SVEUČILIŠTE U ZAGREBU, FAKULTET ELEKTROTEHNIKE
I RAČUNARSTVA

DIPLOMSKI ZADATAK br. 1823

Primjena hibridnog evolucijskog algoritma na problem raspoređivanja u visokoškolskim ustanovama

Vjera Omrčen

Zagreb,2010.

Sadržaj

1 Uvod

Sve vrste obrazovnih ustanova suočavaju se svake godine s problemom izrade plana nastavnih aktivnosti. Problem raspoređivanja nastavnih aktivnosti specifičan za različite vrste obrazovnih ustanova, ali u širem smislu može ga se definirati kao problem dodjele termina održavanja i potrebnih sredstava (prostorija, nastavnog osoblja, učenika i nastavnih pomagala) svim nastavnim aktivnostima poštujući pri tome njihova pojedinačna specfična ograničenja i zahtjeve.

U literaturi se kao tri osnovne podvrste problema raspoređivanja navode:

- problem osnovnoškolskog rasporeda *(school timetabling)*
  - učenici su grupirani u nedjeljive razrede
  - pretpostavka je isti nastavni program za sve učenike u istom razredu
  - nastava se mora održavati u kontinuiranim blokovima
- problem sveučilišnog rasporeda *(university course timetabling)*
  - studenti nisu nužno strogo grupirani
  - putem izbornih predmeta studenti sami djelomično određuju svoj nastavni program
- problem rasporeda ispita *(examination timetabling)*
  - rasporedi u kojima bilo koji student mora prisustvovati na dva ispita istovremeno nedopustivi su
  - raspršeniji rasporedi su bolji

Ručna izrada rasporeda čak i u jednostavnijim slučajevima poput osnovnih ili srednjih škola može biti vrlo dugotrajan i naporan posao. Štoviše, smatra se da je u većini slučajeva konkretna izrada sveučilišnog rasporeda zbog iznimno velikog prostora rješenja i specifičnih zahtjeva pojedinih kolegija NP potpun problem. Upravo iz tog razloga automatizirana izrada rasporeda je područje intenzivnog istraživanja moderne računarske znanosti.

Slijepe metode pretraživanja i iscrpna pretraga neprimjenjivi su zbog velikog prostora rješenja konkretnih problema rasporeda, te se u većini slučajeva problem pokušava riješiti nekom od heurističkih metoda. Pokazuje se da, iako heurističke metode ne garantiraju pronalazak optimalnog rješenja, ipak u praksi mogu proizvesti dovoljno dobre rasporede za upotrebu. Detaljan pregled trendova u rješavanju problema raspoređivanja može se naći u [3, 1, 5, 6, 2].

2 Evolucijsko računarstvo

2.1 Općenito o evolucijskom računarstvu

Evolucija je prirodni proces optimizacije prilagođenosti jedinki okolini u kojoj žive . Proces evolucije odvija se u skladu sa sljedećim osnovnim načelima :

- Lošije prilagođene jedinke imaju veću vjerojatnost odumiranja
- Bolje jedinke imaju veću šansu križati se i prenijeti svoje gene na potomstvo
- Moguće su slučajne promjene (mutacije) genotipa jedinki

Evolucijske mehanizme putem kojih priroda optimira prilagođenost živih vrsta njihovom staništu može se primjeniti na bilo koji optimizacijski problem vodeći se pri tome sljedećom analogijom:

Okolina	$\leftrightarrow$	Optimizacijski problem
Jedinka	$\leftrightarrow$	Rješenje
Prilagođenost	$\leftrightarrow$	Kvaliteta rješenja

Evolucijsko računarstvo je grana računalne znanosti koja se bavi oblikovanjem i proučavanjem heurističkih metoda optimizacije inspiriranih biološkom evolucijom. Po načinu djelovanja metode evolucijskog računarstva ubrajaju se u metode usmjerenog slučajnog pretraživanja prostora stanja. Karakteristično za metode evolucijskog računarstva je to što pretragu prostora rješenja ne započinju iz jedne početne točke, nego iz populacije početnih točaka koje se iterativno poboljšavaju primjenom operatora inspiriranih mehanizmima biološke evolucije.

Metode evolucijskog računarstva dijele se na:

1. Genetske algoritme
    - jedinke u populaciji konstantne veličine nastaju djelovanjem operatora selekcije, križanja i mutacije
2. Evolucijske strategije
    - najčešće korišteni operator je mutacija, operator križanja se ponekad uopće ne koristi

- selekcija jedinki je deterministički određena dobrotom jedinki

3. Genetsko programiranje

   - jedinke populacije ne predstavljaju potencijalna rješenja optimizacijskog problema, već potencijalni *algoritam rješavanja* istog

Glavna prednost navedenih evolucijskih metoda je konvergencija populacije rješenja prema globalnom optimumu. Međutim, kako je za neke optimizacijske probleme jedino iscrpnom pretragom moguće odrediti je li ponuđeno rješenje zaista globalni optimum, nema nikakve garancije da je rješenje dobiveno nakon bilo kojeg broja iteracija nekog od evolucijskih algoritama zaista najbolje moguće. Usprkos tome, metode evolucijskog računarstva pokazale su se kao vrlo moćan alat za rješavnje čitavog niza problema iz inžinjerske prakse u kojima je prostor rješenja prevelik za determinističko pretraživanje, a dovoljno je naći rješenje zadovoljavajuće tj. primjenjive kvalitete.

2.2 Genetski algoritam

Genetski algoritam je vrlo popularna metoda evolucijskog računarstva, koja primjenom operatora selekcije, križanja i mutacije imitira evolucijski proces.

Pri planiranju izrade sustava za optimizaciju zasnovanog na genetskom algoritmu treba prvo definirati način pohrane jedinki populacije u memoriju, operatore prilagođene strukturi jedinke i funkciju dobrote. U literaturi [4, 7]se najčešće susreće *klasični GA* - tj. varijanta genetskog algoritma kod kojega se jedinke zapisuju nizom bitova, a operatori križanja i mutacije temelje se na sklopovski podržanim operacijama, što ubrzava izvođenje na računalu. Nedostatak ovakvog pristupa je u tome što je ponekad rješenja optimizacijskih problema teško i neintuitivno kodirati nizom bitova, ili je prikaz nizom bitova nepogodan za evaluaciju dobrote jedinke. Klasični GA tada bi gubio puno vremena na dekodiranje jedinke pri svakom izračunu dobrote, te je u takvim slučajevima, bolji pristup osmisliti problemu prilagođenu strukturu jedinke i odgovarajuće operatore.

2.2.1 Hibridni genetski algoritam

Hibridni genetski algoritam kombinira prednosti genetskog algoritma sa prednostima operatora lokalne pretrage. Genetski algoritam konvergira prema globalnom optimumu ali je u većini slučajeva spor, dok lokalna pretraga konvergira

brzo prema lokalnom optimumu koji može ali ne mora biti i globalni. Kombiniranjem ova dva pristupa postiže se to da GA pronalazi regiju globalnog optimuma, a lokalna pretraga u toj regiji pronalazi optimumum.

Vodeći se analogijom optimizacijskih problema i evolucije živih bića može se reći da genetski algoritam optimira prilagođenost vrste okolini, dok se lokalna pretraga koncentrira na to da maksimalno prilagodi jedinku okolini za njenog života. U tom smislu lokalna pretraga analogna je procesu socijalizacije ili učenja jedinke.

Postupak optimizacije hibridnim genetskim algoritmom sažeto se može opisati sljedećim pseudokodom:

```
Genetski_algoritam
{
  t = 0
  generiraj pocetnu populaciju potencijalnih rjesenja
  P(0);
  ponavljaj dok nije zadovoljen uvjet zavrsetka
  evolucijskog procesa;
  {
    t = t + 1;
    selektiraj P_(t) iz P(t-1);
    P_child(t) = krizaj jedinke iz P_(t);
    lokalna_pretraga(P_child(t));
    spremi P_child(t) u P(t);
    mutiraj jedinke iz P(t);
  }
ispisi rjesenje;
}
```

3 Formalna definicija problema sveučilišnog rasporeda

3.1 Problem sveucilisnog rasporeda UCTP

Problem sveučilišnog rasporeda moze se opisati uređenom četvorkom *(M,R,T,C)* gdje su:

M	skup nastavnih aktivnosti koje treba rasporediti
R	resursi koje treba rasporediti dodijeliti nastavnim aktivnostima.
T	termini u kojima je rasporedjivanje moguce
C	skup mogucih ogranicenja/zahtjeva nastavnih aktivnosti

Kao podrazred ovog problema moguće je definirati različite probleme raspoređivanja, kao npr:

- problem rasporeda predavanja
- problem rasporeda auditornih vježbi
- problem rasporeda laboratorijskih vjezbi

Kako *UCTP* obuhvaća vrlo širok skup problema , ostvarenje konkretnog sustava za rješavanje cijele ove klase je teško zamislivo te se konkretni sustavi za automatiziranu izradu rasporeda u praksi bave njegovim podproblemima.

3.2 Problem rasporeda laboratorijskih vjezbi (LETP)

U okviru ovog rada formalno se definira problem rasporeda laboratorijskih vježbi. Problem rasporeda laboratorijskih vjezbi (*LETP* - Laboratory exercise timetabling problem) je podproblem *UCTP*-a. *LETP* se opisuje se uređenom šestorkom *(T,L,R,E,S,C)* gdje su:

T	skup *termina* u kojima je rasporedjivanje moguce	
	Termin	$t \in T$ definiran je kao skup uzastopnih vremenskih jedinica (*kvanta*) raspoređivanja.
	Kvant	je osnovna vremenska jedinica raspoređivanja. Pretpostavka je da se trajanje bilo koje nastavne aktivnosti može izraziti kao $n \cdot q$, gdje je q trajanje jednog kvanta a $n \in N$
L	skup resursa (nastavnih pomagala) potrebnih za odrzavanje vježbi	

R	skup raspoloživih prostorija
E	događaji (vježbe) koje treba rasporediti
S	studenti
C	skup ograničenja

- Za sva nastavna pomagala$l \in L$ definirana je količina ,$Q(l) = n_l$, gdje je n_l broj dostupnih nastavnih pomagala.
- Raspoloživm prostorijama $r \in R$ funkcija $f_r : R \to 2^T xN$ pridružuje uređeni par $(T_r, size_r)$ gdje su T_r termini u kojima je prostorija raspoloživa, a $size_r$ broj nedjeljivih radnih mjesta u prostoriji.
- Događaju $e \in E$ pridruženi su
  - skup ograničenja $C_e \subseteq C$ u skladu s kojima dogadjaj mora biti raspoređen,
  - skup upisanih studenata $S_e \subseteq S$,
  - skup demonstratora $Dem_e \subseteq S_e$ i broj smjena koje demonstratori moraju odraditi n_dem_e,
  - skup prikladnih prostorija za održavanje $R_e \subseteq R$,
  - skup prikladnih termina T_e,
  - trajanje dur_e,
  - ograničenje maksimalnog trajanja $span_e$,
  - broj studenata po radnom mjestu spw_e,
  - broj raspoloživog nastavnog osoblja na predmetu $staff_e$,
- Svaki događaj $e \in E$, osim ako nije drugačije nametnuto skupom njegovih ograničenja C_e, može se održati u više *instanci*.
  - *Instanca (primjerak)* i_e događaja e je trojka $(t_{i,e}, R_{i,}\mathrm{e}, S_{i,e})$ gdje su $t_{i,e} \in T_e$, $R_{i,e} \subseteq R_e$, $S_{i,e} \subseteq S_e$.
- Nad Kartezijevim produktom $E \times E$ definirana je relacija poretka događaja$\prec_d$. Dva događaja $e_1, e_2 \in E$ su elementi relacije $\prec_d$ ako i samo ako, se zadnja instanca $e_1$mora održati barem d dana prije prve instance e_2.

- Nad $R \times E$ definira se funkcija potrošnje asistenata po sobi $u : R \times E \to N$. Vrijednost $u(r, e)$ je broj asistenta koji moraju biti prisutni u prostoriji r za održavanje vježbe e. Funkcija nije definirana za $r \notin R_e$.
- Skup ograničenja C podijeljen je na *čvrsta* i *meka* ograničenja.
  - *Čvrsta ograničenja* su skup uvjeta koje svaki raspored mora zadovoljavati. Raspored je *izvediv* ako i samo ako zadovoljava sva čvrsta ograničenja.
  - *Meka ograničenja* su skup uvjeta kojima se opisuje kvaliteta rješenja, ali ne moraju biti nužno zadovoljeni da bi raspored bio izvediv.

U skladu sa organizacijskim potrebama *FER-a* čvrsta ograničenja su definirana na sljedeći način:

1. U svakoj prostoriji r u jednom terminu može se održavati maksimalno jedna instanca nekog događaja.
2. Događaj e smije se održavati samo u prostorijama $r \in R_e$.
3. Događaj e smije se održavati u prostoriji $r \in R_e$ samo u terminima $T_e \cap T_r$.
4. Svaka instanca i događaja e zauzima termin duljine dur_e.
5. Za sve raspoređene instance i događaja e vrijedi $|S_{ie} \backslash Dem_{ie}| \leq \sum_{r \in R_{ie}} spw_e \centerdot size_r$ (u prostoriju r na događaju e stane maksimalno $spw_e \centerdot size_r$ studenata).
6. Za sve raspoređene instance i događaja e vrijedi $\sum_{r \in R_{ie}} u(r, e) \leq staff(e)$ (zbroj potrebnih asistenata u svim prostorijama dodijeljenim instanci vježbe mora biti manji ili jednak broju raspoloživog nastavnog osoblja).
7. Za sve raspoređene instance i događaja e vrijedi $\bigcup_{i \in I(e)} S_{ie} \subseteqq S_e$; (svi studenti koji moraju obaviti vježbu su raspoređeni).
8. Svaki demonstrator $dem_e \in Dem_e$ mora prisustvovati na n_dem_e instanci događaja e.
9. Paralelno korištenje svih resursa $l \in L$ ograničeno je količinom $Q(l)$.
10. Ako su događaji e_1i e_2 u relaciji poretka $\prec_d$ zadnja instanca događaja $e_1$mora završavati d vremenskih jedinica prije početka prve instance događaja e_2.

11. Događaj e mora biti raspoređen unutar maksimalno $span_e$ uzastopnih vremenskih jedinica.

Meka ograničenja su:

1. Student može prisustvovati vježbi samo u terminima kada je slobodan od drugih nastavnih aktivnosti

2. Student može prisustvovati na maksimalno jednoj vježbi u istom terminu

Opisani sustav čvrstih i mekih ograničenja osmišljen je u skladu sa organizacijskim potrebama Fakulteta elektrotehnike i računarstva, ali dovoljno je općenit za opis mnogih problema *LETP* razreda.

Za raspored koji je izvediv ali ne zadovoljava meka ograničenja kažemo da je *konfliktan*. *Konflikt* je vremensko preklapanje trajanja 1 kvant između dvije obaveze studenta. Bolji rasporedi sadžavaju manji broj konflikata.

4 jAgenda projekt

jAgenda projekt pokrenut je u jesen 2007 godine s ciljem oblikovanja automatiziranog sustava za izradu rasporeda laboratorijskih vježbi u skladu sa organizacijskim potrebama FER-2 nastavnog programa. Realizirani sustav napisan je u programskom jeziku Java, a sastoji se od međusobno neovisnih GA i ACO modula, te osim problema rasporeda laboratorijskih vježbi za FER može rješavati različite probleme LETP razreda bilo kojom od te dvije tehnike.

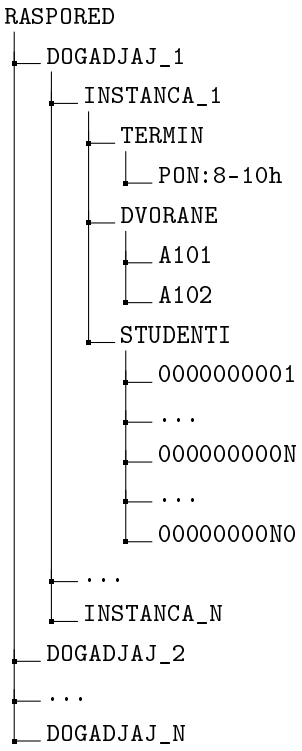
Tipični primjerci problema rasporeda koje aplikacija rjesava sastoje se od približno 2.500 studenata, koje je potrebno rasporediti na do 50 vježbi, u oko 30 prostorija. Ostvarena aplikacija podržava mogućnost uklapanja rasporeda vježbi u prethodno definiran raspored predavanja i međuispita.

jAgenda se uspješno primjenjuje primjenjuje za izradu rasporeda laboratorijskih vježbi na FER-u od zimskog semestra akademske godine 2007/ 2008.

4.1 Raspored kao jedinka populacije

Jedinke populacije su rasporedi generirani u skladu sa zadanim ulaznim čvrstim ograničenjima. Generiranje je nasumično, a mjera dobrote jedinke je broj vremenskih preklapanja u kvantima za studente koji moraju obaviti vježbe. Formalno strukturu rasporeda i prostora rješenja definiramo na sljedeći način:

- *Instanca događaja* $i_{n,e}(t_{n,e}, R_{n,e}, S_{n,e})$ je konkretno održavnje vježbe jednoznačno određeno terminom, dodijeljenim podskupom studenata i dodijeljenim prostorijama.
- *Raspoređeni događaj* $I(e,a) = \{i_{1,e}, .., i_{n,e}\}$je skup instanci događaja u rasporedu a. Raspoređeni događaj je osnovna jedinica nasljeđivanja *(gen)*.
- *Mogući raspored događaja* $I(e)$ označava raspoređeni događaj e u praznom rasporedu.
- *Univerzalni skup instanci* $\mathcal{I}$ je skup svih mogućih $I(e)$ za sve $e \in E$.
- *Raspored* $sch = \{I(e_1, sch), .., I(e_n, sch)\}$ je jedinka populacije i predstavlja uniju raspoređenih događaja. U raspored moraju biti uključene instance svih $e \in E$
- *Prostor rješenja* $\mathcal{SOL}$ je skup svih izvedivih rasporeda.



Slika 1: Hijerarhijska struktura rasporeda

Struktura rasporeda prikazana je na slici 1.

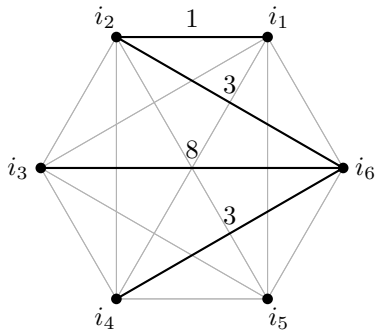
Za raspored a kažemo je u relaciji konflikta $\otimes$ sa genom $I(e)$ ako i samo ako bi umetanje $I(e)$ u raspored a uzrokovalo narušenje čvrstih ograničenja.

4.2 Funkcija kazne

Neka je $I(s, a)$ skup svih instanci $(t_{i,e}, S_{i,e}, R_{i,e})$ u rasporedu a, za koje vrijedi $s \in S_{i,e}$. Za svakog studenta $s \in S$ možemo konstruirati *graf preklapanja obaveza* na sljedeći način:

- svaka instanca $i \in I(s, a)$ je vrh grafa
- težina brida između dva vrha zadana instancama $i_{1,e1}, i_{2,e2} \in I(s, a)$ predstavlja broj kvanata u kojima se vremenski termini dvije instance preklapaju, a računa se po formuli:

$$w(i_{1,e1}, i_{2,e2}) = \max(\min(t_{1,e1} + dur_{e1}, t_{2,e2} + dur_{e2}) - \max(t_{1,e1}, t_{2,e2}), 0)/quant.$$



Slika 2: Primjer grafa preklapanja obaveza. Doprinos funkciji kazne rasporeda je 1+3+3+8=15

- Vrijednost $w(i_{1,e1}, i_{2,e2})$ nazivamo brojem konflikata

Suma težina bridova $confl(s) = \sum_{i=1}^{|I(s,a)|-1} \sum_{j=i+1}^{|I(s,a)|} w(i_i, i_j)$ ovako konstruiranog grafa naziva se *broj konflikata za studenta* s i opisuje koliko je raspored a loš za studenta s. Broj konflikata za sve studenete $s \in S$ jednoliko pridonosi funkciji kazne rasporeda $fitness : \mathcal{SOL} \rightarrow N$.

Kazna se za raspored a računa sumom: $fitness(a) = \sum_{s \in S} confl(s)$. Vrijednost $fitness(a)$ naziva se *broj konflikata u rasporedu* a.

Raspored čija kazna iznos 0 smatra se dovoljno dobrim za upotrebu i pronalazak takvoga zaustavlja izvođenje programa. Primjer grafa preklapanja obaveza dan je na slici 2.

4.3 Operator selekcije

Jedinke se selektiraju K-turnirskom selekcijom. Iz trenutne populacije *P(t)* nasumično se odabire K jedinki, od kojih se najlošija odbacuje, stvarajući tako *"mjesto"* u populaciji za novu jedinku koja će proizvesti operator križanja. (parametar selekcije K može se zadati u konfiguracijskoj datoteci *zagor.cfg*)

```
Selekcija(Populacija p,k)
{
  Lista odabrane_jedinke;
  za(i=1 do k)
  {
     slučajno odaberi jedinku j;
```

```
      dodaj j u odabrane_jedinke;
    }
    izbaci najlošiju jedinku u listi odabrane_jedinke iz populacije;
  }
```

4.4 Operator križanja

Operator križanja $Cross : \mathcal{SOL} \times \mathcal{SOL} \rightarrow \mathcal{SOL}$ pokušava proizvesti jedinku koja sadrži kombinaciju genetskog materijala roditelja a i b. Moguće je da su neki geni roditelja a u relaciji konflikta $\otimes$ sa roditeljem b i obrnuto.

Primjerice: neka je
$I(e_n, a) = \{i_{1e_n}, .., (PON : 9 - 11h, A101, \{00001, ..., 0000N\}), ..i_{ke_n}\}$
$I(e_m, b) = \{i_{1e_m}, .., (PON : 8 - 10h, A101, \{00001, ..., 0000N\}), ..i_{le_m}\}$

- Kako ista prostorija ne smije biti dodijeljena različitim događajima u terminima koji se vremenski preklapaju, potomak ne može naslijediti oba gena $I(e_n, a)$ i $I(e_m, b)$. U ovom slučaju jedan od konfliktnih gena pokušava se nadomjestiti nasumičnom realokacijom.

Operator križanja može se opisati sljedećim pseudokodom:

```
Križaj(raspored a,raspored b)
{
  raspored c = ∅;
  slučajno odaberi E1 ⊂ E;
  E2 = E - E1;
  Za sve e ∈ E1 do
  {
    umetni I(e,a) u raspored c;
  }
  Za sve e∈E2
  {
    ako(c⊗I(e,b))
    {
       slučajno rasporedi I(e);
       ako( I(e)⊗c)
       {
         odbaci c;
```

```
          izlaz("križanje neuspješno");
        }
        inače dodaj I(e) u raspored c;
      }
      inače dodaj I(e,b) u raspored c;
    }
  }
```

4.5 Operator mutacije

Operator mutacije $Mut : \mathcal{SOL} \times N \rightarrow \mathcal{SOL}$ pokušava u rasporedu a ponovno nasumično rasporediti slučajan broj događaja ne veći od nekog cijelog broja $n \in [1, .., razinaMutacije]$. Ukoliko operator mutacije ne uspije preseliti događaje u nove instance ne narušavajući pri tome čvrsta ograničenja, mutacija se poništava i jedinka se nepromijenjena vraća u populaciju.

```
Mutacija(raspored a,razina_mutacije)
  n =slučajna vrijednost u intervalu [1, razina_mutacije];
  za i=1 do n
  {
    slučajno odaberi gen I(e,a);
    obriši I(e,a) iz rasporeda a;
    generiraj slučajno gen I_new(e,a);
    ako(I_new(e,a)⊗a)
    {
       odbaci promjene jedinke a;
       izlaz("mutacija neuspješna")
    }
    inače dodaj I_new(e,a) u a;
  }
```

4.5.1 Razina mutacije

Razina mutacije (maxMutLevel) je adaptivni parametar čija vrijednost se povećava kada dođe do stagnacije u populaciji. Broj generacija u kojima populacija mora stagnirati da bi došlo do povećanja razine mutacije naziva se *prag stagnacije*. U trenutku kada stagnacijski brojač dosegne vrijednost praga stagnacije on se re-

setira, a parametri razina_mutacije i prag_stagnacije adaptiraju se na sljedeći način

```
prag_stagnacije=1.5 * prag_stagnacije;
razina_mutacije=razina_mutacije + 1;
```

Adaptivni parametri postavljaju se na inicijalne vrijednosti zadane pri pokretanju programa čim dođe do napretka populacije.

4.6 Operator lokalne pretrage

Neka je događaju$e$ sa $|S_e|$ upisanih studenata u rasporedu a raspoređen u n instanci $i_{1,e}, .., i_{n,e}$, gdje je kapacitet instance $cap(i_{1,e}) = \sum_{r \in R_{ie}} size_r$. Tada je $|S_e|$ upisanih studenata moguće rasporediti na instance $i_{1,e}, .., i_{n,e}$ na $\frac{|S_e|}{\prod_{i \in I(e,a)} cap(i)!}$ načina.

Npr:

- 200 studenata moguće je rasporediti na 5 instanci kapaciteta 40 na $\frac{200}{40!*40!*40!*40!*40!} \approxeq 2.18 * 10^{135}$ načina.

Optimalni razmještaj studenata po instancama $I(e, a)$ je onaj koji minimizira broj konflikta u rasporedu a. Iz primjera je očito da je pronalazak optimalnog razmještaj studenta po instancama podproblem $LETP - a$ koji se ne može riješiti iscrpnom pretragom.

Minimalna moguća kazna rasporeda a je ona vrijednost koju kazna rasporeda a poprima za optimalan razmještaj studenata po instancama.

Operator lokalne pretrage $Ls : \mathcal{SOL} \times N \rightarrow \mathcal{SOL}$ pokušava ciljanim premještanjem n studenata koji imaju konflikte u raspredu u instancu u kojoj neće imati konflikte poboljšati trenutni razmještaj skupa studenata S_e a time i dobrotu jedinke.

Funkcija $dems : E \rightarrow 2^S$ pridružuje događaju $e \in E$ skup studenata koji obavljaju demonstratorsku dužnost.

Algoritam lokalne pretrage dan je sljedećim pseudokodom:

```
lokalna pretraga(Raspored a, broj_zamjena n)
{
   ako u rasporedu nema konflikta izlaz;
   za (j = 0; j < n; j++)
   {
     student s1 = slučajno odaberi studneta koji ima preklapanje;
```

```
događaj e1 = slučajno odaberi događaj u kojem student ima koflikt;
instanca i1 = instanca u kojoj student ima konflikt;

događaj e2 = null;
instanca i2 = null;
dodaj e1 u listu potencijalni_događaji;
student_je_demos=false;
ako(s1 ∈ dems(e1))
{
  student_je_demos=true;
  dodaj sve ostale kategorije događaja e1 u skup potencijalni_događaji;
}
odgovarajuci_termin_nadjen = false;

Za sve(tmp_e2 ∈ potencijalni_događaji)
{

   slucajno permutiraj skup instanci I(tmp_e2);
   Za sve (i ∈ I(tmp_e2,a) )
   {
     ako(s1 ∉ S_{i,tmp_e2})
     {
         e2 = tmp_e2;
         ako(student je student slobodan u terminu t_{i,tmp_e2})
         {
            odgovarajuci_termin_nadjen = true;
            i2 = i;
            prekini vanjsku petlju;
         }
      }
   }
  ako(odgovarajuci_termin_nadjen) prekini vanjsku petlju;
}
ako(e2=null) nastavi vanjsku petlju; //studenta se nema kuda seliti
ako(i2=null) i2= slucajna instanca e2;

ako( (sve sobe i2 su pune) ili (s1 ∈ dems(e1)) )
```

```
        {
          /*u ovom slucaju student koji se seli u drugu instancu mora pronaci
            zamjenu jer inace doci do narusenja cvrstih ogranicenja*/
          s2 = null
          ako(s1 ∈ dems(e1))
          {
            zamjenski_demosi = dems(e1,i2) \ ( dems(e1,i1) ∩ dems(e1,i2) );
            ako (zamjenski_demosi == ∅) nastavi vanjsku petlju;
            s2 = slucajni student iz skupa zamjenski_demosi;
          }
          inace
            s2 = slucajni student iz  S_{i2,e2};
          preseli studenta s2 iz S_{i2,e2} u S_{i1,e1};
        }
      preseli studenta s1 iz S_{i1,e1} u S_{i2,e2};
    }
```

Pojednostavljeno rečeno, lokalna pretraga djeluje kao virtualna burza grupa koja pokušava što bolje razmjestiti studente po instancama vježbi poštujući pri tome sljedeća pravila:

- ukoliko su sve prostorije u odredišnoj instanci pune, student se mijenja za instancu sa slučajno odabranim kolegom.

- ukoliko ima mjesta u nekoj od prostorija odredišne instance, student se seli bez zamjene.

- demonstrator ne može zamjeniti studenta.

- ukoliko u instanci iz koje demonstrator odlazi ima minimalan broj demonstratora, on mora pronaći zamjenu

- demonstratora može zamjeniti jedino drugi demonstrator.

- demonstratora se nikada ne smije preseliti u instancu u kojoj već obavlja demonstratorsku dužnost.

Parametar *broj_zamjena* određuje koliko će studenata sa preklapanjima u rasporedu operator lokalne pretrage pokušati premjestiti u za njih bolje termine. Operator lokalne pretrage ne garantira poboljšanje jedinke pri svakoj iteraciji; štoviše moguće je čak i privremeno pogoršanje u nekim situacijama.

Npr:

- Student a ima vježbu A u terminu PON:8-11h i vježbu B u terminu PON:10-12h. $\Rightarrow$ ukupan konflikt studenta a iznosi 4 kvanta

- Student b ima vježbu A u terminu UT:8-11h i vježbu B u terminu PON:8-10h $\Rightarrow$ ukupan konflikt studenta b iznosi 0 kvanata

  $\Longrightarrow$ukupno preklapanja u rasporedu ima 4

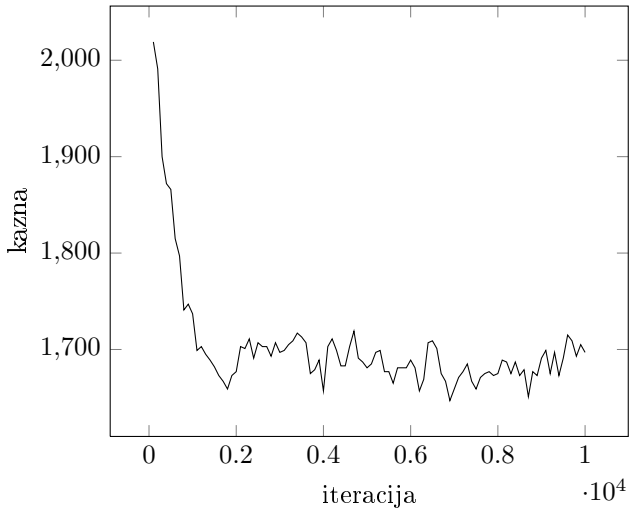
- Lokalna pretraga mijenja studenta a za studenta b, što rezultira sljedećom situacijom:

- Student a ima vježbu A u terminu UT:8-11h i vježbu B u PON:10-12h $\Rightarrow$ukupan konflikt studenta a = 0 kvanata

- Student b ima vježbu A u terminu PON:8-11h i vježbu B u PON:8-10h $\Rightarrow$ukupan konflikt studenta b = 8 kvanata

  $\Longrightarrow$ukupno preklapanja u rasporedu sada ima 8

Kvalitativno ponašanje funkcije kazne jedinke tijekom lokalne pretrage prikazano je slikom 3 . Ukoliko tijekom *i-te* iteracije lokalne pretrage kazna jedinke poprimi vrijednost 0 (tj. pronađeno je traženo rješenje) , rješenje se zapisuje na disk te se prekida daljnji rad sustava, čime je osigurano da oscilacija kazne tijekom lokalne pretrage neće uzrokovati "zaobilaženje" optimalnog rješenja.



Slika 3: Ponašanje kazne jedinke tijekom 10000 iteracija lokalne pretrage

4.7 Generiranje početne populacije

Slučajne jedinke stvaraju se uz poštivanje čvrstih ograničenja. Svakoj instanci vježbe dodjeljuje se skup studenata, termin, i skup dvorana. Generiranje je

nasumično i nije uvijek uspješno, a izvodi se u dva koraka:

1. Generiranje praznog rasporeda, pri kojemu se rezervira dovoljno prostorija i termina za smještaj studenata. (Za rezervirane instance skup dodijeljenih studenata ostaje prazan.)

2. U prazne instance događaja u rasporedu slučajnim odabirom dodaju se studenti i demonstratori.

Algoritam za rezervaciju potrebnog broja dvorana, termina i resursa može se opisati sljedećim pseudokodom:

```
setTermsAndRooms()
E = lista događaja koje treba rasporediti;
n_R =broj_koliko_resursa_se_koristi_u_rasporedu;
n_kvant = broj kvanata u radnom danu;
Resource_pool[n_R][n_kvant] = inicijalino su dostupni svi resursi u svim kvantima;
dok( E nije prazna )
{
    slučajno odaberi iz E događaj e;
    izbaci e iz E;
    D_e = lista dozvoljenih dana za e;
    N_S_e = broj studenata upisnih na dogadjaj e; //ne racunajuci demonstratore
    dok(D_e nije prazan && (N_S_e > 0))
    {
        d_e = slučajni dan iz D_e;
        TS_d_e = skup dozvoljenih termina u danu d za događaj e;
        R_e = lista prostorija prikladnih za događaj e;
        dok(TS_d_e nije prazan && (N_S_e > 0))
        {

            ts_d_e = slučajni termin iz TS_d_e;
            R_i=0;
            S_i=0;
            i= nova instanca (ts_d_e,R_i,S_i)
            za(svaku prostoriju r iz liste R)
            {
```

```
                ako
                (soba slobodna za ts_d_e && zadovoljava ograničenja)
                {
                dodaj r u Ri;
                spw_e = broj studenata po radnom mjestu za događaj e;
                size_r = veličina sobe r;
                rezerviraj min(spw_e*size_r , N_s_e)
                potrebnih resursa u terminu ts_d_e;
                N_S_e = max (N_S_e -(size_r * spw_e) , 0);
                }
              }
            }
        }
        ako (N_S_e > 0)
        {
            //generiranje neuspjesno
            Povećaj prioritet događaja e;
            Odbaci raspored;
        }
    }
```

U slučaju neuspješnog generiranja događaj koji je uzrokovao odbacivanje rasporeda dobiva u idućem pokušaju generiranja veći prioritet. Događaji sa višim prioritetom imaju veću vjerojatnost da će biti ranije izabrani pri raspoređivanju.

Uspješnost generiranja jedinke ovisi o zadanim ograničenjima i varira od 5 - 95% . Nemogućnost generiranja jedinke može ukazivati na to da je možda zadan kontradiktoran skup ograničenja.

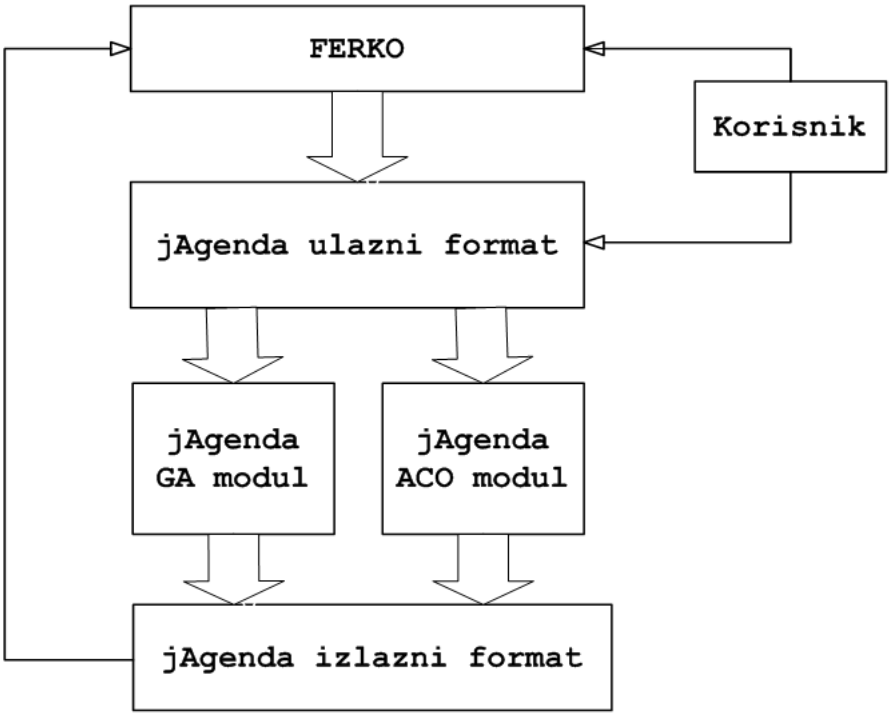
Npr:

- događaj sa trajanjem 2h i 100 upisanih studenata, sa 3 prikladne prostorije kapaciteta 30 studenata, ima postavljeno ograničenje maksimalnog trajanja vježbe na 2h.

jAgenda sustav nema implementirane mehanizme za provjeru kontradiktornosti skupa ograničenja, zbog čega pri ručnom unosu ulaznih podataka korisnik treba biti vrlo pažljiv.

4.8 Ulazni podaci sustava

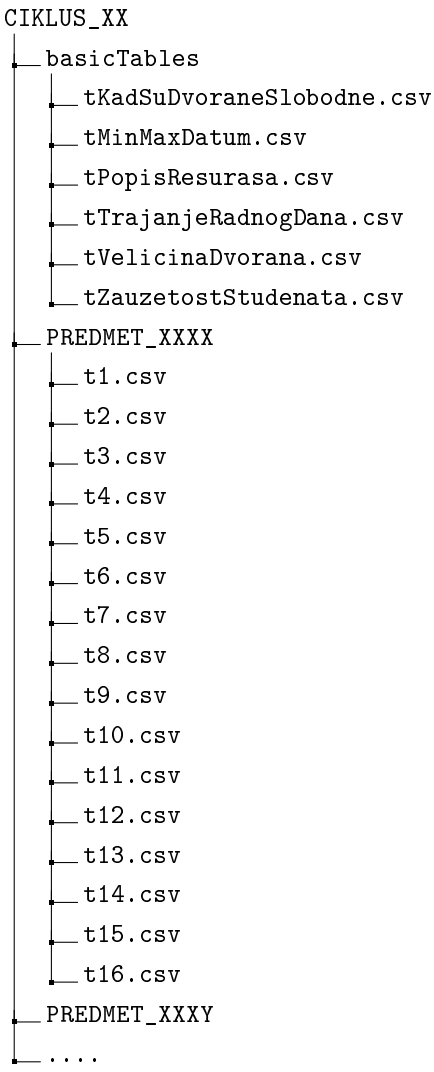
Ulazni podaci jAgenda sustava su **.csv* datoteke kojima se specificira konkretna instanca *LETP* problema koji sustav treba rijesiti. U ranim fazama razvoja jAgenda sustava ulazni podaci zadavali su se rucnim unosom 16 datoteka po kolegiju. Od prosle akademske godine asistenti ogranicenja i zahtjeve vjezbi zadaju posebnim modulom sustava Ferko koji je otporniji na implicitne pogreske uzrokovane zadavanjem nemogucih ogranicenja. Podaci se potom iz baze podataka ferko sustava izvoze u ulazni format jAgenda sustava (ilustrirano na slici 4). Premda se podaci jos uvijek mogu zadavati rucno, ovaj nacin rada ne preporucuje se.



Slika 4: Ulazno/izlazni tok podataka

Korijenski direktorij iz kojega će sustav pročitati ulazne podatke može se zadati pri pokretanju *jAgende,* ali sami ulazni podaci moraju biti organizirani u skladu sa predviđenom direktorijskom strukturom ulaznih podataka.

Ograničenja zadana ulaznim datotekama sustav smatra čvrstim i proizvedeni rasporedi uvijek će biti u skladu s njima.



4.8.1 Ograničenja na razini rasporeda

Direktorij *basicTables* sadrži osnovna ograničenja na razini cijelog rasporeda. Ograničenja definirana u datotekama ovog direktorija vrijede za sve predmete koji se raspoređuju.

- *Ograničenje dostupnosti dvorana* - općenito prostorije u kojima se održavaju vježbe nisu slobodne u svim terminima perioda raspoređivanja. Zbog toga je potrebno ulaznom datotekom *tKadSuDvoraneSlobodne.csv* navesti za svaku prostoriju popis termina kada je ona raspoloživa. Primjer sadržaja

datoteke dan je na slici .

- *Ograničenje perioda raspoređivanja* - određuje period za koji se raspored izrađuje. Zadaje se ulaznom datotekom *tMinMaxDatum.csv*
- *Ograničenje trajanja radnog dana* - zadano je datotekom *tTrajanjeRadnogDana.csv*. Pretpostavlja se je radno vrijeme jednako za sve dane perioda raspoređivanja. Raspoređivanje je u svakom danu perioda, ograničeno na termine između prvog i zadnjeg prihvatljivog vremenskog kvanta.
  - Trajanje vremenskog kvanta je 15minuta, a prvim kvantom dana smatra se vremenski period 00:00 - 00:15.
- *Ograničenje veličine dvorana* - zadano je datotekom *tVelicinaDvorana.csv*. Datoteka sadrži *pretpostavljene* kapacitete prostorija. Ukoliko je potrebno, kapacitet prostorije moguće je preopteretiti u skladu s potrebama vježbe.
- *Ograničenje zauzetosti studenata* - na FER-u se laboratorijske vježbe tipično održavaju paralelno sa nastavom pa studenti nisu slobodni u svim terminima perioda raspoređivanja. Zauzetost studenata drugim nastavnim aktivnostima zadana je datotekom *tZauzetostStudenata.csv*
- *Ograničenje dostupnog broja resursa* - zadano je datotekom *tPopisResurasa.csv*. Pod pojmom resurs općenito se može smatrati bilo koje nastavno pomagalo koje je studentima potrebno za obavljanje vježbe. Paralelno raspoređivanje vježbi ograničeno je dostupnim brojem zajedničkih nastavnih pomagala.
  - Primjerice, jedan od resursa koji koriste vježbe na FER-u su licence komercijalnog software-a (MatLab,NESTO, itd), kojih fakultet posjeduje točno određeni broj. Licenca nije vezana uz konkretno računalo što znači da kupljeni komercijalni software može biti instaliran na proizvoljnom broju fakultetskih računala, ali je nemoguće istovremeno pokrenuti više instanci software-a nego što je dostupno licenci. Zbog ovog ograničenja sustav mora osigurati da se ne rasporedi paralelno više studenata na vježbe koje koriste isti resurs, nego što je tih resursa dostupno.

```
room;date;from;to;
A001;2009-09-28;08:00;20:00;
A001;2009-10-01;08:00;20:00;
A002;2009-09-28;08:00;20:00;
A002;2009-09-29;08:00;20:00;
A002;2009-09-30;08:00;20:00;
A002;2009-10-01;08:00;11:00;
A002;2009-10-01;14:00;20:00;
A002;2009-10-02;15:00;18:00;
```

Slika 5: Format i primjer sadržaja datoteke *tKadSuDvoraneSlobodne.csv*

```
min;max;
2009-09-28;2009-10-02
```

Slika 6: Format i primjer sadržaja datoteke *tMinMaxDatum.csv*

```
pocetak;kraj
32;80
```

Slika 7: Format i primjer sadržaja datoteke *tTrajanjeRadnogDana.csv*

```
ime;vel
A001;12
A002;1
A101;30
A102;30
```

Slika 8: Format i primjer sadržaja datoteke *tVelicinaDvorana.csv*

```
id;date;from;to;additional
0000000001;2009-09-28;11:00;13:00;1|Elektronička mjerenja i komponente
0000000001;2009-09-29;08:00;11:00;1|Analogna i mješovita obrada signala
0000000001;2009-09-29;18:00;20:00;1|Organizacijska psihologija
0000000001;2009-09-30;08:00;11:00;1|Mikrovalna elektronika
```

Slika 9: Format i primjer sadržaja datoteke *tZauzetostStudenata.csv*

4.8.2 Ograničenja specifična za vježbu

Ulazni podaci specifični za svaku vježbu koju treba rasporediti zadaju se u odvojenim direktorijima. Ulaznim datotekama precizno se definiraju organizacijske zahtjevi pojedine vježbe.

Datoteka	Format datoteke
t1.csv	"SIF";"OZN_LAB";"JMBAG";"KATEGORIJA"
t2.csv	"SIF";"OZN_LAB";"OZN_PROSTORIJE"
t3.csv	"SIF";"OZN_LAB";"KATEGORIJA";"OZN_PROSTORIJE";"MAX_BR_MJESTA"
t4.csv	"SIF";"OZN_LAB";"BR_STUD_PO_RADNOM_MJESTU";"TRAJANJE_VJEZBE"
t5.csv	"SIF";"OZN_LAB";"PRIHVATLJIVI_DATUM"
t6.csv	"SIF";"PREDUVJET";"VJEZBA";"MIN_RAZMAK"
t7.csv	"SIF";"OZN_LAB";"OGR_TRAJANJA";"TIP_OGR"
t8.csv	"SIF";"OZN_LAB";"MAX_SOBA_ISTOVREMENO"
t9.csv	"SIF";"OZN_LAB_VJEZBE";"MAX_ASISTENATA_ISTOVREMENO"
t10.csv	"SIF";"OZN_LAB";"OZN_PROSTORIJE";"POTROSNJA_ASISTENATA"
t11.csv	"SIF";"OZN_LAB";"MAX_MJESTA_ISTOVREMENO";"MIN_MJESTA"
t12.csv	"SIF";"OZN_LAB";"OZN_PROSTORIJE";"BR_MJESTA";"BR_REZ_MJ"
t13.csv	"SIF";"OZN_LAB";"KORISTENI_RESURS"
t14.csv	"SIF";"OZN_LAB";"JMBAG_DEMOSA"
t15.csv	"SIF";"OZN_LAB";"BROJ_ODRADA_DEMOSA"
t16.csv	"SIF";"OZN_LAB";"OZN_PROSTORIJE";"BR_DEMOSA_PO_PROSTORIJI"

Tablica 1: Pregled formata ulaznih datoteka za specifikaciju ograničenja vježbi

Dodatna pojašnjenja:

- *t1.csv* - sadrži podatke o studentima upisanima na kolegij i vježbama koje isti moraju odraditi . Moguće je da postoji više varijanti iste laboratorijske vježbe kao što je npr. slučaj kada su studentima dodijeljeni različiti zadaci za vježbu na temelju matičnog broja. Ovakva situacija opisuje se poljem *KATEGORIJA*, u koje se unosi varijanta laboratorijske vježbe koju student mora odraditi. Ako vježba ima samo jednu varijantu svi studenti u datoteci imati će u ovom polju upisanu istu vrijednost.
- *t2.csv* - sadrži podatke o prihvatljivim prostorijama za održavanje vježbi.
- *t3.csv* - tablica omogućava preopterećenje pretpostavljenih kapaciteta soba navedenih u datoteci *tVelicinaDvorana.csv* za potrebe pojedine vježbe.
- *t4.csv* - datoteka kojom se definira trajanje vježbi i broj studenata po radnom mjestu.

- *t5.csv* - datoteka kojome se definiraju prihvatljivi datumi održavanja za pojedine vježbe. Definiranje prihvatljivih termina unutar dana nije podržano.

- *t6.csv* - datotekom se definira se relacija poretka vježbi. Ukoliko je npr. unutar istog ciklusa raspoređivanja potrebno rasporediti dvije vježbe istog kolegija, a uz to je odrađena *vježba_1* preduvjet za održavanje *vježbe_2* sustav će generirati rasporede u kojima zadnja raspoređena instanca *vježbe_1* uvijek završava minimalno MIN_RAZMAK prije početka prve raspoređene instance *vježbe_2.*

- *t7.csv* - datotekom se definira se ograničenje maksimalni vremenski raspon unutar kojega moraju biti raspoređene sve instance vježbe. Primjer situacije u kojoj se ova ograničenja koriste su laboratorijske vježbe koje uključuju provjere znanja, za koje je potrebno da ih svi studenti obave paralelno. Uz postavljeno ograničenje maksimalnog trajanja vježbe, sustav će garantirano sve instance laboratorijske vježbe rasporediti u nekom od prihvatljivih dana vježbe definiranih tablicom *t5*.csv, tako da za vremena početka prve raspoređene instance t_1, završetka zadnje raspoređene instance t_2 , vrijedi $t_2 - t_1 \leq OGR\_TRAJANJA$

- *t8.csv* - datotekom se definira se koliko najviše prostorija može biti dodijeljeno svakoj instanci vježbe pri raspoređivanju.

- *t9.csv* - datotekom se definira se koliko je nastavnog osoblja raspoloživo na kolegiju, čime se osigurava da sustav nikada instancama vježbi ne dodijeli paralelno prostorije za koje treba više asistenata nego što ih je raspoloživo.

- *t10.csv* , *t11.csv* , *t12.csv* - datoteke su zadržane radi kompatibilnosti sa starijim verzijama sustava, popunjavanje se ne preporuča.

- *t13.csv* - definira koja resurse (nastavna pomagala) vježba koristi.

- *t14.csv* - sadrži popis studenata koji obaljvaju dužnost demonstratora na pojedinim vježbama. Student koji obavlja dužnost demonstratora ne zauzima radno mjesto u laboratoriju.

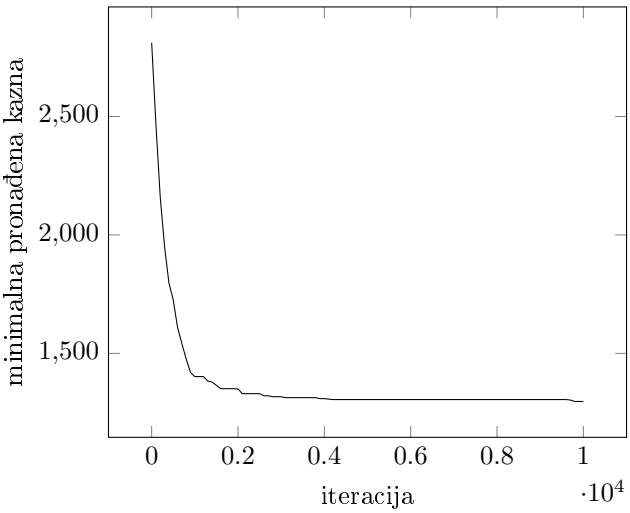
- *t15.csv* - definira na koliko instanci vježbe svaki student demonstrator mora pristustvovati.

- *t16.csv* - definira potrošnju demosa po prostorijama.

5 Analiza učinkovitosti sustava

5.1 Utjecaj operatora lokalne pretrage na performanse sustava

Operator lokalne pretrag ciljanom pretragom okoliša jedinke pokušava pronaći bolje termine za vježbe studentima sa konfliktima. Parametar lokalne pretrage (*broj_zamjena*) je broj pokušaja realokacije studenata sa problemima u rasporedu u drugu instancu koja im bolje odgovara. Pokušaj realokacije studenta u svakoj od iteracija lokalne pretrage može ali ne mora uspjeti.
Slikom 10 prikazano je ponašanje najmanje vrijednosti kazne poprimljene tijekom lokalne pretrage za vrlo velik broj iteracija. Primjer sugerira da nakon nekog broja iteracija korisnost operatora lokalne pretrage drastično opada, što je očekivano jer je potencijalno smanjenje kazne jedinke primjenom operatora lokalne pretrage ograničeno *minimalnom mogućom kaznom* te jedinke. Zbog toga je pri izboru vrijednosti parametra operatora lokalne pretrage *broj_zamjena* vrlo važno procijeniti prikladnu vrijednost istog tako da se ne troši uzalud procesorsko vrijeme, a da s druge strane sustav ne "promaši" optimalno rješenje jer okolina jedinke nije dovoljno temeljito pretražena.



Slika 10: Minimalna vrijednost kazne koju je jedinka poprimila tijekom do uključivo *i-te* iteracije lokalne pretrage

Ispitivanje učinkovitosti sustava za različite vrijednosti parametra *broj_zamjena*

provedeno je na 10 primjeraka problema rasporeda uz postavke sustava dane tablicom 2. Za svaku od ispitivanih vrijednosti parametra *broj_ zamjena* izvršeno je 10 pokretanja sustava za svaki od primjera problema rasporeda. Rezultati su dani u tablicama 3,6,4.

parametar sustava	vrijednost
veličina populacije	30
selekcijski algoritam	10-turnirska selekcija
vjerojatnost mutacije	30%
inicijalni prag stagnacije	100
maksimalna razina mutacije	10
kriterij zaustavljanja	100000 iteracija

Tablica 2: Postavke sustava pri testiranju učinkovitosti sustava za različitu vrijednost parametra *broj_ zamjena*

br_z\ problem	1	2	3	4	5	6	7	8	9	10
0	440	8421	1434	1349	899	8493	1071	374	1982	7129
10	64	425	28	47	216	418	58	38	167	209
20	50	440	30	0	206	371	42	24	123	181
30	44	431	8	2	86	334	6	16	59	143
40	40	188	4	0	72	334	4	16	52	57
50	40	111	14	0	122	294	4	16	55	104
60	44	272	6	0	75	332	4	12	34	88
70	44	271	16	0	54	365	4	12	47	72
80	44	215	4	0	28	140	4	12	30	37
90	44	223	14	0	36	34	4	12	24	57
100	40	125	6	0	40	64	4	16	18	43

Tablica 3: Minimalna kazna jedinke u 10 pokretanja uz vrijednost parametra lokalne pretrage

	1	2	3	4	5	6	7	8	9	10
0	464	8696,9	1501,1	1458,9	963	8657,3	1142,5	384,2	2085,5	7281
10	101,6	736,9	30	135	319,6	618,2	230,9	61,1	214,4	306,3
20	78,6	899,1	43,5	60,1	244,7	634,7	54,8	64,8	280,7	348
30	66,6	649,1	52,5	10,1	206,4	525	55,2	21,8	109,1	196,1
40	52,9	442,8	17,4	9,2	135,4	427,1	33,4	20,7	72,7	159,6
50	45,6	526,8	28,9	4,2	171,1	334,7	10,5	21,7	147,6	186,1
60	57	498,1	11	3,2	128,5	438,7	20,4	14,7	50,1	125,2
70	49,4	380,2	27,5	8,6	88,6	439,2	4	16,2	95,8	110,9
80	45,5	334,5	12,4	5,3	40,3	203,9	5	15,6	38,1	74
90	44,8	425,6	19,7	4,6	46,6	109,9	4	16,1	49,2	62,2
100	40,9	310,9	17,2	3	42,4	155,6	4,8	18,2	34	50,2

Tablica 6: Prosječna vrijednost kazne u 10 pokretanja

br_z\ problem	1	2	3	4	5	6	7	8	9	10
0	494	9090	1592	1580	1034	8842	1206	400	2212	7403
10	140	1014	33	199	428	1018	396	84	314	376
20	100	1284	62	126	287	884	78	122	436	508
30	102	831	169	16	284	743	112	28	183	289
40	76	593	46	20	195	521	64	28	87	279
50	52	871	52	8	201	375	28	26	253	256
60	68	719	18	10	182	508	48	18	63	167
70	58	429	42	14	139	542	4	22	193	156
80	48	516	22	12	48	257	6	20	48	91
90	46	630	30	12	54	186	4	20	76	71
100	44	515	28	12	44	331	6	20	56	60

Tablica 4: Maksimalna kazna jedinke u 10 pokretanja uz vrijednost parametra lokalne pretrage

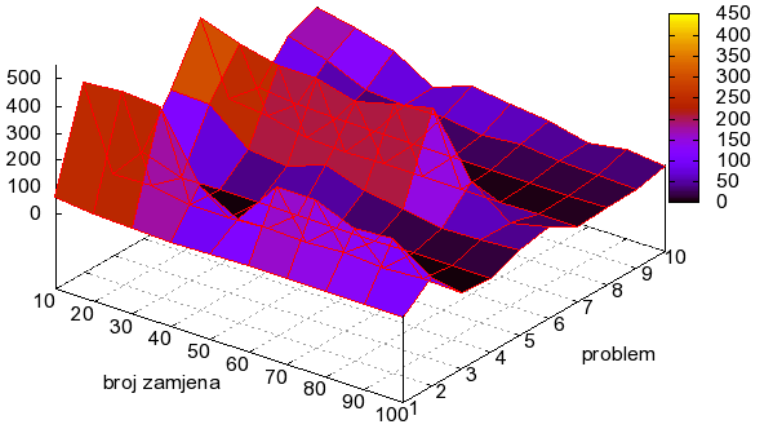
Iz podataka dobivenih obavljenim simulacijama mogu se zaključiti sljedeće stvari o ponašanju sutava:

- Ako isključimo operator lokalne pretrage (*broj_ zamjena=0*) konvergencija populacije prema optimalnom rješenju je znatno sporija.
- Pretjerano velike vrijednosti parametra lokalne pretrage samo usporavaju sustav. Primjer trajanja izvođenja 100000 iteracija na računalnoj jezgri frekvencije 3.33 GHz i sa 2GB RAM-a dodijeljenih procesu dan je u tablici 7.
- U praktičnoj primjeni dobar izbor za parametar lokalne pretrage su vrijednosti između 20 i 50.

parametar	0	10	20	30	40	50
trajanje	00:08:59	00:09:13	00:09:43	00:10:24	00:10:15	00:11:14

parametar	60	70	80	90	100
trajanje	00:11:52	00:12:24	00:12:54	00:13:41	00:14:00

Tablica 7: Primjer ovisnosti trajanja izvođenja 100000 iteracija u ovisnosti o parametru *broj_zamjena*

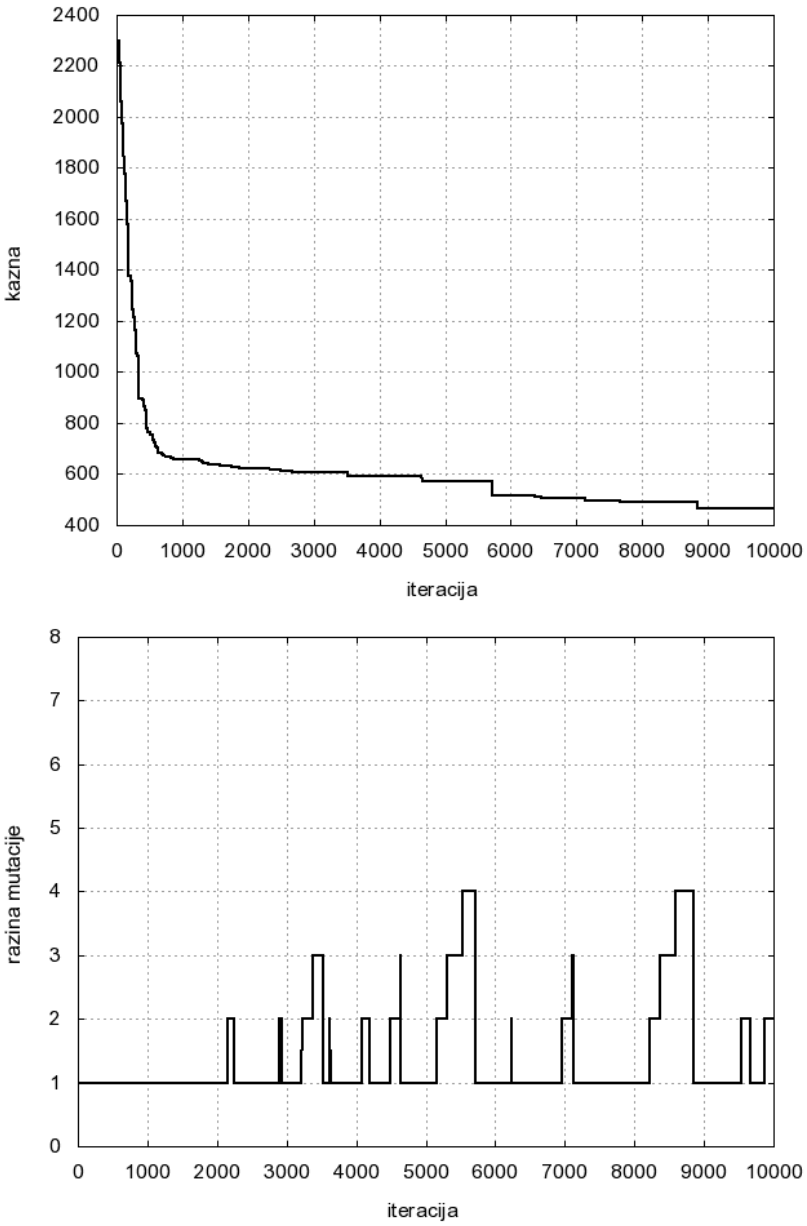


Slika 11: Ovisnost kazne o parametru broj zamjena

5.2 Utjecaj adaptivnih parametara mutacije

Adaptivni parametar *razina_mutacije (maxMutLevel)* sustava *jAgenda* povećava agresivnost operatora mutacije ukoliko se ustanovi da je u populaciji došlo do stagnacije (tj. kazna najbolje jedinke ni prosječna kazna populacije kroz određeni broj iteracija ne opadaju). Do stagnacije dolazi ukoliko je genetski materijal jedinki u populaciji sličan te su zbog toga sve jedinke u blizini nekog lokalnog optimuma. Ukoliko takav lokalni optimum nije ujedno i globalni povećanjem maksimalne razine mutacije pokušava se u populaciju unijeti novi genetski materijal koji će algoritmu omogućiti bijeg iz lokalnog optimuma. Primjer ponašanja minimalne kazne u populaciji i razine mutacije tijekom 10000 iteracija dan je slikom 12.

Sa slike je očito da je u trenucima pada kazne razina mutacije često povišena u odnosu na inicijalnu vrijednost.



Slika 12: Vrijednosti kazne i razine mutacije

Ispitivanje utjecaja parametra *maxMutLevel (maksimalna razina mutacije)* na učinkovitost sustava za različite vrijednosti provedeno je na 10 primjeraka problema rasporeda uz postavke sustava dane tablicom 8 . Za svaku od ispitivanih vrijednosti parametra *maxMutLevel* izvršeno je 10 pokretanja sustava za svaki od primjera problema rasporeda. Rezultati su dani u tablicama 10,9,11.

parametar sustava	vrijednost
veličina populacije	30
selekcijski algoritam	10-turnirska selekcija
vjerojatnost mutacije	30%
inicijalni prag stagnacije	100
broj_zamjena	30
kriterij zaustavljanja	100000 iteracija

Tablica 8: Postavke sustava pri ispitivanju učinkovitosti sustava za različitu vrijednost parametra *maxMutLevel*

maxMut\problem	1	2	3	4	5	6	7	8	9	10
1	448	744	466	244	885	667	376	294	457	338
2	388	934	472	282	843	582	364	322	581	267
3	386	702	492	219	890	611	402	270	420	285
4	68	533	8	2	202	609	62	16	91	118
5	40	524	18	0	74	430	6	12	99	70

Tablica 9: Minimalna kazna jedinke u 10 pokretanja uz vrijednost parametra maksimalna razina mutacije

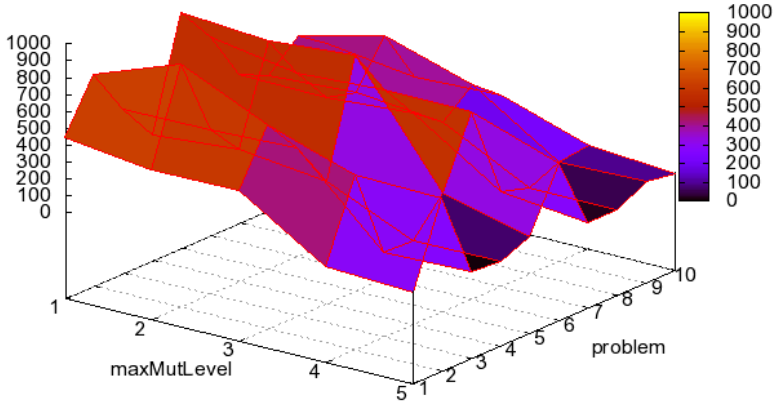
maxMut\problem	1	2	3	4	5	6	7	8	9	10
1	488,4	1267,6	525,4	306,8	1042,1	831,8	537,7	325,1	628,4	443,4
2	462,9	1112,9	555,9	402,5	1005,4	798,9	435,9	359,9	733,6	431,5
3	435,6	1061,7	550,7	308,6	1010,5	761	455,8	306,6	615,3	390,2
4	90,6	969,9	31,2	58,9	296,6	740,7	86,3	25,7	165,2	265,9
5	55	816,6	27,7	12,9	238,5	658,2	72	17,5	133,6	195,8

Tablica 10: Prosječna kazna jedinke u 10 pokretanja uz vrijednost parametra maksimalna razina mutacije

maxMut\problem	1	2	3	4	5	6	7	8	9	10
1	524	2095	595	398	1277	1149	868	366	793	531
2	516	1318	691	607	1199	1110	505	396	877	604
3	492	2038	611	477	1085	970	546	350	727	533
4	118	1617	48	144	463	1013	118	42	214	405
5	68	1064	42	32	312	1009	138	24	170	287

Tablica 11: Maksimalna kazna jedinke u 10 pokretanja uz vrijednost parametra maksimalna razina mutacije

Iz rezultata se može zaključiti da u skladu sa očekivanjem povećanje maksimalne razine mutacije pomaže sustavu da izađe iz lokalnog optimuma, čime se ubrzava konvergencija. Ponašanje sustava za različite vrijednosti parametra *maxMutLevel* ilustrirano je slikom 13.



Slika 13: Ovisnost kazne o maksimalnoj razini mutacije

6 Zaključak

Problem rasporeda u visokoškolskim obrazovnim ustanovam u literaturi je poznat pod imenom *UCTP (University course timetabling problem)*. U sklopu ovog diplomskog rada kao njegov podrazred formalno se definira problem rasporeda laboratorijskih vježbi *LETP (laboratory exercise timetabling problem)*. Na Fakultetu elektrotehnike i računarstva 2007. godine pokrenut je *jAgenda* projekt kojemu je bio cilj izrada sustava za automatizirano rješavanje *LETP-a* na fakultetu. Realizirani istoimeni sustav zasnovan na metodama prirodom inspiriranog računarstva ostvaren je u programskom jeziku Java i generira rasporede laboratorijskih vježbi dovoljno dobre za upotrebu u praksi.

Podsustav zasnovan na genetskom algoritmu uključuje operator lokalne pretrage bez kojega je sustav u praksi neupotrebljiv. U GA-podsustav je također uključen modul za detekciju stagnacije koji povećanjem agresivnosti operatora mutacije omogućava bijeg iz lokalnog optimuma te time znatno ubrzava konvergenciju GA.

Iz činjenice da se *jAgenda* sustav uspješno se primjenjuje na Fakultetu elektrotehnike i računarstva od 2008. godine može se zaključiti da je hibridni genetski algoritam primjenjiv na problem rasporeda u visokoškolskim obrazovnim ustanovama ukoliko se uzmu u obzir njihovi specifični organizacijski zahtjevi.

7 Prilozi

7.1 Pokretanje modula za automatsko logiranje statističkih podataka

U sklopu ovog rada ostvaren je i modul za automatsko logiranje statističkih podataka. Izvršna *stats_config.jar* datoteka nalazi se u *bin/* direktoriju na priloženom CD mediju. Simulacije se pokreću iz konzolnog na sljedeći način:

```
java -jar stats_config.jar
\n <rasporedi_lokacija> <izlazna_datoteka>
\n <konfiguracijska_datoteka>
```

Parametarom *rasporedi_lokacija* zadaje se direktorij u kojemu su ulazni podaci, parametrom *izlazna_datoteka* željena izlazna datoteka, a parametrom *konfiguracijska_datoteka* postavke programa. Primjer sadržaja konfiguracijske datoteke je sljedeći:

```
mode=1 //lokalna pretraga=1, utjecaj maksimalne razine mutacije=2
num_ite=100000 //uvjet
num_runs=1  //broj pokretanja svakog problema za svaku vrijednost parametra
max_param=100 //maksimalna vrijednost ispitivanog parametra
min_param=0  //minimalna vrijednost ispitivanog parametra
param_step=10  //korak povećavanja parametra
```

Program u izlaznu datoteku upisuje zapise oblika:

```
...
1 0 486
1 0 448
1 0 450
1 10 64
1 10 84
...
```

Prvi stupac označava koja instanca problema rasporeda je rješavana, drugi koja je bila vrijednost ispitnog parametra, a treći minimalnu kaznu u populaciji na kraju tog pokretanja.

Literatura

[1] E.K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 127(2):266–280, July 2002.

[2] R. Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, 2007.

[3] B. McCollum. University timetabling: Bridging the gap between research and practice. In H RudovĂĄ E Burke, editor, *PATAT 2006 — Proc. 6th Int. Conf. on the Practice And Theory of Automated Timetabling*, pages 15 – 35. Masaryk University, 2006.

[4] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

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

[6] Andrea Schaerf. A survey of automated timetabling. In *115*, page 33. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 30 1995.

[7] Michael D. Vose. *Simple Genetic Algorithm*. MIT Press, 1999.