algoritama. Razlike između algoritama mogu biti postignute drugačijim parametrima istih algoritama, različitim vrstama algoritama ili različitim evaluiranjem rješenja.

Različiti parametri istih algoritama uključuju druge stope mutacije, veličine populacije, ostvarenja operatora i drugo. Algoritmi koji imaju manje stope mu-

tacije ili manju populaciju predviđeni su za optimizaciju postojećih rješenja, a oni algoritmi koji imaju veće stope mutacije ili veće populacije bolje pretražuju šira područja prostora rješenja. Primjer različitih ostvarenja operatora je rješavanje problema trgovačkog putnika gdje svaki algoritam koristi drugačije ostvarenje operatora mutacije ili križanja.

Kombinacija različitih algoritama postiže se paralelnim izvršavanjem potpuno različitih algoritama. Primjerice, genetski algoritmi dobri su za širu pretragu prostora rješenja, algoritmi mravlje kolonije pokušavaju poboljšati trenutačno najbolje rješenje, a algoritmi umjetnog imunološkog sustava istražuju razne varijacije manje skupine najboljih rješenja.

Treća varijacija algoritama može se ostvariti različitim funkcijama vrednovanja. Svaki algoritam može drugačije računati kvalitetu pojedinog rješenja i na taj način isto rješenje ocijeniti drugačijom kvalitetom. Primjer za upotrebu ovakve metode je rješavanje problema koji ima izrazito skupo računanje funkcije vrednovanja. Mogu se ostvariti manje kvalitetne i vrlo brze aproksimacije funkcije vrednovanja te kvalitetnije i sporije aproksimacije. Tako će neki otoci moći znatno brže pretraživati prostor rješenja, ali neće moći jako precizno procijeniti stvarnu kvalitetu rješenja. Drugi otoci puno sporije pretražuju prostor rješenja, ali imaju precizniju funkciju vrednovanja te mogu obaviti kombiniranje dobrih rješenja i finu pretragu oko najboljih primljenih rješenja.

3.1.3. Sitnozrnata paralelizacija

Sitnozrnata paralelizacija (engl. *fine-grained*) predstavlja model paralelizacije gdje su rješenja podijeljena u n-dimenzionalnoj mreži. Svako rješenje čini vlastitu populaciju, a njom upravlja jedan procesor. U jednoj iteraciji algoritma, svaki procesor izvodi operator križanja isključivo sa susjednim rješenjima. Na slici 3.1c skup potencijalnih rješenja za križanje s crnim rješenjem prikazan je plavom bojom. Ovakav način migracije omogućava postupno širenje dobrih rješenja po cijeloj mreži.

Sitnozrnata paralelizacija zahtijeva brzu komunikaciju između procesora jer se komunikacija između susjednih rješenja odvija u svakoj iteraciji algoritma. Najprirodnije i najefikasnije okruženje za izvršavanje sitnozrnate paralelizacije je na računalima arhitekture SIMD jer će na taj način sve instrukcije biti sinkrono izvršene. Paralelizaciju je moguće ostvariti i na arhitekturi MIMD, ali na kraju iteracije svaki procesor mora čekati da svi ostali procesori završe s radom.

Zbog toga se ovakav model paralelizacije rjeđe upotrebljava nego drugi modeli paralelizacije.

3.1.4. Hibridni paralelni evolucijski algoritam

Hibridni paralelni evolucijski algoritmi kombiniraju više različitih modela paralelizacije. Primjerice, moguće je kombinirati krupnozrnati model paralelizacije s jednopopulacijskom paralelizacijom. Svaki otok ubrzan je jednopopulacijskom paralelizacijom, a otoci međusobno komuniciraju preko krupnozrnatog modela paralelizacije. Primjer ovakvog načina paralelizacije ostvaren je za predviđanje proteinske strukture (Tantar et al., 2007).

3.2. Tehnologije raspodjeljivanja

U današnje vrijeme sve su bliža fizička ograničenja ubrzavanja procesora preko povećavanja frekvencije. Dodatnim povećavanjem frekvencije procesora sve su izraženiji razni problemi poput problema zagrijavanja, neželjeni kvantni fenomeni i sl. Kako bi se zadržala stopa razvoja računalne moći, procesori se ubrzavaju paralelizacijom. Primjer paralelizacija unutar jedne procesorske jezgre je ubrzavanje pomoću cjevovoda, što ne zahtijeva izmjenu postojećih slijednih algoritama. Ipak, ubrzavanje pomoću cjevovoda procesorsku moć može ubrzavati samo do određene granice. Zbog toga je trenutačni trend izrada višejezgrenih procesora. Kako bi se potpuno iskoristile mogućnosti višejezgrenih procesora, algoritmi koji se izvršavaju moraju biti višedretveni.

U trenutku pisanja ovog rada postoje pristupačni procesori sa šest jezgri, ali za kompleksne probleme računalna moć jednog takvog računala nije dovoljna. U tom slučaju upotrebljavaju se: grozd računala (engl. *cluster*), splet računala (engl. *grid*) ili računalni oblak (engl. *cloud computing*). Pritom algoritmi moraju biti izrazito paralelni i međusobno komunicirati.

Zahtjevi bilo kojeg raspodijeljenog sustava su: otvorenost, transparentnost i skalabilnost. Otvoreni sustavi pri komunikaciji upotrebljavaju normirana pravila i time omogućuju jednostavnu integraciju s drugim programskim ostvarenjima. Transparentnost prikriva značajke raspodijeljenog sustava (npr. prikriva promjenu lokacije, dijeljenje resursa s drugim korisnicima ili kvarove) i olakšava razvoj raspodijeljenog sustava. Skalabilni sustavi olakšavaju održavanje dobrih performansi sustava kod povećavanja korisnika ili količine korištenih sredstava.

U ovom poglavlju opisat će se neke najpoznatije tehnologije koje se upotrebljavaju u raspodijeljenim algoritmima na više računala.

3.2.1. Komunikacija prijenosnim protokolima

Prijenosni protokoli UDP i TCP izravno koriste funkcionalnosti prijenosnog sloja mrežnog protokola. Prijenosni protokol UDP ostvaruje nepouzdan prijenos nezavisnih paketa, a TCP ostvaruje pouzdan prijenos konekcijskim protokolom. Razvoj komunikacije direktnom upotrebom protokola UDP i TCP zahtijeva znanje o funkcioniranju mreža, sklono je pogreškama, a potrebno je razviti vlastiti sustav sigurnosti, otpornosti na pogreške, serijalizacije i deserijalizacije. Jednom razvijena komunikacijska infrastruktura direktnim korištenjem protokola UDP i TCP omogućava potpunu kontrolu nad komunikacijom, velike brzine prijenosa i smanjuje zavisnosti o drugim bibliotekama.

3.2.2. Poziv udaljene procedure

Poziv udaljene procedure (engl. *Remote Procedure Call – RPC*) omogućava procesima pozivanje i izvođenje procedure na udaljenom računalu. Olakšava razvijanje raspodijeljene programske potpore jer se programski kod za pozivanje udaljene procedure ne razlikuje od poziva obične procedure. Prednosti ove tehnologije su jednostavnost upotrebe, a nedostatak je blokada klijenta za vrijeme čekanja potvrde, ovisnost o vanjskim bibliotekama i slaba međusobna kompatibilnost između različitih programskih ostvarenja. Jedno od najpoznatijih programskih ostvarenja je CORBA.

Objektno orijentirana varijanta poziva udaljene procedure je poziv udaljene metode (engl. *Remote Method Invocation – RMI*). Umjesto procedure, upotrebljava se udaljeni objekt. Svaki klijent ima sučelje udaljenog objekta, a poslužitelj posjeduje konkretni objekt. Razvoj je također jednostavan jer se korištenje udaljenog objekta ne razlikuje od korištenja lokalnog objekta. Najpoznatije programsko ostvarenje poziva udaljene metode je *Java Remote Method Invocation*.

3.2.3. Komunikacija porukama

Sučelje MPI (engl. *Message Passing Interface*) je norma za komuniciranje između procesa slanjem i primanjem poruka. Nastao je 1992. godine; 1994. godine razvijen je MPI 1.1, a 1997. godine razvijen je MPI 2.0. Iako sučelje sadrži preko sto

funkcija, u praksi se najčešće upotrebljava samo njih dvadesetak. Komunikacija može biti: komunikacija između dva procesa, komunikacija između grupe procesa, asinkrona komunikacija, a sadrži i mehanizam za razvoj modularnih paralelnih programa. S obzirom na to da je MPI sučelje, potrebno je koristiti neko njegovo programsko ostvarenje.

Prednosti sučelja MPI je njegov industrijski standard, široka primjena i stabilno sučelje. Nedostaci su teško ispravljanje problema, skupa komunikacija pri intenzivnom korištenju, lošije performanse sitnozrnatih modela paralelizacije i komplicirano upravljanje opterećenjima. Također, ne postoje kvalitetna programska ostvarenja za sve poznatije programske jezike.

3.2.4. Biblioteka OpenCL

Biblioteka OpenCL (engl. *Open Computing Language*) je otvoreni i besplatni standard za pisanje aplikacija. Omogućava razvoj prijenosnog koda koji može biti preveden u strojni kod za različite platforme poput standardnih ili grafičkih procesora, ali i razne druge poput programabilnih logičkih polja (FPGA). Zbog toga sustavi razvijeni preko biblioteke mogu potpuno iskoristiti sve dostupne resurse na pojedinom računalu. Pod standard spada sučelje za kontrolu sustava i programski jezik za pisanje jezgrenih funkcija. Programski jezik temelji se na programskom jeziku C99, modernom dijalektu programskog jezika C, a sadrži i dodatne funkcije za sinkronizaciju, obradu slika, upravljanje dijelovima posla i specijalne matematičke funkcije.

Prednosti biblioteke su podrška najvećih proizvođača procesora, mogućnost razvoja jedinstvenog programa koji će se izvršavati na običnim i grafičkim procesorima, veliki potencijal i trenutačne mogućnosti. Nedostatak biblioteke je još rani stadij razvoja tehnologije.

4. Galapagos – platforma za raspodijeljeno evolucijsko računanje

Glavni cilj ovog rada je razviti programsko okruženje koje omogućava olakšano pisanje i izvođenje raspodijeljenih algoritama evolucijskog računanja. Osnovni zahtjevi programskog okruženja su:

- podrška izrade različitih vrsta evolucijskih algoritama,
- automatska razmjena rješenja između algoritama,
- podrška za različite migracijske parametre,
- automatsko prikupljanje statističkih podataka i
- razvoj u programskom jeziku Java.

Razvijena je platforma Galapagos, programsko okruženje koje ispunjava sve navedene zahtjeve. Predviđena arhitektura na kojoj će se koristiti je MIMD sa zajedničkom memorijom unutar jednog računala, a raspodijeljenom memorijom između računala. Omogućava jednostavno raspodjeljivanje evolucijskog računanja na više računala, gdje jedno računalo predstavlja voditelja, a ostala računala radnike (engl. *master/worker pattern*). Na radnicima treba biti pokrenut jednostavan program za komunikaciju s voditeljem. Nakon što se na voditelju učita problem, definiraju konfiguracije algoritama, parametri migracije i drugi potrebni podaci, voditelj započinje komunikaciju s radnicima. Šalju se svi potrebni podaci, a na svakom radniku inicijaliziraju se algoritmi. Za vrijeme rada na voditelju se mogu vidjeti trenutačni statistički podaci, najbolja rješenja i sl.

Platforma Galapagos omogućava razvoj bilo kakvih evolucijskih algoritama za proizvoljan problem. Jedini zahtjev problema je njegova pripadnost jednokriterijskim optimizacijama. Detaljniji opis kako se razvijaju novi moduli za platformu Galapagos nalazi se u poglavlju 4.1.

Algoritmi razvijeni za platformu mogu se jednostavno paralelizirati preko krupnozrnatog modela paralelizacije ili trivijalnog paralelnog evolucijskog algoritma. Komunikacija između algoritma implementirana je unutar platforme. Način rada automatske razmjene rješenja opisan je u poglavlju 4.2.

S obzirom na to da neki parametri migracije, poput topologije i učestalosti migracije, nisu ovisni o implementaciji algoritama, njihovo konfiguriranje ostvareno je unutar platforme. Zato postoji jedinstven postupak definiranja dijela migracijskih parametara identičan za sve probleme. Ostali parametri migracije poput broja migrirajućih rješenja, odabira rješenja za migraciju i ugrađivanje primljenih rješenja ostvaruju se unutar modula. Na taj način korisniku je omogućena jednostavna i sveobuhvatna kontrola svih parametara migracije. Detaljniji opis definiranja parametara migracije nalazi se u poglavlju 4.3.

Kako bi se mogli jednostavno analizirati i statistički obraditi paralelni evolucijski algoritmi, razvijen je podsustav za prikupljanje statističkih podataka. Svaki algoritam može zabilježiti određene događaje poput pronalaska novog najboljeg rješenja ili trenutačnih statističkih podataka populacije, a platforma će sve događaje slati na računalo voditelja. Prikupljanje statističkih podataka detaljnije je opisano u poglavlju 4.4.

U poglavlju 4.5 opisan je jednostavan primjer programskog ostvarenja modula za platformu Galapagos. Funkcionalnost modula je pronalazak ekstrema jednostavne funkcije.

Platforma Galapagos razvijena je pomoću stabilnih, besplatnih i otvorenih biblioteka i neovisna o arhitekturi i operacijskom sustavu. Korištene biblioteke i ostali tehnički detalji opisani su u poglavlju 4.6.

4.1. Razvoj algoritama za platformu

Definiranje vlastitog modula svodi se na implementiranje sučelja `CoreInterface`. Sučelje sadrži metode za: učitavanje i prikaz opisa problema, učitavanje, pohranu i prikaz jednog rješenja problema te stvaranje algoritma. Metoda za stvaranje algoritma prima konfiguraciju algoritma, opis problema, objekt za prikupljanje statistike (detaljnije opisan u poglavlju 4.4) i zastavicu za prekid rada, a vraća implementaciju sučelja `Algorithm`.

Opis problema predstavlja skup ulaznih podataka koji jednoznačno opisuju problem koji je potrebno riješiti. Primjerice, ako je sustav predviđen za minimizaciju neke matematičke funkcije, podaci problema su dimenzionalnost domene i

kodomene, implementacija funkcije koja se minimizira i intervali unutar kojih se pretražuje najbolje rješenje. Podaci se svode na jedan objekt koji implementira sučelje `Serializable`. Tijekom izvršavanja objekt se ne mijenja.

Konfiguracije algoritama definiraju skup parametara za svaki algoritam. Primjerice, pod parametre spadaju stopa mutacije, veličina populacije, način križanja i sl. Svaki algoritam će prilikom inicijalizacije primiti vlastiti skup parametara koji ga jednoznačno definiraju. Potrebne parametre definira korisnik.

Zastavica za prekid rada signalizira u kojem je trenutku potrebno prekinuti izvršavanje algoritma. U trenutku kada je zastavica postavljena na `true`, tada je potrebno u što kraćem roku prekinuti algoritam.

Sučelje `Algorithm` definira tri metode: metodu za pokretanje algoritma, metodu za dohvaćanje rješenja predviđenih migriranju i metodu za primanje rješenja.

4.2. Automatska razmjena rješenja

Na svakom radniku postoji podsustav za primanje i slanje rješenja. Prilikom njegove inicijalizacije primaju se potrebni parametri migracije: učestalost migracije i odredište. Prema tim parametrima sustav od algoritama dohvaća rješenja za migraciju i prosljeđuje ih odredištu. Ako se odredište nalazi na istom računalu, migracija se odvija unutar računala, a ako je odredište na nekom drugom računalu, šalje se voditelju. Voditelj prosljeđuje primljena rješenja odgovarajućem radniku. Takav model jednostavan je za implementaciju, efikasno radi, ali nije dobar za komunikaciju između vrlo velikog broja računala jer sav promet između radnika prolazi preko voditelja. Problem bi mogao biti riješen tako da oni radnici koji obavljaju migraciju budu izravno spojeni pa komunikacija između njih zaobilazi voditelja.

Komunikacija između radnika i voditelja odvija se preko prijenosnog protokola TCP. Ostali standardi spomenuti u poglavlju 3.2 nisu korišteni zbog slabe podrške za programski jezik Java. Upotrebom protokola TCP postignuta je velika kontrola poslanog i primljenog prometa, brzina i neovisnost o vanjskim bibliotekama. Ipak, razvoj takve komunikacije zahtijevao je dodatan posao poput serijalizacije i deserijalizacije podataka, formiranje poruka i sl. Nakon ostvarenja i intenzivnog testiranja, takav način komunikacije pokazao se vrlo pouzdanim i stabilnim. Testiranjem na agresivnijim i ekstremnijim konfiguracijama, prilikom čega je stvoren vrlo velik mrežni promet, nije dolazilo do usporavanja, zagušenja ili prekida u komunikaciji.

Sva rješenja koja migriraju trebaju implementirati sučelje `Solution`. Sučelje nasljeđuje sučelje `Comparable<Solution>` i sadrži metodu za dohvat vrijednosti funkcije vrednovanja. Ta vrijednost predstavljena je kao polje brojeva s pomičnim zarezom pa omogućava upotrebu višedimenzionalnih rezultata funkcije vrednovanja. Korisnik sam ostvaruje metodu za usporedbu rješenja.

4.3. Migracijski parametri

Na platformi Galapagos migracijska topologija i učestalost migriranja jednostavno se definiraju. Postoji grafičko sučelje gdje je korisnicima vizualizirana topologija. Svaki algoritam prikazan je krugom, migracije su prikazane strelicama, a učestalost migriranja brojem iznad strelice. Korisnik može definirati proizvoljnu topologiju ili može automatski kreirati neku od preddefiniranih topologija. Automatski se mogu izgraditi potpuno povezana, ciklička, otočna, toroidalna, stablasta ili nepovezana topologija. Nepovezana topologija označava skupinu potpuno izoliranih algoritama i time je moguće ostvariti trivijalni paralelni evolucijski algoritam.

U istom grafičkom prikazu potrebno je odrediti raspodjelu algoritma po radnicima. Postoji funkcija automatske dodjele koja nastoji pridijeliti algoritme tako da svaki algoritam ima vlastitu procesorsku jezgru, ali pritom koristeći što je moguće manji broj radnika. Naravno, korisnik može ručno dodijeliti algoritme radnicima.

4.4. Prikupljanje statističkih podataka

Statistički podaci prikupljaju se u obliku skupa događaja. Svaki događaj definiran je vrstom događaja, vremenom, imenom algoritma i ostalim informacijama vezanim za vrstu događaja. U statističkim podacima trenutačno se mogu bilježiti: pronalazak novog najboljeg rješenja unutar algoritma i statistički podaci trenutačne populacije. Moguće je dodati i nove vrste događaja. Vrijeme koje se bilježi u događaju je broj milisekundi od trenutka pokretanja statistike. Informacija koja se bilježi uz događaj pronalaska novog najboljeg rješenja je vrijednost funkcije vrednovanja novog rješenja, a pod statističke podatke populacije spadaju vrijednosti funkcije vrednovanja najboljeg i najlošijeg rješenja, prosječna vrijednost funkcije vrednovanja svih rješenja populacije, standardna devijacija i trenutačna iteracija algoritma.

Statistički podaci prikupljaju se u preddefiniranim intervalima na svakom radniku i prosljeđuju se voditelju. Voditelj sve podatke spaja u veliki skup događaja. Događaji se mogu pratiti na različite načine: tabličnim prikazom najnovijih podataka, dinamičkim grafičkim prikazom napredovanja kroz vrijeme i pohranom podataka u datoteku.

4.5. Primjer programskog ostvarenja modula

U ovom poglavlju bit će opisano jednostavno programsko ostvarenje modula za platformu Galapagos. Razredi definirani u platformi koji se koriste u izgradnji svih modula su: `CoreInterface`, `Solution`, `Algorithm`, `Configuration` i `AlgorithmStatistics`. Svaki modul mora imati razrede koji predstavljaju: opis problema, rješenje i algoritam. Opis problema je nepromjenjiv objekt koji se prosljeđuje svim algoritmima, sadrži sve potrebne podatke problema i mora implementirati sučelje `Serializable`. Svako rješenje problema implementira sučelje `Solution`, a svi algoritmi implementiraju sučelje `Algorithm`. Osim navedenih razreda, svaki modul mora imati jedan razred koji implementira sučelje `CoreInterface` unutar kojeg se definira izgled grafičkog sučelja za učitavanje podataka problema i prikaz rješenja te metodu za kreiranje algoritama.

Genetski algoritam predviđen za pronalazak minimuma ili maksimuma jednodimenzionalne funkcije ostvaren je u razredu `GeneticAlgorithm` i prikazan je u izvornom kodu 4.1. Izvorni kodovi ostalih razreda nisu ovdje navedeni već su samo kratko opisani. Razred `Function` predstavlja problem koji se rješava i sadrži metodu za računanje funkcije, interval unutar kojeg se pronalazi rješenje i informaciju je li potrebno odrediti maksimum ili minimum funkcije. Razred `SolutionImpl` predstavlja jedno rješenje problema. Razred `Operator` sadrži pomoćne metode koje ostvaruju operatore evolucijskog računanja.

Razred `GeneticAlgorithm` u konstruktoru postavlja sve potrebne parametre i pomoćne varijable. Metoda `runAlgorithm()` implementira rad algoritma. U početku se pokreće statistika i priprema početna populacija. U svakoj iteraciji algoritma dohvaća se stanje varijable `stopRequested` čime se provjerava je li potrebno zaustaviti rad algoritma. Također, novoj populaciji dodaju se primljena rješenja iz drugih algoritama. Kada rješenja iz drugih algoritama stignu u ovaj algoritam, rješenja se automatski dodaju u red predstavljen objektom `queue`. Ovaj algoritam ostvaren je tako da uvijek šalje svoje najbolje rješenje drugim algoritmima.

```
package example;

import hr.fer.zemris.distributedAlgorithms.core.*;
import hr.fer.zemris.distributedAlgorithms.core.statistics.*;
import java.io.Serializable;
import java.util.*;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.atomic.*;
import org.apache.commons.configuration.*;
import ec.util.MersenneTwisterFast;

public class GeneticAlgorithm implements Algorithm {
  private final MersenneTwisterFast rand = new MersenneTwisterFast();
  private final AtomicBoolean stopRequested;
  private final AlgorithmStatistics statistics;
  private final Function f;
  private final double mutationProbability;
  private final int popSize;
  private final AtomicReference<SolutionImpl> best;
  private final ConcurrentLinkedQueue<SolutionImpl> queue;

  public GeneticAlgorithm(Configuration config, Serializable problem,
      AlgorithmStatistics statistics, AtomicBoolean stopRequested) {
    this.stopRequested = stopRequested;
    this.statistics = statistics;
    this.f = (Function) problem;
    mutationProbability = config.getDouble("mutationProbability");
    popSize = config.getInt("populationSize");
    best = new AtomicReference<SolutionImpl>();
    queue = new ConcurrentLinkedQueue<SolutionImpl>();
  }

  @Override
  public void runAlgorithm() {
    statistics.startStatistic();
    SolutionImpl[] population = Operator.init(popSize, rand, f, best);
    statistics.addBestSolution(best.get());
    SolutionImpl[] nextPopulation = new SolutionImpl[popSize];
    while (!stopRequested.get()) {
      int i = 0;
      while (!queue.isEmpty()) nextPopulation[i++] = queue.poll();
      for (; i < popSize; i++) {
```

```
        SolutionImpl[] parents = Operator.selection(population, rand);
        SolutionImpl child = Operator.crossover(parents);
        Operator.mutation(child, mutationProbability, rand);
        nextPopulation[i] = child;
        if (child.isBetterThen(best.get())) {
          best.set(child);
          statistics.addBestSolution(child);
        }
      }
      System.arraycopy(nextPopulation, 0, population, 0, popSize);
    }
  }

  @Override
  public List<Solution> getSolutionsToSend() {
    List<Solution> migrationList = new ArrayList<Solution>();
    if (best.get() != null) migrationList.add(best.get());
    return migrationList;
  }

  @Override
  public void receiveForeignSolutions(List<Solution> solutionList) {
    for (Solution solution : solutionList) {
      queue.add((SolutionImpl) solution);
    }
  }
}
```

Izvorni kod 4.1: Sadržaj datoteke `GeneticAlgorithm.java`

4.6. Tehnologije i biblioteke korištene u razvoju

Osnovni zahtjevi pri izradi aplikacije bili su: neovisnost o arhitekturi računala i operacijskom sustavu, izrada u programskom jeziku Java i upotreba komponenti koje su stabilne, besplatne i otvorenog koda.

4.6.1. Biblioteke programskog jezika Java

Komunikacija putem prijenosnih protokola TCP i UDP implementirana je upotrebom standardnih paketa Jave (paketi `java.socket` i `java.io`). Grafičko sučelje

izrađeno je preko standardne Javine biblioteke Swing i tako je ostvareno sučelje identičnog izgleda bez obzira na operacijski sustav.

Aplikacija na računalima radnicima ostvarena je upotrebom višedretvenosti (postoje dretve za komunikaciju, algoritme, prikupljanje statistike i sl.), a komunikacija između njih mora biti pouzdana, stabilna i brza. Standardni način sinkronizacije preko bloka, metode ili varijable `synchronize` je spora operacija. Paket `java.util.concurrent` sadrži više razreda koji su se pokazali jako brzi i stabilni za ostvarivanje zadanih zahtjeva. Jedan od najčešće korištenih razreda su `AtomicBoolean` i `AtomicReference` za brzu razmjenu referenci i zastavica između dretvi bez potrebe za skupim sinkronizacijama. Vrlo korisnim pokazao se i razred `LinkedBlockingQueue` koji implementira sučelje `BlockingQueue`. Taj razred korišten je intenzivno u dretvi zaduženoj za slanje preko pristupne točke TCP-a.

4.6.2. Ostale biblioteke

Osim standardne biblioteke programskog jezika Java, vrlo korisnim pokazale su se i druge biblioteke. Kao što je definirano u zahtjevima, sve biblioteke su besplatne, stabilne i široko korištene.

Skup biblioteka *Apache Commons*

Skup biblioteka *Apache Commons* često su korištene biblioteke s kojima se pokušava što je više moguće ostvariti princip ponovnog korištenja (engl. *reusability*). Sve biblioteke koje uključuje *Apache Commons* imaju minimalnu ovisnost prema drugim bibliotekama i sučelja prema njima vrlo su stabilna. Upravo to je istovremeno i nedostatak nekih biblioteka. Primjerice, neke biblioteke još uvijek nisu uvele podršku za generičke tipove u Javi.

Biblioteka *Apache Commons Configuration* upotrijebljena je za jednostavno konfiguriranje algoritama. Algoritmi zahtijevaju vrlo kompleksno konfiguriranje kako bi u znanstvene i praktične svrhe bilo moguće isprobavati različite kombinacije parametara. Pomoću ove biblioteke takve konfiguracije vrlo se jednostavno koriste, mijenjaju, pohranjuju u datoteke i učitavaju iz datoteka.

Kako je ispis na standardni izlaz nepraktičan (bez mogućnosti filtriranja poruka, jednostavnog preusmjeravanja dijela poruka u datoteke i sl.) bila je potrebna biblioteka za bilježenje poruka. Odabrana je biblioteka *Apache Commons Logging* jer omogućava jednostavno preusmjeravanje u bilo koju drugu, široko

korištenu biblioteku za bilježenje poruka.

Osim navedenih, iz skupa biblioteka *Apache Commons* korištene su i sljedeće biblioteke: *IO*, *Language* i *Collections*.

Biblioteka za bilježenje poruka

Biblioteka *Apache log4j* najpoznatija je biblioteka za bilježenje poruka poruka. Pruža jako puno mogućnosti kontrole prioriteta, mjesto pohrane i formatiranje poruka. Prioriteti poruka definiraju razinu koliko je zabilježeni događaj bitan (informacija, pogreška i sl.). Mjesto pohrane može biti standardni izlaz, datoteka, mrežna pristupna točka, sustav za bilježenje poruka operacijskog sustava i drugi. Format pohranjenih poruka može biti običan tekst, XML, HTML ili neki drugi format.

Biblioteka za crtanje grafova

JFreeChart je biblioteka otvorenog koda za crtanje grafova. Omogućava crtanje velikog broja različitih vrsta grafova, njihovo kombiniranje, prikaz i pohranu u raznim formatima. Automatski skalira sliku, prikazuje sve potrebne podatke i omogućava veliku kontrolu nad izgledom. Biblioteka je upotrijebljena za dinamičko iscrtavanje kvalitete pronađenih rješenja po vremenu.

Generator pseudoslučajnih brojeva

Jedna od najbržih i najkvalitetnijih implementacija generatora pseudoslučajnih brojeva je Mersennov algoritam (Matsumoto i Nishimura, 1998). Ima vrlo velik period (2^{19937}) i prolazi razne testove generatora pseudoslučajnih brojeva. Nedostaci su: osnovna verzija algoritma nije prikladna za kriptografiju, sporija inicijalizacija i kompleksnije programsko ostvarenje. Može se zaključiti kako je algoritam vrlo dobar za potrebe stvaranja slučajnosti u stohastičkim algoritmima U ovoj biblioteci postoje dvije implementacije: višedretveno sigurna, ali sporija te višedretveno nesigurna, ali brža varijanta.

5. Problem izrade rasporeda ispita

Izrada rasporeda ispita jedan je od organizacijskih problema prisutnih na svakom sveučilištu. Spada pod $\mathcal{NP}$-potpune kombinatorne probleme (Garey i Johnson, 1979). Cilj je svaki predmet smjestiti u jedan od prethodno definiranih termina. U svakom terminu mogu biti održani ispiti iz više predmeta, a iz svakog predmeta postoji samo jedan ispit. Kako bi raspored bio što kvalitetniji, potrebno je zadovoljiti što veći broj ograničenja.

Čvrsta ograničenja su kapaciteti termina, prihvatljivosti termina po predmetima i kolizije ispita za studente. Kapacitetom termina određuje se broj studenata koji istovremeno mogu pisati ispit u jednom terminu. Prihvatljivost termina po predmetu uvodi se jer vrlo često postoje ispiti koji imaju neprihvatljive termine. Kolizija ispita za studente zabranjuje rasporede u kojima neki student ima više ispita u isto vrijeme.

Mekim ograničenjima nastoji se povećati kvaliteta rasporeda nakon što su ispunjena sva čvrsta ograničenja. Primjerice, studentima je vrlo nepoželjna pojava pisanje više ispita u istom danu, a malo manje nepoželjna pojava pisanje ispita u susjednim danima. Također, teže predmete nastoji se što je moguće više međusobno odmaknuti. Problem raspoređivanja ispita po dvoranama rješava se zasebno, kao novi optimizacijski problem (Čupić et al., 2010).

Prvi pokušaji rješavanja problema bio je uporabom bojanja grafova (Werra, 1985) gdje su predmeti bili predstavljeni čvorovima, a termini bojama čvorova. Ovakve metode rješavanja problema nisu efikasne za veće probleme raspoređivanja (Al-Betar et al., 2008).

Potreba za automatskom izradom rasporeda ispita na Fakultetu elektrotehnike i računarstva započela je Bolonjskim procesom u nastavi fakulteta. Velik broj izbornih predmeta i potreba za pisanjem svih ispita unutar jednog ili dva tjedna znatno je otežala ručnu izradu rasporeda ispita. Prva verzija sustava koris-

tila je samo genetski algoritam (Čupić et al., 2009), zatim kombinaciju genetskog algoritma i algoritma mravlje kolonije, a od akademske godine 2009./2010. upotrebljava se sustav opisan u ovom radu.

5.1. Opis problema

Ovdje opisan problem specifičan je za Fakultet elektrotehnike i računarstva te postoje razlike od standardnog problema izrade rasporeda ispita koji se obično spominje u sličnim radovima. Najveće razlike su: integracija subjektivne težine predmeta, različito kažnjavanje problema ovisno o modulima i dodatna kažnjavanja određenih parova predmeta.

Subjektivna težina predmeta je brojčana vrijednost dobivena anonimnom anketom među studentima. Krajem svakog semestra primjenjuje se anketa u kojoj svaki student može procijeniti težinu ispita predmeta koje je imao u semestru. Ponuđeno je pet različitih razina težina, a dobiveni rezultati svode se na broj koji je najčešće u intervalu od 0,2 do 2. Što je broj manji, to je ispit subjektivno lakši.

Studenti na fakultetu podijeljeni su u više modula ili profila. Neki moduli imaju puno, a neki drugi malo studenata. Različiti moduli mogu sadržavati iste predmete. Fakultet studentima dozvoljava upis predmeta i iz drugih modula. Raspored ispita nastoji se izraditi tako da studenti imaju najkvalitetnije rasporede za predmete unutar istog modula.

Svaki modul ima nekoliko obaveznih predmeta. Pri izradi rasporeda nastoji se obavezne predmete međusobno udaljiti što je moguće više. Pri udaljavanju obaveznih predmeta modula ne koristi se informacija o broju dijeljenih studenata kako bi i moduli koji imaju manje studenata imali kvalitetnije raspoređene obavezne predmete.

5.1.1. Formalna definicija problema

Ulazni podaci predstavljeni su skupovima i funkcijama koji slijede. Neka je $P = \{p_1, p_2, \ldots, p_{n_P}\}$ skup od n_P predmeta koji održavaju ispite, $T = \{t_1, t_2, \ldots, t_{n_T}\}$ skup od n_T termina, $S = \{s_1, s_2, \ldots, s_{n_S}\}$ skup od n_S studenata, a n_M modula predstavljeno skupom $M = \{m_1, m_2, \ldots, m_{n_m}\}$. Skup studenata koji polažu ispit iz predmeta p_x dobiva se funkcijom *studentiPredmeta*(p_x), funkcija *modulStudenta*(s_x) vraća modul studenta s_x, a skup predmeta koji čine modul m_x vraća funkcija *predmetiModula*(m_x). Skup $S_{par} = \{(p_i, p_j)\}$ definiramo

kao skup svih parova predmeta p_i i p_j za koje vrijedi *studentiPredmeta*$(p_i) \cap$ *studentiPredmeta*$(p_j) \neq \emptyset$ i $\exists m_x$ takav da vrijedi $\{p_i, p_j\} \subseteq$ *predmetiModula*(m_x). Ovime je definiran skup svih parova predmeta modula koji dijele barem jednog studenta. Funkcija *težinaPredmeta*(p_x) vraća brojčanu vrijednost subjektivne težine predmeta. Što je taj broj veći, predmet je subjektivno teži. Funkcija *godinaPredmeta*(p_x) vraća u koju godinu studija spada predmet p_x, funkcija *razmakUSatima*(t_x, t_y) vraća razmak između termina t_x i t_y izražen u satima, a *razmakUDanima*(t_x, t_y) vraća cjelobrojni razmak termina u danima.

Funkcija *razmakIzmeđuIspita*(p_x, t_y) upotrebljava se pri izradi rasporeda ponovljenih ispita. Vraća vremensku razliku između završnog i ponovljenog završnog ispita predmeta p_x za ispit postavljen u termin t_y. Vremenska razlika predstavlja razliku u radnim danima uz pretpostavku da su neradni dani oni dani u kojima nema definiranih termina.

Zadana ograničenja rasporeda definiramo pomoću nekoliko dodatnih funkcija. Funkcija *terminPrihvatljiv*$(p_x, t_y) = \{\top, \bot\}$ vraća $\top$ ako se ispit predmeta p_x može održavati u terminu t_y. Funkcija *tvrdiKapacitet*(t_x) vraća kapacitet termina ako se u njemu održava samo jedan ispit, a *mekiKapacitet*(t_x) vraća kapacitet termina ako se u njemu održava više ispita.

Svaki raspored međuispita može se predstaviti kao skup uređenih parova $R = \{(p_x, t_y)\}$ koji sadrži točno jedan uređeni par za svaki $p_x \in P$. Definiramo pomoćnu funkciju *termin*(p_x) koja vraća termin pridružen predmetu p_x i funkciju *predmetiUTerminu*(t_x) koja vraća skup predmeta smještenih u termin t_x.

5.1.2. Funkcija vrednovanja rasporeda međuispita

Funkcija *kaznaPredmeta*(p_i, p_j) definira kaznu između dva predmeta ako je vremenski razmak između održavanja njihovih ispita točno jedan dan. Funkcija je implementirana kao što je opisanu u algoritmu 6.

U algoritmu se može vidjeti na koji način je realizirano veće kažnjavanje predmeta unutar modula od predmeta izvan modula. Predmeti onih studenata koji slušaju predmete sa svojeg modula najviše se kažnjavaju, malo manje se kažnjavaju predmeti koji nisu iz istog modula, ali su s istih godina, a najmanje se kažnjavaju predmeti različitih godina. Dobivena kazna proporcionalna je broju studenata koju dijele predmeti i težinom lakšeg predmeta. Na taj način će dva teška predmeta s velikim brojem studenata imati visoku kaznu. Predmeti koji ne

Algoritam 6 Računanje kazne između predmeta

$kazna \leftarrow 0$
za svaki $student \in studentiPredmeta(p_i) \cap studentiPredmeta(p_j)$
  $modulStudenta \leftarrow modulStudenta(student)$
  ako $\{p_i, p_j\} \subseteq predmetiModula(modulStudenta)$ **onda**
    $kazna \leftarrow kazna + 1$
  inače ako $\{p_i, p_j\} \cap predmetiModula(modulStudenta) \neq \emptyset$ **onda**
    ako $godinaPredmeta(p_i) = godinaPredmeta(p_i)$ **onda**
      $kazna \leftarrow kazna + 0.5$
    inače
      $kazna \leftarrow kazna + 0.25$
    kraj
  inače
    ako $godinaPredmeta(p_i) = godinaPredmeta(p_i)$ **onda**
      $kazna \leftarrow kazna + 0.25$
    inače
      $kazna \leftarrow kazna + 0.125$
    kraj
  kraj
kraj
$kazna \leftarrow kazna \cdot \min(težinaPredmeta(p_i), težinaPredmeta(p_j))$

dijele ni jednog studenta nemaju kaznu.

Funkcija $kaznaTermina(t_i, t_j)$ definira kaznu između dva termina u kojima su smještena dva predmeta. Ispit jednog predmeta postavljen je u termin t_i, a drugog predmeta u t_j. U implementaciji je korištena sljedeća funkcija:

$$kaznaTermina(t_i, t_j) = \alpha^{1 - \frac{razmakUSatima(t_i, t_j)}{24}} \tag{5.1}$$

gdje je za α odabrana vrijednost $\alpha = 7$. Rezultat funkcije je velika kazna za termine koji su međusobno vrlo blizu, a puno manja kazna ako su termini udaljeniji. Primjerice, za termine koji su međusobno udaljeni tri sata vrijednost funkcije *kaznaTermina* je oko 5,5, a za termine udaljene dva dana je oko 0,15.

Funkcija $kaznaTerminaParova(t_i, t_j)$ definira kaznu između dva termina za parove predmete modula iz skupa S_{par}. I za ovu funkciju vrijedi da je vrijednost funkcije veća što su termini međusobno bliži. Korištena je sljedeća funkcija:

$$kaznaTerminaParova(t_i, t_j) = \begin{cases} 200, & \Delta d = 0 \\ 100, & \Delta d = 1 \\ \dfrac{60}{1 + \exp\left((0.2 \cdot \Delta t)^3\right)}, & \Delta d \geq 2 \end{cases} \tag{5.2}$$

gdje je

$$\Delta d = razmakUDanima(t_i, t_j). \tag{5.3}$$

Na taj način postignuta je posebno velika kazna za međusobno blizu postavljene parove predmeta. Kada su parovi predmeta postavljeni u isti dan, kazna je 200, a za susjedne dane kazna je 100. Može se primijetiti kako u formuli 5.2 nije upotrijebljena informacija o broju studenata koji pišu ispite iz oba predmeta. Zbog toga će kazna za parove predmeta modula s različitim brojem studenata biti jednaka.

Postupak računanja kazne rasporeda međuispita opisan je u algoritmu 7. Kao što je prikazano u poglavlju 5.2.1, za izradu rasporeda međuispita potrebno je izračunati samo kaznu za meka ograničenja. Kazna se računa za sve parove svih predmeta i to tako da se kazne između predmeta množe s kaznama između termina u koje su predmeti postavljeni. Nakon toga dodaju kazne za sve parove predmeta modula.

5.1.3. Funkcija vrednovanja rasporeda ponovljenih ispita

Funkcija vrednovanja za rasporede ponovljenih ispita upotrebljava sve funkcije kao i funkcija vrednovanja rasporeda međuispita, ali uvodi još jednu funkciju.

Algoritam 7 Računanje kazne rasporeda međuispita

$kazna \leftarrow 0$
za svaki $\{p_1, p_2\} \subseteq P$
  $kaznaPredmeta \leftarrow kaznaPredmeta(p_1, p_2)$
  $kaznaTermina \leftarrow kaznaTermina(termin(p_1), termin(p_1))$
  $kazna \leftarrow kazna + kaznaPredmeta \cdot kaznaTermina$
kraj
za svaki $(p_1, p_2) \in S_{par}$
  $kazna \leftarrow kazna + kaznaIzmeđuTerminaParova(termin(p_1), termin(p_1))$
kraj

Funkcija $kaznaRazmaka(p_x, t_y)$ upotrebljava se za izradu rasporeda ponovljenih ispita gdje je p_x neki predmet, a t_y termin u koji je postavljen ispit predmeta p_x. Funkcija glasi:

$$kaznaRazmaka(p_x, t_y) = \begin{cases} \beta(\alpha - \Delta i)^2, & \Delta i < \alpha \\ \sqrt[4]{brStudenata} \cdot \Delta i \cdot (1 - e^{\alpha - \Delta i}), & \Delta i \geq \alpha \end{cases} \tag{5.4}$$

gdje je *brStudenata* broj studenata predmeta p_x, α parametar kojim se definira željena granica razmaka između završnog i ponovljenog završnog ispita (za manje razmake kazna se značajno povećava), β parametar kojim se određuje intenzitet kazne ako je razmak manji od α i *brStudenata* koji predstavlja razmak između ispita:

$$\Delta i = razmakIzmeđuIspita(p_x, t_y). \tag{5.5}$$

U implementaciji su upotrijebljene vrijednosti $\alpha = 5$ i $\beta = 50.000$. Ovakvim parametrima nastoji se izraditi raspored gdje će svi predmeti imati razmak između završnog i ponovljenog završnog ispita barem pet radnih dana.

Računanje kazne rasporeda ponovljenih ispita prikazan je u algoritmu 8 i razlikuje se od računanja kazne rasporeda ispita jer računa i kaznu čvrstih ograničenja i kaznu mekih ograničenja. Kazna mekih ograničenja računa se kao i za raspored međuispita, ali uvećava se za kazne ovisne o razmaku između završnog i ponovljenog ispita pojedinog predmeta. Kazne čvrstih ograničenja računaju se prema prekoračenju kapaciteta termina i broju pojava gdje student ima više ispita u isto vrijeme. Kazna čvrstih ograničenja ne ovisi o prihvatljivosti termina za predmete jer je to ograničenje ugrađeno u operatore evolucijskog računanja, kao je prikazano u poglavlju 5.2.2.

Algoritam 8 Računanje kazne rasporeda ponovljenih ispita

$čvrstaKazna \leftarrow 0$
$mekaKazna \leftarrow 0$
za svaki $termin \in T$
  $brojStudenata \leftarrow 0$
  za svaki $predmet \in predmetiUTerminu(termin)$
    $brojStudenata \leftarrow brojStudenata + |studentiPredmeta(predmet)|$
  kraj
  ako $|predmetiUTerminu(termin)| = 1$ **onda**
    $kapacitet \leftarrow tvrdiKapacitet(termin)$
  inače
    $kapacitet \leftarrow mekiKapacitet(termin)$
  kraj
  ako $kapacitet < brojStudenata$ **onda**
    $čvrstaKazna \leftarrow čvrstaKazna + (brojStudenata - kapacitet)$
  kraj
  za svaki $\{p_1, p_2\} \subseteq predmetiUTerminu(termin)$
    $dijeljeniStudenti \leftarrow studentiPredmeta(p_1) \cap studentiPredmeta(p_2)$
    $čvrstaKazna \leftarrow čvrstaKazna + |dijeljeniStudenti|$
  kraj
kraj
za svaki $\{p_1, p_2\} \subseteq P$
  $kaznaPredmeta \leftarrow kaznaPredmeta(p_1, p_2)$
  $kaznaTermina \leftarrow kaznaTermina(termin(p_1), termin(p_1))$
  $mekaKazna \leftarrow mekaKazna + kaznaPredmeta \cdot kaznaTermina$
kraj
za svaki $(p_1, p_2) \in S_{par}$
  $mekaKazna \leftarrow mekaKazna + kaznaTerminaParova(termin(p_1), termin(p_1))$
kraj
za svaki $predmet \in P$
  $mekaKazna \leftarrow mekaKazna + kaznaRazmaka(predmet, termin(predmet))$
kraj

5.2. Moduli za platformu Galapagos

U ovom radu razvijena su dva različita modula za platformu Galapagos: jedan je prilagođen izradi rasporeda međuispita i završnih ispita, a drugi izradi ponovljenih završnih ispita.

Iako je problematika vrlo slična, način rada ta dva modula ima puno razlika. Osnovna razlika je što kod izrade međuispita u bilo kojem trenutku sva rješenja unutar populacija imaju ispunjena sva čvrsta ograničenja, a kod izrade ponovljenih završnih ispita nemaju. Kod ponovljenih završnih ispita dozvoljavaju se prekoračenja čvrstih ograničenja jer bi izrada početne populacije trajala iznimno dugo ili je uopće ne bi bilo moguće izgraditi. Druga velika razlika je drugačije računanje funkcije vrednovanja (kao što je opisano u poglavlju 5.1.2).

Sličnosti su implementacije algoritama jer su svi operatori, model i ostali pomoćni razredi izrađeni kao implementacije sučelja. S obzirom na to da su sučelja vrlo slična ili identična, implementacije algoritama su vrlo slične. Algoritmi su implementirani kao što je opisano u poglavlju 2.3.

Kako bi algoritmi radili što brže, korištene su razne optimizacije. Izbjegavala se inicijalizacija novih objekata po iteracijama, upotrijebljena je brža implementacija generatora nasumičnih brojeva, upotrijebljeni algoritmi i strukture podataka imali su što je moguće manju algoritamsku složenost, a sve vrijednosti koje se često upotrebljavaju su unaprijed izračunate i pohranjene. S obzirom na stohastičku prirodu algoritma $\mathcal{MM}$AS i intenzivnu upotrebu matematičke funkcije potenciranja, upotrijebljena je njena aproksimacija[1]. Upotreba aproksimacije znatno je ubrzala algoritam, a algoritam nije pokazao nikakva lošija svojstva.

5.2.1. Modul za izradu rasporeda međuispita

Izrada rasporeda međuispita i završnih ispita upotrebljava generacijski i eliminacijski genetski algoritam, jednostavni imunološki algoritam, algoritam $\mathcal{MM}$AS i algoritam harmonijskog pretraživanja. Ima različite implementacije operatora inicijalizacije populacije, odabira, mutacije, križanja i lokalnog pretraživanja. Izrađeni su tako da ulazna i izlazna rješenja uvijek imaju ispunjena sva čvrsta ograničenja. Zbog toga je inicijalizacija početne populacije sporija, ne radi dobro

[1] Aproksimacija ubrzava računanje potencije za 20 do 40 puta uz pogrešku od oko 5%. Više detalja može se pronaći na mrežnoj stranici `http://martin.ankerl.com/2007/10/04/optimized-pow-approximation-for-java-and-c-c/`.

za velike popunjenosti termina, operatori su kompliciraniji, ali su krajnji rezultati kvalitetniji.

Najčešće korištena inicijalizacija populacije redom postavlja ispite u nasumično odabrane termine počevši od ispita koji imaju najviše zavisnosti prema drugim ispitima. Kada neki ispit nije moguće postaviti bez prekoračenja čvrstih ograničenja, tada se prethodno smješten ispit postavlja u neki drugi termin. Postupak se ponavlja do trenutka kada su svi ispiti smješteni u termine, a niti jedno čvrsto ograničenje nije prekoračeno. Ako ispit nije kreiran u roku od jedne sekunde, postupak se prekida i raspored se počinje inicijalizirati ispočetka.

Operator mutacije koji se obično koristi je mutacija koja pomoću binomne razdiobe određuje broj ispita za izmjenu. Zatim se nasumično odabire taj broj ispita i postavlja se u slučajno odabran termin, ali tako da se pritom ne prekorače čvrsta ograničenja.

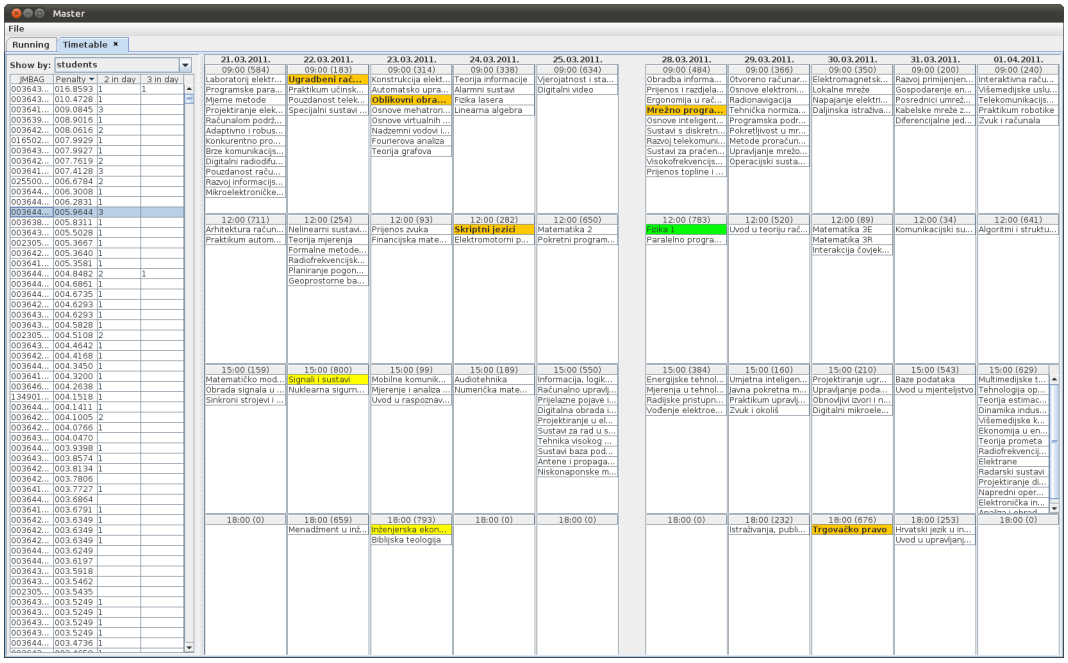
Operator križanja implementiran je tako da se u dijete kopira prvi roditelj. Zatim se za oko 50% predmeta nastoji promijeniti termin tako da bude kao u drugom roditelju. Ako bi se pri promjeni prekoračila čvrsta ograničenja, promjena se ne izvodi.

Zadnji operator koji treba opisati je operator lokalne pretrage. Postoje kvalitetne i sporije lokalne pretrage te manje kvalitetne i brže. Primjer kvalitetnih i sporih lokalnih pretraga inspiriran je pretraživanjem usponom na vrh. Za svaki ispit pokušava se pronaći termin kojim bi se povećala kvaliteta rasporeda. Postupak se ponavlja sve dok postoji promjena bilo kojeg ispita u bilo koji termin, a da pritom kvaliteta rasporeda bude veća.

Unutar modula ostvareno je grafičko sučelje za analizu generiranih rasporeda. Omogućava pregled rasporeda po studentima, parovima predmeta, modulima i kombinacijama upisanih predmeta. Rasporedi po studentima mogu se gledati po kazni koju pridonosi pojedini student i broju ispita u jednom danu. Na slici 5.1 prikazano je razvijeno sučelje. Različite boje predmeta označavaju pripadnost različim godinama studija. Deblja imena predmeta predstavljaju predmete studentovog modula.

5.2.2. Modul za izradu rasporeda ponovljenih završnih ispita

Izrada rasporeda završnih ispita koristi generacijski i eliminacijski genetski algoritam, jednostavni imunološki algoritam i algoritam $\mathcal{MM}$AS. Operatori su puno

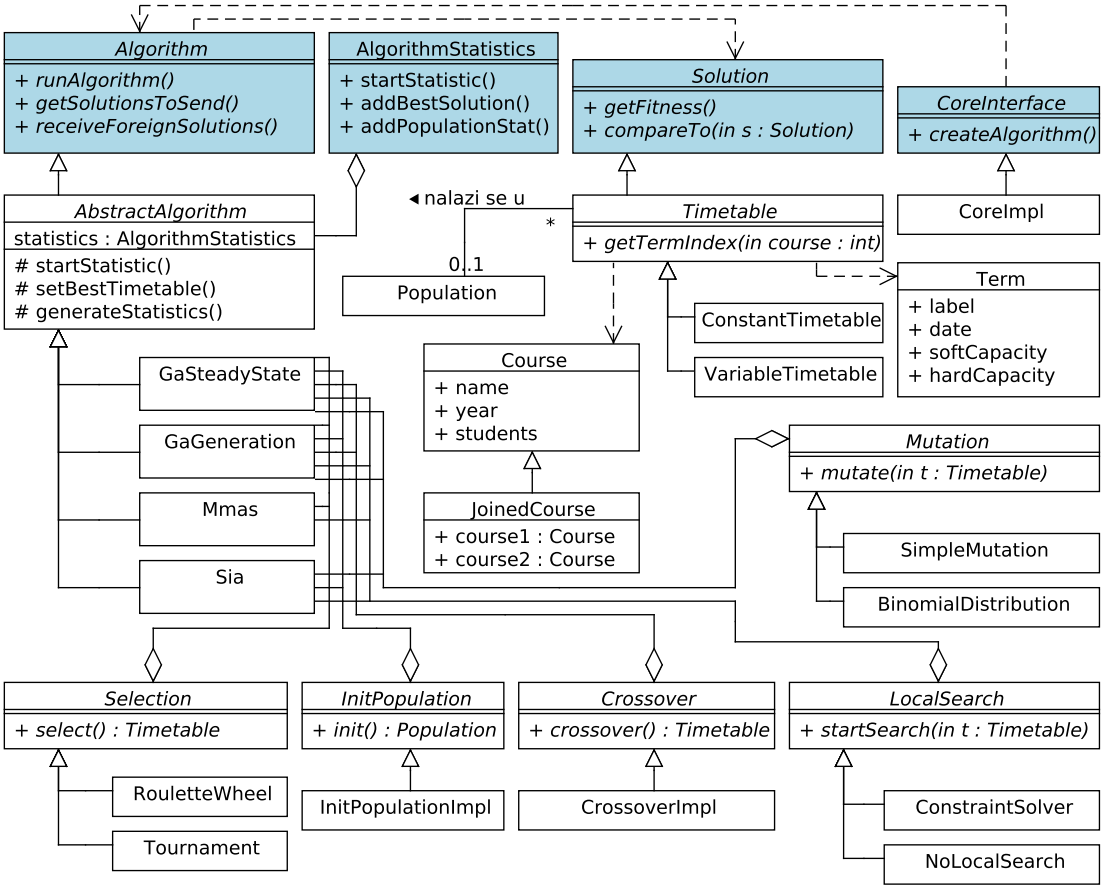


Slika 5.1: Prikaz analiziranja generiranog rasporeda

jednostavniji i brži jer omogućuju prekoračenja svih čvrstih ograničenja osim neprihvatljivih termina za predmete.

Kvaliteta algoritama značajno je poboljšana lokalnim pretraživanjem koje nastoji smanjiti prekoračenja čvrstih ograničenja. Lokalno pretraživanje izvedeno je tako da nastoji mijenjati samo one predmete koji uzrokuju prekoračenje čvrstih ograničenja u samo one termine koji bi mogli poboljšati kvalitetu rasporeda. Upotrebom opisanog lokalnog pretraživanja ispravljanje prekoračenja čvrstih ograničenja znatno je ubrzano.

Na slici 5.2 prikazan je UML dijagram razreda za osnovne razrede korištene pri razvoju modula za izradu ponovljenih završnih ispita. Zbog sažetosti prikaza, navedene su samo osnovne metode i neki njihovi argumenti. Narančastom bojom označena su sučelja i razredi definirani unutar platforme. Razred `AbstractAlgorithm` implementira sve funkcionalnosti koje su potrebne za sve algoritme, uključujući i kreiranje statističkih događaja. Apstraktni razred `Timetable` ima dvije implementacije: `ConstantTimetable` i `VariableTimetable`. Razred `ConstantTimetable` predstavlja raspored koji je nepromjenjiv, višedretveno siguran, troši malo memorije i koristi se u populacijama i komunikaciji između algoritama. Razred `VariableTimetable` je raspored koji sadrži dodatne metode za mijenjanje termina predmeta, metode za računanje kvalitete rasporeda, troši više memorije, nije višedretveno siguran i predviđen je za ponovno



Slika 5.2: UML dijagram razreda modula za izradu ponovljenih završnih ispita

korištenje u svakoj iteraciji algoritma. Primjerice, u njega se pohranjuje rezultat mutacije, lokalnog pretraživanja i križanja dva roditelja.

6. Vrednovanje platforme Galapagos

U ovom poglavlju prikazani su rezultati vrednovanja platforme Galapagos modulima za izradu rasporeda ispita i završnih ispita. Upotrijebljeni su podaci akademske godine 2010./2011. za ispite na Fakultetu elektrotehnike i računarstva.

6.1. Opis testiranih primjera

U akademskoj godini 2010./2011. održana su dva semestra: zimski i ljetni. Svaki semestar podijeljen je u tri ciklusa, a krajem svakog ciklusa održavaju se ispitni tjedni. Prva dva ispitna tjedna su međuispiti, treći ispitni tjedan su završni ispiti, a nakon završnih ispita slijede ponovljeni završni ispiti. Ispitni tjedni obično traju oko dva tjedna i tada se pišu svi ispiti na preddiplomskom i diplomskom studiju. Ponovljeni završni ispiti održavaju se najčešće samo pet radnih dana. Ispiti se pišu svakog radnog dana unutar ispitnih tjedana u 9, 12, 15 ili 18 sati. Termin u 18 sati rezerviran je isključivo za društvene predmete. Društvenih predmeta ima malo i obično su subjektivno lakši od ostalih predmeta.

Tablica 6.1: Podaci o zimskom i ljetnom semestru

Broj	Zimski semestar	Ljetni semestar
Studenata	3.455	3.003
Predmeta	135	140
Kombinacija	1.249	1.504
Jedinstvenih kombinacija	915	1.197
Parova predmeta modula	117	159
Polaganja	17.241	14.086

U tablici 6.1 prikazani su podaci po semestrima. Može se primijetiti vrlo velik broj jedinstvenih kombinacija upisanih predmeta jer je studentima dopušteno upisivanje predmeta izvan vlastitog modula.

Tablica 6.2: Statistika po ispitima

Semestar	Ispit	Termina	Kapacitet	Popunjenost
Zimski	1. međuispit	40	800/1.000	53,88%
	2. međuispit	40	800/1.000	53,88%
	Završni	40	800/1.000	53,88%
	Ponovljeni	20	1.400/2.000	61,58%
Ljetni	1. međuispit	40	800/1.000	44,02%
	2. međuispit	40	800/1.000	44,02%
	Završni	38	700/1.000	52,96%
	Ponovljeni	20	1.400/2.000	50,31%

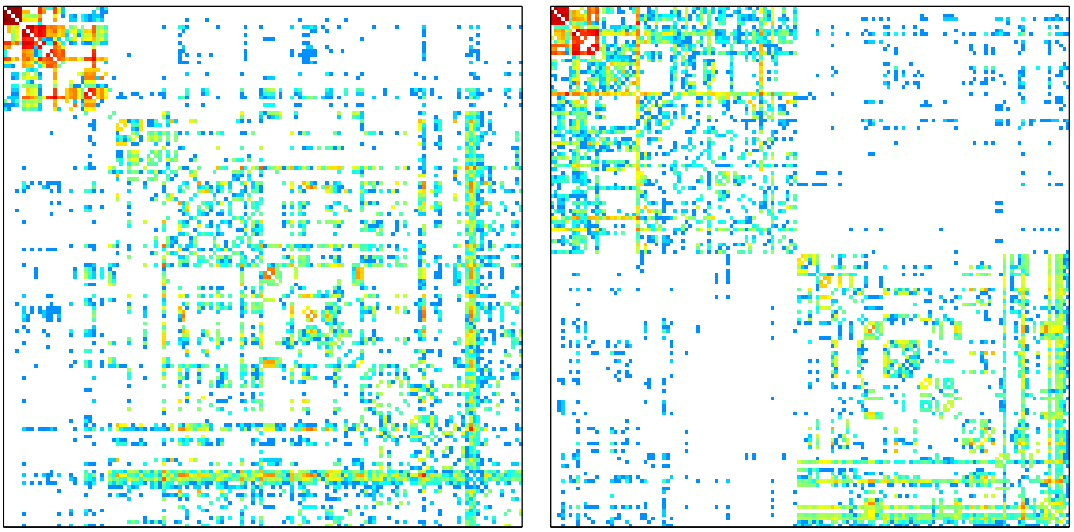
U tablici 6.2 prikazan je broj termina, ukupan kapacitet i popunjenost po ispitnim tjednima. U zimskom semestru radni dani u 1. i 2. međuispitima, kao i u završnim ispitima, jednako su raspoređeni pa je bilo potrebno izraditi samo jedan raspored. Taj raspored mogao je biti primijenjen na sve navedene tjedne ispita. U ljetnom semestru završni ispiti su imali različito raspoređene radne dane od 1. i 2. međuispita pa je bilo potrebno izraditi drugačije rasporede. U tablici 6.3 problemi su imenovani za potrebe eksperimentiranja.

Tablica 6.3: Imenovanje problema

Ime problema	Opis problema
Problem 1	Međuispiti i završni ispiti zimskog semestra ak. god. 2010./2011.
Problem 2	Međuispiti ljetnog semestra ak. god. 2010./2011.
Problem 3	Završni ispiti ljetnog semestra ak. god. 2010./2011.
Problem 4	Ponovljeni završni ispiti zimskog semestra ak. god. 2010./2011.
Problem 5	Ponovljeni završni ispiti ljetnog semestra ak. god. 2010./2011.

Na slici 6.1 grafički je prikazan broj dijeljenih studenata između svih predmeta. Svaki red i stupac predstavljaju po jedan predmet. Predmeti su sortirani prema godini studija (diplomski studij smatra se četvrtom i petom godinom studija). Sjecište retka i stupca bojom vizualizira broj dijeljenih studenata. Što je

boja crvenija, to predmeti imaju više dijeljenih studenata, a plavom bojom označeni su parovi s manjim brojem dijeljenih studenata. Bijelom bojom prikazani su parovi predmeta koji nemaju dijeljenih studenata. Može se primijetiti koji predmeti spadaju u preddiplomski, a koji u diplomski studij. Vidi se kako postoje parovi predmeta koji dijele studente preddiplomskog i diplomskog studija. Uzrok tome je postojanje manjeg broja studenata koji slušaju predmete preddiplomskog i diplomskog studija.



(a) Zimski semestar **(b)** Ljetni semestar

Slika 6.1: Broj dijeljenih studenata između predmeta ak. god. 2010./2011.

6.2. Prethodna istraživanja

U prethodnim istraživanjima testirane su različite topologije s različitim učestalostima migracije na podacima zimskog semestra (Komar et al., 2011). Zaključeno je kako je upotrebom algoritama evolucijskog računanja moguće proizvesti kvalitetne rasporede. Raspodjeljivanjem algoritama rezultati postaju još kvalitetniji. Ipak, nepovezana topologija (trivijalni paralelni evolucijski algoritam) proizvela je kvalitetnije rasporede od krupnozrnatog modela paralelizacije.

6.3. Rezultati

Eksperimenti su izvršeni na jednoj od dvije skupine računala. Jedna skupina ima 20 računala, a druga skupina 16 računala. Sva računala imala su identično sklopovlje; ugrađeni procesor bio je Intel Core2 Quad CPU Q8200 s 4 GiB radne memorije. Operacijski sustav u prvoj skupini bio je Microsoft Windows XP SP3, a u drugoj skupini Microsoft Windows 7. Korištena verzija Jave u obje skupine bila je Java SE Runtime Environment 1.6.0. Računala su bila međusobno povezana u lokalnoj mreži brzine 100 Mbit/s. Svaki eksperiment ponovljen je 30 puta.

6.3.1. Izrada rasporeda međuispita

U ovom poglavlju obrađuju se eksperimenti kojima se testiraju utjecaji parametara migracija (topologija i učestalost migriranja) i različitih problema na kvalitetu rezultata. Upotrijebljena su tri generacijska genetska algoritma (veličina populacije je 10, a stopa mutacije 0,2%), tri eliminacijska genetska algoritma (veličina populacije je 50, a stopa mutacije 0,2%) i po dva algoritma $\mathcal{MM}$AS ($\rho = 0,02$, $\alpha = 1$, $a = 100$ i 50 prolazaka mrava), jednostavnog imunološkog sustava (broj klonova je 10, veličina populacije 100, a stopa mutacije 2%) i harmonijskog pretraživanja (veličina populacije 10, $r_{prihvati} = 99\%$, a $r_{ra} = 5\%$). Svi algoritmi osim harmonijskog pretraživanja imali su aktiviranu lokalnu pretragu.

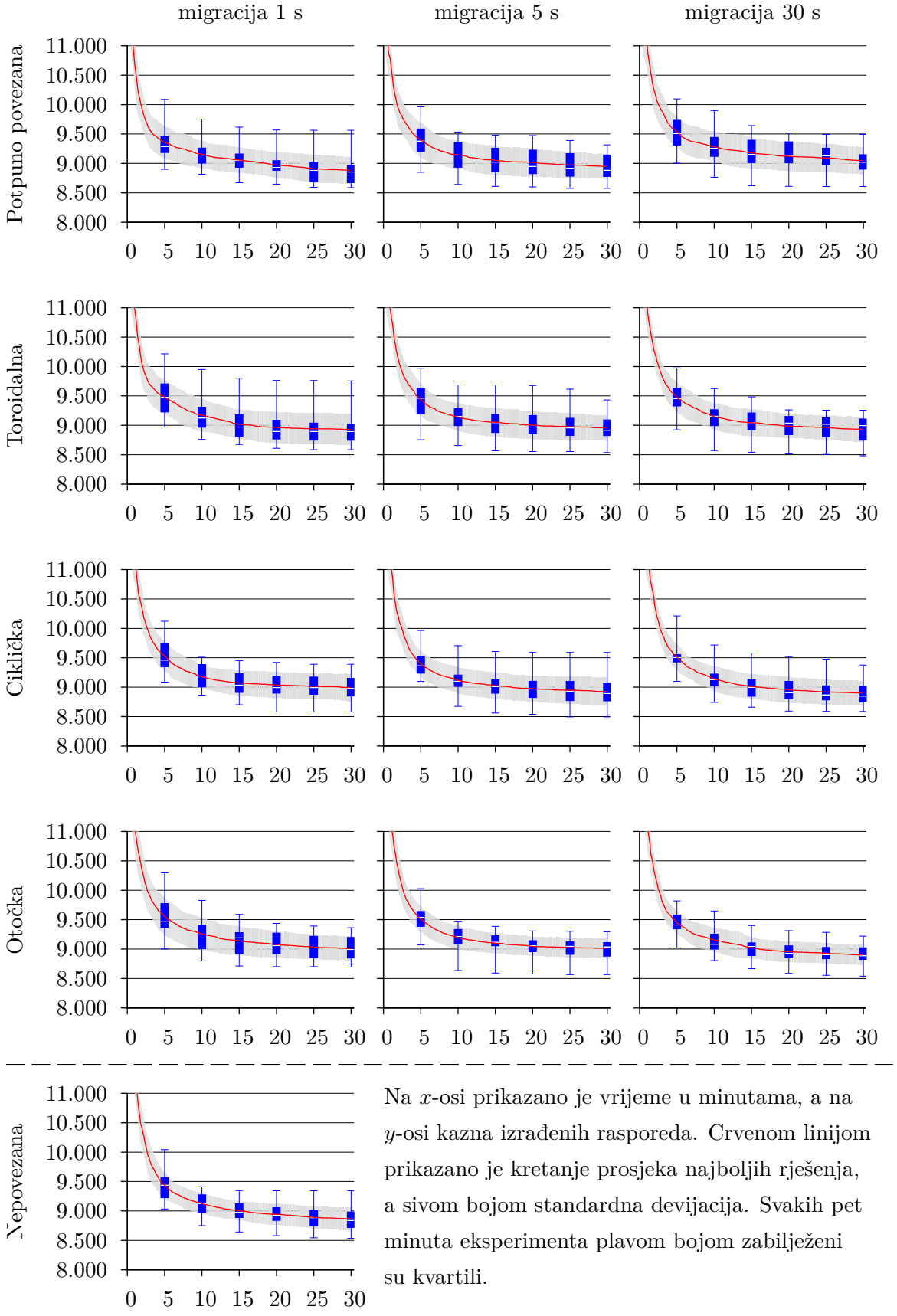
Upotrijebljeni migracijski parametri sastojali su se od različitih topologija i učestalosti razmjene rješenja. Korištene topologije su potpuna, toroidalna, ciklička, otočna i nepovezana topologija, a isprobane su učestalosti razmjene rješenja svakih 1 s, 5 s i 30 s. Sve topologije imale su po 12 algoritama. Toroidalna topologija imala je četiri stupca i tri retka algoritama. Otočna topologija je unutar tri otoka imala po četiri algoritma. Komunikacija između otoka odvijala se pet puta sporije od komunikacije unutar otoka. U svim topologijama, osim u nepovezanoj topologiji, komunikacija između povezanih algoritama odvijala se obostrano.

Problem 1

Tablica 6.4 i slika 6.2 prikazuju rezultate za međuispite i završne ispite zimskog semestra (problem 1). Najbolje rezultate pokazala je nepovezana topologija jer je imala najbolji srednji rezultat, a vrijednost medijana je druga najbolja nakon cikličke topologije s migracijom svakih 30 sekundi. Ipak, rezultati nisu značajno bolji od ostalih kombinacija parametara. Zbog toga se ne može zaključiti koja kombinacija parametara daje posebno dobre rezultate.

Tablica 6.4: Rezultati za problem 1

Topologija	Migracija	Srednji rezultat	Medijan	Najmanji	Najveći
Potpuna	1 s	8.880,3 ± 227,8	8.881,4	8.588,7	9.562,6
	5 s	8.946,3 ± 209,2	8.950,4	8.577,1	9.314,9
	30 s	9.046,0 ± 230,5	9.047,3	8.607,2	9.492,8
Toroidalna	1 s	8.929,3 ± 263,1	8.894,5	8.585,8	9.755,0
	5 s	8.957,6 ± 212,0	8.964,2	8.536,6	9.431,6
	30 s	8.933,8 ± 212,7	9.018,2	**8.480,7**	9.255,2
Ciklička	1 s	8.994,6 ± 228,6	8.988,0	8.577,9	9.390,2
	5 s	8.925,7 ± 237,1	8.906,3	8.494,5	9.590,4
	30 s	8.901,9 ± 207,7	**8.856,7**	8.586,8	9.375,9
Otočna	1 s	9.008,8 ± 202,4	9.000,1	8.695,6	9.364,5
	5 s	9.013,6 ± **162,3**	9.042,8	8.566,0	9.294,6
	30 s	8.900,1 ± 169,5	8.885,1	8.539,7	**9.219,6**
Nepovezana		**8.866,7** ± 191,1	8.859,1	8.534,4	9.343,1



Na x-osi prikazano je vrijeme u minutama, a na y-osi kazna izrađenih rasporeda. Crvenom linijom prikazano je kretanje prosjeka najboljih rješenja, a sivom bojom standardna devijacija. Svakih pet minuta eksperimenta plavom bojom zabilježeni su kvartili.

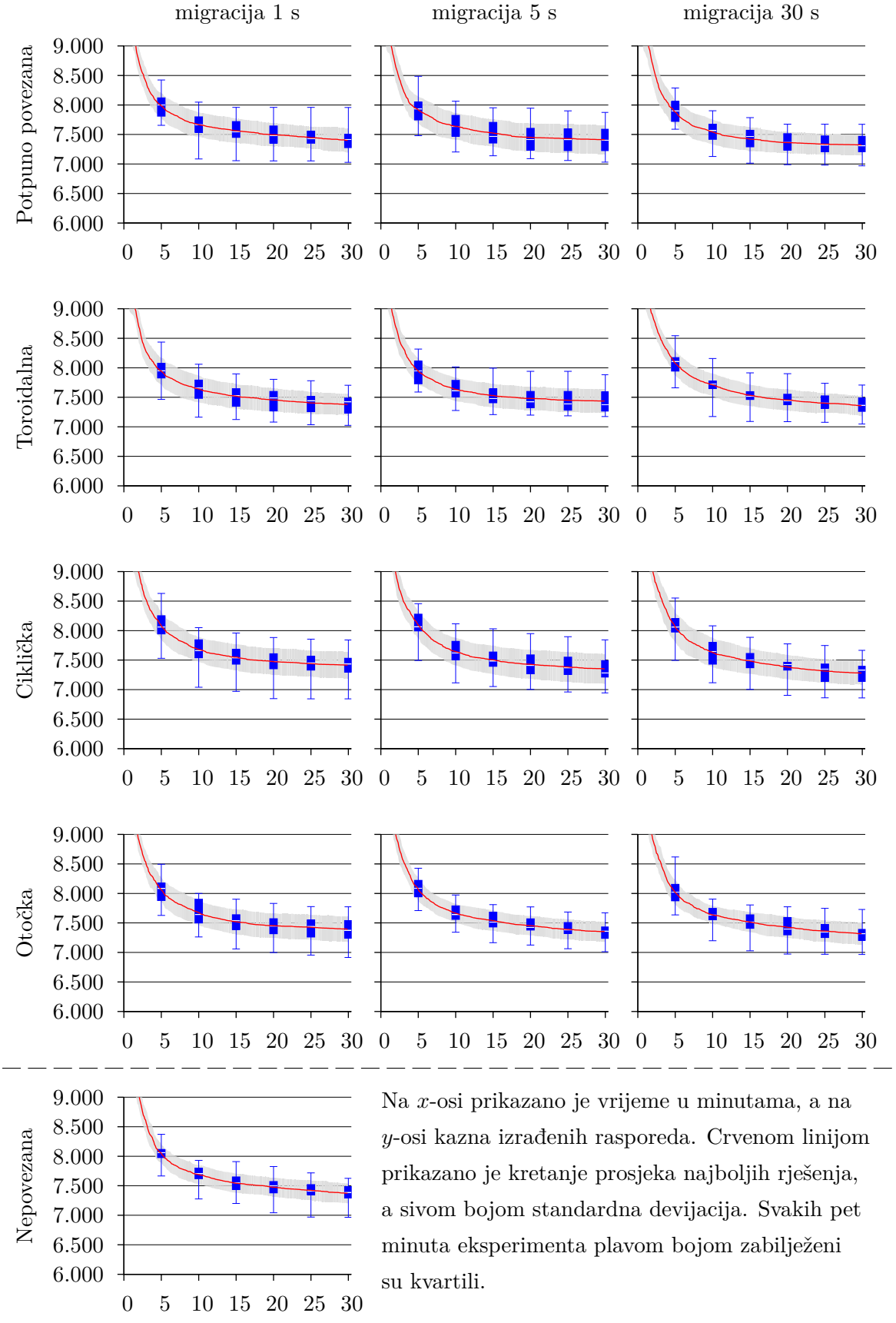
Slika 6.2: Rezultati za problem 1

Problem 2

Tablica 6.5 i slika 6.3 prikazuju rezultate za međuispite ljetnog semestra (problem 2). Najbolji rezultat postigla je ciklička topologija s migracijom svakih 30 sekundi. U tri od pet kriterija najbolje ponašanje pokazala je ciklička topologija, ali uz nešto veću standardnu devijaciju krajnjih rješenja. Primjećuje se kako se kvaliteta rezultata povećava kako se smanjuje učestalost migracije. I dalje se ne primjećuje koja kombinacija parametara daje posebno kvalitetne rezultate.

Tablica 6.5: Rezultati za problem 2

Topologija	Migracija	Srednji rezultat	Medijan	Najmanji	Najveći
Potpuna	1 s	7.400,4 ± 199,8	7.422,8	7.029,2	7.957,5
	5 s	7.411,7 ± 244,8	7.405,8	7.031,7	7.873,6
	30 s	7.323,1 ± 171,5	7.311,7	6.968,7	7.673,3
Toroidalna	1 s	7.376,1 ± 182,5	7.436,5	7.024,5	7.704,6
	5 s	7.436,3 ± 196,6	7.385,7	7.174,4	7.883,3
	30 s	7.356,0 ± 169,9	7.374,7	7.047,0	7.706,9
Ciklička	1 s	7.415,7 ± 224,3	7.443,0	**6.844,3**	7.841,4
	5 s	7.351,5 ± 247,1	**7.285,8**	6.945,5	7.842,8
	30 s	**7.281,2** ± 193,4	7.335,8	6.860,1	7.665,5
Otočna	1 s	7.394,3 ± 209,1	7.431,2	6.915,4	7.773,6
	5 s	7.354,0 ± 168,2	7.364,1	7.013,0	7.670,7
	30 s	7.319,7 ± 185,9	7.318,0	6.965,3	7.729,0
Nepovezana		7.372,7 ± **162,7**	7.393,1	6.966,4	**7.626,3**



Na x-osi prikazano je vrijeme u minutama, a na y-osi kazna izrađenih rasporeda. Crvenom linijom prikazano je kretanje prosjeka najboljih rješenja, a sivom bojom standardna devijacija. Svakih pet minuta eksperimenta plavom bojom zabilježeni su kvartili.

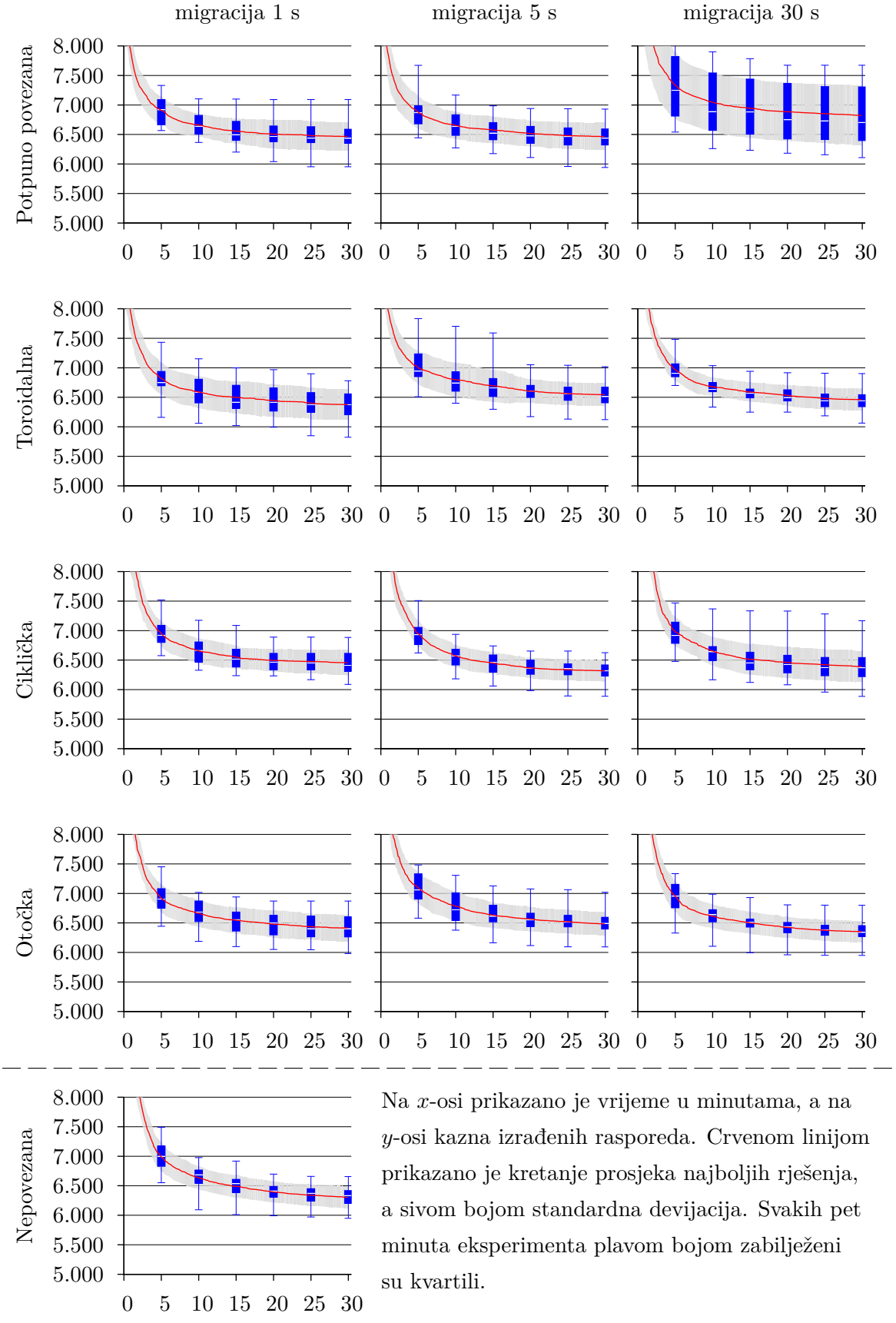
Slika 6.3: Rezultati za problem 2

Problem 3

Rezultati za završne ispite ljetnog semestra (problem 3) prikazani su u tablici 6.6 i slici 6.4. U tri od pet kriterija najboljom se pokazala ciklička topologija s učestalošću migracije svakih 5 sekundi. Prema prosječnoj kvaliteti napravljenih rasporeda nalazi se na drugom mjestu, odmah nakon nepovezane topologije. Posebno zanimljivi rezultati dobiveni su za potpunu topologiju s učestalošću migracije svakih 30 sekundi. Rezultati su daleko najgori prema gotovo svim kriterijima. Nije jasan uzrok ovakvog ponašanja i trebalo bi ga dodatno istražiti.

Tablica 6.6: Rezultati za problem 3

Topologija	Migracija	Srednji rezultat	Medijan	Najmanji	Najveći
Potpuna	1 s	6.468,0 ± 238,7	6.465,5	5.953,1	7.091,3
	5 s	6.466,3 ± 228,6	6.462,0	5.941,4	6.929,3
	30 s	6.823,8 ± 507,7	6.731,4	6.108,5	7.673,3
Toroidalna	1 s	6.377,7 ± 256,3	6.373,7	**5.824,0**	6.780,6
	5 s	6.545,4 ± 190,9	6.513,4	6.121,4	7.018,4
	30 s	6.461,7 ± 184,6	6.474,1	6.063,9	6.902,2
Ciklička	1 s	6.457,6 ± 210,3	6.413,7	6.089,9	6.885,5
	5 s	6.320,2 ± **178,0**	**6.351,4**	5.887,6	**6.625,9**
	30 s	6.391,1 ± 260,6	6.400,4	5.886,6	7.167,6
Otočna	1 s	6.414,8 ± 225,2	6.418,9	5.980,9	6.870,8
	5 s	6.485,9 ± 195,5	6.508,0	6.093,9	7.020,0
	30 s	6.353,9 ± 180,5	6.352,6	5.949,8	6.800,4
Nepovezana		**6.302,6** ± 184,2	6.352,1	5.948,5	6.655,0



Slika 6.4: Rezultati za problem 3

6.3.2. Izrada rasporeda međuispita s poboljšanim parametrima

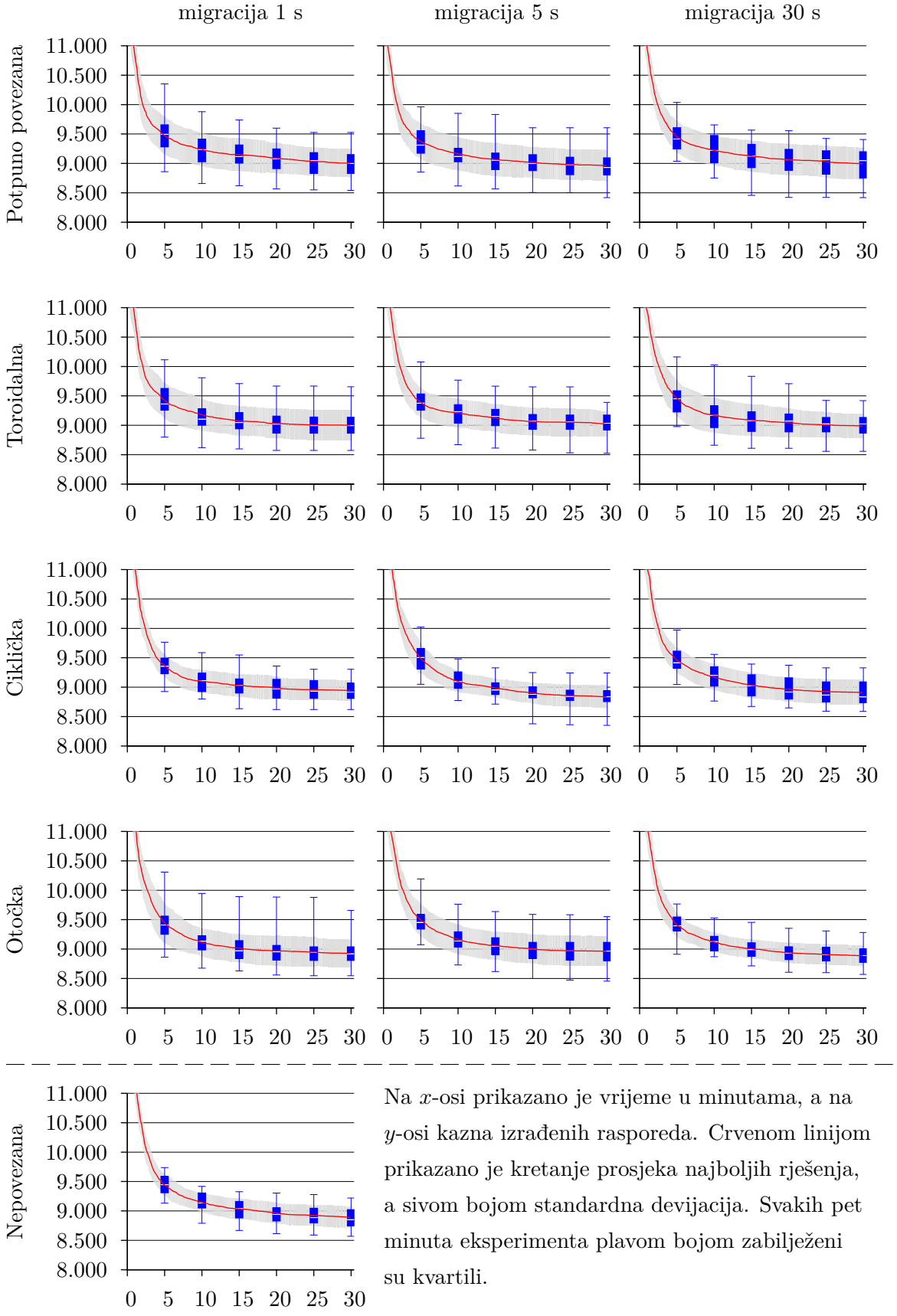
Drugi skup eksperimenata identičan je prvom skupu eksperimenata osim konfiguracija algoritama. Svi parametri algoritama poboljšani su za međuispite ljetnog semestra. Poboljšavanje parametara izvršeno je na izoliranim algoritmima prema svim parametrima koji bi mogli utjecati na kvalitetu rješenja. Stopa mutacije kod generacijskog genetskog algoritma povećana je na 0,5%, kod eliminacijskog genetskog algoritma na 2%, a kod jednostavnog imunološkog sustava na 1%. Harmonijsko pretraživanje podešeno je na $r_{prihvati} = 99,9\%$ i $r_{ra} = 7\%$. Svi algoritmi imali su aktiviranu lokalnu pretragu.

Problem 1

Rezultati za završne ispite ljetnog semestra (problem 1) prikazani su u tablici 6.7 i slici 6.5. Poboljšavanjem parametara neki su rezultati bolji, a neki drugi lošiji. I dalje se ne može izdvojiti najbolja topologija za konkretni problem.

Tablica 6.7: Rezultati za problem 1

Topologija	Migracija	Srednji rezultat	Medijan	Najmanji	Najveći
Potpuna	1 s	9.004,9 ± 239,9	9.028,7	8.542,4	9.527,0
	5 s	8.964,9 ± 264,1	8.957,0	8.418,1	9.606,1
	30 s	8.997,3 ± 270,4	9.062,1	8.418,9	9.407,6
Toroidalna	1 s	9.002,8 ± 257,0	8.998,3	8.572,8	9.656,0
	5 s	9.020,6 ± 210,7	9.058,0	8.523,8	9.389,0
	30 s	8.988,0 ± 202,4	9.037,3	8.557,3	9.419,0
Ciklička	1 s	8.945,6 ± **173,5**	8.930,6	8.619,6	9.304,5
	5 s	8.839,0 ± 191,4	8.843,1	**8.350,5**	9.242,2
	30 s	8.910,1 ± 215,9	**8.836,9**	8.589,5	9.329,4
Otočna	1 s	8.928,0 ± 233,7	8.951,5	8.547,8	9.657,1
	5 s	8.962,4 ± 252,2	8.969,2	8.455,6	9.552,0
	30 s	**8.887,9** ± 175,4	8.902,0	8.569,8	9.282,1
Nepovezana		8.888,0 ± 177,8	8.861,7	8.569,6	**9.219,7**



Na x-osi prikazano je vrijeme u minutama, a na y-osi kazna izrađenih rasporeda. Crvenom linijom prikazano je kretanje prosjeka najboljih rješenja, a sivom bojom standardna devijacija. Svakih pet minuta eksperimenta plavom bojom zabilježeni su kvartili.

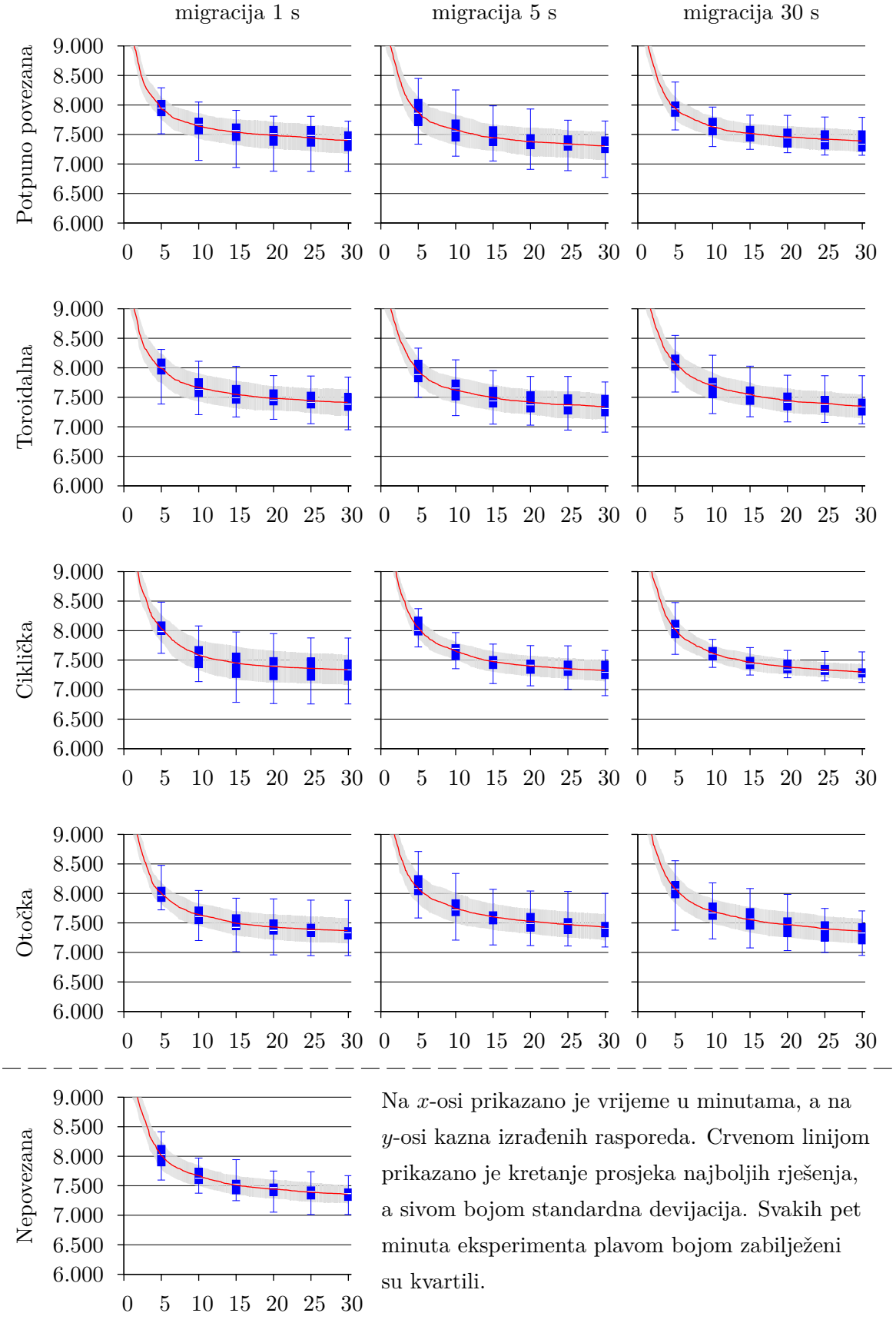
Slika 6.5: Rezultati za problem 1

Problem 2

Rezultati za završne ispite ljetnog semestra prikazani su u tablici 6.8 i slici 6.6. Rezultati su zanimljivi jer se po gotovo svim kriterijima najboljom kombinacijom parametara pokazala ciklička topologija s migracijom svakih 30 sekundi. S obzirom na to da su parametri algoritama bili poboljšani upravo za ovaj problem, rezultati potvrđuju da se krupnozrnatim modelom paralelizacije mogu postići kvalitetniji rezultati od trivijalnog paralelnog evolucijskog algoritma.

Tablica 6.8: Rezultati za problem 2

Topologija	Migracija	Srednji rezultat	Medijan	Najmanji	Najveći
Potpuna	1 s	7.400,7 ± 219,8	7.429,2	6.874,2	7.724,5
	5 s	7.307,7 ± 235,7	7.319,3	**6.772,9**	7.727,6
	30 s	7.389,2 ± 181,1	7.345,8	7.149,0	7.791,5
Toroidalna	1 s	7.413,5 ± 222,5	7.418,6	6.951,0	7.841,6
	5 s	7.340,2 ± 214,4	7.353,1	6.909,5	7.759,4
	30 s	7.350,9 ± 202,8	7.360,4	7.051,2	7.865,2
Ciklička	1 s	7.339,0 ± 253,0	7.386,4	6.757,5	7.873,9
	5 s	7.324,7 ± 178,8	7.294,0	6.898,2	7.664,9
	30 s	**7.299,4 ± 131,4**	**7.281,1**	7.121,8	**7.639,0**
Otočna	1 s	7.371,2 ± 212,9	7.368,9	6.947,5	7.883,5
	5 s	7.429,2 ± 220,8	7.452,7	7.092,5	8.001,9
	30 s	7.362,9 ± 210,8	7.416,4	6.949,6	7.703,6
Nepovezana		7.356,3 ± 148,6	7.386,4	7.012,2	7.671,2



Na x-osi prikazano je vrijeme u minutama, a na y-osi kazna izrađenih rasporeda. Crvenom linijom prikazano je kretanje prosjeka najboljih rješenja, a sivom bojom standardna devijacija. Svakih pet minuta eksperimenta plavom bojom zabilježeni su kvartili.

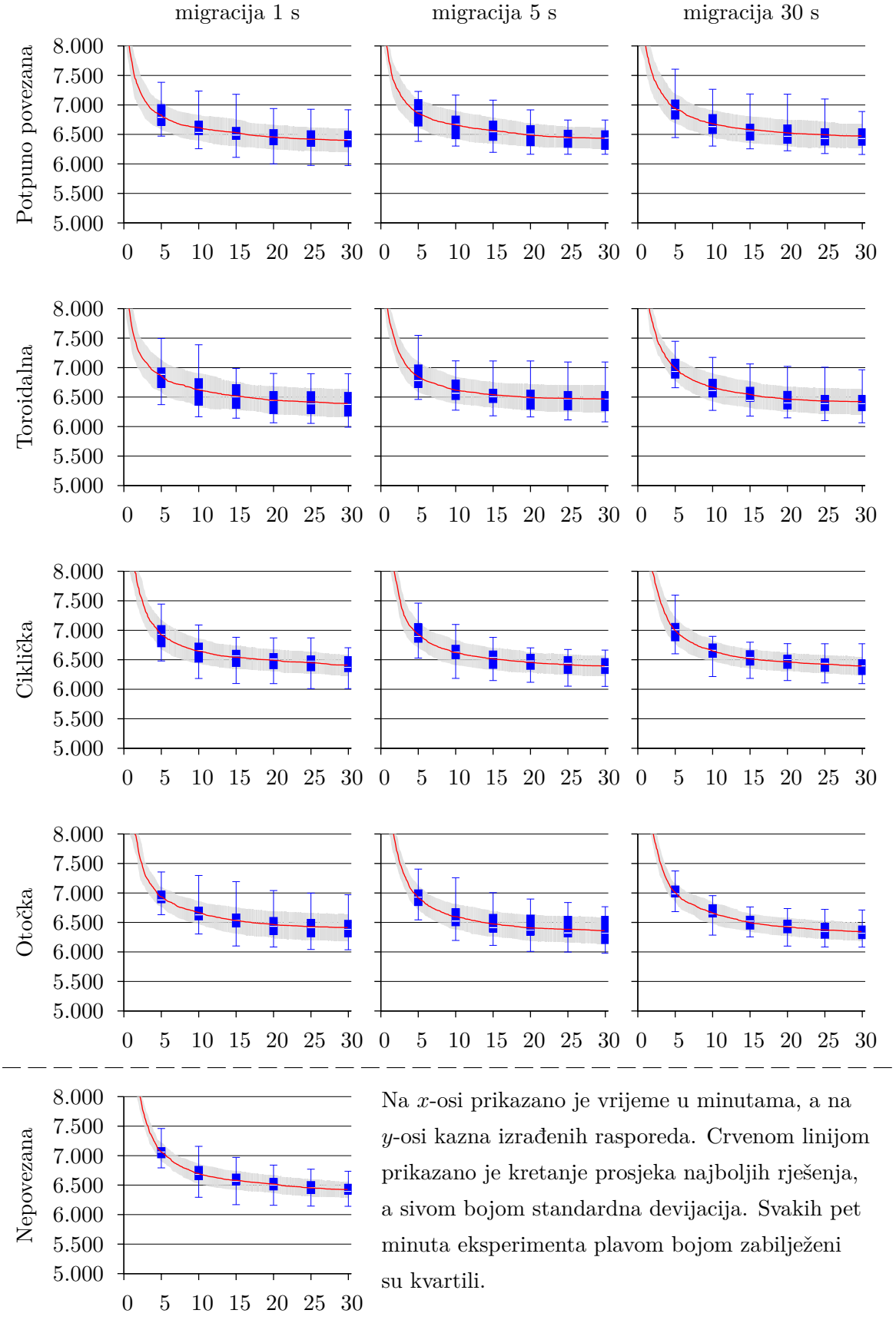
Slika 6.6: Rezultati za problem 2

Problem 3

Rezultati za završne ispite ljetnog semestra (problem 3) prikazani su u tablici 6.9 i slici 6.7. Rezultati nisu značajno izmijenjeni poboljšavanjem parametara algoritama. Zanimljivo je da potpuna topologija s migracijom svakih 30 sekundi, iako i dalje najgora od svih, više nije značajno gora od ostalih kombinacija.

Tablica 6.9: Rezultati za problem 3

Topologija	Migracija	Srednji rezultat	Medijan	Najmanji	Najveći
Potpuna	1 s	6.401,5 ± 199,6	6.415,6	**5.972,5**	6.918,2
	5 s	6.436,9 ± 179,6	6.441,2	6.164,6	6.743,3
	30 s	6.473,8 ± 203,0	6.505,1	6.159,5	6.887,7
Toroidalna	1 s	6.393,9 ± 234,7	6.385,3	5.990,8	6.897,2
	5 s	6.470,3 ± 228,0	6.469,4	6.080,8	7.095,8
	30 s	6.422,6 ± 216,2	6.391,9	6.065,6	6.963,0
Ciklička	1 s	6.411,0 ± 177,2	6.438,2	6.010,6	6.705,0
	5 s	6.396,8 ± 171,5	6.402,8	6.049,6	**6.664,8**
	30 s	6.394,7 ± 154,5	6.403,8	6.096,8	6.771,3
Otočna	1 s	6.413,6 ± 226,6	6.406,8	6.036,7	6.970,5
	5 s	6.361,2 ± 228,0	**6.319,2**	5.978,7	6.767,4
	30 s	**6.344,7** ± 153,0	6.354,4	6.082,0	6.710,9
Nepovezana		6.424,4 ± **134,4**	6.429,7	6.141,3	6.734,6



Slika 6.7: Rezultati za problem 3

6.3.3. Izrada rasporeda ponovljenih ispita

Izrada rasporeda ponovljenih ispita testirana je s istim kombinacijama migracijskih parametara kao i u prvom i u drugom skupu eksperimenata. Korištena su dva generacijska genetska algoritma (veličina populacije je 100, a stopa mutacije 0,5%), dva eliminacijska genetska algoritma (veličina populacije je 200, a stopa mutacije 0,8%), dva algoritma $\mathcal{MM}$AS ($\rho = 0,02$, $\alpha = 1,5$, $a = 100$ i 50 prolazaka mrava) i dva jednostavna imunološka algoritma (broj klonova je 10, veličina populacije 100, a stopa mutacije 2%). Svaki algoritam koji je upotrebljavao operator križanja koristio je 3-turnirski odabir, a svi algoritmi koristili su lokalno pretraživanje. Za svako novo rješenje u algoritmu unutar lokalnog pretraživanja pokušavala su se smanjiti prekršena čvrsta ograničenja izmjenama termina najviše dva različita problematična predmeta, a ako rješenje nije kršilo čvrsta ograničenja, nastojala su se smanjiti prekršena meka ograničenja u izmjenama termina najviše četiri predmeta. Za najbolje pronađeno rješenje u iteraciji svakog algoritma, čvrsta su se rješenja nastojala ispraviti u 10 izmjena, a meka ograničenja u 15 izmjena.

Problem 4

Rezultati za ponovljene završne ispite zimskog semestra (problem 4) prikazani su u tablici 6.10. Može se primijetiti kako dobiveni rasporedi vrlo često nemaju ispunjena sva čvrsta ograničenja. Zbog toga nije bilo moguće izračunati prosjek i standardnu devijaciju rješenja, već je svaki raspored za određenu kombinaciju migracijskih parametara uspoređen sa svim rasporedima ostalih kombinacija. Prosječni rezultat tog uspoređivanja prikazan je u zadnjem stupcu. Za razliku od rezultata u prethodne dvije skupine eksperimenata, rezultati za ponovljene ispite imaju vrlo velike razlike između najboljih i najgorih rješenja.

Problem 5

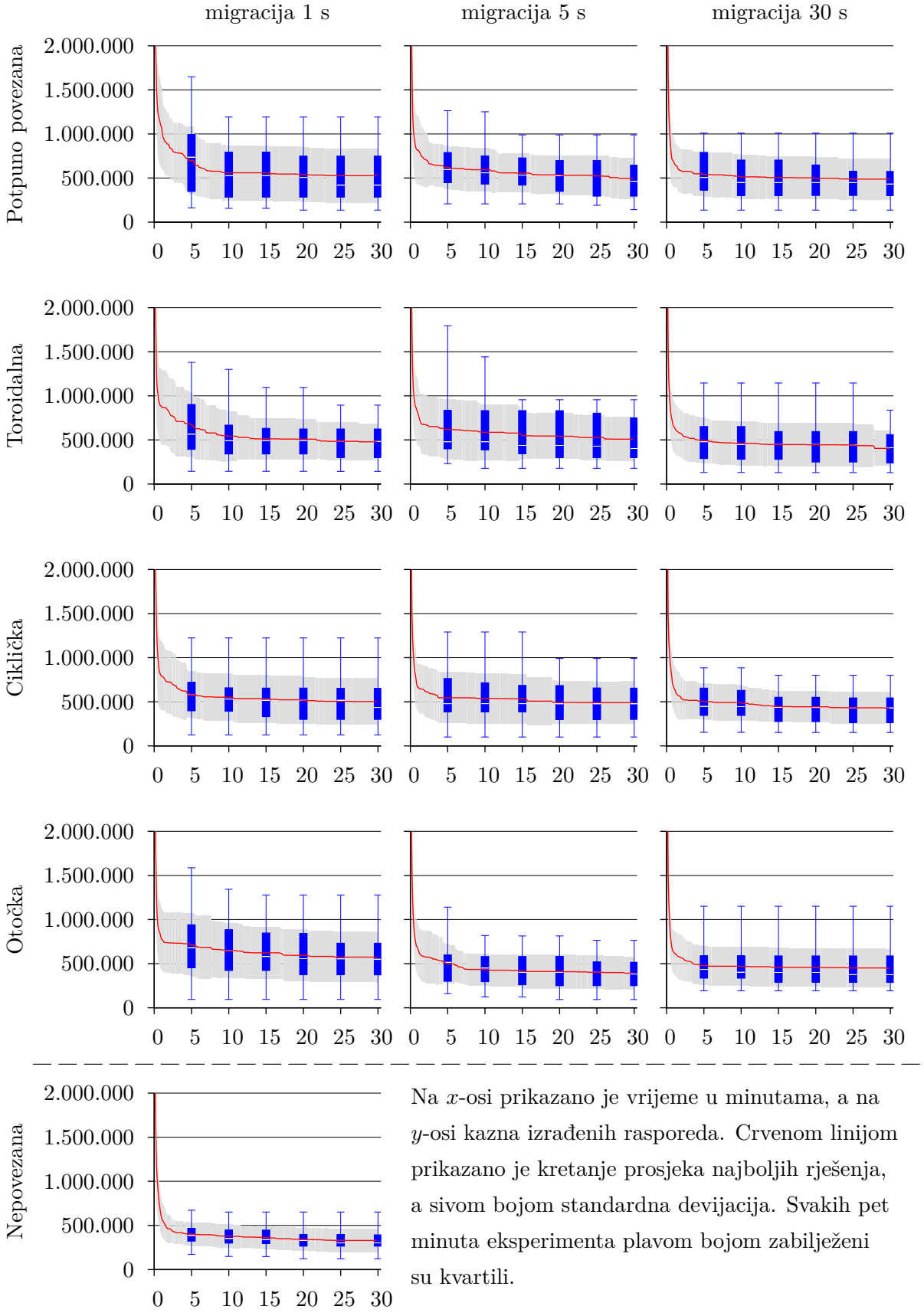
Rezultati za ponovljene završne ispite ljetnog semestra (problem 5) prikazani su u tablici 6.11 i slici 6.8. Za razliku od ponovljenih ispita završnih ispita zimskog semestra, sva pokretanja eksperimenata su u vrlo kratkom vremenu imala rasporede bez čvrstih ograničenja pa je bilo moguće izračunati srednji rezultat, standardnu devijaciju i prikazati statističke podatke prema trajanju eksperimenta. Prema gotovo svim kriterijima, najbolja svojstva pokazuje nepovezana topologija.

Tablica 6.10: Rezultati za problem 4

Topologija	Migracija	Najmanji	Medijan	Najveći	Bolji od
Potpuna	1 s	[1, 256.768]	[1, 1.699.723]	[3, 548.524]	41,46%
	5 s	[0, 1.681.431]	[1, 1.158.772]	[2, 1.598.469]	52,09%
	30 s	**[0, 1.009.195]**	[1, 1.198.885]	[2, 1.658.482]	52,43%
Toroidalna	1 s	[1, 354.856]	[1, 1.557.491]	[2, 2.065.073]	43,70%
	5 s	[0, 2.107.880]	[1, 1.509.633]	[3, 1.057.888]	44,75%
	30 s	[1, 457.964]	[1, 1.214.277]	[2, 1.853.818]	49,99%
Ciklička	1 s	[0, 1.948.553]	[1, 1.362.518]	[2, 1.551.475]	51,51%
	5 s	[0, 1.857.891]	[1, 1.250.784]	[3, 950.846]	54,65%
	30 s	[0, 1.353.986]	[1, 1.418.576]	[2, 1.708.629]	48,91%
Otočna	1 s	[1, 547.541]	[1, 1.551.895]	[3, 300.308]	44,38%
	5 s	[1, 209.530]	[1, 1.204.935]	[2, 1.854.799,5]	50,67%
	30 s	[0, 2.691.523]	**[1, 947.536]**	**[2, 1.488.184]**	56,82%
Nepovezana		[0, 1.774.333]	[1, 1.015.054]	[2, 1.250.591]	**58,64%**

Tablica 6.11: Rezultati za problem 5

Topologija	Migracija	Srednji rezultat	Medijan	Najmanji	Najveći
Potpuna	1 s	527.050 ± 307.924	528.950	136.433	1.192.838
	5 s	496.180 ± 232.443	499.274	142.610	989.268
	30 s	486.698 ± 231.492	449.273	136.789	1.008.949
Toroidalna	1 s	478.172 ± 205.909	491.993	144.000	895.104
	5 s	509.012 ± 247.868	430.180	178.261	956.081
	30 s	423.370 ± 247.373	442.017	129.338	1.239.602
Ciklička	1 s	502.388 ± 262.241	529.570	123.823	1.223.511
	5 s	490.921 ± 239.099	477.820	100.466	991.311
	30 s	429.754 ± 181.309	438.718	153.164	800.569
Otočna	1 s	572.518 ± 285.741	556.793	94.514	1.279.025
	5 s	392.961 ± 182.430	402.761	**94.265**	764.700
	30 s	447.844 ± 214.963	389.748	192.033	1.151.222
Nepovezana		**333.236 ± 132.830**	**317.915**	122.603	**651.517**



Slika 6.8: Rezultati za problem 5

6.3.4. Utjecaj različitih kombinacija algoritama

U zadnjem skupu eksperimenata isproban je utjecaj različitih kombinacija algoritama na kvalitetu krajnjih rješenja. Korišteni su podaci iz međuispita i završnih ispita zimskog semestra (problem 1). Potrebno je napomenuti kako su programska ostvarenja algoritama i funkcije vrednovanja drugačiji od prethodnih skupina eksperimenata pa nije ispravno uspoređivati rezultate s drugim skupinama eksperimenata. Isprobane su sve moguće kombinacije s generacijskim genetskim algoritmom (veličina populacije je 10, a stopa mutacije 2%), algoritma $\mathcal{MM}$AS ($\rho = 0,02$, $\alpha = 1,5$, $a = 100$ i 50 prolazaka mrava), harmonijskog pretraživanja (veličina populacije 10, $r_{prihvati} = 99\%$, a $r_{ra} = 5\%$) i jednostavnog imunološkog sustava (broj klonova je 10, veličina populacije 100, a stopa mutacije 1%) s lokalnom pretragom i bez nje. Upotrijebljena je otočna topologija gdje su tri otoka imala po četiri algoritma. Učestalost migracije unutar otoka obavlja se svake sekunde, a otoci su međusobno komunicirali svake četiri sekunde.

Rezultati bez upotrebe lokalnog pretraživanja

Rezultati bez upotrebe lokalnog pretraživanja prikazani su u tablici 6.12. Najbolje rezultate u tri od pet kriterija pokazao je generacijski genetski algoritam bez ostalih algoritama. Ipak, zbog velike devijacije rezultata niti jedna kombinacija algoritama nije izdvojena kao posebno dobra. Najgore rezultate postigao je algoritam $\mathcal{MM}$AS.

Rezultati s upotrebom lokalnog pretraživanja

Rezultati su upotrebom lokalnog pretraživanja prikazani su u tablici 6.13. Rezultati su potpuno drugačiji od rezultata bez upotrebe lokalnog pretraživanja. Prema gotovo svim kriterijima najbolje rezultate pokazala je konfiguracija u kojoj je bilo harmonijsko pretraživanje bez drugih algoritama. Posebno dobro svojstvo pokazuje u vrlo maloj standardnoj devijaciji rezultata. Sve kombinacije algoritama imaju puno bolje rezultate s upotrebom lokalnog pretraživanja.

Tablica 6.12: Rezultati bez upotrebe lokalnog pretraživanja

Eksperiment	Srednji rezultat	Medijan	Najmanji	Najveći
GA	**8.280,7** ± 350,1	**8.241,0**	**7.671,3**	9.232,1
$\mathcal{MM}$AS	13.415,1 ± 670,5	13.454,3	11.302,5	14.726,5
HS	8.926,6 ± 370,7	8.887,3	8.137,1	9.854,3
SIA	8.476,4 ± **268,5**	8.477,0	8.019,0	**8.904,7**
$\mathcal{MM}$AS, SIA	8.609,7 ± 338,8	8.644,0	7.925,5	9.187,9
GA, SIA	8.465,2 ± 308,8	8.503,0	7.928,2	9.183,0
HS, SIA	8.473,4 ± 287,7	8.484,8	7.957,2	9.438,6
GA, $\mathcal{MM}$AS	8.483,8 ± 290,3	8.431,5	7.688,6	9.170,9
$\mathcal{MM}$AS, HS	9.031,3 ± 307,5	9.004,6	8.416,4	9.684,3
GA, HS	8.453,4 ± 285,8	8.472,9	7.857,4	9.220,6
GA, $\mathcal{MM}$AS, SIA	8.434,3 ± 275,0	8.446,7	7.725,8	8.946,2
$\mathcal{MM}$AS, HS, SIA	8.517,3 ± 359,2	8.416,0	7.895,0	9.263,5
GA, HS, SIA	8.447,5 ± 288,3	8.484,7	7.824,2	9.073,7
GA, $\mathcal{MM}$AS, HS	8.591,2 ± 269,6	8.509,1	8.076,8	9.141,1
GA, $\mathcal{MM}$AS, HS, SIA	8.543,5 ± 333,7	8.557,6	7.964,7	9.534,8

Tablica 6.13: Rezultati s upotrebom lokalnog pretraživanja

Eksperiment	Srednji rezultat	Medijan	Najmanji	Najveći
GA	8.566,7 ± 277,4	8.527,0	8.048,0	9.078,0
$\mathcal{MM}$AS	7.148,7 ± 194,9	7.152,4	6.754,1	7.470,9
HS	**6.888,7 ± 115,5**	**6.921,9**	6.705,1	**7.113,7**
SIA	8.459,8 ± 303,4	8.427,9	8.017,6	9.202,7
$\mathcal{MM}$AS, SIA	7.225,1 ± 187,1	7.244,2	6.721,6	7.690,7
GA, SIA	8.578,5 ± 276,1	8.584,6	8.080,7	9.096,8
HS, SIA	6.979,4 ± 150,8	7.013,5	6.679,2	7.398,0
GA, $\mathcal{MM}$AS	7.210,3 ± 171,2	7.204,8	6.699,4	7.542,6
$\mathcal{MM}$AS, HS	6.957,4 ± 137,1	6.953,3	**6.633,0**	7.258,0
GA, HS	7.130,1 ± 183,2	7.120,7	6.763,2	7.525,7
GA, $\mathcal{MM}$AS, SIA	7.309,0 ± 165,6	7.343,5	7.007,8	7.637,2
$\mathcal{MM}$AS, HS, SIA	6.960,0 ± 146,4	6.956,8	6.735,7	7.323,5
GA, HS, SIA	7.112,5 ± 189,3	7.093,3	6.791,7	7.624,4
GA, $\mathcal{MM}$AS, HS	7.050,7 ± 155,2	7.050,4	6.768,5	7.373,4
GA, $\mathcal{MM}$AS, HS, SIA	7.021,1 ± 186,9	6.985,9	6.713,5	7.450,7

7. Zaključak

Razni optimizacijski problemi koji ne mogu biti riješeni determinističkim algoritmima često se rješavaju pomoću algoritama evolucijskog računanja. Opisan je način predstavljanja rješenja problema na računalu i navedeni su osnovni operatori evolucijskog računanja. Objašnjeni su najpoznatiji algoritmi iz različitih kategorija evolucijskog računanja poput genetskog algoritma, algoritma umjetnog imunološkog sustava, algoritma kolonije mrava i harmonijskog pretraživanja. Opisane su pretpostavke, potencijalne prednosti, vrste i parametri paralelizacije evolucijskog računanja.

Glavni cilj ovog rada bio je razviti platformu za raspodjeljivanje evolucijskog računanja. Osnovni zahtjevi bili su podrška izrade različitih vrsta evolucijskih algoritama, podrška za rješavanje raznih optimizacijskih problema, automatska razmjena rješenja krupnozrnatim modelom paralelizacije, jednostavno definiranje migracijskih parametara i automatsko prikupljanje statističkih podataka. Razvijena je platforma Galapagos koja ispunjava sve navedene zahtjeve. Pisana je u programskom jeziku Java pa je neovisna o operacijskom sustavu. Opisane su mogućnosti platforme, a programsko ostvarenje novih modula objašnjeno je na jednostavnom primjeru.

Opisana su dva optimizacijska problema: izrada rasporeda međuispita i izrada rasporeda ponovljenih ispita. Problemi su formalno definirani, a funkcija vrednovanja kreiranih rješenja detaljno je opisana. Platforma Galapagos ispitana je na tim optimizacijskim problemima. Upotrijebljeni su različiti ulazni podaci, migracijski parametri i kombinacije algoritama. Eksperimenti su pokazali kako migracijski parametri mogu značajno utjecati na krajnje rezultate. U velikom broju eksperimenata najbolja svojstva pokazao je trivijalni paralelni evolucijski algoritam. Ipak, kada su korišteni poboljšani parametri algoritama na jednom problemu, ciklička topologija s učestalošću migracije svakih 30 s pokazala je bolja svojstva od trivijalnog paralelnog evolucijskog algoritma. U zadnjoj skupini eksperimenata isproban je utjecaj različitih kombinacija algoritama i lokalnog

pretraživanja. Primijećeno je kako lokalno pretraživanje značajno poboljšava dobivene rezultate, ali kombinacija različitih algoritama nije se pokazala kao najbolja. Zaključeno je kako platforma Galapagos može jako pomoći u rješavanju različitih optimizacijskih problema paralelizacijom slijednih algoritama evolucijskog računanja. Omogućava jednostavno izvođenje velikog broja eksperimenata i prikupljanje statističkih podataka.

U daljnjim istraživanjima potrebno je detaljnije istražiti zbog čega trivijalni paralelni evolucijski algoritam često pokazuje bolja svojstva od krupnozrnate paralelizacije. Jedno od objašnjenja moglo bi biti da neki algoritmi nisu pogodni za krupnozrnatu paralelizaciju. Primjerice, moguće je da algoritam mravlje kolonije puno lošije radi ako feromonski trag obnavljaju rješenja iz drugih algoritama. Moguće je i da lokalno pretraživanje loše djeluje na krupnozrnatu paralelizaciju jer ubrzava konvergenciju algoritama i onemogućuje kvalitetnu integraciju rješenja drugih algoritama. Daljnji rad na platformi Galapagos trebao bi biti na razvoju modula za druge probleme i proširivanju mogućnosti. Nove mogućnosti mogu biti direktna komunikacija između računala radnika, otpornost na pogreške i proširivanje mogućnosti definiranja migracijskih parametara.

Literatura

M. Affenzeller, S. Winkler, S. Wagner i A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications (Numerical Insights)*. Chapman & Hall, 2009.

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

E. Alba i M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.

J.E. Baker. Reducing bias and inefficiency in the selection algorithm. U *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, stranice 14–21. L. Erlbaum Associates Inc., 1987.

N.A. Barricelli et al. Esempi numerici di processi di evoluzione. *Methodos*, 6 (21-22):45–68, 1954.

W. Bossert. Mathematical optimization: Are there abstract limits on natural selection? U *The Wistar Institute symposium monograph*, svezak 5, stranica 35, 1967.

H.J. Bremermann. Optimization through evolution and recombination. *Self-organizing systems*, stranice 93–106, 1962.

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

M. Čupić, M. Golub i D. Jakobović. Exam timetabling using genetic algorithm. U *31st International Conference on Information Technology Interfaces*, 2009.

M. Čupić, M. Golub i D. Jakobović. Applying AI-techniques as help for faculty administration – a case study. *Central European Conference on Information and Intelligent Systems*, 2010.

L.N. de Castro i F.J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Computation, Special Issue on Artificial Immune Systems*, 6(3):239–251, 2002.

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

M.R. Garey i D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences*. WH Freeman and Company, San Francisco, 1979.

Z.W. Geem. *Music-inspired harmony search algorithm: theory and applications*. Springer Verlag, 2009.

M. Gorges-Schleuter. Asparagos an asynchronous parallel genetic optimization strategy. U *Proceedings of the 3rd International Conference on Genetic Algorithms*, stranice 422–427. Morgan Kaufmann Publishers Inc., 1989.

K. Hwang. *Advanced computer architecture: parallelism, scalability, programmability*, svezak 348. McGraw-Hill, 1993.

M. Komar, Đ. Grbić i M. Čupić. Solving Exam Timetabling Using Distributed Evolutionary Computation. *33rd International Conference on Information Technology Interfaces*, 2011.

M. Matsumoto i T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulatio*, stranice 3–30, 1998. ISSN 1049-3301.

A. Prokopec. *Prilagodljivi genetski algoritam*. Fakultet elektrotehnike i računarstva, 2009.

A.A. Tantar, N. Melab, E.G. Talbi, B. Parent i D. Horvath. A parallel hybrid genetic algorithm for protein structure prediction on the computational grid. *Future Generation Computer Systems*, 23(3):398–409, 2007.

D. De Werra. An introduction to timetabling. *European Journal Of Operational Research*, 19(2):151–162, 1985. URL `http://linkinghub.elsevier.com/retrieve/pii/0377221785901675`.

D.H. Wolpert i W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Izgradnja platforme za raspodijeljene algoritme evolucijskog računanja

Sažetak

Algoritmi evolucijskog računanja često se upotrebljavaju za rješavanje optimizacijskih problema. Kvaliteta rada evolucijskog računanja može se dodatno poboljšati njihovom paralelizacijom. U ovom radu opisano je evolucijsko računanje i objašnjeno je nekoliko najpoznatijih algoritama. Navedene su metode raspodjeljivanja evolucijskog računanja i različiti migracijski parametri. Razvijeno je programsko okruženje koje omogućava jednostavno raspodjeljivanje evolucijskog računanja za rješavanje raznih problema. Platforma je isprobana na konkretnoj implementaciji za rješavanje problema izrade rasporeda ispita. Isprobane su različite kombinacije primjera problema i migracijskih parametara, a rezultati su statistički obrađeni i međusobno uspoređeni.

Ključne riječi: evolucijsko računanje, optimizacijski problemi, raspored ispita

Development of framework for distributed evolutionary computing algorithms

Abstract

Evolutionary computing algorithms are often used for solving optimization problems. The quality of evolutionary computing can additionally be improved by their parallelization. In this paper the evolutionary computing is described and several algorithms are explained. Distributed evolutionary computing methods were listed, as well as different migration parameters. Framework for simple distributed evolutionary computing was developed for solving various problems. The framework was evaluated on particular implementation for solving exam timetabling. Various combinations of problem examples and migration parameters were tested, results were statistically analyzed and compared to each other.

Keywords: evolutionary computing, optimization problems, exam timetabling