SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 436

Algoritmi evolucijskog računanja primijenjeni na problem izrade školskog rasporeda sati

Siniša Pribil

Zagreb, lipanj 2012.

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

Zahvaljujem asistentu dr.sc. Marku Čupiću na ustupljenim materijalima, vremenu, korisnim savjetima i velikoj pomoći koju mi je pružio tijekom izrade ovog rada.

Veliko hvala mojoj Matei na lektoriranju rada, korisnim savjetima, beskrajnom razumijevanju za moje probleme i nepresušnim riječima poticaja i podrške tijekom cijelog rada na ovom projektu.

Naposljetku, hvala mojim roditeljima na razumijevanju i podršci tijekom mog cijelog školovanja.

SADRŽAJ

1. Uvod

Velik dio našeg života određen je različitim oblicima rasporeda: školske i fakultetske obveze, poslovni sastanci, kino projekcije, promet javnog prijevoza i slično. Tijekom prošlosti svi oblici rasporeda izrađivali su se ručno. Iako su zahtjevi bili manji, niti tada to nije bilo jednostavno. Problem izrade rasporeda u današnje je vrijeme još složeniji postupak jer skup elemenata koje je potrebno rasporediti postaje sve veći, a vremenski okvir unutar kojeg ih je moguće rasporediti sve manji. Na temelju navedenih razloga, nije teško objasniti težnju da taj postupak bude što je više moguće automatiziran.

Obrazovne ustanove najmanje su jednom godišnje suočene s vrlo složenim zadatkom. Potrebno je za desetke grupa ili razreda sa stotinama učenika ili studenata organizirati predavanja i druge događaje koje će svi moći pohađati. Također je potrebno obratiti pozornost na raspoloživost nastavnika i dvorana odgovarajućih veličina u zadanom vremenu. U skladu s tim potrebama postoje tri oblika rasporeda koji se redovito izrađuju u sustavu obrazovanja: osnovno- i srednje-školski, fakultetski i rasporedi fakultetskih ispita.

Izrada svakog od navedenih rasporeda započinje s drugačijim pretpostavkama i ograničenjima, iako oni u osnovi moraju zadovoljavati jednake zahtjeve (učenik/student i nastavnik ne smiju imati raspoređena dva događaja u isto vrijeme, dvorana za svaki događaj mora biti odgovarajućeg kapaciteta i slično). Tako se, na primjer, prilikom izrade rasporeda fakultetskih ispita može ili ne mora pretpostaviti da su svi studenti (osim obaveza izazvanih održavanjem drugih ispita koje polažu) slobodni u vrijeme za koje se raspored izrađuje. U osnovno- i srednje-školskom rasporedu u pravilu je poželjno da se sva predavanja održavaju u jednom bloku[1], dok u fakultetskom to nije nužno. Dakako, spomenuta podjela na tri oblika rasporeda u obrazovnom sustavu nije čvrsta, pa tako, na primjer, postoje škole koje svojim učenicima nude veću slobodu prilikom odabira predmeta koje žele slušati. Takvi rasporedi onda imaju određene sličnosti s fakultetskim.

[1] blok predavanja — jedno ili više predavanja iz jednog ili više predmeta koja se održavaju za redom, bez pauze (osim odmora)

Općenito je izrada školskih rasporeda posebno zanimljiva zbog velikog broja pedagoških i organizacijskih ograničenja koja je potrebno zadovoljiti. Većina škola u Republici Hrvatskoj radi u više smjena i ima manjak slobodnog prostora. S obzirom na to da trenutno ne postoji sustav za automatiziranu izradu rasporeda koji bi u potpunosti zadovoljio njihove potrebe, taj iznimno težak i dugotrajan posao svake godine odrađuju školski satničari, koji ponekad nisu u mogućnosti izraditi zadovoljavajući raspored.

Problem izrade rasporeda iznimno je složen kombinatorni problem, a ubraja se i među najteže probleme koje opisuje računarska znanost. Bit je njegove složenosti u velikom broju mogućih rješenja među kojima je potrebno pronaći ono odgovarajuće. Za takve probleme, niti današnja najbrža računala ne mogu iscrpnim pretraživanjem u prihvatljivom vremenu doći do rezultata.

Sredinom prošlog stoljeća počelo se intenzivnije razvijati evolucijsko računarstvo. Ono je temelj za novi način rješavanja problema pronalazilo u evolucijskim postupcima. Ubrzo su evolucijski i slični algoritmi postali rješenje za do tada nerješive probleme na računalu. Pokazalo se kako za rješavanje nekih problema nije dovoljna samo računalna snaga nego i odgovarajuća strategija. Evolucijski algoritmi ne pretražuju sva moguća rješenja za određeni problem, pa tako ne mogu niti jamčiti da će uvijek pronaći optimalno rješenje. Unatoč tome, najčešće je pronađeno rješenje sasvim prihvatljivo u okviru problema koji se rješava i, što je najvažnije, nalaze ga unutar vremena koje smo voljni čekati. Algoritmi koji daju zadovoljavajuće dobra rješenja, ali ne uvijek i optimalna, zovu se približni algoritmi ili heuristike.

Ovaj rad opisuje problem i ostvarenje rješenja automatizirane izrade rasporeda nastave za osnovne i srednje škole korištenjem algoritama evolucijskog računanja. U sklopu rada razvijen je *Sustav za izradu rasporeda* koji se temelji na metaheurističkim algoritmima, a ostvarene su i tri inačice genetskog algoritma s popratnim evolucijskim operatorima i metodama optimizacije.

U poglavlju 2 iznosi se formalan zapis osnovnog problema raspoređivanja i dodatna ograničenja koja su prikupljena u razgovorima s predstavnicima dvije osnovne (*OŠ Voltino* i *OŠ Sveta Nedelja*) i tri srednje (*IV. gimnazija*, *V. gimnazija* i *Prva ekonomska škola*) škole u gradovima Zagreb i Sveta Nedelja. Na kraju poglavlja opisani su načini zapisa ulaznih parametara rasporeda, izgrađenog rasporeda i način vrednovanja rješenja. Metode izrade rasporeda opisane su u poglavlju 3. U poglavlju 4 opisuju se evolucijski algoritmi i njihov način rada, a uz osnovni genetski algoritam, navode se i dvije inačice poboljšanja istog. Poglavlje 5 opisuje programsko ostvarenje *Sustava*, a poglavlje 6 iznosi rezultate ispitivanja njegova učinka.

2. Problem školskog rasporeda sati

Problem izrade rasporeda nastave nije moguće jednoznačno formalno iskazati zbog različitih okolnosti u kojima ga je potrebno riješiti. Ipak, pojedini pojmovi kao što su razred, nastavnik i predavanje, sastavni su dio svakog rasporeda.

Definicija problema u ovom poglavlju formalno se iskazuje kroz nekoliko koraka. U prvom koraku problem se opisuje iznoseći najmanji mogući broj zahtjeva koje bi trebao zadovoljavati svaki važeći raspored. U drugom koraku, pojednostavljeni se problem nadograđuje uzimanjem u obzir zahtijeva koji se vrlo često pojavljuju prilikom izrade rasporeda u stvarnom životu, čime se opisuje osnovni problem izrade rasporeda.

U drugom odjeljku ovog poglavlja opisuju se specifični zahtjevi koji se pojavljuju prilikom izrade rasporeda u osnovnim i srednjim školama u Zagrebu, a koji nisu pokriveni definicijama iskazanim u prvom odjeljku. Za neke zahtjeve navode se i formalni zapisi kojima se proširuju zahtjevi osnovnog problema.

Treći odjeljak opisuje formate datoteka: ulaznih u kojima su zapisani parametri rasporeda kojeg *Sustav* mora izgraditi, kao i izlaznih s već izgrađenim rasporedima. U posljednjem odjeljku pobliže se opisuju načini na koje se moguće rješenje problema može vrednovati.

2.1. Formalna definicija

Prije iskazivanja formalne definicije problema, potrebno je zadati oznake osnovnih pojmova koji se koriste prilikom izrade rasporeda. Neka oznake: $c_1, \ldots, c_m$ označavaju m razreda, oznake: $t_1, \ldots, t_n$ označavaju n nastavnika i neka brojevi: $1, \ldots, p$ predstavljaju p termina (ukupan broj nastavnih sati kroz određeni broj dana za koje se raspored izrađuje). Također, neka $R_{m \times n}$ predstavlja matricu predavanja, gdje vrijednost r_{ij} označava broj predavanja koje nastavnik t_j održava razredu c_i u zadanom vremenskom rasponu (unutar p termina).

2.1.1. Pojednostavljeni problem

Najjednostavniji problem izrade rasporeda može se iskazati kao problem raspoređivanja zadanih predavanja u predviđene termine (nastavne sate) na način da pritom niti jedan razred ili nastavnik ne budu raspoređeni u više od jednog predavanja. Taj je zadatak de Werra (1985) formalno iskazao kao:

$$\text{pronađi} \quad X_{m\times n\times p}$$

$$\text{t. d.} \quad \forall(i,j) \quad \sum_{k=1}^{p} x_{ijk} = r_{ij}, \tag{2.1}$$

$$\forall(i,k) \quad \sum_{j=1}^{n} x_{ijk} \leq 1, \tag{2.2}$$

$$\forall(j,k) \quad \sum_{i=1}^{m} x_{ijk} \leq 1 \text{ i} \tag{2.3}$$

$$\forall(i,j,k) \quad x_{ijk} = 0 \text{ ili } 1,$$

pri čemu je $x_{ijk} = 1$ ako k-ti sat nastavnik t_j predaje razredu c_i, odnosno $x_{ijk} = 0$ inače.

Ograničenjem iskazanim izrazom 2.1 osigurava se da svaki nastavnik održi točno onoliko predavanja koliko mu je prethodno zadano. Preostali izrazi osiguravaju da niti jedan razred (2.2) ili nastavnik (2.3) nemaju raspoređeno više od jednog predavanja u isto vrijeme.

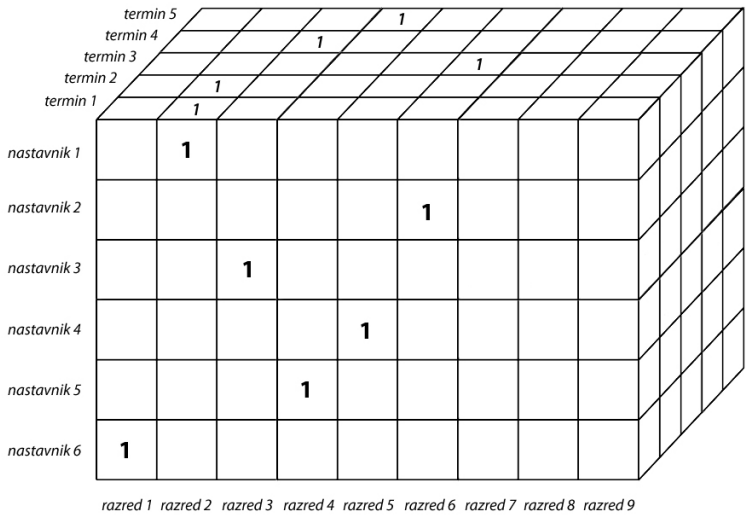
Dokazano je (Even et al., 1975) kako uvijek postoji rješenje za navedeni problem, uz poštivanje uvjeta da niti jedan razred i niti jedan nastavnik nemaju zadano više od p predavanja u koje ih je potrebno rasporediti. Dakle, navedeni problem uvijek ima rješenje ukoliko vrijedi:

$$\forall(i) \quad \sum_{j=1}^{n} r_{ij} \leq p \text{ i}$$

$$\forall(j) \quad \sum_{i=1}^{m} r_{ij} \leq p.$$

Lako je uvidjeti zašto je gornja granica za broj predavanja upravo p. S obzirom da svaki razred i nastavnik u jednom terminu mogu pohađati ili organizirati samo jedno predavanje, najveći broj predavanja u kojima mogu sudjelovati je p i u tom će slučaju biti zauzeti u svakom od p termina. S druge strane, dokaz da rješenje pojednostavljenog problema uvijek postoji uz ove uvjete, nešto je složeniji.

Potrebno je naglasiti kako je prilikom rješavanja pojednostavljenog problema jedino važno osigurati da svi razredi i svi nastavnici mogu biti prisutni na predavanjima u bilo kojem od p termina. Pritom je bilo kakvo rješenje koje zadovoljava prethodno navedene uvjete prihvatljivo. Dodatno, ovaj oblik rasporeda ne uzima u obzir činjenicu da su za održavanje predavanja potrebne učionice, čiji je broj ograničen, kao i nužnost poštivanja niza drugih organizacijskih i pedagoških zahtjeva (poput najvećeg broja radnih sati u danu i slično). Grafički prikaz rasporeda izrađenog na temelju iskazanog pojednostavljenog problema prikazan je slikom 2.1.



Slika 2.1: Grafički prikaz rasporeda na temelju pojednostavljenog problema

2.1.2. Osnovni problem

Unatoč činjenici da je raspored izrađen rješavanjem problema opisanog u prethodnom odjeljku ispravan i teoretski izvediv, veoma je malen broj slučajeva u kojima će takav raspored zadovoljiti stvarne potrebe prosječne škole. Trebalo bi, na primjer, uzeti u obzir činjenicu da postoje sati kada određeni nastavnik ili razred nisu u mogućnosti biti prisutni na predavanju, odnosno da te sate nisu raspoloživi. Navedena pretpostavka značajno otežava rješavanje problema, ali dobiveno rješenje čini mnogo prihvatljivijim.

Neka su $C_{m\times p}$ i T_{nxp} dvije binarne matrice takve da vrijedi $c_{ik} = 1$ (odnosno $t_{jk} = 1$) ukoliko je razred c_i (odnosno nastavnik t_j) raspoloživ u zadano vrijeme k, a $c_{ik} = 0$ (odnosno $t_{jk} = 0$) inače. Tada formalan zapis problema, koji uzima u obzir raspoloživosti nastavnika i razreda, glasi:

$$\text{pronađi} \quad X_{m\times n\times p}$$

$$\text{t. d.} \quad \forall(i,j) \quad \sum_{k=1}^{p} x_{ijk} = r_{ij}, \tag{2.4}$$

$$\forall(i,k) \quad \sum_{j=1}^{n} x_{ijk} \leq c_{ik}, \tag{2.5}$$

$$\forall(j,k) \quad \sum_{i=1}^{m} x_{ijk} \leq t_{jk} \text{ i} \tag{2.6}$$

$$\forall(i,j,k) \quad x_{ijk} = 0 \text{ ili } 1.$$

Važno je primijetiti kako izrazi 2.5 i 2.6 zamjenjuju i ujedno dodatno ograničuju izraze 2.2 i 2.3 iz pojednostavljenog problema. Naime, tim je izrazima osigurano da niti jednom razredu (odnosno nastavniku) ne može biti raspoređena obveza u onaj terminu u kojem nisu raspoloživi.

De Werra (1985) u obzir uzima i unaprijed raspoređena predavanja. To su predavanja kojima je termin čvrsto zadan prije početka izrade rasporeda i ta se predavanja u postupku izrade ne mogu premještati. Takav je oblik ograničenja koristan ukoliko za neka predavanja postoje vanjski utjecaji zbog kojih ih je moguće održati samo u određenim terminima. Unaprijed raspoređena predavanja mogu se formalno iskazati na sljedeći način:

$$\forall(i,j,k) \quad x_{ijk} \geq y_{ijk},$$

gdje je $y_{ijk} = 1$ ukoliko za razred c_i i nastavnika t_j postoji unaprijed raspoređeno predavanje u terminu k, odnosno $y_{ijk} = 0$ inače. Izraz je u skladu sa svim prethodno navedenim jer i u slučaju da predavanje nije unaprijed raspoređeno u termin k, x_{ijk} i dalje može poprimiti vrijednost 1. Pokazano je kako je unaprijed raspoređena predavanja moguće ostvariti mehanizmom raspoloživosti koristeći izmišljene razrede ili nastavnike (na primjer, izmišljeni nastavnik raspoloživ je samo u jednom terminu — onom u kojem se nalazi unaprijed raspoređeno predavanje).

2.2. Dodatna ograničenja

Ovaj odjeljak sadrži opise dodatnih ograničenja kojima se raspored može bolje prilagoditi potrebama pojedine škole. U odjeljku su ujedno opisani i zanimljivi zahtjevi koje prema svojim rasporedima postavlja pet zagrebačkih osnovnih i srednjih škola. Važno je međutim napomenuti kako svako dodatno ograničenje značajno smanjuje broj mogućih rješenja i samim time otežava zadatak izrade valjanog rasporeda. Velik utjecaj na težinu izrade rasporeda ima i svrstavanje zahtjeva u grupe čvrstih i mekih, o čemu će više riječi biti u nastavku poglavlja. U dodatku A nalazi se popis svih prikupljenih zahtjeva organiziran u obliku ulaznih parametara *Sustava za izradu rasporeda*.

2.2.1. Raspoređivanje učionica

U opisu osnovnog problema, navedenom u prethodnom odjeljku, nije naveden zahtjev za raspoređivanjem predavanja u učionice jer se u problemu pretpostavlja da svaki razred u svakom terminu ima osiguranu vlastitu učionicu. Tom se pretpostavkom zahtijeva da škola raspolaže s barem m u potpunosti iskoristivih učionica. Koliko god se čini da je ta pretpostavka osnova za normalan rad svake škole, praksa je nažalost pokazala kako ne raspolažu sve škole takvim resursima. Neka predavanja zahtijevaju posebnu opremu koja nije prisutna u svim učionicama (laboratoriji za fiziku, kemiju, učionice za likovni i glazbeni odgoj i slično) ili čak i same učionice nisu raspoložive cijelo vrijeme (na primjer, dvorana za tjelesnu i zdravstvenu kulturu koja se dijeli s drugom školom ili laboratorij koji u određeno vrijeme koriste grupe za izvannastavne aktivnosti). Učionice su često i različitih veličina, pa je potrebno voditi računa i o njihovom kapacitetu i veličini razreda koji bi mogao pohađati nastavu u njima. Sve iznesene činjenice uvelike utječu na izgled valjanog rasporeda, pa učionice često postaju sastavni dio problema izrade.

Schaerf (1999) spominje problem raspoređivanja u posebne učionice i predlaže rješenje u obliku označavanja učionica s posebnom opremom i bilježenja broja predavanja koja se u pojedinom terminu održavaju u takvim učionicama. Tada takvih predavanja u jednom terminu ne smije biti veći od broja posebnih učionica. Međutim, problem različitih opremljenosti i veličine učionica u hrvatskim školama uvelike nadilazi predloženo rješenje.

Učionica u kojoj se održava predavanje nova je varijabla kojom je potrebno proširiti prostor u koji se smještaju predavanja prilikom izrade rasporeda. Označimo s h_1, …, h_o o različitih učionica s kojima škola raspolaže i zadajmo binarnu matricu $H_{o \times p}$

tako da vrijedi: $h_{lk} = 1$ ako je učionica h_l raspoloživa u zadano vrijeme k, odnosno $h_{lk} = 0$ inače. U tom slučaju moguće je proširiti i matricu raspoređenih predavanja (iz osnovnog problema) na dimenziju koja uključuje učionice, tako da glavna zadaća problema glasi:

$$\text{pronađi} \quad X_{m \times n \times p \times o}.$$

Osim toga, svakoj se dvorani h_l pridružuje i skup njezinih značajki F_l, odnosno opreme kojom raspolaže. Jednako se tako i svakom predavanju r_{ij} koje je potrebno rasporediti pridružuje skup značajki E_{ij} koje treba zadovoljavati učionica u koju se predavanje raspoređuje. Nakon toga osnovni se problem može nadograditi dvama dodatnim ograničenjima:

$$\forall(i, j, k) \quad \sum_{l=1}^{o} x_{ijkl} \leq h_{lk} \text{ i} \tag{2.7}$$

$$\forall(x_{ijkl}) \quad \text{t. d. } x_{ijkl} = 1 \rightarrow E_{ij} \subseteq F_l. \tag{2.8}$$

Izrazom 2.7 osigurava se da se predavanja u određenoj učionici neće održavati u onim terminima u kojima navedena učionica nije raspoloživa. Izraz 2.8 ograničuje raspoređivanje predavanja samo u one učionice koje zadovoljavaju zahtijevane kriterije opremljenosti. Može se primijetiti kako se ovim izrazom ujedno može riješiti i problem kapaciteta učionica i broja učenika u razredu. U značajke učionice zabilježi se njezin kapacitet, a zahtijevanim se značajkama u sklopu predavanja pridruži broj učenika u razredu koji pohađa predavanje. Ostali izrazi iz 2.1.2 proširuju se za dimenziju učionica i istovjetno moraju vrijediti i u ovom slučaju.

2.2.2. Višesatna predavanja

Višesatna predavanja redovito se pojavljuju u školama u Hrvatskoj. Najčešće su to predavanja od dva sata koja se nazivaju blok sati. Višesatna predavanja obično se održavaju u sklopu predmeta koji programom imaju predviđeno tri ili više sati nastave tjedno. Kod takvih predmeta, raspodjela predavanja predviđa po jedan ili dva blok sata u tjednu. Takva se predavanja zatim iskorištavaju za nastavne cjeline i aktivnosti koje iziskuju više od jednog školskog sata kao, na primjer, pisanje školske zadaće ili obrada lektire iz hrvatskog jezika, pisanje većih kontrolnih zadaća iz matematike i slično. Iako se prema nastavnom programu održavaju manje od tri sata tjedno, pojedini su predmeti, poput tehničke ili likovne kulture, zbog svojih obilježja u nekim školama također organizirani u blok satove. Samim su time i višesatna predavanja važan pojam koji je potrebno uključiti u problem izrade rasporeda.

Trenutno opisan formalni problem ne predviđa postojanje višesatnih predavanja. S obzirom na to da jedno polje matrice predavanja $X_{m\times n\times p\times o}$ predstavlja predavanje u trajanju od jednog sata, moguće je višesatno predavanje predstaviti u obliku dva jednosatna predavanja s istim sudionicima u istoj dvorani koja su uzastopna u vremenu. Time je međutim samo pokazano da je u trenutnoj strukturi rješenja moguće prikazati ovakvu vrstu predavanja. Potrebno je još na neki način zadati duljine predavanja koja se trebaju rasporediti i ograničenje da se višesatno predavanje zaista mora održati u uzastopnim nastavnim satima. Duljine predavanja moguće je zadati nizom brojeva koji će predstavljati zadane duljine predavanja koja je potrebno rasporediti. Nazovimo taj niz d_{ij}, pri čemu on određuje duljine predavanja koje razred i pohađa kod nastavnika j u vremenskom rasponu za kojeg se izrađuje raspored. Na primjer: $d_{14} = [2, 2, 1, 1, 1]$ predstavlja niz predavanja koja nastavnik 4 mora održati razredu 1. Vidljivo je kako je potrebno rasporediti ukupno pet predavanja, među kojima su dva u trajanju od dva sata, a tri u trajanju od jedan nastavni sat. U skladu s izrazom 2.4, mora vrijediti i:

$$\forall(i,j,l) \quad \sum_{z=1}^{|d_{ij}|} d_{ijz} = \sum_{k=1}^{p} x_{ijkl} = r_{ij} \text{ i}$$

$$\forall(i,j,z) \quad d_{ijz} > 0,$$

odnosno, suma duljina svih predavanja mora biti jednaka unaprijed zadanom broju sati predavanja r_{ij} koje nastavnik j predaje razredu i. Pri tome d_{ijz} predstavlja duljinu z-tog predavanja. Označimo sa s_{ijz} termin u kojem započinje predavanje d_{ijz}. Osnovni zahtjev raspoređivanja višesatnih predavanja, održavanje predavanja s istim sudionicima u više uzastopnih sati, može se zapisati kao:

$$\forall(i,j) \quad \exists(l) \quad \sum_{z} \sum_{k=s_{ijz}}^{s_{ijz}+d_{ijz}-1} x_{ijkl} = d_{ijz}.$$

Potrebno je pripaziti na još jedan problem. S obzirom na to da se višesatna predavanja prema navedenim ograničenjima mogu smjestiti u bilo koji termin, potrebno je spriječiti raspoređivanje predavanja u termine toliko blizu kraju dana da se zadano uzastopno predavanje prekine kroz dva dana. Uzmimo kao primjer predavanje duljine dva sata koje se raspoređuje u ponedjeljak zadnji sat poslije podne. Prema dosadašnjim formalnim izrazima, predavanje bi se zaista održavalo u dva uzastopna termina (zadnji sat u ponedjeljak i prvi sat u utorak), ali ta dva termina nalaze se u dva različita dana. Potrebno je stoga isključiti takvu mogućnost. Neka se svaki dan sastoji od q radnih sati (p je višekratnik od q). Za sva predavanja onda mora vrijediti:

$$\lfloor s_{ijz} \,/\, q \rfloor = \lfloor (s_{ijz} + d_{ijz} - 1) \,/\, q \rfloor.$$

2.2.3. Neprekidan raspored

U gotovo svim osnovnim i većini srednjih škola, neprekidnost rasporeda za razrede vrlo je važno ograničenje. Neprekidan raspored znači da razred ne smije imati niti jednu pauzu između predavanja unutar istog dana (kako, na primjer, učenici ne bi lutali školom bez nadzora). Do takve situacije neće doći ukoliko je broj sati predavanja u kojima razred mora sudjelovati jednak ukupnom broju raspoloživih sati p. Ipak, najčešće u rasporedu postoje sati kada određeni razred nema predavanje. Takvi sati onda obavezno moraju biti prije ili nakon nastave, a nikako između dva predavanja u istom danu. Drugim riječima, u svakom danu g mora vrijediti:

$$\forall(i) \quad \sum_{j,l} \sum_{k=a_{ig}}^{b_{ig}} x_{ijkl} = b_{ig} - a_{ig} + 1,$$

pri čemu a_{ig} predstavlja indeks termina prvog predavanja u danu g za razred i, dok je b_{ig} indeks zadnjeg termina predavanja. Schaerf (1996) to ograničenje uzima u obzir na način da za svaki razred zadaje sate koji obavezno moraju biti popunjeni predavanjima (u sredini dana), dok je ostale moguće popuniti po potrebi.

Većina škola istovremeno ne ustraje na potpunim neprekidnim rasporedima za nastavnike, ali zahtijeva da oni budu donekle uređeni. Tako se obično dopuštaju pauze od jedan sat dnevno do najviše dva do tri sata tjedno. Pauze se, dodatno, obvezno dodjeljuju nastavniku u danu kada ima više od unaprijed određenog broja predavanja. Neke škole zahtijevaju da svaki (ili uglavnom svaki) sat u tjednu po jedan nastavnik ima pauzu, kako bi u slučaju potrebe mogao održati zamjensko predavanje.

U razgovoru s predstavnicima zagrebačkih škola, pojavio se još jedan zanimljiv zahtjev vezan uz raspoređivanje pauzi u nastavi. Odnosi se na pauze zadane vremenskim rasponom. Riječ je o ograničenju u kojem bi se zadavao vremenski raspon unutar jednoga dana (početnim i završnim terminom) u kojem bi se zahtijevalo određeno vrijeme pauze. Trebalo bi se moći odrediti treba li se cjelokupno vrijeme ostvariti kao jedinstvena, neprekinuta pauza, ili je dovoljno samo da ukupan zbroj duljina pauzi bude jednak traženom vremenu. Takvo ograničenje korisno je u slučajevima kada se raspored izrađuje za više udaljenih objekata je pa, na primjer, potrebno određeno vrijeme nastavniku da doputuje s jednog mjesta na drugo. Slično je i s učenicima u slučaju da je dvorana za tjelesnu i zdravstvenu kulturu udaljena od ostalih učionica, pa im je potrebno određeno vrijeme da do nje dođu. Pritom nije moguće predvidjeti više od jednog sata tjelesne i zdravstvene kulture za taj razred (koji bi uključivao i vrijeme potrebno za dolazak), jer bi se time ujedno rezervirao i nastavnik koji za to vrijeme može održavati nastavu drugim razredima. S obzirom na to da predstavljaju željene i

jasno zadane slučajeve, pauze zadane vremenskom rasponom ne podrazumijevaju se kao kršenje zahtjeva za neprekidnost rasporeda.

2.2.4. Podjele razreda

Programi u osnovnim i srednjim školama u pravilu su u mnogome slični. Ukoliko postoje značajnije razlike u broju predavanja i predmetima koji se preklapaju u sklopu više programa, škole najčešće uspješno uspiju organizirati razrede koji se sastoje od učenika istog programa. Unatoč uspješnom organiziranju razreda po programima, učenici često moraju birati između pohađanja, na primjer, vjeronauka i etike ili jednog od više ponuđenih stranih jezika (ponekad i početne i napredne grupe za svakog od njih). To dovodi do situacije da će se u pojedinim terminima razred morati podijeliti na dvije ili više grupa. Takve podjele obično slijede spajanja manjih grupa iz više razreda koje zajedno slušaju isti predmet. Na primjer, dijelovi razrednih odjeljenja a i b međusobno se spajaju u dvije cjeline koje posebno pohađaju etiku i vjeronauk.

U dosadašnjim iskazanim primjerima razredi su bili cjeloviti. Zadana predavanja održavala su se cjelokupnim razredima i bila su neovisna o drugim predavanjima (izuzevši činjenicu da nije moguće rezervirati predavanje u terminu u kojem je jedan od sudionika već zauzet drugim predavanjem). U tom slučaju razred više ne može biti najmanja čestica grupe učenika kojoj se održava nastava. To postaje razredni odjeljak odnosno skup učenika istog razreda koji, gledajući tip dijeljenja, pohađaju iste predmete. Tako, na primjer, podjela razreda prema stranom jeziku može biti na odjeljak koji uči engleski i odjeljak koji uči njemački jezik. Ukoliko se s γ_i označi skup svih učenika razreda c_i, a s δ^i_{uv} v-ti odjeljak u-tog tipa razreda c_i, tada moraju vrijediti sljedeći izrazi:

$$\forall(u) \quad \bigcup_v \delta^i_{uv} = \gamma_i, \tag{2.9}$$

$$\forall(u) \quad \forall(v_1, v_2) \quad \delta^i_{uv_1} \cap \delta^i_{uv_2} = \emptyset. \tag{2.10}$$

Izraz 2.9 ukazuje na potrebu cjelokupne podjele razreda na odjeljke unutar tipa, to jest, svi učenici razreda moraju se nalaziti u jednom od odjeljaka. Također, izraz 2.10 iskazuje činjenicu da odjeljci moraju biti disjunktni, odnosno da niti jedan učenik razreda ne smije pripadati dvama ili više odjeljaka istog tipa.

S obzirom na to da su sada odjeljci razreda postali najmanja čestica grupe učenika kojoj se održava nastava, osnovni problem izrade rasporeda mora se promijeniti na način da se, umjesto razreda, bilježe raspoloživosti i zadaci odjeljaka razreda. Treba se promijeniti i način zadavanja predavanja koja je potrebno rasporediti kako bi se omo-

gućilo održavanje nastave u kojoj sudjeluje veći broj grupa učenika (u ovom slučaju odjeljaka razreda). Ovo je potrebno kako bi se mogla normalno održavati redovna predavanja cijelom razredu (navođenjem svih odjeljaka jednog tipa). Navedene promjene za sobom povlače i druge poteškoće, a one su opisane u sljedećem odjeljku.

2.2.5. Vezana predavanja

Važno je uočiti dodatna ograničenja koja prema rasporedu postavlja zahtjev za zadavanjem odjeljaka razreda naveden u prethodnom primjeru. Ukoliko se razredi a i b spajaju prilikom pohađanja etike i vjeronauka, predavanja iz navedenih predmeta za oba razreda moraju biti raspoređena u isto vrijeme. Neovisno o tome što se razred dijeli na dva odjeljka, ukoliko je zadan zahtjev za neprekidnost rasporeda, svaki učenik navedenog razreda mora imati neprekidan raspored. Postoje dvije mogućnosti za raspoređivanje navedenih predavanja uz istovremeno poštivanje neprekidnosti rasporeda:

- *predavanja iz oba predmeta moraju se održavati istodobno u različitim učionicama, ili se*
- *predavanja mogu održavati u različitim terminima, ali oni moraju biti na početku ili kraju nastave*.

Prva je mogućnost jednostavnija, ali zahtijeva da nastavnici oba predmeta na koje se razredi dijeli, pa i sam razred, budu raspoloživi u isto vrijeme. U drugom slučaju takvog zahtjeva nema, ali u obzir dolaze samo prvi i zadnji satovi zbog očuvanja neprekidnosti rasporeda za drugu polovicu razreda koja ne pohađa navedeni predmet. Moguće je primijetiti kako se spomenuti primjer jednostavno može poopćiti za slučaj dijeljenja razreda na više od dva odjeljka.

Nešto je drugačiji zahtjev ostvarenje usporednih predavanja unutar jednog razreda iz dva ili više predmeta. Vježbe iz informatike i fizike jedan su od primjera ovog tipa raspoređivanja. Po jedan odjeljak razreda naizmjenično svaki tjedan pohađa određen broj sati vježbi iz jednog ili drugog predmeta. Ovo je također slučaj u kojem se razred dijeli na predavanja dva predmeta, ali se ovom prilikom ne spaja niti s jednim drugim razredom. Mogućnosti za razrješavanje ovog slučaja su jednake kao i kod razdvajanja i spajanja više razreda: potrebno je naći termin koji odgovara svim nastavnicima na čija se predavanja razred dijeli ili se navedena predavanja moraju rasporediti na početke i krajeve dana.

Oba navedena slučaja predstavljaju primjere vezanih predavanja. Dva ili više predavanja vezana su ukoliko između njih postoji ograničenje prema kojem je prije ras-

poređivanja jednog potrebno razmotriti mogućnosti raspoređivanja ostalih predavanja. Ograničenja mogu nalagati raspoređivanje predavanja u isti, isključivo različiti termin ili uzastopno. Tako, na primjer, dijeljenje razreda prilikom vježbi informatike i fizike, ukoliko se predavanja želi rasporediti u isti termin, predstavlja vezu između predavanja koje predstavlja vježbe iz fizike jednoj polovici razreda i predavanja koje predstavlja vježbe iz informatike drugoj polovici razreda. Iako se predavanja inače raspoređuju pojedinačno, spomenuta veza nalaže da se predavanja rasporede u isti termin.

Vezana predavanja mogu biti iznimno korisna i prilikom raspoređivanja predavanja iz predmeta koji u nastavnom programu imaju malu satnicu, pa se održavaju svaki drugi tjedan naizmjenično. U nekim školama takav primjer predstavljaju predavanja iz tehničke i likovne kulture. Svaki razred pohađa po dva sata predavanja iz navedenih predmeta jednom u dva tjedna, ali na način da se predmeti, radi očuvanja ukupnog tjednog broja sati, predaju u različitim tjednima (jedan tjedan tehnička, a drugi likovna kultura). Vezom između predavanja tehničke i likovne kulture za svaki razred može se iskazati spomenuto ograničenje, ali ujedno i ne čvrsto zadati u kojem će se tjednu održavati koji predmet. Takav je zahtjev manje ograničavajući i samim je time skup mogućih rješenja problema veći.

2.2.6. Upravljanje opretećenjima

Osim neprekidnosti rasporeda, prilikom izrade istog često se obraća pozornost i na opterećenja učenika i nastavnog osoblja. Dok u, primjerice, fakultetskom rasporedu najmanji ili najveći broj nastavnih sati u danu nisu značajno ograničenje, poželjno je da školski rasporedi poštuju ove stavke. To je ponajprije važno iz pedagoških razloga, jer redovitim obavljanjem svakodnevnih obveza učenici usvajaju radne navike. Tako najmanji broj nastavnih sati osigurava svakodnevni boravak učenika u školi, dok najveći broj služi sprječavanju predugih radnih dana u kojima učenici ne bi mogli zadržati koncentraciju.

Uz pažnju koja se obraća na najmanji i najveći broj sati, prilikom izrade školskog rasporeda prednost se daje i rasporedima koji imaju ujednačen broj sati kroz dane tijekom tjedna, kako za razrede, tako i za nastavnike (Bufé et al., 2001). Ovisno o školi i njezinim potrebama, moguće je tražiti ili izbjegavati rasporede s više višesatnih predavanja unutar istog dana. Školski satničari često se trude što je više moguće olakšati raspored svojim učenicima na način da pokušaju predavanja zahtjevnijih predmeta razmjestiti po različitim danima umjesto u isti dan. Općenito je raspodjela predavanja u tjednu važan čimbenik za kvalitetan raspored. Što su predavanja istog predmeta više

međusobno udaljena u tjednu, to je raspored bolji. To je tako jer učenicima daje više vremena za pisanje domaćih zadaća i za učenje, a nastavnicima za pripremu novog predavanja.

Nastavnici obično imaju mogućnost izražavanja želje da u jednom danu predaju istoj generaciji učenika (zbog istog gradiva) ili da, ukoliko predaju više od jednog predmeta, u jednom danu predaju samo jedan predmet. Takvi se zahtjevi zadovoljavaju samo ukoliko je to moguće i ne stvara previše problema satničaru. Iako su korisna za učenike i nastavno osoblje jer osiguravaju pogodnija radna vremena, i ova ograničenja dodatno smanjuju skup prihvatljivih rasporeda.

2.2.7. Stupnjevanje raspoloživosti

Tijekom razgovora s predstavnicima zagrebačkih škola, uočeno je još jedno dodatno ograničenje koje bi trebalo zadovoljiti prilikom izrade rasporeda. Mnogi su satničari izrazili želju za uvažavanjem poželjnijih raspoloživih termina za nastavnike. Drugim riječima, prethodno opisani sustav raspoloživosti nastavnika nije dovoljno prilagodljiv. Bilo bi bolje da se, za razliku od samih podataka je li nastavnik raspoloživ ili ne, izražavaju stupnjevi raspoloživosti. Tako bi, na primjer, unutar vremena kada je općenito raspoloživ, nastavnik mogao izraziti više ili manje poželjne termine za održavanje nastave. Razlozi su najčešće obiteljske prirode, poput odvoženja djece u vrtić i slično.

Prilagodba osnovnog problema u cilju zadovoljavanja ovog zahtjeva trebala bi uključivati proširenje opsega vrijednosti matrice $T_{n\times p}$, tako da raspoloživosti t_{jk} nastavnika j u pojedinom terminu k mogu poprimiti vrijednosti iz intervala $[0, 1]$, a ne samo 0 ili 1. Vrijednost 0 bi i dalje označavala neraspoloživost nastavnika u zadanom terminu, dok bi svaka vrijednost veća od 0 označavala raspoloživost nastavnika, ali ujedno i poželjnost termina (veća vrijednost — veća poželjnost). S obzirom na navedeno, pri izradi rasporeda trebalo bi se težiti zadovoljavanju sljedećeg izraza:

$$\forall(i, j, l) \quad \max(\sum_{k=1}^{p} x_{ijkl} \cdot t_{jk}).$$

Načelo stupnjevanja raspoloživosti vrlo se lako može proširiti na razrede i prostorije, ukoliko je to potrebno. Kao što je i spomenuto, navedeno ograničenje nije potrebno obavezno poštivati, ali može biti dobra mjera pri detaljnijoj međusobnoj usporedbi dvaju rješenja.

2.3. Parametri i očekivani rezultati

Kao što je opisano u prethodnom odjeljku, postupak izrade rasporeda zahtijeva velik broj ulaznih podataka. Grafički oblik organizacije podataka koji je čitljiv čovjeku, često nije pogodan za programsku primjenu. S druge strane, *XML* zapis podataka podjednako je lako čitljiv čovjeku i računalu. U ovom se odjeljku opisuje struktura datoteke u kojoj su opisani parametri za koje *Sustav* treba pronaći valjani raspored. Sažetak datoteke koja se koristila prilikom testiranja *Sustava* nalazi se u dodatku B. U drugom dijelu ovog odjeljka navode se i opisuju dva oblika zapisa kojima se može predstaviti izgrađeni raspored. Sažetak i grafički prikazi izlaznih datoteka izloženi su u dodatku C.

2.3.1. Ulazni podaci

Svi parametri potrebni za izradu rasporeda organizirani su u jednoj *XML* datoteci koja se automatski stvara na temelju podataka unesenih kroz grafičko sučelje *Sustava* (Bronić, 2012). Ulazna *XML* datoteka sastoji se od nekoliko dijelova:

- općenitih podataka o rasporedu,
- podataka o terminima (engl. *periods*),
- podataka o turnusima (engl. *turnuses*),
- podataka o učionicama (engl. *rooms*),
- podataka o predmetima (engl. *subjects*),
- podataka o nastavnicima (engl. *teachers*),
- podataka o razredima (engl. *classes*),
- podataka o predavanjima (engl. *lectures*) i
- podataka o vezama između predavanja (engl. *connections*).

Opći podaci o rasporedu obuhvaćaju osnovne podatke poput broja tjedana za koje se raspored izrađuje (*XML* oznaka *weekCount*) i broja radnih dana u tjednu (*dayCount*). Ovdje se nalazi i podatak treba li veće blokove što je više moguće međusobno udaljiti u danu i tjednu (*avoidingLargeBlocksEnabled*). Ukoliko je potrebno paziti na veće blokove, potrebno je zadati podatak koliko treba predavanje trajati da bi se smatralo velikim blokom i podvrgnulo dodatnim provjerama (*largeBlockSize*). Osnovni podaci dodatno sadrže podatak treba li se prilikom raspoređivanja predavanja u učionice paziti na kapacitet učionice i broj učenika u razredu (*roomSizeManagementEnabled*).

Podaci o terminima u danu sadrže popis svih termina, njihove identifikatore (*id*) i vremena njihovih početaka i završetaka. Većinom svi pojmovi navedeni u parametrima rasporeda sadrže podatak o identifikatoru, koji služi za povezivanje prethodno navedenih pojmova u druge cjeline. Tako, na primjer, podaci o turnusima sadrže identifikator, naziv turnusa (*name*), listu identifikatora termina koji pripadaju navedenom turnusu (*periods*) i listu identifikatora termina u s kojima nastava u tom turnusu može započeti (*classStartPeriods*).

Svaka učionica također je zadana svojim identifikatorom (*id*), nazivom (*name*), a dodatno se može navesti i njezin kapacitet (*capacity*). Uz navedene podatke, podaci o učionici sadrže i podatke o njezinoj raspoloživosti (*availabilities*). Za svaki tjedan za koji se izrađuje raspored postoji jedna *XML* oznaka *weekAvailability*. U njoj se nalaze podaci o raspoloživosti u jednom tjednu. Oznaka *periodPreferences* sadrži pak oznake *preference* u kojima se nalaze nizovi brojeva koji određuju raspoloživost objekta u zadanom vremenu. Svaka oznaka *preference* predstavlja jedan dan (pa je stoga i njihov broj jednak broju radnih dana u tjednu). Unutar oznake *preference* nalazi se niz od onoliko cijelih brojeva koliko je zadanih termina u jednom danu. Pritom broj 0 označava da objekt nije raspoloživ u navedeno vrijeme, a broj koji je veći od 0 predstavlja raspoloživost. Međusoban odnos brojeva unutar jedne oznake *preference* utječe na stvarnu raspoloživost objekta koja će se koristiti kao podatak prilikom izrade rasporeda. Pri učitavanju parametara, *Sustav* će najveću vrijednost u danu zabilježiti kao raspoloživost 1, a ostale će vrijednosti skalirati s obzirom na najveću, tako da sve budu u intervalu $[0, 1]$. To znači da će najveći broj unutar jedne oznake *preference* uvijek biti skaliran na vrijednost 1, neovisno o njegovoj apsolutnoj vrijednosti ili odnosu naprema vrijednostima u drugim oznakama. Oznaka *dayPreferences* sadrži na jednak način zapisane podatke o raspoloživosti dana. U više željene dane Sustav bi trebao rasporediti veći broj predavanja, pa je njome ujedno moguće oblikovati odnos raspoloživosti između nastavnih sati u različitim danima.

Primjer podataka o raspoloživosti za raspored od dva tjedna nalazi se u okviru 2.1. U njemu je vidljivo kako je objekt raspoloživ ponedjeljkom i srijedom u prvom tjednu i utorkom, srijedom i četvrtkom u drugom tjednu. Od toga je utorkom i četvrtkom raspoloživ samo u poslije podnevnim, odnosno jutarnjim satima. U smislu poželjnosti raspoloživih termina, u prvom su tjednu poželjniji termini u sredini dana, dok su u drugom tjednu poželjniji termini bliže početku i kraju dana.

Podatke o učionicama slijede podaci o predmetima. Svaki predmet ima svoj zadani identifikator (*id*), naziv (*name*), listu identifikatora učionica koje predstavljaju glavne predmetne učionice (*defaultRooms*) i raspoloživosti (*availabilities*). Od *Sustava za*

izradu rasporeda očekuje se da, ukoliko su predmetne učionice zadane, predavanja navedenog predmeta rasporedi što je više moguće upravo u te učionice. Dodatno se za predmet može zadati i njegova složenost (*complexity*), a *Sustav* bi trebao, koliko je moguće, ujednačiti učeničko opterećenje kroz dane u tjednu.

Okvir 2.1: Primjer podataka o raspoloživosti za raspored na bazi dva tjedna

```
<availabilities>
  <weekAvailability>
    <periodPreferences>
      <preference>1,2,3,4,5,6,7,6,5,4,3,2,1</preference>
      <preference>0,0,0,0,0,0,0,0,0,0,0,0,0</preference>
      <preference>1,2,3,4,5,6,7,6,5,4,3,2,1</preference>
      <preference>0,0,0,0,0,0,0,0,0,0,0,0,0</preference>
      <preference>0,0,0,0,0,0,0,0,0,0,0,0,0</preference>
    </periodPreferences>
    <dayPreferences>1,0,1,0,0</dayPreferences>
  </weekAvailability>
  <weekAvailability>
    <periodPreferences>
      <preference>0,0,0,0,0,0,0,0,0,0,0,0,0</preference>
      <preference>0,0,0,0,0,0,1,2,3,4,5,6,7</preference>
      <preference>7,6,5,4,3,2,1,2,3,4,5,6,7</preference>
      <preference>7,6,5,4,3,2,1,0,0,0,0,0,0</preference>
      <preference>0,0,0,0,0,0,0,0,0,0,0,0,0</preference>
    </periodPreferences>
    <dayPreferences>1,0,1,0,0</dayPreferences>
  </weekAvailability>
</availabilities>
```

Skupovi podataka koji opisuju nastavnike nešto su opširniji. Kao i većina ostalih objekata u parametrima rasporeda, i nastavnici imaju zadan identifikator (*id*), naziv (*name*) i raspoloživosti (*availabilities*). Uz to, *XML* oznaka koja opisuje podatke o nastavnicima sadrži i podatke zahtijeva li nastavnik predavanja istoj generaciji u istom danu (*sameGenerationInDayRequired*) ili istog predmeta u istom danu (*sameSubjectInDayRequired*). Oznaka *dayTurnusAlternationEnabled* sadrži podatak želi li nastavnik izmjenu turnusa kroz dane u tjednu. Naime, neki nastavnici žele raditi naizmjenično ujutro i poslije podne svaki dan. Tada se ova zastavica postavlja na `true`. Slično je i s izmjenom turnusa kroz dane na tjednoj bazi (*daysForWeekTurnusAlter-*

nation). Ti se zahtjevi prije svega odnose na ponedjeljke i petke, kako bi se izbjegla situacija u kojoj jedan nastavnik, na primjer, svaki tjedan radi ponedjeljkom ujutro i petkom popodne. Nastavnicima se također zadaje najmanji i najveći broj radnih sati u danu (*workingPeriodNumber*), kao i najmanji i najveći broj pauzi u danu (*dayBreakNumber*) i tjednu (*weekBreakNumber*). Najveći broj praznih sati koji će se protumačiti kao pauza u rasporedu zadaje se unutar oznake *maximumBreakSize*. Zadaje se i najveći broj radnih sati u danu unutar kojih će se nastavniku obavezno rasporediti pauza (*mandatoryBreakThreshold*). Ukoliko je potrebno, nastavniku se mogu zadati i točan broj radnih (*numberOfWorkingDays*) i slobodnih (*numberOfFreeDays*) dana koje u rasporedu mora imati, kao i pauze zadane vremenskom rasponom (*breakRequirements*).

U okviru 2.2 nalazi se primjer zapisa pauze zadane vremenskim rasponom. Svaka pauza nalazi se u posebnoj oznaci *requirement*. U uglatim zagradama nalaze se indeks tjedna i indeks dana u kojem se treba rasporediti pauza. Podatke o tjednu i danu slijede identifikatori termina unutar kojih se pauza mora rasporediti i broj koji označava zadanu duljinu tražene pauze. Na kraju se u zaobljenoj zagradi nalazi indikator treba li pauza biti raspoređena u jednom bloku ili ne. Tako je u okviru 2.2 zadana pauza koja bi se trebala rasporediti u utorak prvog tjedna, između termina *j1* i *j3*. Pauza bi trebala trajati dva sata i biti raspoređena u jednom bloku.

Okvir 2.2: Primjer podataka o pauzi zadanoj vremenskim rasponom

```
<breakRequirements>
  <requirement>[1,2]: j1 - j3 => 2 (true)</requirement>
</breakRequirements>
```

Osim osnovnih podataka koje dijele s drugim objektima (*id*, *name* i *availabilities*), podaci o razredu dodatno sadrže podatak o tome treba li razred imati neprekidan raspored (*continousScheduleRequired*), najmanji i najveći broj radnih sati (*workingPeriodNumber*), kao i identifikator razredne učionice (*mainClassroom*). Razredna učionica je učionica u kojoj bi razred trebao pohađati nastavu iz predmeta kojima nije zadana predmetna učionica ili je ona zauzeta. Dodatno se mogu navesti i pomoćne učionice (*auxiliaryClassrooms*) u kojima će se nastava održavati ako je i razredna učionica zauzeta. Pod oznakom *studentCount* može se navesti ukupan broj učenika u razredu. Podjele razreda na odjeljke navedene su pod oznakom *partitions*. Svaka podjela navedena je pod vlastitom oznakom *partition* u kojoj se odjeljci navode u obliku: *nazivOdjeljka:brojUčenika* i odvajaju znakom ';'. Razredi također mogu imati predviđene pauze zadane vremenskim rasponom, a one se tada navode pod oznakom *breakRequirements*. Ukoliko je predmetima zadana složenost, razredima se može zadati i najveći

dopušten zbroj složenosti predmeta u jednom danu (*maximumLectureComplexitySum*), kako bi se upravljalo opterećenjem učenika.

Podaci o predavanjima najznačajniji su među parametrima rasporeda. Svako predavanje zadano je svojim identifikatorom (*id*), identifikatorom predmeta koji se predaje (*subject*), listom identifikatora nastavnika koji održavaju predavanje (*teachers*), listama razreda (*classes*) i odjeljaka (*partitions*) koji pohađaju predavanje i listom predviđenih učionica u koje predavanje može biti raspoređeno (*classrooms*). Naveden je i broj učionica koje su potrebne za pravilno održavanje predavanja (*necessaryClassroomCount*), a koje *Sustav za izradu rasporeda* može izabrati iz liste predviđenih učionica. Uz dosad navedene podatke, zadaje se duljina trajanja predavanja (*lectureDuration*) i lista indeksa tjedana u koje se predavanje može rasporediti (*weeks*). Za svaki od tjedana u kojem se predavanje može održavati zadaje se i po jedna *weekAvailability* oznaka kojom se određuju termini u koje ga je poželjno rasporediti. Zadaje se i podatak ulazi li predavanje u redovni program nastave i treba li biti uključeno u provjeru neprekidnosti rasporeda za razrede (*regularScheduleManagementEnabled*). Valja napomenuti kako nije potrebno navoditi sve podatke o nastavnicima, razredima i predviđenim učionicama za sva predavanja. Ako neka predavanja nemaju zadan razred (npr. informacije koje nastavnik održava roditeljima) ili učionicu (poput škole u prirodi), takve podatke moguće je izostaviti. Važno je samo da svako predavanje ima zadane podatke o barem jednom od ova tri čimbenika.

Naposljetku, veze između predavanja zadane su svojim identifikatorom (*id*), listom identifikatora vezanih predavanja (*lectureIds*), pravilom (*rule*) i uvjetom pravila (*condition*). Pravilo veze govori moraju li ili ne smiju predavanja biti u nekom odnosu, dok uvjet pravila zadaje taj odnos. Tako se, na primjer, mogu zadati veze sa značenjima da predavanja:

- *'moraju/ne smiju biti u istom terminu'*,
- *'moraju/ne smiju biti u istom danu'*,
- *'moraju/ne smiju biti u istom tjednu'* ili
- *'moraju/ne smiju biti uzastopno'*.

2.3.2. Izlazni podaci

Izlazni podaci *Sustava*, odnosno izgrađeni raspored, mogu se pohraniti u dva oblika:

- kao *XML* datoteka ili
- kao *Excel* dokument.

Pritom svaka od ovih datoteka ima svoje prednosti i služi posebnoj namjeni.

XML datoteka služi za sažeto pohranjivanje podataka o rasporedu koji se kasnije mogu jednostavno učitati u *Sustav* i, ukoliko je potrebno, ručno mijenjati. Sastoji se od samo jedne glavne oznake, *lectures*. U njoj su, za svako predavanje navedeno u ulaznoj *XML* datoteci (zadano oznakom *lectureId*), navedeni podaci o njegovom vremenu (oznaka *slot*) i mjestima održavanja (oznaka *locations*, oblikovana kao lista identifikatora učionica u kojima se predavanje održava). Vrijeme održavanja navedeno je kao apsolutni indeks termina unutar cijelog rasporeda, slično onome što je opisano u odjeljku 2.1.

Excel datoteka namijenjena je olakšavanju čitanja rasporeda ljudima. Organizirana je u tri lista gdje su pojedinačno prikazani rasporedi za nastavnike (*Teachers*), razrede (*Classes*) i učionice (*Classrooms*). U svakoj od tri tablice, nastavni satovi navedeni su u stupcima, a sudionici nastave u redcima. Predavanja su u ćelijama navedena identifikatorom predmeta koji se u tom terminu održava. Dodatno, za nastavnike i učionice navodi se i razred kojem se održava predavanje. Tablični prikaz rasporeda predstavljen u *Excel* datoteci pogodan je za ispis ili dodatne provjere i analize. Isječak takve datoteke prikazan je slikom 2.2.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1		ponedjeljak																			utorak
2	836	08:00	08:50	09:50	10:40	11:30	12:20	13:10	14:00	14:50	15:50	16:40	17:30	18:20	08:00	08:50	09:50	10:40	11:30	12:20	13:10
3	Radojka Papić										(info)	6b (srz)									8a (hrv)
4	Ivanka Perdija	5a (srz)		5a (hrv)	7b (hrv)	7b (hrv)	7b (hrv)									7c (hrv)	5a (hrv)	5a (hrv)			
5	Branka Vukadinović										6c (hrv)	8c (hrv)									
6	Morana Strahija	3b (eng	3c (eng																		7b (eng
7	Maja Burazin			4b (eng	4c (eng	7c (eng	7c (eng		6b (eng	8c (eng	8d (eng				4c (eng	7a (engl	2b (eng				
8	Jelena Šumelj		1a (nje	8b 8c	3a (nje	2a (nje		7a (nje									3a (nje	5b (nje	7a (nje	5b (nje	5a (nje
9	Gordana Barišić-Lazar					4a (nje	6a 6b	5b (engl													
10	Maja Poljski			7c (mat	7c (mat												(info)	7b (mat		7a (mat	5b (mat

Slika 2.2: Isječak izlazne *Excel* datoteke (prikaz dijela rasporeda za nastavnike)

2.4. Vrednovanje rješenja

Kvaliteta rješenja, odnosno izgrađenog rasporeda može se vrednovati kroz čitav niz kriterija. Samo neki od njih su sljedeći:

- broj nepoštivanja raspoloživosti (nastavnika, razreda, učionica, predmeta i predavanja),
- broj nepoštivanja neprekidnih rasporeda (za nastavnike i razrede),
- broj nepoštivanja najmanjeg i najvećeg broja radnih sati (nastavnika i razreda),
- broj nepoštivanih veza između predavanja,
- broj broj predavanja raspoređenih u nepredviđene dvorane,
- broj predavanja raspoređenih u pogrešne tjedne,
- broj višesatnih predavanja "prelomljenih" između dva dana (problem opisan u odjeljku 2.2.2),
- broj predavanja istog predmeta koja su raspoređena u isti dan i slično.

Za većinu navedenih kriterija teško je jednoznačno utvrditi koji predstavljaju važnija, a koji manje važna ograničenja. Predstavnici različitih škola u razgovorima su iznosili različite stavove oko važnijih i manje važnih ograničenja koji ih vode prilikom izrade rasporeda. Tako, na primjer, neke škole dozvoljavaju manji broj pauzi u rasporedima učenika, dok je potpuno poštivanje neprekinutog rasporeda osnovni zahtjev za druge.

Iz svih navedenih kriterija mogu se jedino izdvojiti oni koji predstavljaju jača nepoštivanja ulaznih parametara. To su: broj predavanja raspoređenih u pogrešne tjedne i broj višesatnih predavanja "prelomljenih" između dana. Za njih se sa sigurnošću može reći kako predstavljaju čvrsta ograničenja (engl. *hard constraints*)[1]. Raspored koji ih u potpunosti ne zadovoljava ne može se prihvatiti kao zadovoljavajući. Za rasporede koji ne zadovoljavaju ova dva navedena ograničenja možemo reći da ne zadovoljavaju osnovne zahtjeve nastavnog plana i programa. Ipak, takvo rješenje korisniku *Sustava* može ukazati na pogrešku u zadavanju problema ili nemogućnost pronalaska rješenja koje bi ih zadovoljavalo.

Smještaj ostalih kriterija u čvrsta ili meka ograničenja uvelike ovisi o pojedinoj školi i njezinom načinu rada. Stoga je pri izgradnji *Sustava* nužno korisniku pružiti mogućnost određivanja koja ograničenja smatra važnijima. Postoje dva načina vrednovanja rješenja na temelju većeg broja kriterija: *vektorom s* n *članova* i *težinskom sumom*.

[1] čvrsta ograničenja — ograničenja čijim nepoštivanjem raspored postaje neupotrebljiv

Vektor od n članova jednostavan je način prikaza kvalitete rješenja. Svi čimbenici kvalitete organiziraju se u jedan vektor na način da se one važnije nalaze na početku, a one manje važne na kraju vektora. Prilikom usporedbe dvaju rješenja, uspoređuju se vrijednosti njihovih vektora kvalitete, počevši od onih najvažnijih. Kvalitetnije je rješenje ono s manjom vrijednostima važnijih članova. Ako je neki par članova jednake vrijednosti, uspoređuju se članovi jednog stupnja manje vrijednosti. Tako, na primjer, vrijedi sljedećih nekoliko nejednakosti za vektor kvalitete od pet članova (pri čemu znak "manje" ukazuje na bolje rješenje):

$$[0, 1, 72, 6, 8] < [1, 0, 56, 4, 10],$$
$$[0, 0, 12, 0, 13] < [0, 0, 12, 0, 14],$$
$$[0, 0, 15, 0, 7] < [0, 0, 16, 0, 1] \text{ itd.}$$

Iako jednostavan u primjeni, *vektor od n članova* često može predstavljati problem prilikom rješavanja problema s većim brojem ograničenja. Glavni je razlog tome načelo procjene kvalitete koje se ovdje koristi. Na primjer, rješenje koje je nezamjetno lošije u prvom članu, a višestruko bolje u svim ostalima, svejedno će biti procijenjeno lošijim od onog koje ima djelomično bolju vrijednost prvog člana.

Težinska suma rješava navedeni problem. Svakom kriteriju pridružuje se faktor s kojim se izračunata vrijednost množi. Svi se umnošci zatim zbroje, pa je tako kvaliteta rješenja predstavljena samo jednom vrijednošću koja se onda uspoređuje s vrijednostima drugog rješenja. Ukoliko je neko rješenje u samo jednom članu bolje od drugog i ako on nema pridružen previsok faktor, lako je moguće da će kvaliteta ostalih članova prevladati. U nastavku slijedi primjer:

$$(10 \cdot 0) + (8 \cdot 1) + (5 \cdot 72) + (3 \cdot 6) + (1 \cdot 8) = 384 >$$
$$(10 \cdot 1) + (8 \cdot 0) + (5 \cdot 56) + (3 \cdot 4) + (1 \cdot 10) = 312$$

Iz primjera je vidljivo kako je treći član, s manjim faktorom ali značajno nižom vrijednosti, utjecao na odluku. Iako je značajno manje ograničavajuća, problem *težinske sume* je činjenica da zahtijeva dobro procijenjen raspored težinskih faktora.

3. Metode izrade rasporeda

Ovo poglavlje sastoji se od dva odjeljka. U prvom odjeljku opisana su dva načina na koje se može pristupiti rješavanju problema izrade rasporeda i njihova primjena na formalno opisan problem u poglavlju 2. U istom odjeljku opisane su i vrste sustava za izradu rasporeda s obzirom na stupanj sudjelovanja koji očekuju od strane korisnika. Naposljetku se iznose i opisuju složenosti pojednostavljenog i osnovnog problema, kao i mogućnosti rješavanja pojednostavljenog problema. Drugi odjeljak ovog poglavlja opisuje načine kojima se mogu rješavati složeni problemi raspoređivanja.

3.1. Oblici pristupa i složenost

U smislu međudjelovanja sustava s korisnikom, mnogi autori smatraju da problem izrade rasporeda nije moguće u potpunosti automatizirati. Kako bi poduprli takvo stajalište, najčešće iznose dva obrazloženja. Prvo je postojanje čimbenika koji ljudima jasno razlučuju dva rješenja na bolje i lošije, ali ih je vrlo teško ili čak nemoguće izraziti ograničenjima u *automatiziranom* (engl. *automated*) sustavu. Drugi argument iznosi činjenicu da je prostor mogućih rješenja iznimno velik. U takvoj situaciji, ljudski faktor može biti presudan za uspješno usmjeravanje postupka pretraživanja prema kvalitetnijim rješenjima koja automatski sustav možda ne bi pronašao. Zbog oba navedena razloga, gotovo svi danas raspoloživi sustavi korisniku nude neku mogućnost izmjene konačnog rješenja. Neki sustavi od korisnika zahtijevaju puno veći stupanj uključenosti u postupak izrade pa se nazivaju interaktivni (engl. *interactive*) sustavi za izradu rasporeda, ali u ovom je radu opisana tematika automatiziranih sustava.

Kao što je detaljno opisano u poglavlju 2, problem izrade rasporeda nastave čini organizacija unaprijed zadanog broja predavanja unutar zadanog vremenskog raspona (tipično jedan ili dva tjedna), pritom ujedno zadovoljavajući veći broj različitih ograničenja. Ovakav problem izrade rasporeda naziva problem traženja (engl. *search problem*).

U drugim slučajevima rješavanju problema pristupa se kao traženju najboljeg (ili

gotovo najboljeg) rješenja pa se takav problem naziva *problem optimizacije* (engl. *optimization problem*). Od svakog se rasporeda traži da zadovolji sva *čvrsta* ograničenja, a dodatni zadatak je optimiranje (maksimizacija ili minimizacija) funkcije kvalitete koja ovisi o uspješnosti zadovoljavanja *mekih* ograničenja[1]. Tako je Junginger (1986), za rješavanje osnovnog problema optimizacijom, predložio sljedeću funkciju kvalitete:

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

u kojoj se određena vrijednost d_{ijk} dodjeljuje satima k u kojima je nastavniku t_j manje poželjno održavati predavanje razredu c_i. Očito je kako ovakav oblik funkcije kvalitete provjerava samo raspoloživosti nastavnika, dok sva ostala ograničenja opisana u poglavlju 2 ne uzima u obzir. Samim time nije nije primjenjiva na složeni problem izrade rasporeda u hrvatskim školama.

Colorni et al. (1992) iznose nešto složeniju funkciju kvalitete rješenja, koja uključuje nekoliko čimbenika kvalitete rasporeda. Tako autori predlažu praćenje:

- *didaktičke kvalitete* (npr. raspodjela predavanja preko cijelog radnog tjedna i podjednak broj predavanja u danu),
- *organizacijske kvalitete* (npr. pauza za barem jednog nastavnika svaki sat kako bi po potrebi mogao održati zamjenu) i
- *kvalitete osobnih rasporeda* (npr. osiguranje slobodnog dana za nastavnika koji predaje u dvije škole).

Funkcije kvalitete koje u obzir uzimaju više čimbenika, obično i bolje razlučuju dva moguća rješenja i samim time daju bolje rezultate.

Probleme pretraživanja lako je pretvoriti u probleme optimizacije. Prema Schaerf (1999), potrebno je funkciju kvalitete zadati na način da iskazuje udaljenost trenutnog rješenja od onog "zadovoljavajućeg" (koje u potpunosti zadovoljava sva ograničenja). Uočljivo je kako je moguć i obrnut postupak.

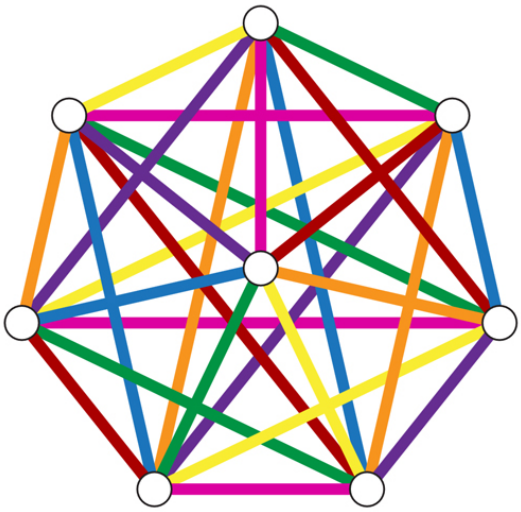
U oba slučaja (pretraživanje i optimizacija) moguće je odrediti temeljni problem. Temeljni problem je problem pronalaska odgovora na pitanje postoji li rješenje (u slučaju pretraživanja) ili postoji li rješenje sa zadanom vrijednošću funkcije kvalitete (u slučaju optimizacije). Kada se spominje složenost određenog problema, zapravo se govori o složenosti njegovog temeljnog problema. Even et al. (1975) su, svođenjem 3-SAT problema (Garey i Johnson, 1979) na osnovni problem iskazan u poglavlju 2,

[1] meka ograničenja — ograničenja koja nije potrebno u potpunosti zadovoljiti, ali njihovo zadovoljavanje pridonosi kvaliteti rješenja

dokazali da je problem izrade rasporeda nastave *NP-potpun* pri gotovo svim varijantama njegova postavljanja. To znači da je najbolje rješenje iscrpnom pretragom moguće pronaći tek za probleme vrlo malog opsega, dok je za sve ostale potrebno koristiti heurističke metode koje ne jamče da će uvijek pronaći optimalno rješenje.

Pojednostavljeni problem, moguće je riješiti brže, svođenjem na rješavanje problema bipartitnog multigrafa. Razrede i nastavnike moguće je predstaviti vrhovima grafa, dok su predavanja predstavljena bridovima. Drugim riječima, razred c_i povezan je s nastavnikom t_j s onoliko paralelnih bridova koliko je zadanih predavanja koja nastavnik t_j održava razredu c_i. Even et al. (1975) opisuje metodu rješavanja problema temeljenu na pronalasku niza najvećih skupova bridova koji nemaju zajedničkih vrhova. Time se zapravo traži najveći broj predavanja koja se mogu održavati istovremeno. Hopcroft i Karp (1971) su dokazali kako je niz najvećih skupova, a time i odgovarajući raspored, moguće pronaći u polinomijalnom vremenu u odnosu na na veličinu grafa. S obzirom na to da je ukupno potrebno pronaći p takvih skupova (p je ukupan broj termina) i da je multigraf također polinomilajan (zbog matrice $X_{m \times n \times p}$), cijela metoda također se može provesti u polinomijalnom vremenu.

Pokazalo se još (de Werra, 1985) kako je problem moguće svesti i na problem bojanja grafa. I u ovom su slučaju nastavnici i razredi predstavljeni vrhovima, a predavanja bridovima grafa. Imajući na raspolaganju p boja (svaki nastavni sat predstavljen je jednom bojom), potrebno je obojiti bridove grafa na način da niti jedan par susjednih bridova nije obojan istom bojom. Prema tome, x_{ijk} poprima vrijednost 1 ukoliko je neki brid između vrhova c_i i t_j obojan bojom k. Takav oblik problema također ima polinomijalnu složenost. Primjer grafa obojenog na ovaj način prikazan je slikom 3.1.



Slika 3.1: Graf u kojem su susjedni bridovi obojeni različitim bojama

3.2. Metode i njihova primjena

Pojednostavljeni problem iz poglavlja 2 ujedno je i jedini oblik problema izrade rasporeda kojeg je moguće riješiti determinističkim metodama u zadovoljavajućem vremenu izvršavanja. Bilo kakav dodatan početni zahtjev svrstava problem u već spomenuti razred *NP-potpunih* problema, koje nije moguće riješiti iscrpnim pretraživanjem. U ovom odjeljku opisane su stohastičke metode kojima se danas najčešće rješava problem izrade rasporeda nastave za osnovne i srednje škole. Na samom početku opisana je izravna heuristika koja oponaša "ručnu" izradu rasporeda. U drugom dijelu odjeljka opisuju se metaheuristike, metode koje nisu usko vezane uz problem koji se rješava.

3.2.1. Izravne heuristike

Izravne su heuristike metode koje u potpunosti oponašaju ljudski način rješavanja problema. Njima se "ručni" posao automatizira računalom. U slučaju izrade rasporeda nastave, uglavnom se sve metode ručne izrade svode na raspoređivanje jednog po jednog predavanja, sve dok je takvo smještanje moguće. Kada više nije moguće pronaći odgovarajući termin za sljedeće predavanje, pristupa se zamjeni termina dvaju ili više već raspoređenih predavanja kako bi se takav termin oslobodio.

Jedna od takvih metoda opisana je u (Junginger, 1986). Opći postupak izrade moguće je opisati u obliku provođenja sljedeća tri postupka:

A: rasporedi *najsloženija* predavanja u termine koji su za njih *najpoželjniji*,

B: kada se u pojedini termin može rasporediti samo jedno predavanje, smjesti to predavanje u navedeni termin i

C: premjesti već raspoređeno predavanje u drugi slobodan termin kako bi se oslobodio termin za novo, još neraspoređeno predavanje.

Predavanje je "složeno" u slučaju kada se na njega odnosi velik broj ograničenja, čime ga postaje teže ispravno rasporediti. Na primjer, to može biti slučaj kada ga održava nastavnik koji nema puno raspoloživih termina, a istovremeno ima mnogo drugih predavanja koja mora održati. S druge strane, termin je "poželjan" kada se, s obzirom na raspoloživosti ostalih razreda i nastavnika, mali broj ostalih predavanja može smjestiti u njega.

Sustav opisan u (Junginger, 1986) raspoređuje predavanja u termine primjenjujući postupke A i B koliko god je to moguće. Kada više nije moguće niti jedno predavanje rasporediti koristeći navedene postupke, dalje se raspoređuje koristeći postupak C.

Postupak A osnova je sustava i korišten je u gotovo svim sustavima koji se temelje na izravnim heuristikama. Razlike su jedino moguće u određivanju složenosti predavanja i poželjnosti termina. Postupak B koristi se kao osiguranje da sustav prilikom primjene postupka A ne dođe do stanja iz kojeg ne može nastaviti s izradom rasporeda. Postupak C najčešće uključuje i neke oblike povratnog praćenja kako bi se mogle ispraviti pogreške do kojih je došlo korištenjem postupka A.

Papoulias (1980) također predlaže jedan oblik izravne heuristike za izradu rasporeda nastave. Autor posebno naglašava potrebu za pravilnom raspodjelom predavanja istog predmeta kroz tjedan. Tako je u izračun poželjnosti pojedinog termina uključen i podatak održava li isti nastavnik istom razredu neko drugo predavanje u tom danu.

3.2.2. Metaheuristike

Kao što je i opisano na praktičnom primjeru u prethodnom odjeljku, heuristike (ili približne metode, prema Čupić (2009)) stohastičke su metode rješavanja složenih problema koje imaju razmjerno nisku računsku složenost. Njihovo je opće svojstvo činjenica da ponovnim pokretanjem postupka nad istim skupom ulaznih podataka ne moramo dobiti jednako rješenje. Također, heuristike ne jamče da će uvijek pronaći optimalno rješenje. Međutim, optimalno rješenje često niti nije nužno. U velikom broju slučajeva prihvatljivo je dovoljno dobro rješenje koje one nude.

Posebna vrsta heuristika su metaheuristike. Dok su osnovne heuristike usko vezane uz problem koji rješavaju, Čupić (2009) opisuje metaheuristike kao skupove algoritamskih koncepata koji se mogu koristiti za zadavanje heurističkih metoda primjenjivih na širok skup problema. Dakle, metaheuristike su heuristike opće namjene koje usmjeravaju pretraživanje prema prostoru u kojem se nalaze dobra rješenja. Njihov velik broj nastao je na temelju promatranja prirodnih procesa. Priroda je jako dobar primjer optimizacije jer je današnji život na Zemlji nastao optimizacijom prvotnih oblika života. Tako su samo neke od poznatijih metaheuristika: *simulirano kaljenje*, *tabu pretraga*, *optimizacija kolonijom mrava*, *optimizacija rojem čestica*, *optimizacija umjetnim imunološkim sustavom* ili *evolucijsko računanje*, koje je detaljnije opisano u idućem poglavlju.

4. Evolucijski algoritmi

Počeci evolucijskog računarstva sežu još u 30-e godine prošloga stoljeća. Unatoč tome, tek je pad cijena elektroničkih računala u 60-ima pozitivno djelovao na razvoj ovog područja. Rasprostranjenošću nove tehnologije omogućeno je korištenje računalne simulacije kao alata za analizu sustava puno složenijih od onih koje je do tada analizirala matematika.

Među znanstvenicima posebno zainteresiranim za ovo područje bili su evolucijski biolozi koji su u njemu vidjeli mogućnost razvoja i proučavanja novih prirodnih evolucijskih sustava i inženjeri iz područja računarske znanosti koji su moć evolucije željeli iskoristiti za razvoj boljih rješenja postojećih problema, uključujući i područja umjetne inteligencije.

Na početku poglavlja navedena je kratka povijest razvoja evolucijskih algoritama. Drugi odjeljak opisuje jednostavan evolucijski sustav na kojem se temelje sve tri grane evolucijskih algoritama: *evolucijske strategije*, *evolucijsko programiranje* i *genetski algoritmi*. Treći odjeljak pobliže opisuje način rada genetskog algoritma. U ovom se odjeljku i detaljnije objašnjavaju osnovni evolucijski operatori koji djeluju u postupku rada genetskog algoritma. Naposljetku se iznose i pobliže opisuju dvije inačice genetskog algoritma koje utječu na brži rad i bolje rezultate prilikom izrade rasporeda nastave za osnovne i srednje škole.

4.1. Povijesni razvoj

Tridesetih godina prošloga stoljeća ideje Sewella Wrighta (opisane u Wright (1932)) pomogle su u popularizaciji evolucijskog računarstva. On je opisao kako evolucijski sustav istražuje prostor s više lokalnih ekstrema[1] i dinamički oblikuje nakupine jedinki oko takvih područja. Time se evolucijski sustav lako može zamisliti kao optimizacijski postupak, što je stajalište koje i danas ima svoje pobornike i protivnike. Unatoč tome, gledanje na evolucijski sustav kao optimizaciju pridonio je razvoju algoritama

[1] tzv. *višemodalni sustav*, prema Golub (2004b)

koji su automatizirali mnoge mukotrpne optimizacijske postupke. Kao rezultat toga, optimizacijski su problemi i danas prevladavajuće područje upotrebe evolucijskog računarstva.

U šezdesetima je započeo intenzivniji razvoj različitih upotreba evolucijskih sustava. Nekoliko grupa ljudi uvidjelo je kako se i jednostavni evolucijski modeli mogu predstaviti u računalnom obliku i iskoristiti za rješavanje složenih problema.

Na Tehničkom sveučilištu u Berlinu, prema De Jong (2006), Rechenberg i Schwefel počeli su oblikovati ideju kako bi evolucijski postupci mogli biti iskorišteni za rješavanje optimizacijskih problema s realnim parametrima (Rechenberg, 1965). Iz tih ideja nastale su evolucijske strategije, jedna od podskupina evolucijskih algoritama koja i danas predstavlja najmoćniji alat za optimiziranje funkcija.

Na UCLA-u je u isto vrijeme Fogel uočio potencijal evolucije u rješavanju problema iz područja umjetne inteligencije (Fogel et al., 1966). Njegove ideje bile su vezane uz korištenje evolucijskih sustava za unaprjeđenje inteligentnih agenata, koji su bili oblikovani kao strojevi s konačnim brojem stanja. Tada razvijen programski okvir, nazvan "evolucijsko programiranje", unaprjeđuje se i danas, a osim evolucije strojeva s konačnim brojem stanja koristi se i za rješavanje mnogih drugih problema.

Holland je, na sveučilištu u Michiganu, u evolucijskom sustavu vidio ključ oblikovanja samoprilagođavajućih sustava sposobnih za snalaženje u stohastičkom okruženju (što je opisao u Holland (1962) i Holland (1967)). On je naglasio i važnost potrebe da se takvi sustavi s vremenom prilagođavaju svojoj okolini na temelju informacija koje od nje primaju. To je dovelo do nove podskupine evolucijskih algoritama nazvane reproduktivni planovi, koji čine osnovu današnjih genetskih algoritama.

Sva ta rješenja bila su vrlo idealizirani modeli evolucijskih postupaka u prirodi. Cilj nije bio oblikovati vjerne kopije bioloških postupaka, nego prilagoditi ostvarenje na način da se oblikuju one sastavnice koje su važne za rješavanje problema. Kao posljedica toga danas većina evolucijskih algoritama radi s prilično jednostavnim pretpostavkama (utvrđena veličina populacije, jednospolne jedinke, statički okoliš kvalitete i druge, o kojima će više riječi biti u sljedećim odjeljcima).

4.2. Jednostavan evolucijski sustav

Prije navođenja detalja i opisa primjena evolucijskih algoritama, potrebno je shvatiti osnovne pojmove vezane uz svaki evolucijski sustav.

Charles Darwin je, kako piše Čupić (2009), teoriju evolucije predstavio iznijevši pet pretpostavki na kojima se temelje i evolucijski algoritmi:

– plodnost vrsta — potomaka ima uvijek više nego što je potrebno,

– veličina populacije je približno stalna,

– količina hrane je ograničena,

– kod vrsta koje se seksualno razmnožavaju nema istovjetnih jedinki, nego postoje varijacije i

– najveći dio varijacija prenosi se nasljeđivanjem.

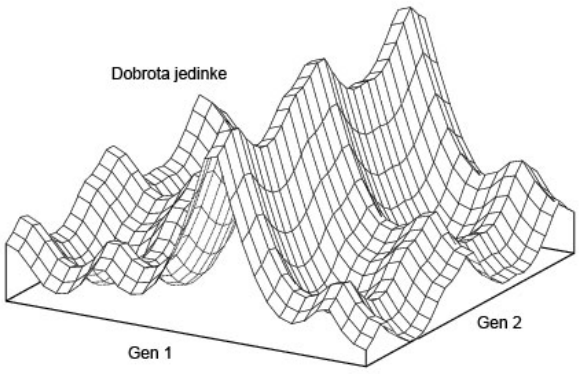
Iz navedenih razloga u svakoj će populaciji postojati borba za preživljavanje — ukoliko je količina hrane ograničena, preživjet će i razmnožavati se samo one jedinke koje će biti dovoljno jake da se izbore za hranu.

Osnovno pitanje koje se postavlja je: na koji način predstaviti jedinku (ili organizam) koja će biti dio populacije? Najjednostavniji način bio bi odrediti L osobina od kojih bi svaka bila izabrana zbog svojeg svojstva da pomogne pri određivanju kvalitete jedinke — njezine dobrote (engl. *fitness*). Kasnije će biti prikazano kako će taj faktor utjecati na sposobnost opstanka jedinke u populaciji.

Svaka jedinka bit će predstavljena vektorom osobina sa sebi svojstvenim vrijednostima. Na primjer:

<*visina, težina, boja kože, boja očiju, boja kose*>.

Navedeni vektor možemo zamisliti kao skup genetskih osobina (*genotip*), vanjskih, stečenih osobina (*fenotip*) ili kao njihovu kombinaciju. On razapinje pet-dimenzionalni prostor značajki u kojem svakoj točki možemo pridružiti neku vrijednost, vrijednost dobrote za jedinku koja poprima takve značajke. Takav prostor zove se *okoliš dobrote*, a grafički prikaz prostora dobrote za genotip od dvije značajke predstavljen je na slici 4.1.



Slika 4.1: Okoliš dobrote za dvije značajke

Na temelju tako opisanih pojmova, moguće je odrediti zakone kretanja evolucijskog sustava odnosno osnovni evolucijski algoritam, koji je predstavljen algoritmom 4.1.

Važno je uočiti kako je algoritam stohastički, odnosno da će, kao i ostale metaheuristike, prilikom više pokretanja s istim ulaznim podacima davati različite rezultate. To ujedno može biti i pozitivno i negativno. Pozitivno može biti u smislu testiranja jer se brzo može ispitati ponašanje sustava u različitim uvjetima. S druge strane, ukoliko dođe do pogreške prilikom izvođenja, vrlo će je teško biti ponoviti kako bi se utvrdio i ispravio uzrok. Upravo zbog nedeterminizma, važno je ne donositi zaključke o ponašanju algoritma samo na temelju jednog ili dva pokretanja.

Algoritam 4.1 Osnovni evolucijski algoritam

Stvori početnu populaciju od M jedinki

- koristeći jednoliku razdiobu nad cijelim prostorom značajki, te izračunaj vrijednost dobrote za svaku jedinku

ponavljaj

Odaberi jednog člana populacije za roditelja

- koristeći jednoliku razdiobu nad cijelom populacijom (svaka jedinka ima jednaku vjerojatnost da bude izabrana)

Koristeći genotip izabranog roditelja, stvori dijete

- na način da kloniraš roditelja i nad tako stvorenom jedinkom provedi mutaciju

Izračunaj dobrotu stvorene jedinke

Izaberi člana populacije koji će odumrijeti

- na način da nasumično (jednolikom razdiobom nad cijelom populacijom roditelja) izabereš jednog kandidata iz roditeljske populacije koji će se boriti za opstanak sa stvorenom jedinkom. U roditeljskoj populaciji ostaje jedinka s većom dobrotom.

kraj ponavljanja

Mutacija u ovom kontekstu predstavlja izmjenu jednog gena (jedne vrijednosti u vektoru osobina). Izbor gena za mutaciju provodi se nasumično, pri čemu svaki gen ima jednaku vjerojatnost da bude izabran, a postupak mutiranja predstavlja mala promjena vrijednosti gena. Na primjer, ako bi vrijednosti bile realni brojevi, mutacija bi mogla biti dodavanje ili oduzimanje nekog iznosa δ.

Vidljivo je iz primjera kako evolucijski algoritam ne pretražuje temeljito cjelokupni prostor značajki, nego "skače" iz točke u točku provođenjem mutacije nad jedinkama

populacije. Međutim, s obzirom da se ovakav sustav (kao i ostale metaheuristike) koristi za rješavanje izrazito složenih problema u kojima iscrpna pretraga nije primjenjiva, ovakvo ponašanje je upravo poželjno. Kako se nova jedinka nakon stvaranja odmah natječe za mjesto u populaciji s jednom od slučajno izabranih članica i u populaciji ostaje "jača", srednja se vrijednost dobrote populacije svakim korakom algoritma približava jednom od vrhova okoliša dobrote.

4.3. Genetski algoritam

Promatrajući pseudokod jednostavnog evolucijskog sustava opisanog u prethodnom odjeljku, uočljivo je nekoliko značajnih nedostataka. Na primjer, kako se svaki put nasumično odabire roditelj na temelju kojeg će se stvoriti nova jedinka, moguć je slučaj u kojem se najbolja ili iznimno dobra jedinka u populaciji nikad ne izaberu za razmnožavanje. Također, kako se novo dijete natječe za opstanak s jednim trenutnim pripadnikom populacije, moguće je i da neka loša jedinka opstane jer je "ždrijeb" nikad nije izabrao za natjecanje.

Ukratko, u osnovnom evolucijskom sustavu dobrota jedinke važna je samo pri opstanku jedinke u populaciji, ali ne i prilikom odabira jedinki za razmnožavanje (jednom kad jedinka postane dio populacije, ima jednaku mogućnost za razmnožavanje kao i sve ostale). Jedinke u populaciji osnovnog evolucijskog sustava izumiru tek kada se odaberu za natjecanje i budu zamijenjene novom jedinkom s većom dobrotom. U takvim populacijama nema prirodnog starenja, a vrijednost prosječne dobrote cijele populacije jednoliko je rastuća odnosno padajuća funkcija koja se kreće prema jednom od optimuma okoliša dobrote. Važno je pritom naglasiti da prosječna dobrota teži jednom od optimuma, a to može predstavljati problem. Dopuštajući jedinkama da žive beskonačno dugo smanjuje se raznolikost populacije. Unutar nje postupno prevladava jedna "obitelj" jedinki koja ne mora nužno težiti globalnom optimumu okoliša dobrote, a naposljetku će cijela populacija biti "zarobljena" u lokalnom optimumu.

Genetski algoritam prilično uspješno rješava navedene probleme, što ga čini jednom od najčešće korištenih metaheuristika.

4.3.1. Osnovni algoritam

Poput ostalih evolucijskih algoritama, i genetski algoritam inspiriran je postupkom prirodne evolucije. Od osnovnog evolucijskog algoritma razlikuje se prije svega načinom stvaranja novih jedinki. Dok se nove jedinke u osnovnom evolucijskom algoritmu stva-

raju klonirajući i mutirajući roditeljsku jedinku, u genetskom se algoritmu sve jedinke razmnožavaju spolno, kombiniranjem genetskog materijala dvaju ili više roditelja. Taj postupak zove se križanje (engl. *crossover*). Zadržan je i postupak mutacije koji u ovom slučaju oponaša djelovanje vanjskih čimbenika na samostalnu jedinku.

Prethodno naveden problem mogućeg neograničenog života jedinki u osnovnom evolucijskom algoritmu može se riješiti na dva načina. Jedan od njih je da se s vremena na vrijeme dopusti da nova jedinka s nižom dobrotom zamijeni stariju jedinku, boljih svojstava. Evolucijski modeli u kojima se u jednom koraku izmjenjuju jedna ili nekoliko jedinki nazivaju modeli *stacionarnog stanja* (engl. *steady state*). Drugi način ima drugačiji pristup, a odnosi se na to da se cijela roditeljska populacija u svakom koraku u potpunosti zamijeni populacijom djece. Takav model zove se *generacijski*, a primjenjuju ga genetski algoritam i neke vrste (μ, λ) *evolucijskih strategija* (više o generacijskim genetskim algoritmima u Holland (1975), De Jong (1975) i Goldberg (1989)).

Uz navedeno, genetski algoritam primjenjuje i pravi biološki značaj dobrote jedinke, a to je sposobnost jedinke da preživi i razmnožava se. Pri odabiru jedinki za razmnožavanje u genetskom se algoritmu u obzir uzima dobrota, na način da će jedinka s većom dobrotom češće biti izabrana za stvaranje potomstva. Pseudokod s primjerom generacijskog genetskog algoritma prikazan je algoritmom 4.2.

Algoritam 4.2 Primjer generacijskog genetskog algoritma s križajnjem dvaju roditelja

$P \leftarrow$ populacija n nasumičnih početnih rješenja
dok ($!uvjetZavrsetka$) **radi**
  $P_n \leftarrow \emptyset$
  za $i = 1$ do n **radi**
    $r_{i1}, r_{i2} \leftarrow$ odabir dva roditelja proporcionalno njihovoj kvaliteti (bolje jedinke imaju veće šanse za stvaranje potomstva)
    $d_i \leftarrow$ *krizanje*(r_{i1}, r_{i2})
    $d_i \leftarrow$ *mutacija*(d_i)
    $P_n \leftarrow P_n \bigcup d_i$
  završi za
  $P \leftarrow P_n$
završi dok
vrati najbolju jedinku iz P

Lako je uočiti kakve posljedice za sobom povlače novosti uvedene u genetskom algoritmu. Kako se cijela postojeća populacija zamjenjuje novom, funkcija srednje

vrijednosti dobrote populacije više neće biti jednolika nego će oscilirati. Takvo ponašanje je poželjno, jer algoritam čini otpornijim na zastoje u lokalnim optimumima.

4.3.2. Evolucijski operatori

Evolucijski su operatori "alati" evolucije. Pomoću njih se populacija mijenja i napreduje. Glavni su evolucijski operatori: *selekcija*, *križanje* i *mutacija*. U ovom je odjeljku detaljno opisano njihovo djelovanje.

Selekcija

Prethodno je navedeno kako genetski algoritam pri odabiru jedinki za razmnožavanje uzima u obzir njihovu dobrotu s ciljem da bolje jedinke više sudjeluju u stvaranju nove populacije. Postupak takvog odabira naziva se *selekcija*. Veća vjerojatnost odabira boljih jedinki za razmnožavanje zajedničko je svojstvo svih vrsta selekcije. Postupci se međusobno razlikuju prema načinu odabira boljih jedinki. Prema Golub (2004a), ovisno o metodi odabira jedinki koje će se razmnožavati i tako stvoriti nove jedinke, selekcijske postupke možemo podijeliti na: *proporcionalne* i *rangirajuće*.

Proporcionalne selekcije odabiru jedinke s vjerojatnošću koja je proporcionalna njihovoj dobroti. Tako je vjerojatnost odabira pojedine jedinke ovisna o kvocijentu dobrote jedinke i prosječne dobrote svih jedinki populacije. S druge strane, vjerojatnost odabira kod rangirajućih selekcija ovisi o položaju jedinke u poretku jedinki sortiranih po dobroti. One se dodatno dijele na *sortirajuće* i *turnirske* selekcije.

Važan pojam vezan uz postupak selekcije je selekcijski pritisak. On predstavlja odnos preživljavanja dobrih i loših jedinki. Označimo s p_i odnosno p_j vjerojatnosti odabira jedinki i i j. Jedinka i bolja je od jedinke j, pa tako vrijedi $p_i > p_j$. Selekcijski pritisak je veći što je kvocijent $\frac{p_i}{p_j}$ veći. Što je selekcijski pritisak veći, to će se češće birati bolje jedinke, a iz populacije će se brže uklanjati one lošije. Velikog je utjecaja na rad genetskog algoritma i kvalitetu rješenja. Visok selekcijski pritisak potaknut će bržu konvergenciju algoritma boljim rješenjima, ali će žrtvovati kvalitetu najboljeg pronađenog rješenja (mogućim izazivanjem prerane konvergencije u lokalni optimum). S druge strane, preniskim selekcijskim pritiskom troši se vrijeme na nepotrebne korake algoritma, a sam postupak presporo konvergira. U nastavku odlomka slijedi opis nekoliko različitih postupaka selekcije.

Jednostavna proporcionalna selekcija, kako joj i samo ime kaže, jedan je od najjednostavnijih postupaka selekcije. Sve jedinke postavljaju se na zamišljeno kolo ruleta na način da je veličina koju jedinka zauzima na obodu kola proporcionalna njezinoj

dobroti. Što je jedinka veće kvalitete, zauzet će veći dio kola. Kolo se zatim okrene i odabire se ona jedinka na čijem se je području oboda kolo zaustavilo. Ukoliko u populaciji imamo ukupno n jedinki i dobrotu i-te jedinke označimo s d_i (uz uvjet: $\forall i\ d_i \geq 0$), vjerojatnost p_i odabira i-te jedinke iznosi:

$$p_i = \frac{d_i}{\sum_{j=1}^{n} d_j}.$$

Iako jednostavan za ostvarenje (ne i za izvršavanje), ovaj oblik selekcije vrlo je osjetljiv na skalu. S obzirom da je vjerojatnost odabira jedinke izravno povezana s vrijednostima dobrote drugih jedinki, problem nastaje ukoliko se u populaciji nalaze jedinke s visokim vrijednostima dobrote. U takvim slučajevima, vjerojatnosti odabira najbolje i najlošije jedinke u populaciji postaju vrlo slične i selekcija prelazi u slučajan odabir, pri kojem nema selekcijskog pritiska, a time i genetski algoritam prelazi u slučajno pretraživanje.

Među rangirajućim selekcijama posebno je pogodna linearno sortirajuća selekcija. Kod ove metode vjerojatnost selekcije jedinke proporcionalna je njezinom položaju u poretku jedinki sortiranih prema dobroti. Tako se vrijednost odabira pojedine jedinke može izračunati prema izrazu:

$$p_i = \frac{i}{\sum_{i=1}^{n} i} = \frac{2i}{n(n+1)},$$

pri čemu se najbolja jedinka nalazi na položaju N, a najgora na položaju 1. Linearna sortirajuća selekcija nije osjetljiva na skalu i, iako veće računske složenosti od proporcionalne, razmjerno je lagana za programsko ostvarenje pa se i često koristi u radu algoritma.

k-turnirske selekcije još su jedan oblik selekcije. Pogodne su za programsko ostvarenje, ne zahtijevaju prethodno sortiranje populacije i vrlo im je lako mijenjati selekcijski pritisak. Prilikom k-turnirske selekcije, iz populacije se slučajnim odabirom izdvaja k jedinki. Među njima se zatim bira najbolja. Ukoliko je potrebno više roditeljskih jedinki za razmnožavanje, k-turnirska selekcija pokreće se više puta.

Selekcijski se pritisak kod k-turnirske selekcije mijenja veličinom turnira. Što je veći turnir, veći je pritisak. Povezanost veličine turnira i selekcijskog pritiska vrlo je lako prepoznati: ukoliko se u turniru bira samo najbolja jedinka, veći broj jedinki u turniru kroz veći broj pokretanja ujedno znači veću vjerojatnost odabira upravo najboljih jedinki. U praksi se tako najčešće koriste 2- i 3-turnirske selekcije.

Mutacija

Mutacija je evolucijski operator koji ima zadaću očuvati raznolikost genetskog materijala unutar populacije. Ona djeluje na jednu jedinku, mijenjajući je u određenom postotku. U prošlom odjeljku opisan je jedan način djelovanja operatora mutacije na jedinku. Djelovanje mutacije uvelike ovisi o obliku u kojem se jedinka predstavlja u memoriji računala (o njezinom *kromosomu*).

Osnovni način zapisa jedinke je u obliku *binarnog kromosoma*. Svi podaci koji opisuju jedinku organizirani su kao niz nula i jedinica. Ovaj način zapisa pogodan je za rad genetskog algoritma jer se uvelike pojednostavljuje postupak djelovanja evolucijskih operatora (neovisno koji problem se rješava, kromosomi imaju jednaku strukturu). Ali ovakav način zapisa prati više problema. Prvi je nužnost oblikovanja funkcija za kodiranje i dekodiranje korisnih podataka iz kromosoma što, ovisno o problemu koji se rješava, može biti prilično složeno. Drugi, možda važniji problem je nemogućnost kontrole valjanosti rješenja. Naime, velika je vjerojatnost da će se provedbom promjena nad binarnim zapisom stvoriti neispravno rješenje, čime se usporava rad algoritma.

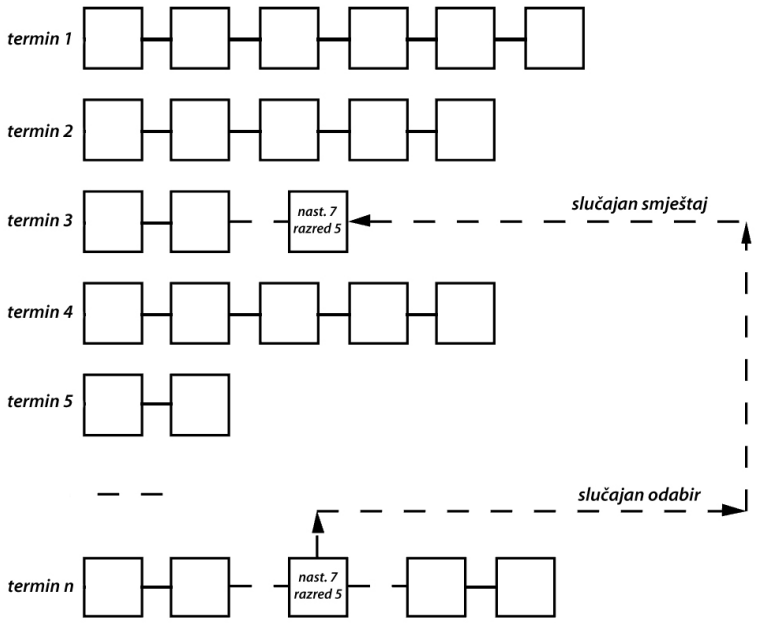
Za razliku od binarnog, *hibridni* zapis kromosoma sastavljen je od složenijih tipova i struktura podataka. Takav zapis usko je vezan uz problem koji se rješava i nije primjenjiv prilikom rješavanja drugih problema. Kromosom kao brojčani vektor, spomenut u prethodnom odjeljku, klasičan je primjer takvog zapisa. Hibridni zapis kromosoma čitljiviji je od binarnog i omogućuje laku provjeru valjanosti rješenja. S obzirom na to da je struktura hibridnog kromosoma ovisna o problemu koji se rješava, takav oblik zapisa zahtijeva i prilagođavanje metoda evolucijskih operatora.

Operator mutacije koji djeluje na binarni kromosom može se svesti na jednostavnu izmjenu pojedinih vrijednosti bitova, kao što je i prikazano na slici 4.2. Naravno, ovo nije jedini način na koji se on može mutirati. Moguća je zamjena bitova na pojedinim mjestima, umetanje pojedinog bita s nekog drugog mjesta u kromosomu i slično.

Slika 4.2: Mutacija jednog bita

Kod hibridnog zapisa kromosoma postupak je malo drugačiji. Uzmimo za primjer hibridni zapis kromosoma u kojim je predstavljen školski raspored. Raspored je predstavljen kao niz termina, a svaki termin sadrži listu predavanja koja se održavaju u to

vrijeme. Očito je kako „izmjena“ jednog predavanja po uzoru na izmjenu jednog bita u binarnom kromosomu ovdje nije moguća. Umjesto toga, djelovanje operatora mutacije moglo bi se predstaviti kao postupak premještanja nasumično odabranog predavanja u nasumično odabrani termin. Može se uočiti i prednost hibridnog zapisa kromosoma: prilikom pokušaja premještanja predavanja može se provjeriti smije li uopće navedeno predavanje biti smješteno u odabrani termin. Ukoliko ne može, do te promjene neće doći nego će se odabrati drugi termin ili predavanje za premještanje. Postupak mutacije hibridnog kromosoma prikazan je na slici 4.3.



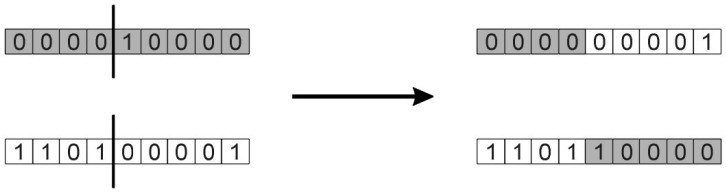
Slika 4.3: Mutacija rasporeda

Križanje

Na početku odjeljka spomenuto je kako se razmnožavanje jedinki u genetskom algoritmu provodi spolno, kombinirajući genetski materijal dvaju ili više roditelja. Stvaranje nove jedinke na taj način naziva se *križanje* (engl. *crossover*). Ono predstavlja vrlo važnu sastavnicu genetskog algoritma, s obzirom na to da jednospolno razmnožavanje rezultira raznovrsnijim jedinkama. Cilj križanja je, kombinirajući dijelove više različitih genetskih kodova, stvoriti potomka koji će kvalitetom nadmašiti svoje roditelje.

U slučaju binarnog kromosoma, križanje se može provesti na nekoliko načina. Jedan je od njih križanje s jednom točkom prekida (engl. 1 *point crossover*). Postupak se izvodi na način da se nasumično odabere položaj oko kojeg će se binarni niz svakog kromosoma podijeliti na dva dijela. Zatim se lijevom dijelu prvog kromosoma pridru-

žuje desni dio drugog i obrnuto, čime se dobivaju dvije nove jedinke. Postupak takvog križanja prikazan je na slici 4.4. Na sličan način može se izvesti i križanje oko više točaka (engl. 2 *point crossover,* n *point crossover*).



Slika 4.4: Križanje oko jedne točke

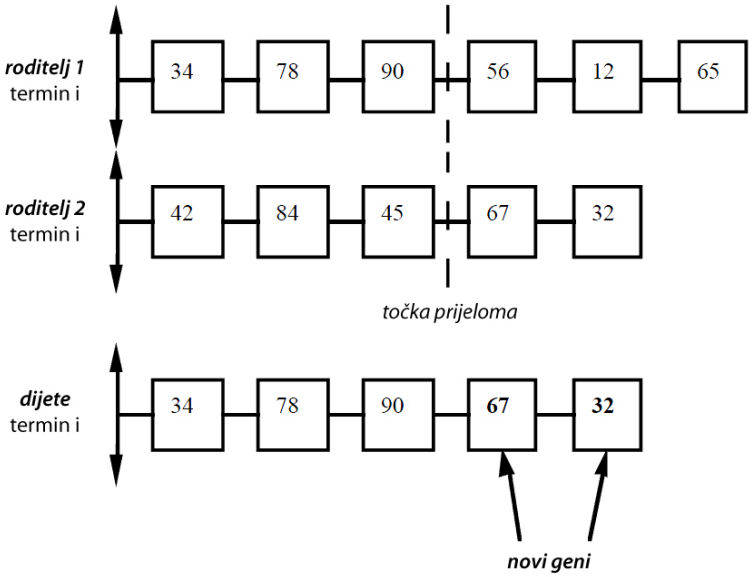
Još jedan oblik križanja binarnih kromosoma je uniformno križanje (engl. *uniform crossover*). Ono se provodi na način da se, uz kromosome roditelja, izgradi još jedan binarni vektor R jednake duljine. Bitovi koji su kod oba roditelja jednakih vrijednosti, prenose se na dijete. U slučaju da se bitovi roditelja razlikuju, na dijete se prenosi vrijednost zapisana u vektoru R. Ukoliko roditeljske kromosome zapišemo kao binarne vektore A i B, uniformno se križanje može opisati sljedećim izrazom:

$$DIJETE = AB + R(A \oplus B).$$

Kod hibridnog kromosoma, postoji cijeli niz načina na koje se može provesti križanje. U slučaju rješavanja problema izrade školskog rasporeda, Abramson (1991) predlaže postupak križanja sličan križanju s jednom točkom prekida kod binarnog kromosoma. Nad listama predavanja odabere se jedna točka prekida. Dalje se za svaki termin liste predavanja križaju poput binarnog kromosoma, kao što je i prikazano na slici 4.5.

4.4. Očuvanje važnih genetskih značajki

U prošlom je odjeljku opisan osnovni genetski algoritam koji na općenit način oponaša postupke prirodne evolucije. S obzirom na to da osnovna primjena genetskog algoritma nije simulacija prirodnih postupaka nego rješavanje problema optimizacije, postavlja se pitanje: može li se na neki način njegov postupak dodatno poboljšati? U ovom se odjeljku opisuju dvije inačice genetskog algoritma koje su posebno prilagođene s ciljem očuvanja genetskih značajki koje čine dobra rješenja. Oba algoritma dovoljno su poopćena da se mogu primijeniti na rješavanje bilo kojeg optimizacijskog problema.



Slika 4.5: Križanje dvaju rasporeda

4.4.1. Algoritam odabira potomaka (OS)

Algoritam odabira potomaka (engl. *offspring selection - OS*) prilagodba je osnovnog genetskog algoritma koja uvodi dodatnu selekciju potomaka nakon postupaka križanja i mutacije. Takav oblik selekcije provjerava uspješnost provedenih operacija. Kako bi se osigurao napredak algoritma s većim udjelom uspješnih jedinki, od operatora križanja i mutacije zahtijeva se stvaranje dovoljnog broja potomaka koji će u nekom obliku nadmašiti kvalitetu svojih roditelja. Iz tog se razloga uvodi novi parametar nazvan *omjer uspješnosti* (engl. *sucess ratio,* $SuccRatio \in [0, 1]$). Omjer uspješnosti zadan je kao postotak članova nove populacije koji su nastali uspješnim razmnožavanjem svojih roditelja.

Kako bi opisali pojam uspješnog razmnožavanja, Affenzeller i Winkler (2009) uvode drugi parametar, *faktor usporedbe* (engl. *comparison factor,* $CompFactor \in [0, 1]$). Prag kvalitete koji nova jedinka mora nadmašiti nalazi se između vrijednosti dobrote njezinih roditelja. Ukoliko je vrijednost faktora usporedbe jednaka 0, nova jedinka mora biti bolja od lošijeg roditelja. Ukoliko je vrijednost 1, tada jedinka kvalitetom mora nadmašiti oba roditelja. Autori predlažu da se faktor usporedbe mijenja tijekom izvođenja algoritma, postupno povećavajući njegovu vrijednost. Na takav se način postiže učinak široke pretrage na početku i sve više usmjerene prema kraju rada algoritma. Ova ideja vrlo je slična načelu rada metaheuristike nazvane *simulirano kaljenje*. Detaljniji opis simuliranog kaljenja i primjena algoritma na rješavanje problema izrade rasporeda nastave navedeni su u Pribil (2011).

Nakon što je nova populacija popunjena zadanim udjelom ($SuccRatio$) uspješnih jedinki, ostatak populacije ($(1 - SuccRatio) \cdot |POP|$, pri čemu je $|POP|$ veličina populacije) dopunjuje se jedinkama nasumično izabranim iz skupa lošijih jedinki ili stvaranjem novih neovisno o njihovoj kvaliteti. Skup lošijih jedinki sadrži jedinke koje su također stvorene u postupku razmnožavanja, ali nisu bile dovoljno uspješne (s obzirom na faktor usporedbe $CompFactor$). *Stvarni selekcijski pritisak* (engl. *actual selection pressure,* $ActSelPress$) računa se kao omjer broja jedinki koje su stvorene (da bi se postigao zadani udio uspješnih jedinki) i ukupnog broja jedinki u populaciji. Taj omjer prikazan je sljedećim izrazom:

$$ActSelPress = \frac{|POP_{i+1}| + |POOL_i|}{|POP_i|},$$

pri čemu $|POOL_i|$ označava veličinu skupa lošijih jedinki.

Najveći selekcijski pritisak (engl. *maximum selection pressure,* $MaxSelPress$) određuje najveći broj jedinki koje se u jednom koraku algoritma mogu stvoriti. Ova vrijednost ujedno može poslužiti i kao dobar pokazatelj za kraj izvođenja algoritma. Naime, činjenica da niti nakon $MaxSelPress \cdot |POP|$ novih stvorenih jedinki nije prikupljeno $SuccRatio \cdot |POP|$ jedinki koje kvalitetom nadmašuju svoje roditelje, ukazuje da daljnjim izvođenjem algoritam neće značajnije napredovati. Pseudokod algoritma odabira potomaka naveden je algoritmom 4.3.

4.4.2. Algoritam očuvanja važnih alela (RAPGA)

U jednom koraku generacijskog genetskog algoritma, sve se jedinke populacije zamjenjuju novima, čime nastaje nova populacija. S obzirom na tu činjenicu, može se postaviti pitanje koji se dijelovi genetskih zapisa iz populacije i trebaju očuvati u generaciji $i + 1$ i na koji se način, s obzirom na već postojeće mehanizme selekcije, križanja i mutacije, to može postići. *Algoritam očuvanja važnih alela* (engl. *Relevant Alleles Preserving Genetic Algorithm, RAPGA*) kroz generacije pokušava očuvati što veću raznolikost populacije, istovremeno zadržavajući što je više moguće genetskog zapisa.

Osnovna je zamisao algoritma mijenjanje veličine populacije kroz korake, ovisno o raspoloživosti jedinki koje doprinose raznolikosti. Nove jedinke prihvaćaju se kao novi članovi populacije ukoliko svojom kvalitetom nadmašuju kvalitetu svojih roditelja ili ako u populaciju uvode, s obzirom na trenutno stanje, nove genetske zapise. U smislu nadmašivanja kvalitete roditelja, primjenjuje se isto načelo kao i kod algoritma selekcije potomaka (koristeći dinamički *faktor usporedbe* $CompFactor$). Nadalje, smatra

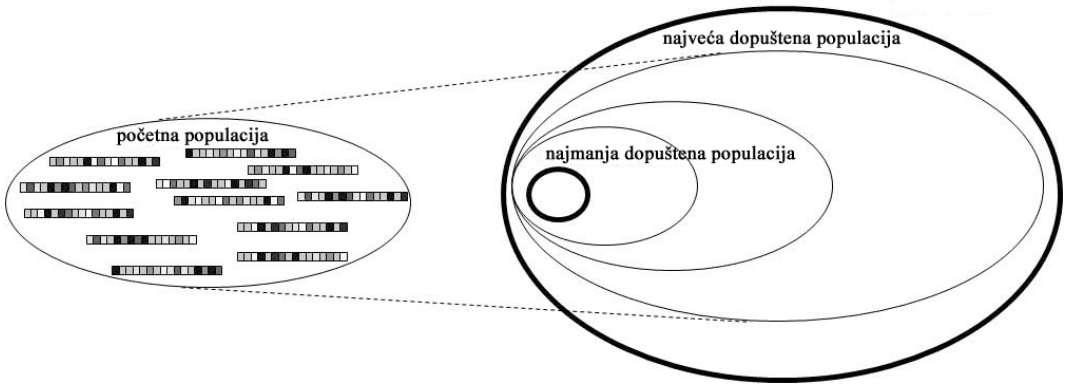
se za jedinku da u populaciju uvodi nove genetske zapise ukoliko je njezin kromosom sastavljen od dijelova koji do tada u tom obliku nisu bili prisutni u populaciji. Sve dok se mogu stvoriti "uspješne" ili "nove" jedinke, populaciji je dozvoljeno da naraste do veličine koja je određena parametrom $MaxPop$. Jedinke koje ne zadovoljavaju navedene zahtjeve jednostavno se odbacuju. Cjelokupni pseudokod algoritma prikazan je algoritmom 4.4.

Kako bi algoritam radio stabilno i stvarao dobre rezultate, prilikom njegovog programskog ostvarenja potrebno je razmotriti sljedeće mogućnosti i čimbenike.

- Algoritam bi na raspolaganju trebao imati nešto drugačije postupke selekcije roditelja u odnosu na standardni genetski algoritam, u smislu da se odabir oba roditelja ne provodi nužno istim selekcijskim postupkom. Dva različita postupka selekcije nazivaju se *muška* i *ženska* selekcija. U mnogim primjerima spominje se kako se korištenjem proporcionalne i nasumične selekcije uspijevaju postići zadovoljavajući rezultati. Također, s obzirom na opisani postupak rada algoritma, razumno je i u potpunosti isključiti selekciju (koristeći nasumičnu selekciju za oba roditelja) jer sam postupak prihvaćanja samo određenih novih jedinki predstavlja dovoljan selekcijski pritisak.
- S obzirom na činjenicu da se nove jedinke prihvaćaju u populaciju samo ako su raznolike i dovoljno kvalitetne, ima smisla u rad algoritma uključiti veći broj različitih operatora križanja i mutacije, čak i neovisno o njihovoj ukupnoj uspješnosti. S obzirom da će sve jedinke biti dodatno provjerene prije ulaska u novu populaciju, moguće je korištenje operatora koji u prosjeku ne daju dobre rezultate, ali s ponekad mogu stvoriti kvalitetnu jedinku. To povećava ukupan broj jedinki koje će se stvoriti u jednom koraku algoritma, a može uvelike pomoći u proširenju prostora pretrage algoritma i time smanjiti mogućnost prerane konvergencije. Također, ukoliko se koristi veći broj operatora pojedine vrste, preporuča se njihov nasumični odabir, a ne sklonost onim uspješnijima.
- Kao što je i prikazano na slici 4.6, u cilju optimalnog rada algoritma, preporučljivo je zadati najveću ($MaxPop$) i najmanju ($MinPop$) vrijednost veličine populacije. U slučaju kada se ne zadaje gornja granica, veličina populacije bi se vrlo lako nepotrebno napuhala, posebno u prvih nekoliko koraka. Donja granica od najmanje 2 jedinke potrebna je zbog poštivanja osnovnih načela na kojima se temelji rad ovog algoritma (dva različita roditelja potrebna su za stvaranje nove jedinke). Unatoč tome, zbog veće raznolikosti i boljeg rada algoritma, preporučljivo je donju granicu postaviti na vrijednost veću od 2.

– Ovisno o problemu koji se rješava i načinu zapisa kromosoma, postoji nekoliko mogućnosti provjere unosi li nova jedinka vrijedne genetske zapise u populaciju. Najbolji način svakako je potpuna usporedba kromosoma nove jedinke s kromosomima ostalih jedinki trenutno u populaciji. Ipak, takva usporedba može biti vrlo zahtjevna i samim time neprihvatljiva. Druga je mogućnost usporedba jedinki na temelju vrijednosti njihovih dobrota. Dvije jedinke koje imaju istu vrijednost dobrote, smatraju se jednakima. Međutim, takav način usporedbe može mnogo različitih jedinki odbaciti kao jednake. S obzirom na to, gdje god je moguće, preporučljivo je koristiti potpunu usporedbu kromosoma.

– Kako bi se otkrio slučaj u kojem više nije moguće pronaći nove i uspješne jedinke za popunjavanje nove populacije, potrebno je uvesti parametar koji će određivati najveći trud koji je u takvu potragu moguće uložiti ($MaxEffort$). Taj parametar određuje najveći broj jedinki koje će se stvoriti prilikom izgrađivanja nove populacije (neovisno o tome hoće li one biti prihvaćene u populaciju ili ne).



Slika 4.6: Prikaz promjene veličine populacije kod algoritma RAPGA

Algoritam 4.3 Algoritam odabira potomaka (OS)

inicijaliziraj ukupan broj iteracija $brojIteracija \in N$
inicijaliziraj broj trenutne iteracije $i = 0$
inicijaliziraj veličinu populacije $|POP|$
inicijaliziraj omjer uspješnosti $SuccRatio \in [0, 1]$
inicijaliziraj najveći selekcijski pritisak $MaxSelPress \in [1, \infty]$
inicijaliziraj donju vrijednost fakt. usp. $donjaGranica \in [0, 1]$
inicijaliziraj gornju vrijednost fakt. usp. $gornjaGranica \in [donjaGranica, 1]$
inicijaliziraj faktor usporedbe $CompFactor = donjaGranica$
inicijaliziraj stvarni selekcijski pritisak $ActSelPress = 1$
stvori početnu populaciju POP_0 veličine $|POP|$
dok $(i < brojIteracija) \wedge (ActSelPress < MaxSelPress)$ **radi**
  inicijaliziraj sljedeću populaciju POP_{i+1} i
  inicijaliziraj skup lošijih jedinki $POOL$
  dok $(|POP_{i+1}| < (|POP| \cdot SuccRatio)) \wedge ((|POP_{i+1}| + |POOL|) < (|POP| \cdot MaxSelPress))$ **radi**
    stvori dijete križanjem članova populacije POP_i i mutacijom
    usporedi vrijednosti dobrote djeteta c s dobrotama roditelja rod_1 i rod_2 (bez gubitka općenitosti pretpostavimo da je rod_1 kvalitetniji od rod_2):
    ako $f_c \leq (f_{rod_2} + |f_{rod_1} - f_{rod_2}| \cdot CompFactor)$ **tada**
      umetni dijete u $POOL$
    inače
      umetni dijete u POP_{i+1}
    završi ako
  završi dok
  $ActSelPress = \frac{|POP_{i+1}|+|POOL_i|}{|POP_i|}$
  popuni ostatak populacije POP_{i+1} jedinkama $POOL$:
  dok $(|POP_{i+1}| \leq |POP|)$ **radi**
    ako $(|POOL| > 0)$ **tada**
      umetni u POP_{i+1} nasumično odabranu jedinku iz $POOL$
    inače
      umetni u POP_{i+1} novo stvorenu jedinku
    završi ako
  završi dok
  prilagodi $CompFactor$ ovisno o zadanoj strategiji
  $i = i + 1$
završi dok

Algoritam 4.4 Algoritam očuvanja važnih alela (RAPGA)

inicijaliziraj broj trenutne iteracije $i = 0$
inicijaliziraj početnu veličinu populacije $InitPop$
inicijaliziraj najmanju dopuštenu veličinu populacije $MinPop$
inicijaliziraj najveću dopuštenu veličinu populacije $MaxPop$
inicijaliziraj najveći dopušteni trud $MaxEffort$
inicijaliziraj donju vrijednost fakt. usp. $donjaGranica \in [0, 1]$
inicijaliziraj gornju vrijednost fakt. usp. $gornjaGranica \in [donjaGranica, 1]$
inicijaliziraj faktor usporedbe $CompFactor = donjaGranica$
inicijaliziraj trenutni trud $CurrEffort = 0$
stvori početnu populaciju POP_0 veličine $InitPop$
dok $(|POP_i| \geq MinPop)$ **radi**
  inicijaliziraj sljedeću populaciju POP_{i+1}
  $CurrEffort = 0$
  dok $(|POP_{i+1}| < MaxPop) \wedge (CurrEffort < MaxEffort)$ **radi**
    $CurrEffort = CurrEffort + 1$
    stvori dijete križanjem članova populacije POP_i i mutacijom
    usporedi vrijednosti dobrote djeteta c s dobrotama roditelja rod_1 i rod_2 (bez gubitka općenitosti pretpostavimo da je rod_1 kvalitetniji od rod_2):
    ako $f_c \geq (f_{rod_2} + |f_{rod_1} - f_{rod_2}| \cdot CompFactor)$ **ili** (dijete c uvodi novu genetsku informaciju u populaciju POP_{i+1}) **tada**
      umetni dijete u POP_{i+1}
    završi ako
  završi dok
  prilagodi $CompFactor$ ovisno o zadanoj strategiji
  $i = i + 1$
završi dok

5. Primjena genetskog algoritma na problem izrade rasporeda nastave

U okviru rada razvijeno je programsko ostvarenje *Sustava za izradu rasporeda nastave za osnovne i srednje škole*. Ostvaren je u programskom jeziku *Java* i sastoji se od nekoliko sastavnica. U ovom je poglavlju detaljnije opisana njegova arhitektura i način rada.

Prvi odjeljak detaljnije opisuje arhitekturu *Sustava*. Opisuju se sastavnice koje čine *Sustav*, kao i njihova uloga. Navode se i glavna sučelja i metode koje pojedini razredi trebaju ostvariti kako bi se uklopili. Nadalje, navedene su strukture podataka koje se koriste. Najvažnija je ona koja predstavlja izgrađen raspored, a ostvarene su njezine dvije inačice. Ovdje je opisana i struktura s parametrima rasporeda koja se koristi pri radu i heuristički algoritam koji stvara početna rješenja.

U drugom su odjeljku detaljnije opisana ostvarenja genetskih operatora selekcije, križanja i mutacije, zajedno s njihovim vršnim sučeljima i upraviteljima. Zadnja dva odjeljka opisuju procjenitelje rješenja i optimizacijske metode koje se koriste pri izradi rasporeda.

5.1. Ostvarenje rješenja

S obzirom na složenost problema izrade rasporeda nastave, glavne osobine *Sustava* koje su se pri razvoju pokušale postići su prilagodljivost i proširivost. Ove osobine važne su jer su u razgovorima s predstavnicima škola uočene značajne razlike u zahtjevima koje njihovi rasporedi trebaju zadovoljavati, što je opisano u poglavlju 2. Kako su razgovori provedeni s predstavnicima svega nekoliko škola u Zagrebu i Svetoj Nedelji, moguće je da u Hrvatskoj postoje škole za čije ispravne rasporede opisani zahtjevi ne bi bili dovoljni. *Sustav* stoga treba biti dovoljno prilagodljiv da se na jednostavan način mogu promijeniti pojedina postojeća ponašanja i dovoljno proširiv da se lako može nadograditi novim mogućnostima.

Upravo iz navedenih razloga, razvijeni *Sustav* nije prilagođen samo radu s evolucijskim algoritmima, već s metaheuristikama općenito. Slično je i s obilježjima rasporeda koji se izgrađuje; nova obilježja mogu se jednostavno dodati ukoliko se za to ukaže potreba.

5.1.1. Sastavnice *Sustava*

Arhitektura *Sustava* podijeljena je u četiri samostalna modula koji su organizirani u obliku četiri programska projekta:

- modul *Core* (projekt *SchoolSchedCore*),
- modul *Data* (projekt *SchoolSchedData*),
- modul *Engine* (projekt *SchoolSchedEngine*) i
- modul *Gui* (projekt *SchoolSchedGui*).

Modul *Core*

Modul *core* predstavlja jezgru *Sustava*. Njegova je izvedba od kritične važnosti za cjelokupni rad Sustava jer se svi ostali moduli u velikom dijelu oslanjaju na ovdje zadana sučelja i metode. To se odnosi na strukture podataka koje predstavljaju parametre rasporeda, kao i na sučelja razreda koji ostvaruju djelatnost algoritama i njihovih operatora.

Središte djelovanja modula *Core*, a samim time i cjelokupnog *Sustava*, razred je `AbstractSchedulingContext`. Ovaj apstraktni razred predstavlja kontekst izrade rasporeda. U njemu su sadržani svi podaci koji su potrebni za ispravan rad *Sustava*. Kontekst raspoređivanja sadrži i osnovne metode za pokretanje izrade rasporeda, a služi i kao sabirnica za sve ostale dijelove *Sustava* (nudeći metode za pristup parametrima, procjeniteljima rješenja, tvornici rasporeda, brojaču vremena i slično). Ovisno o željenom ponašanju *Sustava*, ovaj je razred potrebno naslijediti novim koji će ostvariti zaštićenu apstraktnu metodu:

- `abstract void startScheduling(int iterationCount).`

U njoj je potrebno pokrenuti izradu rasporeda koristeći jedan ili više algoritama zadanih u kontekstu raspoređivanja i na željeni način oblikovati dojavljivanje napretka korisniku. Napredak rada *Sustava* dojavljuje se metodom `fireProgressMade`, kojoj se predaje instanca razreda `SchedulingProgressEvent` i podatak treba li se događaj dojaviti odmah svim prijavljenim stranama ili unutar redovnih intervala do-

javljivanja. Razred `SchedulingProgressEvent` sadrži podatke o stanju izrade: trenutni korak, proteklo vrijeme i najbolje dosad pronađeno rješenje.

Izrada rasporeda pokreće se pozivanjem javne metode `startScheduling` nad primjerkom razreda `AbstractSchedulingContext`. Kao parametre metodi treba predati najveći željeni broj koraka algoritama, najduže željeno vrijeme izvođenja i indikator želi li se izrada pokrenuti sinkrono ili asinkrono. Ukoliko se izrada pokreće sinkrono, metoda će završiti izvođenje tek kada završi cjelokupna izrada rasporeda. Takvo ponašanje pogodno je za testiranje *Sustava* gdje se izrada pokreće uzastopno nekoliko puta. Asinkrono pokretanje koristi se prilikom izrade koja je potaknuta iz grafičkog sučelja *Sustava*, jer bi u protivnom do kraja izrade cjelokupna aplikacija bila zamrznuta. Unutar ove metode pokreće se brojač vremena i poziva se prije opisana zaštićena metoda `startScheduling`.

Unutar jednog postupka raspoređivanja moguće je koristiti jednu ili više metaheuristika. Svaki algoritam djeluje neovisno unutar svojeg konteksta. Algoritam tijekom izvođenja postupka raspoređivanja ima pristup kontekstu raspoređivanja (parametrima, procjeniteljima, vremenu izvođenja i slično), ali ne i kontekstima drugih algoritama. Kontekst algoritma opisan je razredom `AbstractAlgorithmContext`, a predviđa sljedeće metode:

- `abstract void initialize()` — inicijalizacija algoritma, vezanje s ostalim sastavnicama *Sustava* i priprema za početak rada,
- `abstract void generateInitialPopulation()` — stvaranje početne populacije jedinki,
- `abstract Solution getBestSolution()` — dohvaćanje najboljeg trenutno pronađenog rješenja,
- `abstract void step()` — izvršavanje jednog koraka algoritma,
- `abstract void use(Solution solution)` —uključivanje vanjskog rješenja u trenutnu populaciju i
- `abstract boolean isFinished()` — provjera kraja izvođenja.

U kontekstu algoritma nalaze se metode i podaci usko vezani uz njegove osobine. U slučaju evolucijskih algoritama, kontekst će sadržavati algoritme za stvaranje početne populacije, upravitelje operatorima selekcije, križanja, mutacije i, ako su ostvarene, algoritme lokalne pretrage s pripadajućim upraviteljima. Upravo na takav način ostvaren je razred `GeneticAlgorithmContext`.

Procjenitelji rješenja dio su konteksta raspoređivanja. Time se postiže jedinstven

način procjene kvalitete rješenja za sve uključene algoritme. Glavni procjenitelj, oblikovan razredom `ScheduleEvaluator`, sadrži niz svih procjenitelja zadanih razredom `AbstractEvaluator` i nudi metode za izračun dobrote rješenja. Izračun se pokreće pozivanjem metode `calculateFitness`, kojoj se kao parametar predaje raspored koji se želi procijeniti. Unutar metode pokreću se pojedinačni izračuni dobrote nad svakim od zadanih procjenitelja, a rješenje oblikovano razredom `Fitness` predstavlja vektor kvalitete jedinke (svaki pojedinačni rezultat kao jedan član vektora). Ukoliko se kvalitetu jedinke želi predstaviti u obliku težinske sume, više procjenitelja moguće je obuhvatiti u sumu koristeći razred `EvaluationAccumulator` koji također nasljeđuje razred `AbstractEvaluator`. Važno je napomenuti kako razred `Fitness` ne predstavlja nužno dobrotu jedinke, već rezultat procjene rješenja, a na programeru procjenitelja je da odredi je li to dobrota ili kazna i s obzirom na to naslijedi metodu `compareTo`. Uz `ScheduleEvaluator`, u kontekstu raspoređivanja nalazi se i razred za podršku višedretvenoj procjeni cijele populacije, `MultiThreadEvaluator`. Ukoliko način rada algoritma i arhitektura računala to dozvoljavaju, ovakav način procjene može doprinijeti brzini izvođenja.

Osim navedenih razreda, modul *Core* sadrži i mnoge zajedničke pomoćne razrede i metode koje se koriste u drugim modulima, poput metoda za operacije nad nizovima, pretvaranje termina u odgovarajuće dane i tjedne i slično. Cjelokupni dijagram modula *core* prikazan je na slici 5.1.

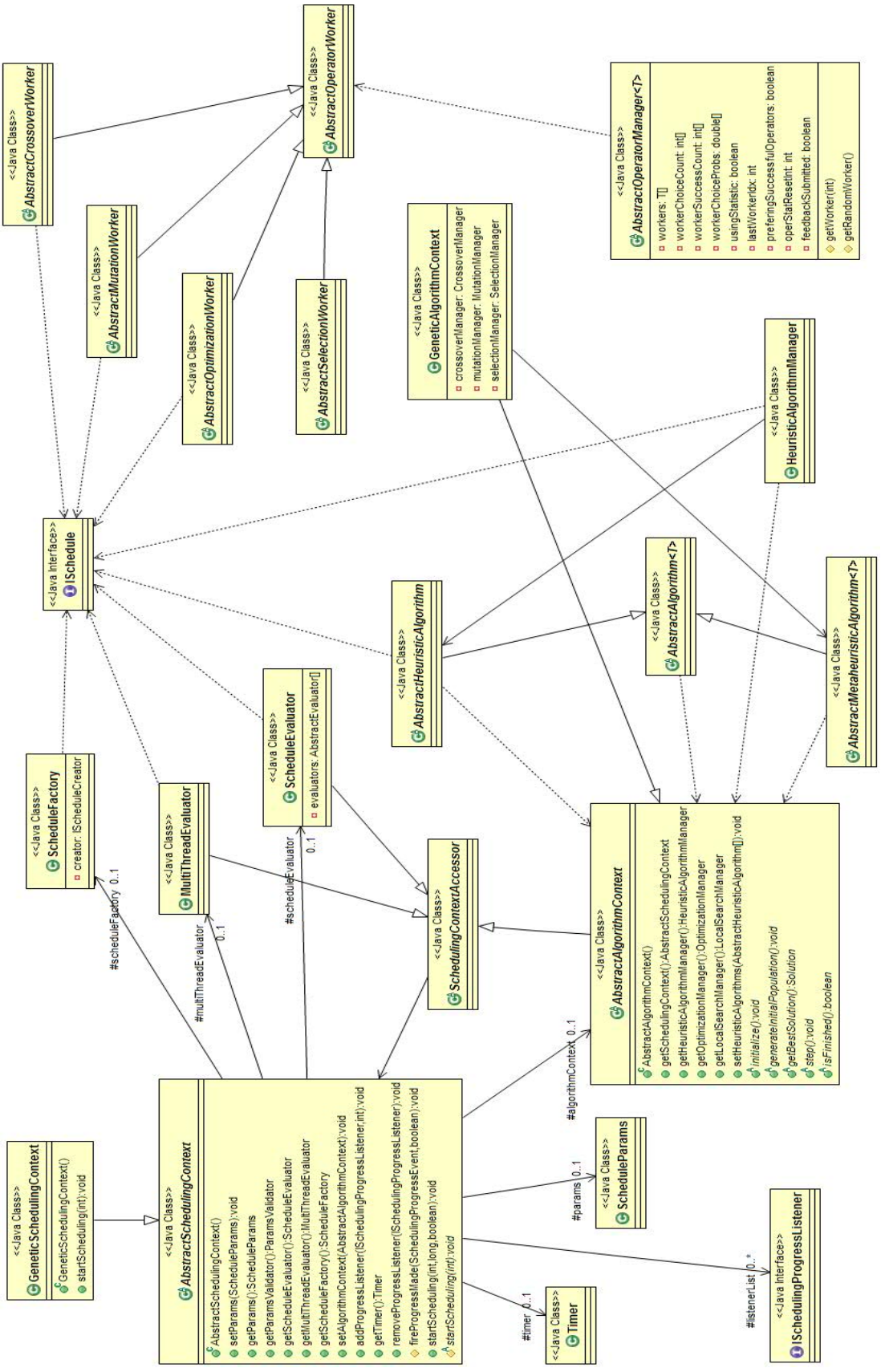
Modul *Data*

Modul *Data* sadrži mehanizme za čitanje parametara iz i zapis izgrađenih rasporeda u datoteke. To podrazumijeva strukture podataka potrebne za preslikavanje datoteka u memoriju računala (i obrnuto), kao i metode koje to preslikavanje provode. Moguće je podržati i bilo koji drugi oblik ulaznih ili izlaznih datoteka. U ovom modulu potrebno je ostvariti postupak pretvaranja podataka iz datoteke u odgovarajuću strukturu koja se koristi u *Sustavu* (a zadana je u modulu *Core*).

Za novu vrstu parametarske datoteke potrebno je ostvariti novi razred prema sučelju `IParamsHandler`, koje predviđa sljedeće metode:

- `void toExternal(ScheduleParams p, OutputStream s)` i
- `ScheduleParams fromExternal(InputStream i).`

Slično je i s izlaznim podacima. Da bi se podržao novi oblik datoteke za zapis rješenja, potrebno je naslijediti sučelje `AbstractScheduleStreamHandler` s predviđenim metodama:



Slika 5.1: Dijagram razreda modula *Core*

- `void toExternal(ISchedule schedule, OutputStream s)` i
- `ISchedule fromExternal(InputStream stream)`.

Iako je formalno potrebno naslijediti obje metode, ostvarenje istih nije nužno ukoliko se takvo ponašanje neće koristiti. Jednom kada se ostvari razred koji povezuje novu datoteku s unutarnjim strukturama podataka, moguće ga je koristiti prema potrebama, samostalno ili u kombinaciji s mehanizmima za zapis drugih vrsta datoteka.

Modul *Engine*

U modulu *Engine* nalazi se cjelokupna programska logika na kojoj se temelji izrada rasporeda. Tako se ovdje nalaze svi algoritmi koji izrađuju raspored (heuristički i metaheuristički) zajedno s ostvarenjima svojih operatora, kao i ostvarenja procjenitelja rješenja.

Heuristički algoritmi koriste se za stvaranje početne populacije rješenja. Oni samostalno izgrađuju cjelokupna rješenja za koja nije potrebno da budu kvalitetna ili uopće izvediva. Važno je samo da se sva predavanja zadana u parametrima smjeste u neke termine i da se na takav način uspostavi temelj na kojem metaheuristike mogu započeti svoj rad.

Razvijeni *Sustav* nudi mogućnost korištenja većeg broja heurističkih algoritama. Svaki takav algoritam treba naslijediti sučelje `AbstractHeuristicAlgorithm` koje propisuje samo jednu metodu:

- `abstract ISchedule generateSchedule(ISchedule sched)`.

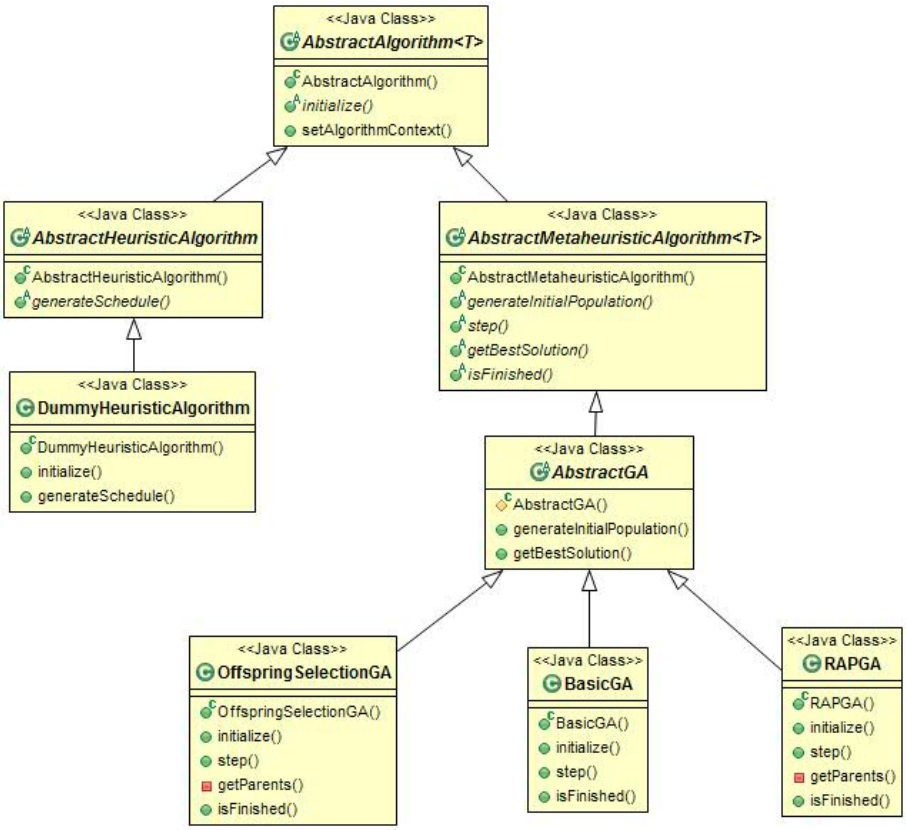
Metoda kao parametar prima prazan raspored i zadatak joj je popuniti ga. U slučaju da s obzirom na zadane parametre i način rada algoritma nije moguće izgraditi raspored, metoda treba baciti iznimku `CouldNotGenerateScheduleException`.

Sučelje za metaheurističke algoritme propisuje više metoda. S obzirom na to da se cijeli *Sustav* temelji na radu metaheurističkih algoritama, njihovo se sučelje (`AbstractMetaheuristicAlgorithm`) sastoji od približno jednakih metoda koje nudi i opći algoritamski kontekst:

- `abstract void generateInitialPopulation()`,
- `abstract Solution getBestSolution()`,
- `abstract void step()`,
- `abstract void use(Solution solution)` i
- `abstract boolean isFinished()`.

Kontekst zatim pozive pojedinih metoda prosljeđuje svojoj metaheuristici.

Metaheuristički algoritmi razvijeni u sklopu ovog rada (osnovni genetski algoritam, algoritam odabira potomaka i algoritam očuvanja važnih alela) obuhvaćeni su vršnim apstraktnim razredom `AbstractGA` koji nasljeđuje prethodno opisano sučelje za metaheuristike. Kako su svi razvijeni algoritmi inačice genetskog algoritma, ovaj razred sadrži i neke zajedničke varijable, poput trenutne populacije, broja elitističkih jedinki i vjerojatnosti križanja i mutacije, a ostvaruje i metode za stvaranje početne populacije i dohvat najboljeg trenutno pronađenog rješenja. Ostalim algoritmima preostaje ostvariti metode za izvršavanje jednog koraka i ispitivanje kraja izvršavanja. Cjelokupna struktura algoritama prikazana je dijagramom razreda na slici 5.2.



Slika 5.2: Dijagram razreda koji prikazuje strukturu ostvarenih algoritama u *Sustavu*

Modul *Gui*

U modulu *Gui* nalaze se ostvarenja dijaloga i upravitelja koji čine grafičko sučelje *Sustava*. U sučelju je na jednostavan način moguće zadati parametre i pokrenuti izradu rasporeda. S obzirom na to da grafičko sučelje nadilazi temu ovog grada, više o njemu nalazi se u (Bronić, 2012).

5.1.2. Strukture podataka

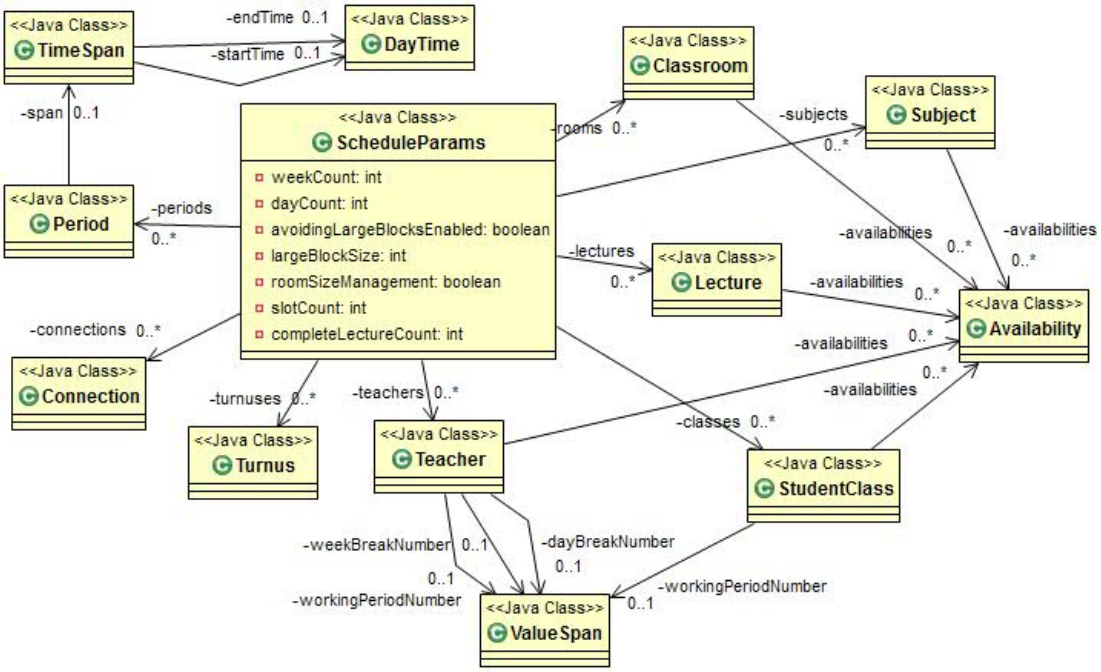
Uz prethodno opisana sučelja, modul *Core* zadaje i strukturu podataka u kojoj se nalaze parametri rasporeda kojeg je potrebno izgraditi. Glavni je objekt te strukture razred `ScheduleParams`. Njegovo ustrojstvo vrlo je slično ustrojstvu *XML* datoteke s ulaznim parametrima. On sadrži metode za dohvat općih podataka o rasporedu (broj tjedana, broj dana u tjednu, duljina "većeg" bloka predavanja...), dok su podaci o turnusima (razred `Turnus`), učionicama (razred `Classroom`), predmetima (razred `Subject`), nastavnicima (razred `Teacher`), razredima (razred `StudentClass`), predavanjima (razred `Lecture`) i vezama između predavanja (razred `Connection`) organizirani u nizove.

Svakom je objektu u parametrima dodijeljen jedinstveni broj (*indeks*) prema kojemu se mogu dohvatiti detaljni podaci o njemu. Zbog lakše buduće paralelizacije *Sustava*, objekti su međusobno povezani indeksima, a ne izravnim vezama na objekte. Tako će se u narednim odlomcima prilikom opisivanja metoda spominjati sljedeći indeksi: `slotIdx` (indeks vremenskog odsječka[1]), `lectureIdx` (indeks predavanja), `classIdx` (indeks razreda), `teacherIdx` (indeks nastavnika) i slični.

Jedina je razlika između ustrojstva parametara rasporeda u memoriji računala i *XML* datoteci u načinu zapisivanja podataka o odjeljcima razreda. Dok su u *XML* datoteci navedeni u sklopu podataka o razredu, u `ScheduleParams` odjeljci razreda predstavljeni su ravnopravno s ostalim cjelokupnim razredima, uz dodatni podatak kojem cjelokupnom razredu i njegovoj podjeli pripadaju. Na takav se način algoritamskim postupcima olakšava rad s pojmom odjeljaka, a kada se za time pojavi potreba, podaci o podjelama jednostavno se dohvaćaju. S obzirom na to, razred `StudentClass` između ostalog sadrži i ove metode:

- `Integer getParent()` — ukoliko objekt predstavlja odjeljak, metoda vraća indeks objekta koji predastavlja cjelokupni razred kojem odjeljak pripada, inače vraća `null`;
- `Integer getChildGroupInParent()` — ukoliko je objekt odjeljak, metoda vraća indeks podjele koje je ovaj odjeljak dio, a u suprotnom, metoda vraća `null`;
- `int[][] getChildren()` — ukoliko objekt predstavlja cjelokupni razred, metoda vraća sve podjele i indekse njima pripadajućih odjeljkaka, inače

[1] Vremenski odsječak (engl. *timeslot*) je apsolutni položaj pojedinog termina u rasporedu. Ukupan broj vremenskih odsječaka u rasporedu jednak je umnošku broja tjedana za koje se izgrađuje raspored, broja dana u tjednu i broja radnih sati u danu.



Slika 5.3: Struktura parametara rasporeda

vraća `null`. Jedna podjela predstavljena je nizom identifikatora odjeljaka koji joj pripadaju. Sve podjele jednog razreda predstavljene su kao niz nizova indeksa odjeljaka, što je i tip podataka koji ova metoda vraća.

Dijagram razreda podatkovne strukture ulaznih parametara prikazan je slikom 5.3 (detaljniji opis parametara rasporeda naveden je u poglavlju 2).

Najvažnija je struktura podataka u *Sustavu* ona koja sadrži podatke o izgrađenom rasporedu, a zadana je sučeljem `ISchedule`. Ujedno se ista struktura koristi prilikom rada algoritama, jer sadrži metode za ispitivanje mogućnosti rezervacije predavanja u nekom terminu, kao i rezervacije i oslobađanja istih. Razredi koji nasljeđuju ovo sučelje moraju ostvariti sljedeće metode:

- `int getSlotForLecture(int lectureIdx)` — za zadano predavanje vraća vremenski odsječak u kojem je predavanje rezervirano ili -1 ukoliko predavanje nije rezervirano;
- `int[] getLocationsForLecture(int lectureIdx)` — vraća niz indeksa učionica u kojima se održava zadano predavanje ili prazan niz ukoliko predavanje ne zahtjeva učionice;
- `int getOccupiedRoomCountForSlot(int slotIdx)` — vraća broj slobodnih dvorana u zadanom vremenskom odsječku,
- `int[] getLecturesInSlot(int slotIdx)` — za zadani vremenski

odsječak, vraća niz predavanja koja se u njemu održavaju (niz može biti i prazan ako takvih predavanja nema);

- `boolean isClassroomOccupiedInTime(int roomIdx, int slotIdx, int duration)` — vraća `true` ukoliko je učionica zauzeta[2] u zadanom vremenskom odsječku i trajanju, inače vraća `false`;
- `int getLectureForClassroomInSlot(int roomIdx, int slotIdx)` — vraća indeks predavanja koje se održava u zadanoj učionici i vremenskom odsječku ili -1 ako je učionica u tom terminu slobodna;
- `boolean isTeacherOccupiedInTime(int teacherIdx, int slotIdx, int duration)` — vraća `true` ako je nastavnik zauzet u zadanom vremenskom odsječku i trajanju, inače vraća `false`;
- `int getLectureForTeacherInSlot(int teacherIdx, int slotIdx)` — vraća indeks predavanja koje zadani nastavnik održava u određenom vremenskom odsječku ili -1 ako je nastavnik u to vrijeme slobodan;
- `boolean isClassOccupiedInTime(int classIdx, int slotIdx, int duration)` — vraća `true` ako je razred zauzet u zadanom vremenskom odsječku i trajanju, inače vraća `false`;
- `int getLectureForClassInSlot(int classIdx, int slotIdx)` — vraća indeks predavanja koje razred pohađa u zadanom vremenskom odsječku ili -1 ako je u to vrijeme slobodan;
- `boolean lectureCanBeReserved(int lectureIdx, int slotIdx, int[] classroomIdxs)` — vrši procjenu mogućnosti rezervacije zadanog predavanja u navedenom vremenskom odsječku i zadanim učionicama. Niz učionica nije nužno zadati (može se predati `null` kao niz indeksa učionica). Međutim, ako su one nužne za održavanje predavanja, procjena u tom slučaju ne mora biti u potpunosti točna (zbog nedostatka podataka). Metoda vraća `true` ukoliko je procijenjeno da se predavanje može rezervirati, inače vraća `false`;
- `boolean reserveLecture(int lectureIdx, int slotIdx, int[] classroomIdxs)` — rezervira predavanje u zadani odsječak i učionice. Prije postupka rezervacije, unutar metode potrebno je pozvati metodu

[2]Pojam zauzetosti u smislu strukture podataka koja opisuje raspored različit je od pojma raspoloživosti koji se spominje u poglavlju 2. Zauzetost znači opterećenost drugim rezerviranim obvezama unutar rasporeda, neovisno o tome je li zauzeta strana raspoloživa ili ne u navedeno vrijeme.

`lectureCanBeReserved` kako bi se ispitala mogućnost željene rezervacije. Povratna je vrijednost podatak je li rezervacija uspješno obavljena;

- `void reserveLectureUnchecked(int lectureIdx, int slotIdx, int[] classroomIdxs)` — provodi rezervaciju predavanja bez ikakvih provjera. Poziv ove metode preporuča se jedino u slučajevima kada su sve potrebne provjere već prethodno obavljene (ili je predavanje neposredno prije oslobođeno iz istog termina). Ukoliko rezervaciju nije moguće ispravno provesti, metoda baca iznimku `SchedulingException`;
- `int freeLecture(int lectureIdx)` — oslobađa navedeno predavanje i vraća indeks odsječka u kojem je bilo rezervirano i
- `ISchedule clone()` — vraća kopiju trenutnog rasporeda.

U *Sustavu* postoje dva ostvarenja ovog sučelja. Razred `BasicSchedule` ostvaruje navedene metode na najosnovniji mogući način. Termini i učionice u kojima se predavanja održavaju organizirani su u obliku nizova cijelih brojeva. Dodatno, za svaki vremenski odsječak prati se broj slobodnih dvorana, kao i predavanja koja su u tom terminu raspoređena nastavnicima, razredima i učionicama. Zadnje strukture predstavljaju određenu zalihost, ali pridonose brzini izvođenja i jednostavnosti dohvaćanja željenih podataka. Ova inačica rasporeda dozvoljava rezervaciju predavanja dok god se nastavniku, razredu ili učionici ne pokušaju rasporediti dva ili više predavanja u isto vrijeme.

Razred `SmarterSchedule` nadogradnja je prethodno spomenutog i nudi detaljnije ostvarene metode za ispitivanje mogućnosti rezervacije predavanja u određeni termin. Ovakvo ostvarenje rasporeda neće dopustiti rezervaciju predavanja u parametrima nedozvoljeni tjedan ili vremenske odsječke u kojima bi ono bilo razlomljeno između dva dana. Također, unutar metode `lectureCanBeReserved` ispituje se i zauzetost cjelokupnog razreda i njegovih odjeljaka. Tako će ovo ostvarenje rasporeda odbiti rezervirati predavanje u termin u kojem bi cjelokupni razred i neki od njegovih odjeljaka imali raspoređenu nastavu u isto vrijeme.

5.1.3. Stvaranje početnih rješenja

Heuristički algoritmi služe stvaranju početnih rješenja s kojima metaheuristike započinju svoj rad. Ta rješenja ne moraju biti kvalitetna, ali ih je potrebno moći predstaviti zadanim oblikom zapisa. U sklopu *Sustava* razvijen je heuristički algoritam predstavljen razredom `DummyHeuristicAlgorithm`. Ovaj algoritam početna rješenja

Algoritam 5.1 Stvaranje početnih rješenja

```
rasp = noviPrazanRaspored()
perm ← permutiraj(nizOdsjecaka)
pocPretrage = 0
za (i = 0; i < brojPredavanja; i++) radi
  predavanje = redoslijedPredavanja[i]
  rezervirano = false
  za (j = 0; j < brojOdsjecaka; j++) radi
    odsjecak = perm[j]
    ako (rasp.moguceRezervirati(predavanje, odsjecak)) tada
      rasp.rezerviraj(predavanje, odsjecak)
      rezervirano = true
    završi ako
  završi za
  ako (!rezervirano) tada
    uvećaj složenost smještaja predavanja predavanje
    poredaj niz redoslijedPredavanja prema složenosti
    dojavi nemogućnost izrade rasporeda
  završi ako
završi za
vrati rasp
```

stvara nasumičnim smještajem predavanja u vremenske odsječke, istovremeno bilježeći složenost njihovog smještanja, po uzoru na izravnu heuristiku opisanu u poglavlju 3.

Predavanja se pokušavaju smjestiti redoslijedom od najsloženijeg prema najjednostavnijem. Pri inicijalizaciji algoritma, težine smještaja svih predavanja postavljaju se na vrijednost 0. Ukoliko se prilikom izrade rješenja utvrdi da se neko predavanje ne može smjestiti niti u jedan termin, povećava se vrijednost njegove složenosti smještaja i dojavljuje se nemogućnost izrade rasporeda. Prilikom sljedećeg pokretanja postupka, složenije će se predavanje prije pokušati smjestiti i time pospješiti izradu. Pseudokod cjelokupnog postupka stvaranja početnog rasporeda prikazan je algoritmom 5.1.

5.2. Evolucijski operatori

Evolucijski operatori koje genetski algoritmi koriste obuhvaćeni su trima vršnim apstraktnim razredima koji određuju njihovo ponašanje. Algoritmi im pristupaju preko njihovih upravitelja (zadanih vršnim razredom `AbstractOperationManager`) koji, prema potrebi, imaju mogućnost davanja prvenstva uspješnijim operatorima. Prije pokretanja *Sustava*, potrebno je za svaki algoritamski kontekst zadati operatore koji se žele koristiti. To se provodi pozivom metode `setWorkers(T[] workers)` nad odgovarajućim upraviteljem, kojoj se kao parametar predaje niz željenih operatora. Tijekom rada je, prema potrebi, metodama nad odgovarajućim upraviteljem moguće dohvatiti željeni ili nasumično odabrani operator. U odjeljku su detaljnije opisana programska ostvarenja genetskih operatora selekcije, križanja i mutacije.

5.2.1. Selekcija

Operatori selekcije nasljeđuju razred `AbstractSelectionWorker`, koji predviđa sljedeće tri metode:

- `abstract int selectPopulationMember(Solution[] population)` — vraća indeks jedinke koja se odabire za razmnožavanje;
- `abstract int eliminatePopulationMember(Solution[] population)` — vraća indeks jedinke koja se odstranjuje iz populacije i
- `abstract boolean populationSortNedded()` — vraća podatak je li, s obzirom na način rada operatora, prije odabira potrebno sortirati populaciju prema kvaliteti.

Za potrebe ovog rada razvijeni su sljedeći operatori selekcije: proporcionalna, linearna sortirajuća, nasumična i turnirska s i bez istovjetnih jedinki.

5.2.2. Križanje

Operatori križanja nasljeđuju apstraktni razred `AbstractCrossoverWorker` kod kojeg je potrebno ostvariti samo jednu metodu:

- `abstract ISchedule performCrossover(ISchedule[] schedules)` — za zadani niz roditeljskih jedinki (barem dvije) vraća novu jedinku dobivenu njihovim križanjem.

U *Sustavu* je ostvaren jedan operator križanja, `BinaryLikeCrossover`. Način rada ovog operatora vrlo je sličan križanju binarnog kromosoma opisanog u poglavlju

4. Predavanja su najmanje čestice na koje je moguće razložiti raspored. Ako se niz predavanja predoči kao binarni niz, lako je uočiti da bi se i nad školskim rasporedom moglo provoditi križanje s n točki prekida.

Za slučaj jedne točke prekida i dva roditelja, potrebno je odabrati jednu točku koja će podijeliti nizove predavanja na dva dijela. Zatim se u novoj jedinki predavanja rezerviraju u one termine u kojima su bila raspoređena u roditeljskim jedinkama, u skladu s točkom prekida. Problem nastaje ukoliko neko predavanje više nije moguće rezervirati u isti termin u kojem je bilo raspoređeno kod odgovarajućeg roditelja. U tom se slučaju traži slučajan termin za rezervaciju. Ukoliko takav termin nije moguće naći, postupak križanja se prekida.

Ovakav način rada moguće je poopćiti na slučaj s m roditelja i n točki prekida. U tom je slučaju moguće stvoriti cjelobrojni niz koji, s obzirom na točke prekida, sadrži podnizove s indeksima roditelja iz kojih se dohvaćaju podaci o smještaju predavanja (na primjer, prvih a članova niza ima vrijednost 0, sljedećih b članova vrijednost 1 i tako dalje do članova s vrijednošću $m-1$).

Korištenjem cjelobrojnog niza koji određuje na temelju kojeg roditelja se dohvaćaju podaci o pojedinom predavanju i stvaranjem niza s nasumičnim vrijednostima (između 0 i $m-1$), moguće je ostvariti i križanje nalik uniformnom binarnom. Upravo su posljednja dva načina ostvarena u sklopu *Sustava*. Pseudokod postupka križanja iskazan je algoritmom 5.2.

5.2.3. Mutacija

Razred `AbstractMutationWorker` predstavlja vršni razred za sve operatore mutacije. Prilikom njegovg nasljeđivanja, potrebno je ostvariti metode:

- `abstract boolean isAplicableOn(int lectureIdx)` — vraća informaciju je li trenutni postupak mutacije primjenjiv na zadano predavanje i
- `abstract boolean modify(ISchedule schedule, int lectureId)` — izvodi mutaciju rasporeda nad zadanim predavanjem.

Prema uzoru na način rada operatora križanja, i mutacije se vrše nad predavanjima kao najmanjim česticama rasporeda. Prva metoda potrebna je iz razloga što je moguće napisati postupak mutacije koji nad pojedinim predavanjima neće moći provesti zadanu aktivnost. Primjer za to je operator `LectureRelocatingMutation`. Zadatak ovog postupka je premjestiti predavanje u neku drugu učionicu u istom terminu. Nema ga smisla provoditi nad predavanjima za koja nisu potrebne učionice jer takvo provođenje neće izazvati nikakvu promjenu u rasporedu.

Algoritam 5.2 Križanje rasporeda

$rasp =$ ***noviPrazanRaspored()***
ako ($nacinRada == nPOINT$) **tada**
  $odabirRoditelja =$ ***stvoriNPointPoljeOdabira()***
inače
  $odabirRoditelja =$ ***stvoriUniformPoljeOdabira()***
završi ako
za ($i = 0; i < brojPredavanja; i$++) **radi**
  $predavanje = nizPredavanja[i]$
  $odsjecak = rodRasporedi[odabirRoditelja[i]]$.***dohvatiOdsjecak***($predavanje$)
  ako ($rasp$.***moguceRezervirati***($predavanje$, $odsjecak$)) **tada**
    $rasp$.***rezerviraj***($predavanje$, $odsjecak$)
  inače
    ako predavanje moguće rezervirati u neki drugi termin **tada**
      rezerviraj predavanje u taj termin
    inače
      nije moguće provesti križanje
    završi ako
  završi ako
završi za
vrati $rasp$

Uz navedeni, u *Sustavu* su razvijena još dva operatora mutacije. Drugi postupak (`LectureMovingMutation`), pokušava razmjestiti zadano predavanje u neki drugi, slučajno odabran termin. Operator `LectureSlotSwappingMutation` uz zadano predavanje odabire još jedno i pokušava im međusobno zamijeniti termine. S obzirom na to da im je djelovanje prvotno usmjereno na termin u kojem se predavanje nalazi, oba ova operatora moguće je primijeniti na bilo koje predavanje.

5.3. Vrednovatelji rješenja

Vrednovatelji rješenja iznimno su važni su za rad *Sustava* jer o njihovim rezultatima ovisi što će se smatrati više ili manje kvalitetnim rasporedom. Trenutno u *Sustavu* postoji 13 različitih ostvarenja procjenitelja. Svaki izračunava posebnu ocjenu rasporeda s obzirom na čimbenike koje prati. U nastavku slijedi njihov popis s kratkim opisima djelovanja:

– procjenitelji raspoloživosti — za svaki rezervirani termin ispituju raspoloživost objekta u navedeno vrijeme i bilježe nepoštivanje istog:

    1. brojač nepoštivanja raspoloživosti razreda (`ClassAvailabilitiesViolationCounter`),
    2. brojač nepoštivanja raspoloživosti predavanja (`LectureAvailabilitiesViolationCounter`),
    3. brojač nepoštivanja raspoloživosti učionica (`RoomAvailabilitiesViolationCounter`) i
    4. brojač nepoštivanja raspoloživosti nastavnika (`TeacherAvailabilitiesViolationCounter`);

– procjenitelji neprekinutih rasporeda — provjeravaju neprekidnost rasporeda za svaki dan u kojem se održava nastava i bilježe nepoštivanje ovog zahtjeva, ukoliko je zadan:

    5. brojač nepoštivanja neprekidnog rasporeda za razrede (`ClassContinousScheduleViolationCounter`) i
    6. brojač nepoštivanja neprekidnog rasporeda za nastavnike (`TeacherContinousScheduleViolationCounter`);

– procjenitelji radnih sati — broje i bilježe broj slučajeva u kojima se ne poštuje najmanji i najveći broj radnih sati u danu ili tjednu:

    7. brojač nepoštivanja broja radnih sati (u danu) za razrede (`ClassWorkingPeriodsViolationCounter`) i
    8. brojač nepoštivanja broja radnih sati (u danu i tjednu) za nastavnike (`TeacherWorkingPeriodsViolationCounter`);

– procjenitelji termina predavanja — s obzirom na različite čimbenike, provjeravaju termine u kojima su predavanja rezervirana i broje one koji nisu u skladu s parametrima:

    9. brojač predavanja raspoređenih u pogrešan tjedan (`IllegalLectureWeekCounter`),
    10. brojač predavanja raspoređenih u pogrešnu učionicu (`IllegalLectureLocationCounter`),

11. brojač višesatnih predavanja razlomljenih između dva dana (`DaySplittedLectureCounter`),
12. brojač više predavanja istog predmeta u istom danu (`MultipleLecturesOfSameSubjectInDayCounter`) i
13. brojač nepoštivanja veza između predavanja (`ConnectionViolationCounter`).

S obzirom na to da navedeni procjenitelji broje slučajeve u kojima se ne poštuju zadani parametri, idealan raspored bio bi onaj za koji bi sve vrijednosti procjene bile jednake 0. Zato objekt `Fitness` u ovom slučaju predstavlja kaznu rasporeda, a bolje rješenje je ono koje ima manju vrijednost procjene kvalitete, neovisno o tome uspoređuju li se vektori s n sastavnica ili težinske sume. Kvalitetu rasporeda moguće je dodatno stupnjevati ostvarenjem novih procjenitelja.

5.4. Lokalne pretrage

Evolucijski algoritmi stohastički pretražuju prostor rješenja problema, pri čemu selekcija i križanje usmjeravaju pretragu prema boljim rješenjima, a mutacija utječe na povećanje raznolikosti populacije i samim time nastavak pretraživanja. Zbog svoje stohastičke prirode, ovakav način rada može ponekad zaobići dobra među-rješenja koja bi značajnije utjecala na daljnji tijek izvođenja i konačni rezultat. Iz tog razloga, navedenim se operatorima često pridružuje i neki oblik metoda lokalnih pretraga.

Lokalne su pretrage metode optimiranja jedinki koje djeluju u neposrednom susjedstvu jedinke unutar prostora rješenja. U tom se prostoru iscrpno pretražuju rješenja i jedinka se pomiče u smjeru najboljeg pronađenog. S obzirom na to da je takav postupak iznimno računalno zahtjevan, postupak se provodi samo nad dijelom populacije.

U slučaju, na primjer, optimizacije funkcije jedne varijable, lokalnu je pretragu jednostavno ostvariti. Počevši od trenutne točke, pretražuje se prostor vrijednosti za do δ manjih i većih od postojeće. Kada se pronađe najbolja, njome se zamjenjuje trenutno rješenje. Kod izrade rasporeda postupak nije tako jednostavan iz više razloga. Jedan je činjenica da je raspored višedimenzionalan problem, pa je male izmjene moguće provesti na čitav niz načina, ne znajući pritom koliko se zaista odmaknulo od trenutnog rješenja u prostoru pretraživanja. Drugi je zahtjevnost procjene kvalitete, koja iziskuje najveću količinu računalnih resursa u cijelom postupku izrade.

S obzirom na navedeno, optimizacija rješenja u *Sustavu* provodi se na način da se za odabrano predavanje ili grupu predavanja pokušaju pronaći bolji termini, s kojima

bi naposljetku cijeli raspored postao kvalitetniji. Operator optimizacije, opisan vršnim razredom `AbstractOptimizationWorker`, predviđa jednu metodu koju moraju ostvariti svi postupci:

- `double assess(ISchedule schedule, Lecture lectureObj, int slotIdx)`.

Ova metoda predstavlja lokalnu provjeru prikladnosti termina zadanom predavanju s obzirom na područje procjene. U slučaju da termin u potpunosti zadovoljava potrebe predavanja, metoda treba vratiti vrijednost 0.

Cjelokupan postupak optimizacije jedinke pokreće se pozivom metode `boolean optimize(ISchedule schedule)` koja vraća podatak je li postupak završio uspješno. Pseudokod ove metode opisan je algoritmom 5.3. Važno je primijetiti kako je, s obzirom da metoda `assess` provodi samo lokalnu procjenu, na kraju postupka potrebno dodatno procijeniti cjelokupno dobiveno rješenje. U *Sustavu* trenutno postoje tri postupka lokalne pretrage:

- `AvailabilitiesOptimization` — poboljšanje s obzirom na raspoloživosti razreda, nastavnika, učionica i predavanja,
- `ContinousScheduleOptimization` — poboljšanje s obzirom na broj radnih sati razreda i nastavnika i
- `MultipleLecturesOfSameDayOptimization` — poboljšanje s obzirom na raspored predavanja istog predmeta u danu.

Algoritam 5.3 Optimiranje rasporeda

$oldFitness =$ ***procijeni***$(schedule)$
$predavanja \leftarrow$ ***slucajanOdabirNPredavanja***()
$termini \leftarrow$ trenutni termini $predavanja$
za ($i = 0$; $i < predavanja.$***length***; i++) **radi**
  $schedule.$***free***$(predavanja[i])$
završi za
za ($i = 0$; $i < predavanja.$***length***; i++) **radi**
  $bestRes =$ ***assess***$(schedule, predavanja[i]\ termini[i])$
  $bestSlot = termini[i]$
  za ($slot = 0$; $slot < bojSlotova$; $slot$++) **radi**
    $res =$ ***assess***$(schedule, predavanja[i], slot)$
    ako ($res <$ **<** $bestRes$) **tada**
      $bestRes = res$
      $bestSlot = slot$
    završi ako
  završi za
  rezerviraj $predavanja[i]$ u $bestSlot$
završi za
$newFitness =$ ***procijeni***$(schedule)$
ako (newFitness **<** oldFitness) **tada**
  optimiranje uspješno
inače
  optimiranje neuspješno, vrati $predavanja$ u stare termine ($termini$)
završi ako

6. Utjecaj parametara na rad algoritma

U poglavlju 4 opisana su načela rada tri inačice genetskog algoritma koje su ostvarene u sklopu *Sustava*. Učinak sva tri algoritma ispitan je u ovisnosti o njihovim parametrima. U ovom poglavlju opisani su testni parametri s kojima su algoritmi ispitivani, kao i način na koji su pokusi provođeni. U drugom odjeljku opisuje se i stvarni problem jedne zagrebačke osnovne škole kojega se pokušalo riješiti u pokusima. Treći odjeljak iznosi rezultate dva kruga testiranja. U prvom krugu ispitivalo se djelovanje zajedničkih parametara na rad pojedinih inačica genetskog algoritma, a u drugom je ispitivana učinkovitost ostvarenih algoritama lokalnih pretraga. Naposljetku je izneseno objašnjenje dobivenih rezultata.

6.1. Testni parametri

Kod svih metaheurističkih algoritama, parametri imaju velik utjecaj na kvalitetu dobivenih rješenja. Loš skup parametara može značajno pokvariti učinak inače uspješnog algoritma i obrnuto. S obzirom na to da je za njihovo ispravno zadavanje potrebno određeno znanje iz područja evolucijskog računarstva, taj se zadatak ne može prepustiti korisniku *Sustava*. Umjesto toga, potrebno je pronaći pogodne postavke koje će se koristiti u svakom postupku izrade rasporeda, bez sudjelovanja korisnika.

Unatoč činjenici da su sva tri ostvarena algoritma razvijena na istim temeljima, vrlo je mali skup parametara koji su im zajednički. To su: *veličina početne populacije*, *vjerojatnost križanja*, *vjerojatnost mutacije* i *broj elitističkih jedinki*. Kako operator križanja čini okosnicu rada genetskog algoritma, vjerojatnost njegove primjene najčešće se postavlja na veliku vrijednost (oko 90%) i ne mijenja se značajnije. Slično je i s brojem elitističkih jedinki. Dovoljna je jedna za očuvanje najboljeg dosad pronađenog rješenja. Veličina početne populacije parametar je koji može značajnije utjecati na rad *Sustava*. Pritom treba uočiti kako on nema veliki utjecaj na rad algoritma *RAPGA*, jer

će već nakon prvog koraka populacija vjerojatno biti različite veličine. Takav slučaj moguće je djelomično nadzirati promišljenim zadavanjem najveće i najmanje dopuštene veličine populacije. S druge strane, mijenjanje vjerojatnosti mutacije moglo bi kod sva tri algoritma dovesti do uočavanja određenih pravilnosti i poboljšanja ukupnih rezultata.

U cilju postizanja što točnijih rezultata, algoritmi *OS* i *RAPGA* djelomično su izmijenjeni za potrebe testiranja. Naime, navedene inačice genetskog algoritma mogu lako odrediti trenutak u kojem se izvođenje može zaustaviti, jer nastavak ne bi doveo do značajnije boljih rezultata. Kod algoritma *OS* to je trenutak u kojem trenutni selekcijski pritisak (`ActSelPress`) dosegne najveću dopuštenu vrijednost (`MaxSelPress`). Algoritam *RAPGA* zaustavlja se u trenutku kad nije moguće stvoriti populaciju najmanje dopuštene veličine. Oba su slučaja posljedica nemogućnosti stvaranja potrebnog broja "uspješnih" jedinki unutar dopuštenog broja pokušaja. Prilikom testiranja, ova ograničenja su ukinuta i navedenim algoritmima je omogućeno beskonačno izvođenje. U slučaju nemogućnosti stvaranja dovoljnog broja "uspješnih" jedinki, populacija (`MinPop` u slučaju *RAPGA*) popunjava se lošijim jedinkama i postupak se nastavlja.

Školski je raspored iznimno složen sustav sastavljen od velikog broja predavanja koja je moguće razmjestiti na velik broj načina. Nerijetko međusobnom zamjenom dvaju ili više predavanja dobivamo, iako strukturno različite, kvalitetom jednake rasporede, što predstavlja problem permutacija pri izradi rasporeda. Prostor mogućih rješenja prepun je lokalnih optimuma koji uvelike otežavaju rad operatora križanja. Križanje je u ovakvim slučajevima iznimno teško jer uključuje združivanje značajki dviju ili više jedinki, a premještanje već jednog predavanja uvelike može utjecati na ukupnu kvalitetu rješenja. Iz navedenih je razloga u postupak testiranja uključen i jedan oblik *evolucijske strategije*.

Evolucijske su strategije algoritmi evolucijskog računanja koje se oslanjaju isključivo na operator mutacije. Inačica korištena u ovom postupku testiranja sastoji se od populacije zadane veličine iz koje se operatorom selekcije bira jedna jedinka. Klon odabrane jedinke mutira se i, ukoliko je zadovoljavajuće kvalitete, postaje dio populacije pri čemu se najlošija jedinka izbacuje. Postupak se nastavlja odabirom nove jedinke iz populacije. Cilj uvođenja inačice *evolucijskih strategija* u postupak testiranja je ispitati u kojoj je mjeri operator križanja koristan prilikom izrade školskog rasporeda.

U prvom su krugu testirani parametri koji utječu na veličinu populacije (20, 50, 100, 200 jedinki i dodatno populacija od 2 jedinke kod *evolucijske strategije*) i vjerojatnost mutacije (0.001, 0.002, 0.004 i 0.008). Ostali parametri, koliko je to bilo

moguće, međusobno su izjednačeni između algoritama. Tako je, na primjer, parametar $MaxEffort$ algoritma ***RAPGA*** usklađen s parametrom $MaxSelPress$ algoritma ***OS*** na način da je u oba algoritma izjednačen najveći broj jedinki koje će se u jednom koraku moći stvoriti. Na sličan način usklađeni su parametri $MinPop$ i $SuccRatio$ istih algoritama. Parametar $SuccRatio$ algoritma ***OS*** postavljen je na vrijednost $0,5$, dok je parametar $MaxSelPress$ postavljen na vrijednost 6. Kod oba algoritma faktor usporedbe mijenjao se između istih granica koje su iznosile: $donjaGranica = 0$ i $gornjaGranica = 0,75$. Prvi krug testiranja čini 68 testova, a cjelokupan popis parametara i testnih slučajeva nalazi se u dodatku D.

U drugom krugu testirani su postupci lokalnih pretraga za dva najbolja testna primjera iz prvog kruga. S obzirom na to da su u sklopu rada razvijene 3 lokalne pretrage, krug čini 14 testova, po 7 za svaki od dva najbolja rezultata iz prvog kruga, što predstavlja sve slučajeve međusobnog združivanja postupaka lokalnih pretraga.

6.2. Provođenje pokusa

Prvi krug testiranja sastavljen je od ukupno 680 pokusa, dok drugi čini njih 170, što znači da je svaki testni primjer pokretan 10 puta. Izvođenje pokusa vremenski je ograničeno. Svaki pokus izvodi se točno jedan sat. Nakon isteka vremena, izvršavanje algoritma se zaustavlja i bilježi se najbolji pronađeni rezultat. Svi algoritmi pri svakom pokretanju ponovno pripremaju strukture podataka, tako da su rezultati izvođenja u potpunosti neovisni. Kvaliteta rasporeda izražava se korištenjem težinske sume, a težinski faktori rezultata pojedinih procjenitelja prikazani su u tablici 6.1.

U oba kruga testiranja izrađuje se raspored na temelju istih parametara. Oni predstavljaju stvaran problem raspoređivanja u *Osnovnoj školi Voltino* za školsku godinu 2011./2012. U školi postoje 22 razredna odjeljenja, od kojih je 10 *nižih*[1] i 12 *viših*[2] razreda. Predavanja održavaju 24 predmetna nastavnika i 10 učitelja razredne nastave.

Nastava se može održavati u ukupno 24 učionice, među kojima je 10 razrednih (za nastavu *nižih* razreda), 12 predmetnih učionica i 2 dvorane za tjelesnu i zdravstvenu kulturu. Poželjno je da se nastava svakog predmeta za *više* razrede održava u za to predviđenim učionicama, ali ovaj zahtjev nije obavezno poštivati. Također, u nekim predmetnim učionicama (na primjer, učionica za glazbeni, likovni odgoj i informatiku) nije moguće održavati nastavu iz drugih predmeta. Sva razredna nastava, osim stranih jezika, održava se u razrednim učionicama.

[1] od 1. do 4. razreda

[2] od 5. do 8. razreda

Tablica 6.1: Težinski faktori rezultata procjenitelja u ukupnom rezultatu

Rd. br.	Procjenitelj	Težinski faktor
1	brojač nepoštivanja raspoloživosti razreda	50
2	brojač nepoštivanja raspoloživosti predavanja	50
3	brojač nepoštivanja raspoloživosti učionica	50
4	brojač nepoštivanja raspoloživosti nastavnika	50
5	brojač nepoštivanja neprekidnog rasporeda za razrede	35
6	brojač nepoštivanja neprekidnog rasporeda za nastavnike	35
7	brojač nepoštivanja radnih sati (u danu) za razrede	8
8	brojač nepoštivanja radnih sati (u danu i tjednu) za nastavnike	8
9	brojač predavanja raspoređenih u pogrešan tjedan	50
10	brojač predavanja raspoređenih u pogrešnu učionicu	10
11	brojač višesatnih predavanja razlomljenih između dva dana	100
12	brojač više predavanja istog predmeta u istom danu	25
13	brojač nepoštivanja veza između predavanja	30

Raspored se izrađuje za dva tjedna nastave po 5 radnih dana, u kojima se *višim* razredima izmjenjuju turnusi. Radni dan sastoji se od ukupno 13 nastavnih sati, po 6 u jutarnjem i poslije podnevnom turnusu i jedan u međuturnusu. U istom turnusu nastavu pohađaju peti i sedmi, odnosno šesti i osmi razredi. *Niži* razredi u oba tjedna pohađaju nastavu ujutro.

Svi razredi moraju imati neprekinuti raspored. Pritom se učiteljima razredne nastave dopušta da sami sastave rasporede za svoje razrede, uz jednu iznimku. S obzirom na to da vjeronauk i strane jezike *nižim* razredima predaju nastavnici predmetne nastave, predavanja ovih predmeta raspoređuju se sa svim ostalima prilikom izrade školskog rasporeda. To znači da, sa stajališta parametara rasporeda, *niži* razredi ne moraju imati neprekinuti raspored iz razloga što će sve moguće "pauze" popuniti učitelji razredne nastave drugim predavanjima.

Unutar navedena dva tjedna nastave potrebno je rasporediti 743 predavanja u ukupnom trajanju od 836 nastavnih sati. Osmi se razredi prilikom pohađanja nastave tjelesne i zdravstvene kulture dijele na muške i ženske odjeljke i ujedno spajaju s odgovarajućim odjeljcima susjednih razreda. Tako se spajaju muški i ženski odjeljci razreda $8.a$ i $8.b$, odnosno $8.c$ i $8.d$. Dodatno, predavanja likovne i tehničke kulture za sve razrede održavaju se u blokovima od dva školska sata svaki drugi tjedan. U cilju očuvanja

jednakog broja radnih sati u tjednu, između ovih predavanja za svaki razred postoji veza s pravilom da se ne smiju održavati u istom tjednu. Zbog svoje opsežnosti, cjelokupna parametarska datoteka nazvana `voltinoParamsFix4.xml` nalazi se na mediju priloženom ovom radu.

Osim parametara, prikupljen je i stvarni raspored *Osnovne škole Voltino* za školsku godinu 2011./2012. Opis njegove kvalitete i usporedba s rezultatima dobivenim u testiranjima nalaze se u nastavku poglavlja.

6.3. Rezultati

U sljedećim odlomcima navedeni su i komentirani rezultati provedenih mjerenja. Dobiveni rezultati prikazani su tablicama i grafovima i uspoređeni su s trenutnim rasporedom *Osnovne škole Voltino*.

6.3.1. Prvi krug testova

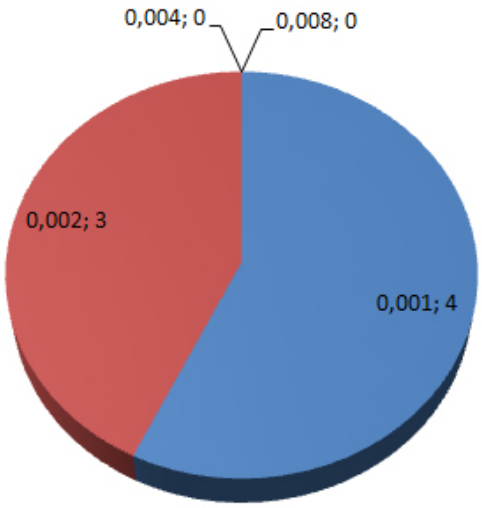
Po završetku testiranja, dobiveni rezultati grupirani su prema testnim primjerima kako bi se za svaki primjer izračunao medijan, aritmetička sredina i standardno odstupanje. Rezultati su grafički prikazani slikom 6.3, a detaljniji prikaz najboljih 10% rezultata naveden je u tablici 6.2.

Tablica 6.2: Podaci o najboljih 10% rezultata prvog kruga testiranja

		Rezultat		
Rang	Test	Medijan	Aritmetička sredina	Standardno odstupanje
1	6	7607	$7712,9$	$547,168$
2	5	7864	$7840,2$	$320,303$
3	1	8147	$8089,1$	$377,256$
4	9	8204	$8059,8$	$559,720$
5	2	$8294,5$	$8242,7$	$403,746$
6	10	$8428,5$	$8441,9$	$281,231$
7	37	$8448,5$	$8454,8$	$541,386$

Iz prikazanih rezultata lako je vidljiva značajna prevlast *evolucijske strategije* nad algoritmima koji u svojem radu koriste operator križanja. Od 7 najboljih rezultata, prvih 6 pripada pokusima u kojima se koristio ovaj algoritam. Ovakvi rezultati potvrđuju

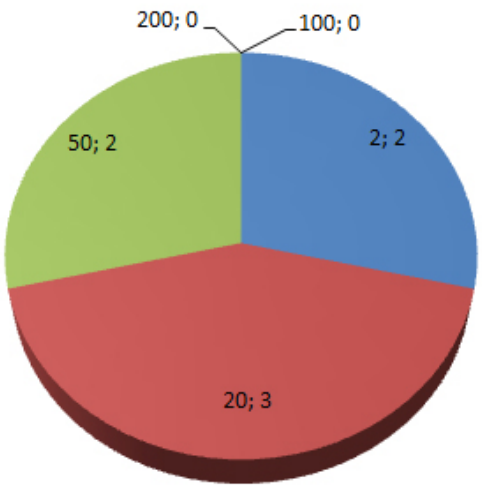
činjenicu da zbog prirode problema koji se rješava operator križanja ne može značajnije doprinijeti pronalasku boljih rješenja. Osjetljivost rasporeda na male promjene jasno se vidi i iz grafa 6.1, koji prikazuje koje su vjerojatnosti mutacija korištene u testovima s najboljim rezultatima. Iako su sve ispitivane vjerojatnosti razmjerno male, vidljivo je kako su najuspješnije mutacije one koje minimalno mijenjaju trenutno rješenje ($0,001$ i $0,002$). Ako se uzme u obzir podatak kako jedan postupak mutacije premješta najviše dva predavanja odjednom i da testirani raspored sadrži ukupno 743 predavanja, može se izračunati kako su najuspješnija mutiranja ona koja mijenjaju prosječno 2-3 predavanja u cijelom rasporedu. S obzirom na roditeljske jedinke, svaki će operator križanja premjestiti značajno veći broj predavanja, pa se u toj činjenici može pronaći razlog lošijeg učinka algoritama koji ga koriste.



Slika 6.1: Vjerojatnosti mutacija korištene u najboljih 10% testova

Graf na slici 6.2 prikazuje udio pojedinih veličina populacija koje su korištene u testovima s najboljim rezultatima. Vidljivo je kako su najuspješniji testovi oni s manjim populacijama, dok oni s populacijama od 100 i 200 jedinki uopće nisu zastupljeni. Pritom treba uzeti u obzir i podatak koji nije vidljiv iz grafa, već iz tablice 6.2. Najbolji rezultati postižu se s populacijama veličine 20 i 2 jedinke, dok su testovi s populacijom od 50 jedinki nešto lošiji. Sve to ukazuje na činjenicu kako je učestalost izmjene populacije važnija od njezine raznolikosti. Razlog tome vjerojatno je također visoka osjetljivost rasporeda na promjene. S obzirom na to da već vrlo mala mutacija može jedinku značajno premjestiti u prostoru kvalitete, velika raznolikost populacije nije potrebna. Dapače, velik broj usporednih točaka pretraživanja usporava postupak. Ipak, s obzirom na to da su najbolja dva rezultata postignuta s populacijama od 20 jedinki,

može se zaključiti kako sa smanjivanjem populacije nije dobro pretjerati.



Slika 6.2: Veličine populacija korištene u najboljih 10% testova

Najbolji rezultati prema korištenim algoritmima prikazani su tablicom 6.3. Od algoritama koji svoj rad temelje na operatoru križanja, najuspješniji je bio algoritam *OS*. Osnovni *GA* i algoritam *RAPGA* postigli su vidljivo lošije i međusobno slične rezultate. Razlog boljih rezultata *OS*-a vjerojatno je njegov način rada, u kojem udio sljedeće generacije jedinki mora biti bolji u određenoj mjeri bolji od svojih roditelja. Time se pretraga dodatno usmjerava prema boljim rješenjima i rezultati su jasno vidljivi.

Tablica 6.3: Najbolji rezultati algoritama

Test	Algoritam	Medijan
6	*evolucijska strategija*	7607
37	*OS*	8448,5
21	*osnovni GA*	11113
53	*RAPGA*	11497,5

Algoritam *RAPGA* također ima jedan oblik odabira jedinki za sljedeću generaciju. Pritom osim boljih, prihvaća i jedinke koje u populaciju uvode određenu raznolikost. Jedinka predstavlja raznolikost u populaciji ukoliko se identična jedinka već ne nalazi u njoj. S obzirom na iznimno velik broj mogućih rješenja problema raspoređivanja, vrlo je velika vjerojatnost da će gotovo svaka nova jedinka biti prihvaćena na temelju ovog kriterija, neovisno o njezinoj kvaliteti. Time rad ovog algoritma postaje vrlo sličan *osnovnom genetskom algoritmu*, što je i vidljivo u postignutim rezultatima.

Moguće je ostvariti još jednu inačicu algoritma *RAPGA*. U njoj bi se za novu generaciju odabirale samo one jedinke koje zadovoljavaju oba uvjeta: kvalitetu s obzirom na roditelje i raznolikost s obzirom na već postojeće jedinke u populaciji. Nažalost, uvodni testovi pokazali su kako i u ovom slučaju algoritam ne postiže zadovoljavajuće rezultate. Naime, ovakva strategija može pomoći u početku izvođenja, ali je previše ograničavajuća u kasnijim fazama i izaziva brzo zaustavljanje. Čak i u slučaju da se algoritam prilagodi beskonačnom izvođenju, što je i napravljeno za potrebe testiranja, zbog prejakog ograničenja, ponašanje algoritma brzo prelazi u ono nalik osnovnom *GA*.

6.3.2. Drugi krug testova

Testovi pod rednim brojevima 5 i 6 postigli su najbolje rezultate u prvom krugu testiranja. Drugi je krug temeljen na parametrima ova dva testa, uz pokretanje razvijenih algoritama lokalnih pretraga. Cjelokupni rezultati drugog kruga prikazani su u tablici 6.4. Korištenje pojedinog algoritma lokalne pretrage u testu označeno je kvačicama u stupcima koji određuju redom:

- *Raspol.* — lokalnu pretragu poboljšanja raspoloživosti razreda, nastavnika, učionica i predavanja (`AvailabilitiesOptimization`),
- *Nepr. rasp.* — lokalnu pretragu poboljšanja neprekidnosti rasporeda za razrede i nastavnike (`ContinousScheduleOptimization`) i
- *Više pred.* — lokalnu pretragu poboljšanja rasporeda predavanja istog predmeta u danu (`MultipleLecturesOfSameDayOptimization`).

Većina vrijednosti pokazuje jasno poboljšanje rezultata s obzirom na one dobivene bez pokretanja algoritama lokalnih pretraga. U najboljem slučaju poboljšanje iznosi oko 12%, dok su najlošiji rezultati postignuti koristeći algoritme lokalnih pretraga tek neznatno lošiji od najboljih rezultata dobivenih bez njih. Iako su rezultati u tablici poredani prema rastućoj vrijednosti medijana ukupnog rezultata, s obzirom na vrlo sličan učinak najbolja četiri testa i razmjerno veliku devijaciju prvog, teško je sa sigurnošću utvrditi koji je skup parametara najbolji. Pritom treba uzeti u obzir činjenicu kako su podaci o učinku algoritama u oba kruga izračunati na temelju 10 pokretanja svakog testa, što je prilično malen broj. Veći broj pokretanja nažalost nije bio moguć zbog ograničenja u raspoloživosti potrebnog broja testnih računala.

Unatoč spomenutom problemu, vrijednosti u tablici pokazuju pozitivan učinak algoritma lokalne pretrage po raspoloživostima. Svi testovi u kojima se koristi ova pre-

Tablica 6.4: Rezultati drugog kruga testiranja

		Lokalne pretrage			Rezultat		
Rang	Test	Raspol.	Nepr. rasp.	Više pred.	Med.	Arit. sr.	Stand. ods.
1	1	✓			6683	6867,4	600,177
2	2	✓	✓		6829	6763	274,837
3	10	✓	✓	✓	6881,5	6808,4	328,081
4	4	✓		✓	6942	6953,4	225,312
5	3	✓	✓	✓	7025,5	7063,7	379,528
6	9	✓	✓		7057	7089,3	275,655
7	8	✓			7171,5	7239,3	347,358
8	11	✓		✓	7219	7227	365,172
9	6		✓	✓	7583	7745	578,558
10	5		✓		7625,5	7581,7	672,076
11	12		✓		7831	7915,4	290,922
12	14			✓	7854	7780,3	382,216
13	7			✓	8109,5	8166,7	402,200
14	13		✓	✓	8188,5	8162,5	477,331

traga postigli su najbolje rezultate. Međutim, ovi rezultati ne mogu se pripisati isključivo učinkovitosti algoritma. Težinski faktori kojima se množe kazne raspoloživosti veći su od onih kojima se množe kazne neprekidnosti rasporeda ili većeg broja predavanja istog predmeta u danu. Posljedično tome, jedinično poboljšanje lokalne pretrage raspoloživosti ima veći učinak na konačan rezultat.

Dodatan problem predstavlja međuovisnost kriterija. Vrlo je velika vjerojatnost da će se unaprjeđujući raspored prema jednom kriteriju, povećati kazna drugog. To se prije svega odnosi na kriterije koji se množe većim faktorima, jer će njihovo unaprjeđenje rezultirati boljom ukupnom kvalitetom rasporeda, unatoč pogoršanju nekog drugog kriterija manje važnosti. Time se ujedno dobiva raspored razmjerno velike ukupne kvalitete s razmjerno lošim svojstvima prema kriterijima manje važnosti.

Dok je s jedne strane navedeno ponašanje logično, u stvarnosti je jednako loš raspored koji ima razmjerno malen broj kršenja važnijih ograničenja ili razmjerno velik broj kršenja ograničenja manje važnosti. Sve to upućuje na zaključak kako je iznimno važno na ispravan način odabrati težinske faktore na temelju kojih se izračunava ukupna kvaliteta rasporeda. Međutim, ovaj zadatak postaje vrlo težak kada se uzme u

obzir činjenica da satničari različito vrednuju iste kriterije.

S obzirom na sve navedeno, iznimno je važan rezultat testa broj 10. U njemu se koriste sva tri algoritma lokalnih pretraga, a postignuti je rezultat među najboljima. To je značajno jer ukazuje na činjenicu da je moguće koristiti sve postupke lokalnih pretraga, unaprjeđujući kriterije veće i manje važnosti, a istovremeno očuvati ukupnu kvalitetu rasporeda.

6.3.3. Usporedba rezultata sa stvarnim školskim rasporedom

Najbolja četiri skupa postavki drugog kuga testiranja korištena su u konačnom ispitivanju učinkovitosti *Sustava*. Svaki od testova ponovno je pokrenut 10 puta, u trajanju od 2 sata po pokretanju. Rezultati su većinom sumjerljivi, ali je test broj 10 postigao rezultate s najmanjim odstupanjima, što je posebno važno prilikom korištenja *Sustava*. U tablici 6.5 uspoređen je medijan dobivenih rezultata s izračunatom kvalitetom stvarnog rasporeda koji se ove školske godine koristi u *Osnovnoj školi Voltino*.

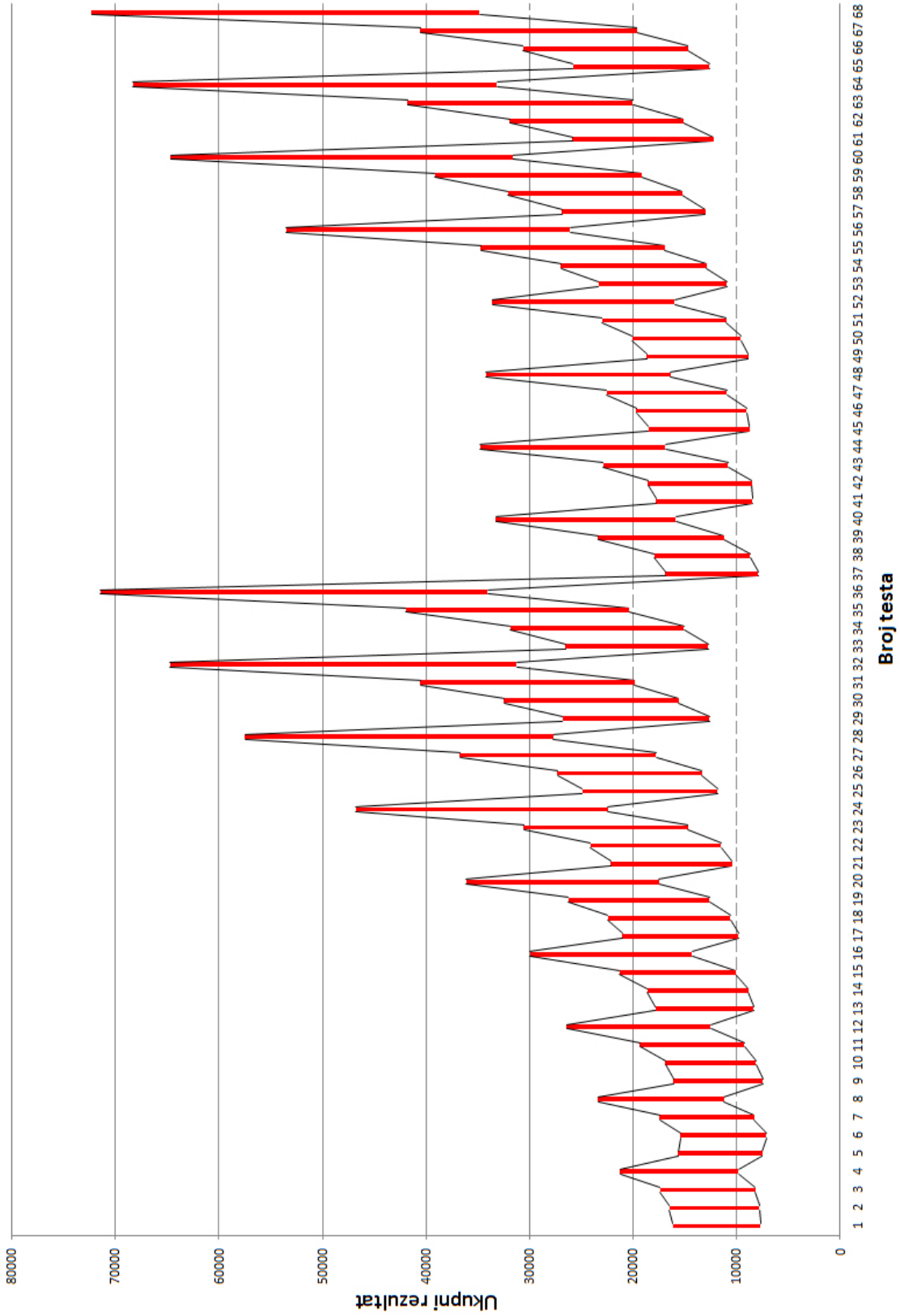
Tablica 6.5: Usporedba kvalitete medijana izrađenih rasporeda i stvarnog rasporeda *Osnovne škole Voltino*

Kriterij	Izrađeni rasp.	Stvaran rasp.
Ukupni rezultat	4777	4623
Broj predavanja u manje poželjnim učionicama	70	110
Broj predavanja prelomljenih između dva dana	0	0
Broj predavanja raspoređenih u pogrešan tjedan	0	0
Broj nepoštivanih veza između predavanja	19	6
Broj nepoštivanja raspoloživosti nastavnika	1	0
Broj nepoštivanja raspoloživosti razreda	3	0
Broj nepoštivanja raspoloživosti učionica	0	0
Broj nepoštivanja raspoloživosti predavanja	2	0
Broj nepoštivanja neprekidnosti rasporeda za razrede	2	0
Broj nepoštivanja neprekidnosti rasporeda za nastavnike	10	26
Broj više predavanja istog predmeta u istom danu	139	97
Broj nepoštivanja broja radnih sati za nastavnike	32	0
Broj nepoštivanja broja radnih sati za razrede	7	1

Vidljivo je kako je prema ukupnoj kvaliteti izrađeni raspored neznatno lošiji od stvarnog školskog rasporeda. Međutim, s obzirom na to da je ukupan rezultat izračunat

kao težinska suma kvaliteta prema kriterijima, potrebno je i njih usporediti.

Gledajući pojedinačno rezultate prema kriterijima, *Sustav* bolje raspoređuje predavanja u poželjne učionice i izrađuje kvalitetniji raspored za nastavnike. Većina ostalih kriterija nažalost lošije je zadovoljena. Subjektivnom procjenom prema zadovoljenju pojedinih kriterija, izrađeni raspored još uvijek nije spreman za korištenje u školi. Unatoč tome, zasigurno bi bio vrlo dobra početna točka za ručnu izradu. Također, proširenjem *Sustava* ciljanim heurističkim algoritmima koji bi djelovali na problematične slučajeve, kvaliteta bi se mogla dodatno unaprijediti.



Slika 6.3: Rezultati prvog kruga testova (aritmetička sredina i standardna devijacija rezultata prvog kruga testiranja)

7. Zaključak

Gotovo svi događaji u našim životima organizirani su nekim oblikom rasporeda. Unatoč tome, izrada rasporeda još i danas je iznimno težak zadatak i njegovo rješavanje nije jednostavno automatizirati. Iz tog razloga, većina osnovnih i srednjih škola u Republici Hrvatskoj svake godine izdvaja značajna novčana sredstva za izradu rasporeda nastave. U ovom je radu detaljno opisan problem izrade rasporeda nastave za osnovne i srednje škole. Problem je predstavljen u više oblika, počevši od najjednostavnijeg i nadograđujući ga dodatnim zahtjevima. U sklopu rada provedeni su razgovori s predstavnicima pet osnovnih i srednjih škola u gradovima Zagreb i Sveta Nedelja s ciljem prikupljanja detaljnih podataka o potrebama i načinima izrade rasporeda. Neki od tih zahtjeva formalno su opisani i ovom radu.

U radu su izneseni i oblici pristupa problemu izrade rasporeda, kao i opis njegove složenosti. Dodatno, opisan je jedan heuristički postupak izrade rasporeda, kao i tri opće metode iz područja evolucijskog računarstva (genetski algoritam, algoritam odabira potomaka i algoritam očuvanja važnih alela) kojima se izrada može automatizirati. Svaki algoritam opisan je u svom izvornom obliku, a opisani su i postupci kojima se može prilagoditi rješavanju navedenog problema. Razvijen je i *Sustav za automatiziranu izradu rasporeda nastave*, u sklopu kojeg su ostvareni spomenuti algoritmi. *Sustav* je moguće proširiti novim algoritmima i vrednovateljima rješenja.

Provedena su dva kruga ispitivanja učinkovitosti ostvarenih algoritama u ovisnosti o zadanim parametrima. Dodatno se uz spomenute algoritme ispitivao učinak *evolucijske strategije* kao algoritma koji ne koristi operator križanja. U svim testovima rješavao se stvaran problem raspoređivanja u *Osnovnoj školi Voltino*.

Prvi je krug sastavljen od 68 testova kojima se ispitivao utjecaj veličine populacije i vjerojatnosti mutacije jedinke. Najbolje rezultate postigla je *evolucijska strategija*, neovisno o ispitivanim parametrima, dok su algoritmi temeljeni na operatoru križanja postigli vidljivo lošiji učinak. Uzrok tome vjerojatno je činjenica kako je raspored vrlo osjetljiv na male promjene. S obzirom na to da operator križanja u velikoj mjeri mijenja jedinku, veliki su izgledi da pritom neće biti uspješan. Ovaj zaključak podržava i

činjenica kako su bolji rezultati postignuti koristeći manje vjerojatnosti mutacije. Također, bolji su rezultati postignuti uz manje veličine populacija. Na temelju toga bi se moglo zaključiti kako je učestalost izmjene populacije važnija od njezine raznolikosti, što je također povezano s osjetljivošću rasporeda na promjene.

U drugom je krugu kroz 14 testova ispitivan utjecaj postupaka lokalnih pretraga na kvalitetu izrađenog rasporeda. Rezultati pokazuju njihov pozitivan učinak na ukupan rezultat, iako nije moguće sa sigurnošću odrediti njihovu pojedinačnu uspješnost. Naime, postupci lokalnih pretraga djeluju na samo neke od kriterija na temelju kojih se određuje kvaliteta rasporeda. Unaprjeđujući raspored s obzirom na jedan kriterij, često se smanjuje kvaliteta s obzirom na druge. Pritom unaprjeđenje ukupne kvalitete ne ovisi samo o uspješnosti postupka lokalne pretrage, već i o težinskim faktorima kojima se množe procjene kvalitete rasporeda prema kriterijima.

Naposljetku su najbolji dobiveni rezultati uspoređeni sa stvarnim rasporedom koji se u ovoj školskoj godini koristi u *Osnovnoj školi Voltino*. Iako *Sustav* zasad nije uspio doseći kvalitetu ručno izrađenog rasporeda, dobivena su rješenja obećavajuća.

Prvi sljedeći korak zasigurno bi trebao biti detaljno ispitivanje rada *Sustava* prilikom rješavanja problema drugih škola. U cilju dodatnog poboljšanja kvalitete rješenja, nameće se mogućnost ostvarenja većeg broja heuristika koje bi oponašale ljudski postupak unaprjeđena rasporeda. Time bi se izravno djelovalo na zadovoljavanje kriterija koji su se u dosadašnjim slučajevima pokazali problematičnima. To se prije svega odnosi na veći broj predavanja istog predmeta u jednom danu.

S druge strane, trebalo bi razviti učinkovit način određivanja težinskih faktora kojima se množe vrijednosti kvalitete po kriterijima. S obzirom na to da bi korisnicima *Sustava* trebalo omogućiti izmjenu važnosti pojedinih kriterija, ovo predstavlja velik problem.

Ostvarenje drugih metaheuristika također bi mogao doprinijeti kvaliteti dobivenih rješenja. Pritom bi se trebalo usmjeriti algoritmima koji su temeljeni na operatoru mutacije, s obzirom da križanje previše mijenja rješenje i time često narušava kvalitetu. Moguće je i razviti mehanizam usporednog izvršavanja različitih algoritama i međusobne razmjene rješenja, čime bi se utjecalo na otpornost *Sustava* na zapinjanje u lokalnim optimumima. Strukture podataka za takav način rada već su ostvarene u sklopu *Sustava*.

LITERATURA

D. Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. *Management Science*, stranice 98–113, 1991.

M. Affenzeller i S. Winkler. *Genetic algorithms and genetic programming: modern concepts and practical applications*, svezak 6. Chapman & Hall/CRC, 2009.

T. Bronić. *Grafičko korisničko sučelje za uređivanje školske satnice, diplomski rad*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2012.

M. Bufé, T. Fischer, H. Gubbels, C. Häcker, O. Hasprich, C. Scheibel, K. Weicker, N. Weicker, M. Wenig, i C. Wolfangel. Automated solution of a highly constrained school timetabling problem-preliminary results. *Applications of Evolutionary Computing*, stranice 431–440, 2001.

A. Colorni, M. Dorigo, i V. Maniezzo. A genetic algorithm to solve the timetable problem. *Politecnico di Milano, Milan, Italy TR*, stranice 90–060, 1992.

M. Čupić. *Prirodom inspirirani optimizacijski algoritmi, skripta u sklopu predmeta Umjetna inteligencija*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2009.

K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. University Microfilms International, 1975.

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

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

S. Even, A. Itai, i A. Shamir. On the complexity of time table and multi-commodity flow problems. U *Foundations of Computer Science, 1975., 16th Annual Symposium on*, stranice 184–193. IEEE, 1975.

L.J. Fogel, A.J. Owens, i M.J. Walsh. *Artificial intelligence through simulated evolution*. John Wiley, 1966.

M.R. Garey i D.S. Johnson. Computers and intractability. *A Guide to the Theory of NP-Completeness, Bell Laboratories, Murray Hill, NJ*, 1979.

D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley, 1989.

M. Golub. *Genetski algoritam, drugi dio, skripta*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2004a.

M. Golub. *Genetski algoritam, prvi dio, skripta*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2004b.

J.H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314, 1962.

J.H. Holland. Nonlinear environments permitting efficient adaptation. U *Computer and information sciences, II: proceedings*. Academy Press, 1967.

J.H. Holland. *Adaptation in natural and artificial systems*. Broj 53. University of Michigan press, 1975.

J.E. Hopcroft i R.M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite. U *Switching and Automata Theory, 1971., 12th Annual Symposium on*, stranice 122–125. IEEE, 1971.

W. Junginger. Timetabling in germany: a survey. *Interfaces*, stranice 66–74, 1986.

DB Papoulias. The assignment-to-days problem in a school time-table, a heuristic approach. *European Journal of Operational Research*, 4(1):31–41, 1980.

S. Pribil. *Izrada rasporeda nastave za osnovne i srednje škole, seminarski rad*. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2011.

I. Rechenberg. Cybernetic solution path of an experimental problem",(royal aircraft establishment translation no. 1122, bf toms, trans.). *Farnsborough Hants: Ministery of Aviation, Royal Aircraft Establishment*, 1122, 1965.

A. Schaerf. *Tabu search techniques for large high-school timetabling problems*. Computer Science, Department of Interactive Systems, CWI, 1996.

A. Schaerf. A survey of automated timetabling. *Artificial intelligence review*, 13(2): 87–127, 1999.

S. Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. U *Proceedings of the sixth international congress on genetics*, svezak 1, stranice 356–366, 1932.

Dodatak A
Popis zahtjeva i ostalih ulaznih parametara za izradu rasporeda

Dodatak A sadrži popis svih zahtjeva koji su prikupljeni u razgovorima s predstavnicima pet osnovnih i srednjih škola u gradovima Zagreb i Sveta Nedelja, kao i pregledom mogućnosti trenutno postojećih računalnih rješenja za izradu rasporeda nastave. Zahtjevi su, zbog preglednosti, navedeni kao popis ulaznih parametara *Sustava za izradu rasporeda* i podijeljeni u sedam glavnih cjelina na koje se odnose.

1. **Raspored**
   (a) tip rasporeda (jedan, dva, više tjedana)
      i. broj dana u tjednu
   (b) broj i termini početaka satova
      i. "granice" turnusa (na primjer, za $2.d$ i $3.c$)
      ii. sati od kojih raspored može početi ("dozvole" za dolazak na $2.$, $3.$ sat...)
   (c) zastavica za izbjegavanje dva "duža" predavanja u danu
      i. trajanje "dužeg" predavanja
   (d) zastavica kontrole kapaciteta dvorana i veličina grupa
   (e) broj dvorana (za slučaj da nisu sve unesene)
2. **Nastavnik**
   (a) raspoloživost u školskim satima
      i. poželjni sati

ii. poželjni dani u tjednu

iii. mogućnost zadavanja broja slobodnih dana (bilo kojih)

iv. mogućnost zadavanja broja dana koje mora raditi (bilo kojih)

(b) zastavica zahtijevanja predavanja istoj generaciji u istom danu

(c) ukoliko predaje više predmeta, zastavica predavanja istog predmeta u istom danu

(d) ukoliko radi u oba turnusa, zastavica izbjegavanja dva dana za redom u istom turnusu

(e) najveći i najmanji broj sati u danu

(f) pauza ukoliko je broj sati u jednom danu veći od određenog broja

(g) najveći i najmanji broj rupa u danu

(h) najveći i najmanji broj rupa u tjednu

(i) najveća veličina rupe

3. **Predmet**

(a) raspoloživost u školskim satima

i. poželjni sati

ii. poželjni dani u tjednu

(b) faktor zahtjevnosti predmeta

(c) "položaj" predmeta u rasporedu (unutar nastave, međuturnus, suprotan turnus)

4. **Razred**

(a) podjele (grupe vezane uz predmete, npr.: eng-njem, tjM-tjZ, fiz-inf. . .)

i. ovisno o $1.d$ mogu ili ne moraju sadržavati brojeve učenika

(b) raspoloživost u školskim satima

i. poželjni sati

ii. poželjni dani u tjednu

(c) zastavica obaveznog neprekidnog rasporeda

(d) najveći broj sati u danu

(e) zadana učionica

(f) pomoćne učionice

(g) najveće dnevno opterećenje

5. **Dvorana**

6. raspoloživost u školskim satima

   (a) poželjni sati

   (b) poželjni dani u tjednu

7. kapacitet dvorane - ovisno o $1.d$ može ili ne mora biti navedeno

8. **Predavanje**

   (a) predmet (jedan)

   (b) nastavnici (jedan ili više)

   (c) razredi/grupe (jedan ili više)

      i. izravno zadani razredi/grupe kojima se predaje

   (d) raspoložive učionice (jedna ili više)

      i. izravno zadane učionice

      ii. broj potrebnih učionica za odabrati

      iii. zastavica "ne prati učionice" (tada se samo pazi na ukupan broj, ne i razmještaj predavanja - onda nije potrebno navesti $6.d.i$)

   (e) trajanje predavanja

      i. indikator dozvole da algoritam može sam "razlomiti" predavanje u više manjih blokova (u tom slučaju preslikavanje nije $1:1$ - jedno zadano predavanje se preslikava u više predavanja u rasporedu)

         – najveća i najmanja veličina tako nastalih manjih blokova

   (f) tjedan u kojem se održava (ako je raspored višetjedni)

      i. mogućnost odabira: "jednom u rasporedu" (znači jednom u dva/više tjedana) - za 1.5 (i slično) sati tjedno

   (g) čvrsto zadan/i termin/i održavanja predavanja (ima ih najviše onoliko za koliko se tjedana radi raspored) - isključuje $6.c.ii$ i $6.e.i$ jer se odnosi na točno zadani termin

9. **Veze**

   (a) izravno zadano dva ili više predavanja, ili

   (b) predmeti i razredi na koje se odnosi pravilo

   (c) pravila:

      i. moraju/ne smiju biti u istom danu

      ii. moraju/ne smiju biti u isto vrijeme

      iii. moraju/ne smiju biti uzastopno

         – ako moraju biti uzastopno, zastavica je li važan poredak

      iv. moraju/ne smiju biti u istom tjednu (samo za $6.f.i$)

Dodatak B
Sažetak ulazne parametarske datoteke

U nastavku se nalazi primjer ulazne parametarske datoteke u *XML* zapisu. Zbog opširnosti iste ovdje je naveden samo njezin sažetak, a cjelokupna datoteka korištena prilikom testiranja *Sustava* nalazi se na mediju priloženom ovom radu.

Okvir B.1: Sažetak ulazne parametarske datoteke

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xmlScheduleParams>
 <weekCount>2</weekCount>
 <dayCount>5</dayCount>
 <avoidingLargeBlocksEnabled>false</avoidingLargeBlocksEnabled>
 <largeBlockSize>2</largeBlockSize>
 <roomSizeManagementEnabled>false</roomSizeManagementEnabled>
 <periods>
  <period id="j1">08:00 - 08:45</period>
  <period id="j2">08:50 - 09:35</period>
...
  <period id="m1">13:10 - 13:55</period>
  <period id="p1">14:00 - 14:45</period>
  <period id="p2">14:50 - 15:35</period>
...
 </periods>
 <turnuses>
  <turnus id="jut">
   <name>Jutarnji turnus</name>
   <periods>j1,j2,j3,j4,j5,j6</periods>
   <classStartPeriods>j1,j2</classStartPeriods>
  </turnus>
...
```

```
 </turnuses>
 <rooms>
...
  <room id="hrv1">
   <name>Hrvatski 1</name>
   <availabilities>
    <weekAvailability>
     <periodPreferences>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
     </periodPreferences>
     <dayPreferences>1,1,1,1,1</dayPreferences>
    </weekAvailability>
    <weekAvailability>
     <periodPreferences>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
     </periodPreferences>
     <dayPreferences>1,1,1,1,1</dayPreferences>
    </weekAvailability>
   </availabilities>
  </room>
...
 </rooms>
 <subjects>
...
  <subject id="hrv">
   <name>Hrvatski jezik</name>
   <defaultRooms>hrv1,hrv2</defaultRooms>
   <availabilities>
    <weekAvailability>
     <periodPreferences>
```

```
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
     </periodPreferences>
     <dayPreferences>1,1,1,1,1</dayPreferences>
    </weekAvailability>
    <weekAvailability>
     <periodPreferences>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
     </periodPreferences>
     <dayPreferences>1,1,1,1,1</dayPreferences>
    </weekAvailability>
   </availabilities>
  </subject>
...
 </subjects>
 <teachers>
...
  <teacher id="rpapic">
   <name>Radojka Papic</name>
   <availabilities>
    <weekAvailability>
     <periodPreferences>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
     </periodPreferences>
     <dayPreferences>1,1,1,1,1</dayPreferences>
    </weekAvailability>
    <weekAvailability>
```

```
      <periodPreferences>
       <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
       <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
       <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
       <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
       <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</preference>
      </periodPreferences>
      <dayPreferences>1,1,1,1,1</dayPreferences>
     </weekAvailability>
    </availabilities>
    <sameGenerationInDayRequired>
     false
    </sameGenerationInDayRequired>
    <sameSubjectInDayRequired>
     false
    </sameSubjectInDayRequired>
    <dayTurnusAlternationEnabled>
     false
    </dayTurnusAlternationEnabled>
    <daysForWeekTurnusAlternation>
     false,false,false,false,true
    </daysForWeekTurnusAlternation>
    <workingPeriodNumber>2 - 7</workingPeriodNumber>
    <mandatoryBreakThreshold>7</mandatoryBreakThreshold>
    <dayBreakNumber>0 - 1</dayBreakNumber>
    <weekBreakNumber>1 - 2</weekBreakNumber>
    <maximumBreakSize>1</maximumBreakSize>
   </teacher>
...
 </teachers>
 <classes>
...
   <class id="8a">
    <name>8.a</name>
    <partitions>
     <partition>musko:null;zenski:null</partition>
    </partitions>
    <availabilities>
```

```
    <weekAvailability>
     <periodPreferences>
      <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
      <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
      <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
      <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
      <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
     </periodPreferences>
     <dayPreferences>1,1,1,1,1</dayPreferences>
    </weekAvailability>
    <weekAvailability>
     <periodPreferences>
      <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
      <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
      <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
      <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
      <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
     </periodPreferences>
     <dayPreferences>1,1,1,1,1</dayPreferences>
    </weekAvailability>
   </availabilities>
   <continuousScheduleRequired>
    true
   </continuousScheduleRequired>
   <workingPeriodNumber>1 - 7</workingPeriodNumber>
  </class>
...
 </classes>
 <lectures>
...
  <lecture id="hrv:rpapic:6a:1">
   <subject>hrv</subject>
   <teachers>rpapic</teachers>
   <classes>6a</classes>
   <classrooms>
    hrv1,hrv2,eng,povGeo,mat,njem,teh,bioKem,fiz
   </classrooms>
   <necessaryClassroomCount>1</necessaryClassroomCount>
```

```
    <lectureDuration>2</lectureDuration>
    <weeks>1</weeks>
    <availabilities>
     <weekAvailability>
      <periodPreferences>
       <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
       <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
       <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
       <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
       <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</preference>
      </periodPreferences>
      <dayPreferences>1,1,1,1,1</dayPreferences>
     </weekAvailability>
    </availabilities>
    <regularScheduleManagementEnabled>
     true
    </regularScheduleManagementEnabled>
   </lecture>
...
   <lecture id="geo:asimac:8a:3">
    <subject>geo</subject>
    <teachers>asimac</teachers>
    <classes>8a</classes>
    <classrooms>
     hrv1,hrv2,eng,povGeo,mat,njem,teh,bioKem,fiz
    </classrooms>
    <necessaryClassroomCount>1</necessaryClassroomCount>
    <lectureDuration>1</lectureDuration>
    <weeks>2</weeks>
    <availabilities>
     <weekAvailability>
      <periodPreferences>
       <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
       <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
       <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
       <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
       <preference>1,1,1,1,1,1,1,0,0,0,0,0,0</preference>
      </periodPreferences>
```

```
      <dayPreferences>1,1,1,1,1</dayPreferences>
     </weekAvailability>
    </availabilities>
    <regularScheduleManagementEnabled>
     true
    </regularScheduleManagementEnabled>
   </lecture>
...
 </lectures>
 <connections>
...
   <connection id="5a:lik-teh:1">
    <lectureIds>
     lik:ngolubic:5a:1,teh:ssostaric:5a:1
    </lectureIds>
    <rule>MUST_NOT_BE</rule>
    <condition>SAME_WEEK</condition>
   </connection>
...
 </connections>
</xmlScheduleParams>
```

Dodatak C
Primjeri izlaznih datoteka

U ovom dodatku nalaze se primjeri izlaznih datoteka u kojima je zapisan izrađeni raspored. Trenutno su podržana dva oblika zapisa: *XML* i *Excel*. Slično kao i u slučaju parametarske datoteke, ovdje su navedeni samo sažeti prikazi izlaznih datoteka, a cjelokupni zapisi nalaze se na priloženom mediju.

Okvir C.1: Sažetak izlazne *XML* datoteke s izrađenim rasporedom

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xmlSchedule>
 <lectures>
  <lecture>
   <lectureId>hrv:rpapic:6a:1</lectureId>
   <slot>45</slot>
   <locations>hrv2</locations>
  </lecture>
...
  <lecture>
   <lectureId>geo:gcindric:6a:4</lectureId>
   <slot>118</slot>
   <locations>povGeo</locations>
  </lecture>
...
 </lectures>
</xmlSchedule>
```

Slika C.1: Primjer izlazne *Excel* datoteke (raspored nastavnika)

Slika C.2: Primjer izlazne *Excel* datoteke (raspored razreda)

Slika C.3: Primjer izlazne *Excel* datoteke (raspored prostorija)

Dodatak D
Popis testnih slučajeva prvog kruga

U nastavku su navedeni popisi testnih slučajeva prvog kruga testiranja. U prvom su krugu ispitivani utjecaj veličine populacije i vjerojatnosti mutacije na učinak algoritama. S obzirom na to da ispitivani algoritmi ne ovise o jednakim parametrima, pojedini parametre bilo je potrebno mijenjati u ovisnosti o veličini početne populacije koja se koristi u testu. Tablica D.1 prikazuje parametre algoritma *RAPGA* koji se mijenjaju u ovisnosti o zadanoj veličini početne populacije. U tablici D.2 naveden je popis testnih slučajeva prvog kruga testiranja.

Tablica D.1: Dodatni parametri algoritama *RAPGA* koji se mijenjaju u ovisnosti o veličini početne populacije

Početna populacija	Parametri
20	$MinPop = 10$ $MaxPop = 30$ $MaxEffort = 60$
50	$MinPop = 25$ $MaxPop = 75$ $MaxEffort = 150$
100	$MinPop = 50$ $MaxPop = 150$ $MaxEffort = 300$
200	$MinPop = 100$ $MaxPop = 300$ $MaxEffort = 600$

Tablica D.2: Parametri zadani u pojedinim testnim slučajevima prvog kruga testiranja

Test	Algoritam	Početna populacija	Vjerojatnost mutacije
1	*evolucijska strategija*	2	0.001
2	*evolucijska strategija*	2	0.002
3	*evolucijska strategija*	2	0.004
4	*evolucijska strategija*	2	0.008
5	*evolucijska strategija*	20	0.001
6	*evolucijska strategija*	20	0.002
7	*evolucijska strategija*	20	0.004
8	*evolucijska strategija*	20	0.008
9	*evolucijska strategija*	50	0.001
10	*evolucijska strategija*	50	0.002
11	*evolucijska strategija*	50	0.004
12	*evolucijska strategija*	50	0.008
13	*evolucijska strategija*	100	0.001
14	*evolucijska strategija*	100	0.002
15	*evolucijska strategija*	100	0.004
16	*evolucijska strategija*	100	0.008
17	*evolucijska strategija*	200	0.001
18	*evolucijska strategija*	200	0.002
19	*evolucijska strategija*	200	0.004
20	*evolucijska strategija*	200	0.008
21	*osnovni GA*	20	0.001
22	*osnovni GA*	20	0.002
23	*osnovni GA*	20	0.004
24	*osnovni GA*	20	0.008
25	*osnovni GA*	50	0.001
26	*osnovni GA*	50	0.002
27	*osnovni GA*	50	0.004
28	*osnovni GA*	50	0.008
29	*osnovni GA*	100	0.001
30	*osnovni GA*	100	0.002
31	*osnovni GA*	100	0.004

Nastavak na sljedećoj stranici...

Tablica D.2 – Nastavak

Test	Algoritam	Početna populacija	Vjerojatnost mutacije
32	*osnovni GA*	100	0.008
33	*osnovni GA*	200	0.001
34	*osnovni GA*	200	0.002
35	*osnovni GA*	200	0.004
36	*osnovni GA*	200	0.008
37	*OS*	20	0.001
38	*OS*	20	0.002
39	*OS*	20	0.004
40	*OS*	20	0.008
41	*OS*	50	0.001
42	*OS*	50	0.002
43	*OS*	50	0.004
44	*OS*	50	0.008
45	*OS*	100	0.001
46	*OS*	100	0.002
47	*OS*	100	0.004
48	*OS*	100	0.008
49	*OS*	200	0.001
50	*OS*	200	0.002
51	*OS*	200	0.004
52	*OS*	200	0.008
53	*RAPGA*	20	0.001
54	*RAPGA*	20	0.002
55	*RAPGA*	20	0.004
56	*RAPGA*	20	0.008
57	*RAPGA*	50	0.001
58	*RAPGA*	50	0.002
59	*RAPGA*	50	0.004
60	*RAPGA*	50	0.008
61	*RAPGA*	100	0.001
62	*RAPGA*	100	0.002
63	*RAPGA*	100	0.004

Nastavak na sljedećoj stranici...

Tablica D.2 – Nastavak

Test	**Algoritam**	**Početna populacija**	**Vjerojatnost mutacije**
64	*RAPGA*	100	0.008
65	*RAPGA*	200	0.001
66	*RAPGA*	200	0.002
67	*RAPGA*	200	0.004
68	*RAPGA*	200	0.008

Algoritmi evolucijskog računanja primijenjeni na problem izrade školskog rasporeda sati

Sažetak

Izrada rasporeda za osnovne i srednje škole složen je kombinatorički problem. Danas postoje mnoga računalna rješenja za automatiziranu izradu rasporeda. Međutim, ona ne zadovoljavaju potrebe većine škola u Republici Hrvatskoj. Ovaj rad opisuje problem izrade rasporeda sati za škole. Opisuju se oblici i složenost problema, kao i postupci evolucijskog računarstva kojima se može pristupiti njegovu rješavanju. U sklopu rada razvijen je *Sustav za izradu rasporeda nastave* i provedeno je ispitivanje utjecaja parametara algoritama na njegov učinak. Konačni rezultati uspoređeni su sa stvarnim školskim rasporedom.

Ključne riječi: genetski algoritam, problem izrade rasporeda, školski raspored, evolucijski algoritmi, heuristike, lokalna pretraga, križanje, mutacija, GA, OS, RAPGA, evolucijska strategija

Evolutionary computation algorithms applied to the school scheduling problem

Abstract

School scheduling problem is a highly complex combinatorial problem. Today there are many computer solutions for the automated scheduling. However, those solutions do not meet the needs of most school in the Republic of Croatia. This paper describes the problems of the school timetabling. It describes the types and complexity of the problem and evolutionary computation methods with which it can be solved. A software system for generating school timetables is developed. Experiments are performed to study the influence of parameters of the algorithm's performance. Final results were compared with actual school schedule.

Keywords: genetic algorithm, scheduling problem, school timetabling, evolutionary algorithms, heuristics, local search, crossover, mutation, GA, OS, RAPGA, evolutionary strategy