SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 466

Heurističke metode lokalne pretrage primijenjene na problem izrade rasporeda sati za škole

Alan Tus

Zagreb, rujan 2012.

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

Hvala mojim roditeljima na neizmjerno puno strpljenja, pažnje i truda za vrijeme mojeg školovanja, a posebno pred njegov kraj.

Hvala asistentu dr.sc. Marku Čupiću za sve što me naučio tokom studija, a posebno hvala što nije izgubio nadu u mene i ovaj rad.

Hvala Miletu na povremenim savjetima i podršci.

Hvala Kristini što je znala reći prave riječi i natjerati me da nastavim.

SADRŽAJ

1. Uvod **1**

2. Problem izrade rasporeda **3**

2.1. Uvod u terminologiju 3

2.2. Osnovni problem 5

2.3. Čvrsta i meka ograničenja 7

2.4. Dodatna ograničenja 7

2.4.1. Neprekidan raspored 7

2.4.2. Višesatna predavanja 8

2.4.3. Podjele razreda i vezana predavanja 8

2.4.4. Raspoređivanje u prostorije 9

2.4.5. Ostala ograničenja 9

3. Heuristički i metaheuristički algoritmi **10**

3.1. Optimizacija kolonijom mrava 10

3.2. Osnovni algoritam optimizacije kolonijom mrava 12

3.2.1. Konstrukcijski algoritam 13

3.2.2. Feromonski model 15

3.3. Mravlji sustav 20

3.4. Elitistički mravlji sustav 20

3.5. Poredani mravlji sustav 21

3.6. $\mathcal{MAX}-\mathcal{MIN}$ mravlji sustav 21

3.7. Sustav kolonije mrava 23

3.7.1. Pseudoslučajni proporcionalni odabir 23

3.7.2. Globalno osvježavanje feromona 23

3.7.3. Lokalno osvježavanje feromona 24

4. Algoritmi lokalne pretrage **25**
4.1. Pretraga promjenjivih susjedstva 25
4.1.1. Spust po promjenjivim susjedstvima 27
4.1.2. Osnovna pretraga promjenjivih susjedstva 27
4.1.3. Reducirana pretraga promjenjivih susjedstva 28
4.1.4. Sveobuhvatna pretraga promjenjivih susjedstva 29
4.1.5. Dekompozicijska pretraga promjenjivih susjedstva 29
4.1.6. Iskrivljena pretraga promjenjivih susjedstva 31

5. Implementacija **33**
5.1. Implementacija rješenja 33
5.1.1. Osnovni dijelovi *Sustava* 33
5.1.2. Višedretvenost 38
5.2. Vrednovanje rješenja 38
5.3. Algoritam optimizacije mravljom kolonijom 40
5.3.1. Konstrukcijski algoritam 40
5.3.2. Izračun dostupnosti 43
5.3.3. Izgradnja mravljeg rješenja 44
5.4. Algoritmi lokalne pretrage 44
5.4.1. Jednostavna optimizacija 44
5.4.2. Algoritmi pretrage promjenjivih susjedstva 44

6. Rezultati i rasprava **46**
6.1. Testni parametri 46
6.2. Način testiranja 51
6.3. Rezultati 51

7. Zaključak **54**

Literatura **56**

A. Izvorni kodovi **58**

B. Primjer ulazne i izlazne datoteke **65**

C. Rezultati testiranja **70**

1. Uvod

Rasporede koristimo svakodnevno za organizaciju i planiranje. Pojavljuju se u obliku školskih rasporeda, rasporeda kazališnih i kino predstava, voznih redova i slično. Imaju dvije osnovne svrhe, da pojednostavne svakodnevni život i definiraju tko, što i kada radi te da optimiziraju iskorištavanje resursa. Izrada tih rasporeda se danas gotovo svuda automatizira, ali u školama na području Republike Hrvatske se raspored nastave danas još uvijek većinom izrađuje ručno.

Izrada rasporeda je uglavnom ručna zato što čovjek još uvijek može napraviti bolji raspored od kompjutera. Sva dostupna programska rješenja za izradu rasporeda ne mogu zadovoljiti zahtjeve koji su specifični za škole u Republici Hrvatskoj. Detaljan pregled dostupnih programskih rješenja i usporedba mogućnosti sa zahtjevima Hrvatskih škola je dana u [14].

Izrada rasporeda sati u osnovnim i srednjim školama je prepuštena satničarima. To su najčešće profesori matematike i informatike koji posjeduju neka znanje koja su im od pomoći pri izradi rasporeda. Satničari se pri izradi rasporeda služe raznim pomagalima, ali ponajprije iskustvom. Najčešće je slučaj da se izradom rasporeda bave već neko dulje vrijeme pa znaju koja predavanja je najteže rasporediti, koji su najčešći problemi i njihovo najbolje rješenje. Također, poznaju svoje kolege, znaju njihove zahtjeve i želje, upoznati su sa zakonskim ograničenjima i načinom rada koji treba poštivati te na taj način mogu izraditi bolji raspored.

Prije početka planiranja i implementacije novog sustava za izradu rasporeda bilo je potrebno upoznati se s gore spomenutim zahtjevima koji se javljaju pri izradi rasporeda nastave. Obavljen je niz razgovora sa satničarima u osnovnim i srednjim školama u Zagrebu:

- OŠ Voltino,
- OŠ Sveta Nedelja,
- IV. gimnazija,
- V. gimnazija i

– Prva ekonomska škola.

Analiza zahtjeva s kojima smo se susreli je napravljena u [14], a analiza problema izrade rasporeda nastave za škole u RH je napravljena u [11]. Dio tih analiza će biti predstavljen i u ovom radu zbog lakšeg razumijevanja. Iako se sve škole u kojima su obavljeni razgovori sa satničarima nalaze na području Zagreba, definicija problema i zahtjeva se može poopćiti na područje cijele Republike Hrvatske.

Ovaj rad dio je većeg projekta. Cilj je stvoriti sustav za izradu rasporeda nastave koji će se moći koristiti na području Republike Hrvatske. Prve inačica jezgre sustava i izrađena je u sklopu [12]. Ovim radom je nadograđena jezgra sustava te implementirana i opisana nova implementacija izrade rasporeda nastave, spojem algoritma optimizacije kolonijom mrava i lokalne pretrage. U sklopu [1] napravljeno je grafičko sučelje koje omogućava zadavanje ulaznih parametara.

U idućem poglavlju, poglavlje 2, će biti predstavljen problem koji se pokušava riješiti u ovom radu. Poslije, u poglavljima 3 i 4 su objašnjeni algoritmi koji se mogu koristiti za rješavanje opisanog problema. U poglavlju 5 je objašnjena cjelokupna implementacija čiji su parametri i rezultati detaljnije razrađeni u poglavlju 6. Na kraju, u poglavlju 7, se nalazi zaključak.

U daljnjem tekstu rada, riječ *Sustav* se odnosi na sustav za izradu rasporeda implementiran u sklopu ovog rada.

2. Problem izrade rasporeda

Ovo poglavlje daje uvod u terminologiju i formalnu definiciju problema izrade rasporeda nastave koji se javlja u ovom radu. Opisani su i zahtjevi i ograničenja koji se javljaju pri izradi rasporeda.

2.1. Uvod u terminologiju

Ovo je kratki uvod koji služi kako bi pojasnili značenje nekih termina koji će se koristiti u daljnjem tekstu.

Termini Termin je vremenski period od 45 minuta[1] u kojem se može održavati nastava. Ukupan broj termina jednak je:

$$brojTjedana * brojDanaUTjednu * brojTerminaUDanu \tag{2.1}$$

Primjerice za dvotjedni raspored, s pet dana u tjednu i 13 predavanja dnevno ima ukupno 130 termina. U svakom terminu može biti više predavanja uzevši u obzir da su svi potrebni resursi dostupni.

Blok Blok predstavlja jedan ili više uzastopnih termina u kojem se održava jedno predavanje.

Tjedni nastave Raspored može biti jednotjedni ili višetjedni. Škole često imaju dvotjedni raspored gdje se u drugom tjednu nastave zrcali raspored iz prvog tjedna uz manje izmjene. Pri izradi rasporeda će se uz predavanja spominjati tjedan u kojem se održava nastava, ali može biti i potpuno različit. Ukoliko je nastava dvotjedna (može biti i jednotjedna) razredi mogu različite tjedne imati predavanja u različitim turnusima.

[1] Ako se radi o skraćenim satima može biti i 35 minuta.

Turnusi Turnusi su zapravo smjene u kojima se održava nastava. U praksi se najčešće radi u jednoj ili dvije smjene (jutarnja i popodnevna), a u ekstremnim slučajevima i u tri smjene. Ponekad se javlja pojam *međuturnusa* koji predstavlja jedan ili više termina na spoju dva turnusa u kojima je dozvoljeno preklapanje nastave različitih turnusa. Turnus je definiran nizom uzastopnih termina koji mu pripadaju. Jedan termin može pripadati samo jednom turnusu ili se nalaziti u međuturnusu. Pripadnost razreda pojedinom turnusu se izražava u dostupnostima tako da je dostupnost u terminima izvan pripadnog turnusa jednaka 0.

Dostupnosti Dostupnosti služe kako bi definirali raspoloživost resursa u pojedinom terminu. Za sve resurse je moguće definirati dostupnost od 0 do 1 gdje 0 znači da resurs nije na raspolaganju u tom terminu, a sve vrijednosti veće od 0 znače da je resurs raspoloživ. Što je vrijednost veća to je taj termin poželjniji. Dostupnosti se definiraju po danima u tjednu i terminima. Primjerice na ovaj način nastavnici mogu definirati termine kada im je poželjnije održavati nastavu. Pri izradi rasporeda dostupnosti predstavljaju heurističku informaciju i jedno od čvrstih ograničenja.

Predmeti Predmeti predstavljaju predmete u školi poput matematike, biologije i kemije. Kod izrade rasporeda je ovaj podatak važan kako bi se izbjeglo više predavanja istog predmeta u jednom danu.

Razredi Razredi predstavljaju skupine učenika koje zajedno slušaju nastavu. Za svaki razred se definiraju dostupnosti. Osim "običnih" razreda pri izradi rasporeda se susrećemo i s razrednim particijama. Primjerice postoji podjela za tjelesnu i zdravstvenu kulturu (TZK) na muški i ženski dio razreda ili podjela na dio razreda koji sluša etiku i dio koji sluša vjeronauk. Cilj je spojiti po dva muška i dva ženska dijela razreda da zajedno imaju nastavu TZK kako bi se što bolje iskoristila dvorana. Za razredne particije nije moguće definirati dostupnosti. Smatra se da ako je cijeli razred dostupan u nekom terminu da su onda i sve njegove particije dostupne.

Nastavnici Nastavnici su svi ljudi koji održavaju nastavu. Za nastavnike se definiraju dostupnosti.

Prostorije Prostorije su sve učionice, kabineti, dvorane i ostali prostori koji služe za održavanje nastave. Pri zadavanju parametara za izradu rasporeda nije uvijek potrebno definirati prostorije ukoliko se raspoređivanje predavanja u prostorije ne želi prepustiti algoritmu. Za prostorije se definiraju dostupnosti.

Predavanja Predavanje je definirano s:

- jednim ili više nastavnika,
- trajanjem bloka (jedan ili više uzastopnih termina),
- oznakom tjedna u kojem se održava nastava,
- oznakom koja definira radi li se o predavanju koje ne spada u redovni raspored[2],
- dostupnostima,
- predmetom[3],
- jednim ili više razreda ili razrednih particija[3],
- jednom ili više prostorija[3] te
- brojem potrebnih prostorija[3].

Valja primijetiti kako predmeti koji imaju više predavanja tjedno po razredu moraju imati definirano po jedno *predavanje* za svako od tih predavanja.

Veze između predavanja Veze između predavanja su detaljnije definirane u poglavlju 2.4.3.

2.2. Osnovni problem

Raspored je zapravo alokacija resursa u vremenu (i prostoru). Za ovaj pojednostavljeni problem naši resursi će biti razredi, nastavnici i predavanja. Koristiti ćemo sljedeće oznake:

- $1, \ldots, c$ predstavljaju c razreda (engl. *class*),
- $1, \ldots, t$ predstavljaju t nastavnika (engl. *teacher*),
- $1, \ldots, s$ predstavljaju s termina (engl. *slot*) te
- $1, \ldots, l$ predstavljaju l predavanja (engl. *lecture*).

Definirajmo i tablicu P_{c*t} gdje vrijednost p_{ij} predstavlja broj predavanja koje nastavnik t_j održava razredu c_i unutar dostupnog broja termina s. Uzevši u obzir definiciju predavanja možemo reći da je konačni raspored zapravo niz predavanja l kojima su

[2]Ovdje se radi o predmetima poput izbornih predmeta i drugih vannastavnih aktivnosti te se ti predmeti smiju održavati izvan redovne nastave. Poželjno je da budu barem neposredno prije ili poslije redovne nastave.

[3]Nije obavezno.

pridruženi termini s. Potrebno je rasporediti predavanja u termine na način da nastavnici i razredi ne budu raspoređeni u više od jednog predavanja u istom terminu. [3] taj problem definira kao:

$$\text{t.d. } \forall(i,j) \sum_{k=1}^{t} x_{ijk} = p_{ij}, \tag{2.2}$$

$$\forall(i,k) \sum_{i=1}^{c} x_{ijk} \leq 1, \tag{2.3}$$

$$\forall(j,k) \sum_{j=1}^{t} x_{ijk} \leq 1 \text{ i} \tag{2.4}$$

$$\forall(i,j,k) x_{ijk} = 0 \text{ ili } 1, \tag{2.5}$$

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

Izraz (2.2) ograničava da svaki nastavnik održi točno onoliko predavanja koliko je predviđeno, r_{ij}. Druga dva izraza ograničavaju da svaki razred (2.3) i nastavnik (2.4) mogu u svakom trenutku biti na najviše jednom predavanju.

Za gore definirani problem je dokazano [7] kako uvijek postoji rješenje. Jedini uvjet jest da niti jedan razred i niti jedan nastavnik nemaju zadano više od s predavanja na koja ih je potrebno rasporediti. Mora vrijediti:

$$\forall(i) \sum_{j=1}^{t} p_{ij} \leq s \text{ i} \tag{2.6}$$

$$\forall(j) \sum_{i=1}^{c} p_{ij} \leq s. \tag{2.7}$$

Ako znamo da je s ukupan broj raspoloživih termina i da moramo poštivati ograničenja (2.3) i (2.4), onda je lako zaključiti da će ukupan zbroj svih predavanja nekog nastavnika (2.6) biti manji ili jednak s. Isto to vrijedi i za razrede (2.7), zbroj svih predavanja koja taj razred ima će biti manji ili jednak ukupnom broju raspoloživih termina. Dakle, u najgorem slučaju će biti zauzeti u svakom terminu.

Ovo je pojednostavljena definicija problema izrade rasporeda kojim se bavi ovaj rad jer ne uzima u obzir dostupnosti resursa, dodatne zahtjeve i ostala čvrsta i meka ograničenja koja su zadana ovim radom. Rješenje ovog problema je zadovoljeno ukoliko su sva predavanja raspoređena u ponuđene termine zadovoljavajući gornja ograničenja.

U školama se za prikaz cijelog rasporeda nastave najčešće koristi matrica R_{t*s}, gdje vrijednost r_{ij} definira razred kojem nastavnik t_j drži u terminu t_i. Ova tablica se može naći izvješena u svim zbornicama škola koje smo posjetili.

2.3. Čvrsta i meka ograničenja

Ograničenja koja se zadaju algoritmu se dijele na čvrsta i meka. *Čvrsta ograničenja* (engl. *hard constraints*) su ona ograničenja koja imaju prioritet i ne smiju se prekršiti. Konačni rezultat je valjan ako i samo ako zadovoljava ova ograničenja. *Meka ograničenja* (engl. *soft constraints*), se mogu gledati kao preporuke ili želje koje raspored čine boljim. Konačni raspored smije kršiti neka ili čak sva meka ograničenja, ali ako raspored poštuje više tih ograničenja imati će veću vrijednost evaluacijske funkcije

Čvrsta ograničenja variraju od škole do škole i zato je prepušteno korisniku da kroz parametre stupnjevanjem dostupnosti ili eksplicitnim zahtjevom zada zahtjeve koji će predstavljati *čvrsta* ograničenja. U *čvrsta* ograničenja najčešće ubrajamo sljedeće zahtjeve:

- ograničenja zadana dostupnostima razreda, profesora i prostorija,
- najveći i najmanji broj sati nastave dnevno i tjedno za profesore i razrede,
- broj dozvoljenih rupa dnevno i tjedno u rasporedu profesora,
- zabrana prelijevanja bloka iz dana u dan ili iz turnusa u turnus te
- smještanje predmeta u zadani tjedan[4].

Meka ograničenja čine svi ostali zahtjevi koji su spomenuti u sljedećem odjeljku (2.4).

2.4. Dodatna ograničenja

Ovo poglavlje spominje samo najvažnija ograničenja važna za razumijevanje rada *Sustava*. Detaljan opis svih ograničenja je dostupan u [14, 12].

2.4.1. Neprekidan raspored

Neprekidan raspored za razrede znači da pojedini razred za vrijeme nastave nema niti jednu pauzu između predavanja unutar jednog dana. To je vrlo važno ograničenje, pogotovo za osnovne škole, jer se onda potrebno pobrinuti za učenike za vrijeme pauze. Ukoliko su popunjeni svi sati predviđeni za nastavu u pojedinom danu, neće nastati rupe u rasporedu, ali ako postoje termini kada razred nema predavanja, ti termini moraju obavezno biti prije ili poslije nastave.

[4] Ovo vrijedi za višetjedne rasporede.

Neprekidan raspored za nastavnike nije obavezan, ali je poželjno da i njihov raspored bude gotovo neprekidan. Dozvoljene su pauze od jednog sata dnevno ili dva do tri sata tjedno kako bi raspored bio prihvatljiv. Pri zadavanju parametara može se zadati, traži li se za pojedini razred ili nastavnika izrada neprekidnog rasporeda.

2.4.2. Višesatna predavanja

Zbog dinamike nastave ponekad je potrebno održavati nastavu nekog predmeta u nizu od dva ili više sati za redom. Ovakva predavanja se nazivaju *blok* predavanja. U srednjim školama primjer ovakve nastave su hrvatski jezik (pisanje školske zadaće), kemija (izvođenje pokusa), matematika (veće cjeline gradiva) i drugi. Tada je poželjno imati višesatna predavanja, ali, osim kada je to izričito zadano, to nije poželjno ponašanje.

Ako se radi, na primjer, o predmetu s dva sata predavanja tjedno, tada u nekim slučajevima nije poželjno da ti sati budu u nizu, ili čak isti dan jer se želi opterećenje na učenike rasporediti kroz tjedan. Također, slučaj u kojem se višesatna predavanja spajaju s drugim predavanjima istog predmeta nije poželjan. Pri izradi rasporeda se pazi da se takvi rasporedi izbjegavaju i moguće je zadati veličinu najvećeg dopuštenog bloka i zatražiti izbjegavanje većih bokova nastave.

2.4.3. Podjele razreda i vezana predavanja

U nekim srednjim školama se nude različiti nastavni programi, različiti stupnjevi pojedinih predmeta (jezici) i slično. Zato među pojedinim predavanjima mogu postojati različite veze i pravila kada se neko predavanje može održati u odnosu na ono drugo. Najčešće, škole uspijevaju sastaviti razrede na način da cijeli razredi slušaju jedan nastavni program, ali kada to nije moguće javljaju se podgrupe unutar razreda. Primjerice, jedan dio razreda će imati etiku, a drugi dio vjeronauk ili će jedan dio razreda imati početni engleski jezik, a drugi dio napredni engleski jezik. Ukoliko postoji više razreda s istim razrednim podgrupama pri izradi rasporeda se pokušava objediniti podgrupe koje imaju isto predavanje kako bi ponovno činile novi cijeli razred i time što bolje iskoristiti resurse[5]. U parametrima se mogu definirati podgrupe za neki razred i kasnije ih dodijeliti nekom predavanju. Za takva predavanje se mogu definirati pravila.

Za bilo koja dva predavanja može se definirati međusobna veza tako da se spoje s nekom kombinacijom uvjeta iz sljedeće dvije grupe:

[5]Neke od prednosti su: manje sati za nastavnika, kraći raspored za učenike i više slobodnih prostorija.

– *mora biti* ili *ne smije biti* te

– *isti termin* ili *isti dan* ili *isti tjedan* ili *slijedno (uređeno)* ili *slijedno (nije uređeno)*[6].

Spomenimo nekoliko primjera:

– mora biti isti dan,

– ne smije biti isti tjedan,

– mora biti slijedno (uređeno) te

– mora biti isti termin.

2.4.4. Raspoređivanje u prostorije

U školama koje imaju manjak prostora, raspoređivanje predavanja u prostorije je dodatan izazov pri izradi rasporeda nastave. Kada smo definirali osnovni problem, postavili smo pretpostavku da svaki razred ima svoju prostoriju koja mu je uvijek na raspolaganju. Međutim, neka predavanja zahtijevaju posebnu prostoriju koja je prikladno opremljena[7] i može ju koristiti samo jedan razred odjednom. Zato se pri zadavanju zahtjeva za izradu rasporeda može uključiti opcija raspoređivanja predavanja u prostorije. Tada je potrebno zadati popis raspoloživih prostorija, željene prostorije za pojedina predavanja i (ukoliko postoje) zadati matičnu prostoriju za svaki razred.

2.4.5. Ostala ograničenja

Od ostalih ograničenja možemo spomenuti da je moguće zadati interval unutar kojeg se smije nalaziti određeni broj radnih sati[8] u danu za razrede i nastavnike, zatražiti pauze za profesore u određeno vrijeme te zatražiti da profesor predaje samo jednoj generaciji u jednom danu.

[6]Ovdje se pod slijedno misli da dva predavanja (ne)moraju slijediti jedno iza drugoga.

[7]Na primjer, za nastavu tjelesne i zdravstvene kulture je potrebna dvorana, a za informatiku učionica opremljena računalima.

[8]Broj radnih sati u danu je broj sati provedenih na nastavi, ne računajući pauze.

3. Heuristički i metaheuristički algoritmi

Ovo poglavlje opisuje odabrane algoritme koji su primjenjivi na izradu rasporeda. Samo neki od njih će biti implementirani, a o tome će biti više u poglavlju 5.

Heuristički algoritmi su algoritmi koji ne daju najbolje rješenje, ali u najboljem slučaju daju dobro rješenje u razumnom vremenskom periodu.[13] Služe za pronalaženje početnog ili okvirnog rješenja kad klasične metode zakažu ili su prespore. Zbog brzine se gubi na dobroti, potpunosti, preciznosti i/ili točnosti rješenja. Možemo ih podijeliti na dvije vrste:

Konstrukcijski algoritmi Kao što im ime govori, ovi algoritmi konstruiraju rješenje. Najčešće su brzi i jednostavni, ali usko povezani uz problem koji rješavaju.

Algoritmi lokalne pretrage Ovi algoritmi se najčešće javljaju uz neki konstrukcijski algoritam jer pokušavaju iterativnim postupkom poboljšati dobiveno početno rješenje. Za razliku od konstrukcijskih algoritama, ovi algoritmi su sporiji, ali daju bolja rješenja i u svom radu koriste vrlo općenita načela koja su neovisna o problemu koji rješavaju.

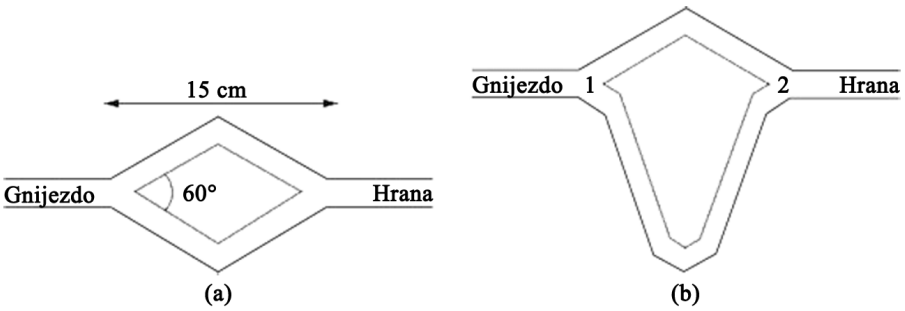
Metaheuristika je računarska metoda koja iterativnim postupkom pokušava poboljšati postojeće rješenje s obzirom na neku evaluacijsku funkciju. Metaheuristike imaju vrlo malo ili nijednu pretpostavku o problemu čije rješenje optimiziraju te mogu pretražiti vrlo velik prostor rješenja, ali zato ne garantiraju da će u konačnici pronaći optimalno rješenje. Mnogi metaheuristički algoritmi u svom radu koriste neki oblik stohastičke optimizacije.

3.1. Optimizacija kolonijom mrava

Optimizacija kolonijom mrava je prirodom inspiriran algoritam koji oponaša svojstva kolektivne inteligencije. Jedinke lokalno stvaraju promjene u svojoj okolini i na taj

način uzrokuju pojavu globalnih svojstava. Na taj način moguće je rješavati kompleksne raspodijeljene probleme pomoću jednostavnih lokalnih interakcije bez potrebe za centraliziranim globalnim sustavom. Osnovna ideja samoorganizirajućih načela koja dozvoljavaju visokokoordinirano ponašanje pravih mrava može poslužiti kao izvor inspiracije za izradu novih algoritama čiji dijelovi surađuju kako bi riješili problem.

Mravi imaju vrlo slabo razvijen vid ili su potpuno slijepi tako da se njihova komunikacija svodi na korištenje kemikalija koje nazivamo *feromonima*. Tako će primjerice, mravi putem od gnijezda do hrane za sobom ostavljati trag feromona. Na svim putovima gdje je barem jedan mrav prošao postojati će feromonski trag od gnijezda do hrane i s vremenom će se istaknuti najjači trag koji će svi mravi slijediti. Pokus koji najbolje predstavlja proučavano ponašanje mrava su opisali [4] pomoću pokusa s dvostrukim mostom (slika 3.1). Most je povezivao gnijezdo Argentinskih mrava (*I. humilis*) i izvor hrane te su u različitim pokusima mijenjali omjer $r = \frac{l_l}{l_s}$ između duljina dvaju krakova mosta gdje je l_l dulji krak, a l_s kraći krak.

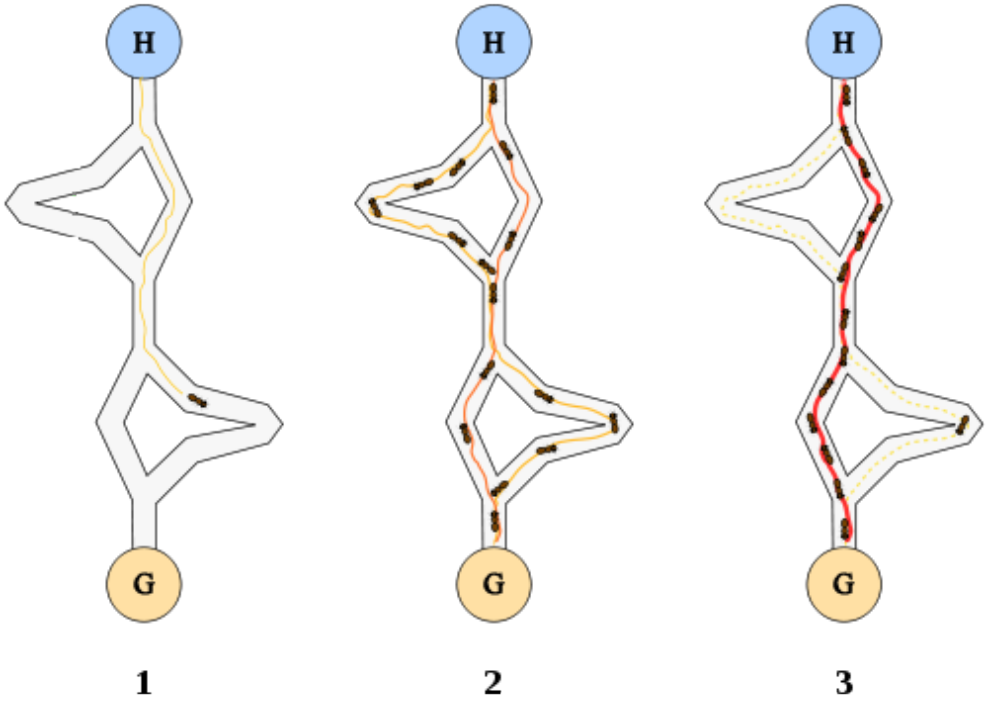


Slika 3.1: Dvostruki most

U prvom eksperimentu je most podijeljen na dva jednaka kraka ($r = 1$). Mravi su pušteni da se slobodno kreću od gnijezda do hrane. Pratili su broj mrava koji je odabrao jedan ili drugi krak mosta. Rezultat je bio taj da su u početku mravi birali nasumičan krak, ali s vremenom su počeli kretati isključivo jednim od krakova. Rezultat možemo protumačiti na sljedeći način. Na početku pokusa na nijednom od krakova nije bilo tragova feromona, tako da nisu imali poželjniji put. S vremenom je veći broj mrava izabrao jedan od krakova i time ostavio veću količinu feromona na tom kraku. Veća količina feromona mravima znači da je to poželjniji put i uskoro će svi ići tim putem.

U drugom pokusu je omjer krakova bio $r = 2$, tako da je dulji krak bio dvostruko dulji od kraćeg. Ponovno, na početku pokusa nije bilo feromona na krakovima i mravi su nasumično birali put. Ubrzo su svi mravi počeli birati kraći krak (slika 3.2).

Izveden je još niz pokusa koji su pokazali kako isparavaju feromoni i utiču na



Slika 3.2: Tijek pokusa s dvostrukim mostom

kretanje mrava

3.2. Osnovni algoritam optimizacije kolonijom mrava

Prva inačica algoritma i gotovo sve njegove nadogradnje imaju iste korake pri radu. Postupak je opisan pseudokodom 3.1. Inicijalizacijom se stvaraju strukture podataka potrebne za rad. Unutar glavne petlje se obavlja konstrukcija mravljih rješenja, potom se nad svim ili samo najboljim mravljim rješenjem obavlja lokalna pretraga. Nakon lokalne pretrage se obavlja isparavanje i polaganje novih feromonskih tragova. Na kraju se obavljaju pozadinske akcije (engl. *daemon actions*) koje nisu obavezni dio algoritma, ali mogu poslužiti kao prostor za proširenje algoritma.

Gledajući rad algoritma, možemo ga podijeliti na osnovne dijelove:

- konstrukcijski algoritam,
- algoritam lokalne pretrage i
- feromonski model.

Konstrukcijski algoritam i osnovni feromonski model će biti objašnjeni u sljedećim poglavljima, a algoritmi lokalne pretrage u poglavlju 4. Pojedinačni algoritmi optimizacije kolonijom mrava će također biti objašnjeni u sljedećim poglavljima.

Algoritam 3.1 Algoritam optimizacije kolonijom mrava

inicijaliziraj
dok nije zadovoljen završni kriterij **radi**
    konstruiraj mravlja rješenja
    obavi lokalnu pretragu
    osvježi feromone
    pozadinske akcije
kraj dok

3.2.1. Konstrukcijski algoritam

Osnovni konstrukcijski algoritam je heuristički konstrukcijski algoritam koji postepeno pokušava rasporediti sva predavanja u dostupne termine. Pri radu se služi samo informacijama o dostupnosti resursa, ali ih neće uvijek poštivati. Također, ne obazire se na dodatna ograničenja i ne pazi na preljev bloka predavanja u drugi turnus ili dan.

Konačni raspored koji će konstruirati ovaj algoritam ne zadovoljava rješenje osnovnog problema jer krši stroga ograničenja, ali će konstruirani raspored biti "izvediv u vremenu i prostoru".

Pri odabiru redoslijeda smještanja predavanja u raspored se koristi kolektivna inteligencija mrava koja bilježi koliko puta pojedino predavanje nije bilo moguće smjestiti u raspored koji se gradio. Svaki neuspješan pokušaj povećava brojač uz to predavanje i povećava njegovu složenost. Informacija o složenosti predavanja se koristi pri pseudo slučajnom proporcionalnom odabiru tako da će složenija predavanja biti razmještena u raspored među prvima.

Uvodi se *pravilo slučajnog proporcionalnog odabira*, a time i novu heurističku informaciju, feromonske tragove koje polažu mravi.

Slučajno proporcionalni odabir

Pravilo slučajno proporcionalnog odabira (engl. *random proportional rule*) služi mravu k da odabere termin s_j za predavanje l_i. Pravilo je dano sljedećom jednadžbom:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{ako } j \in N_i^k \tag{3.1}$$

gdje τ_{ij} predstavlja vrijednosti feromona za predavanje l_i u terminu s_j, a η_{ij} predstavlja neku heurističku informaciju. [5]

U ovom slučaju radi se o vrijednosti koju vraća izračun dostupnosti[1]. Parametri α

[1] Opisan u poglavlju 5.3.2.

i β imaju sljedeću ulogu. Ako je $\alpha = 0$ onda veću vjerojatnost imaju oni termini koji imaju bolju dostupnost resursa. Ako je $\beta = 0$ onda samo feromonski tragovi utiču na odabir termina. Umnožak u brojniku daje objedinjenu heurističku informaciju s naglaskom na dostupnosti ili feromonske tragove. Ovaj omjer predstavlja vjerojatnost da termin s_j bude odabran u susjedstvu mrava k tako da vrijednost brojnika predstavlja sumu heurističkih vrijednosti cijelog susjedstva. Susjedstvo mrava čine svi termini koji još nisu popunjeni.

Rad konstrukcijskog algoritma

Nakon inicijalizacije potrebnih podatkovnih struktura ulazi se u glavnu programsku petlju koja obavlja funkciju raspoređivanja svih predavanja u vlastite termine.

Algoritam 3.2 Mravlji konstrukcijski algoritam

```
inicijaliziraj
dok rasporedenaPredavanja < ukupnoPredavanja radi
    odabranoPredavanje ← slucajno proporcinalni odabir predavanja
    odabraniTermin ← slucajno proporcinalni odabir termina
    noviRaspored ← rasporedi(odabraniTermin, odabranoPredavanje)
    ako odabranoPredavanje nije rasporedeno u odabraniTermin onda
        zablijezi promasaj uz odabranoPredavanje
        ponovno pokreni algoritam
    inače
        oznaci odabranoPredavanje kao rasporedeno
    kraj ako
kraj dok
```

Predavanje koje će sljedeće biti raspoređeno se bira pomoću *pravila slučajnog proporcionalnog odabira*. Razlika u odnosu na jednadžbu (3.1) jest da se ne koriste vrijednosti feromona, a heurističku informaciju predstavlja podatak o "kompleksnosti" pojedinog predavanja. Kompleksnost predavanja je zapravo broj promašaja za pojedino predavanje, broj neuspjelih pokušaja raspoređivanja tog predavanja. Informacije o broju promašaja se centralno sinkroniziraju nakon svakog završetka rada konstrukcijskog algoritma tako da na početku rada algoritma se radi o nasumičnom odabiru predavanja, ali s vremenom se češće među prvima odabiru predavanja s više promašaja. Ovaj način rada je uveden jer je primijećeno kroz testiranja da ako se kompleksnija predavanja među prvima smjeste u raspored, da postoji veća vjerojatnost da će sva

predavanja biti raspoređena.

Termin za pojedino predavanje se bira pomoću *pravila slučajnog proporcionalnog odabira* kako je opisano u jednadžbi (3.1).

3.2.2. Feromonski model

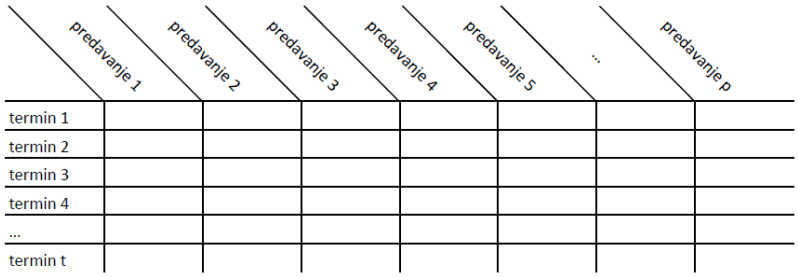
Kao što je već spomenuto, feromoni predstavljaju stvarne feromone koje za sobom ostavljaju mravi. Oni nam govore koliko je neki put do rješenja dobar i zajedno s heurističkom informacijom trebaju usmjeriti algoritam k boljem rješenju. Svi feromonski modeli imaju sljedeće stvari zajedničke. Imaju neku strukturu feromonskih tragova, metodu za postavljanje početnih vrijednosti, metodu za polaganje novih tragova i metodu za isparavanje tragova. Međusobno se razlikuju u implementaciji pojedinih metoda. Struktura se modelira obzirom na problem koji se rješava. U sljedećim poglavljima će biti opisani neki općeniti pojmovi i procesi vezani uz feromonske tragove. Specifične implementacije pojedinih metoda će biti obrađene unutar poglavlja o specifičnoj vrsti algoritma optimizacije kolonijom mrava u kojoj se koristi.

Struktura feromonskih tragova

Feromonski tragovi su predstavljeni dvodimenzionalnom matricom F decimalnih vrijednosti gdje redci predstavljaju s termina, a stupci predstavljaju l predavanja. Odlučio sam se za ovakav oblik jer se izradi rasporeda pristupa iz konteksta predavanja. Kao što smo već definirali, predavanje ima unaprijed zadane predmet, profesora(e) i razred(e), a promjenjivi su termin i prostorija u kojoj će se održavati. U zahtjevima su zadani željeni termini i prostorije, a na algoritmu je da odabere onaj spoj koji će se najbolje uklapati u cijeli raspored. Svako predavanje ne mora imati zadane prostorije[2] tako da se ta informacija ne zapisuje u feromonskim tragovima. Također, raspoređivanje predavanja po prostorijama ne predstavlja prioritet i uvijek se može, do neke mjere, optimizirati lokalnom pretragom.

Planira se provesti testiranja s višedimenzionalnom strukturom feromonskih tragova, ali vjerujemo da je ovo najbolji omjer između informacija i kvalitete rezultata. S više dimenzija bi se feromonski trag mogao previše raspršiti i time bi se izgubilo na kvaliteti rješenja i povećala složenost rada sa strukturom feromonskih tragova.

[2]Primjerice, informacije za roditelje nemaju dodijeljenu prostoriju.



Slika 3.3: Feromonski model

Početna vrijednost feromona

Početna vrijednost feromona se postavlja na unaprijed određenu vrijednost koja se najčešće izračunava pomoću neke funkcije koja ovisi o pronađenom rješenju. Početno rješenje ne mora biti idealno ili prihvatljivo, već je dovoljno da zadovoljava neke osnovne kriterije kako bi nam pružilo okvirne informacije o rezultatima koje možemo očekivati za ovaj problem. Za izradu početnog rješenja se koristi jednostavan heuristički algoritam koji je opisan u poglavlju 5.3.1.

Isparavanje feromonskih tragova

Postupak isparavanja feromonskih tragova također odgovara procesu koji se odvija u prirodi. Feromoni koje su mravi položili će s vremenom isparavati tako da dok drugi mrav dođe do nekog feromonskog traga on će imati manji intenzitet nego kada je položen. Ukoliko se radi o vrlo prometnoj ruti kojom ide mnogo mrava, onda će intenzitet biti jači jer se češće polažu novi feromoni i ostale rute će polako ispariti.

Ovaj postupak osigurava da feromoni nekog dobrog rješenja (ali ne najboljeg) nemaju prevelike vrijednosti kako ne bi došlo do stagnacije algoritma u lokalnom optimumu. Osnovna ideja jest da se svi feromonski tragovi isparavaju nakon proteka vremena ili u našem slučaju, nakon iteracije, koraka ili neke druge akcije. Točan postupak isparavanja biti će opisan u nastavku rada uz svaki algoritam.

Polaganje feromonskih tragova

Postoje različiti načini polaganja feromonskih tragova [6]. Razlikujemo tri vrste:

1. Algoritam temeljen na gustoći mrava *ant-density* polaže unaprijed zadanu vrijednost feromona nakon svakog koraka mrava.

2. Algoritam temeljen na brojnosti mrava *ant-quantity* polaže vrijednost feromona obrnuto proporcionalnu vrijednosti odabranog koraka[3] nakon svakog koraka mrava.

3. Algoritam temeljen na ciklusima mrava (engl. *ant-cycle*) polaže feromone nakon što je završila konstrukcija rješenja. Vrijednost položenih feromona je obrnuto proporcionalna kvaliteti rješenja (prijeđenog puta) ili neka unaprijed zadana funkcija najčešće ovisna o kvaliteti rješenja.

Testiranja su pokazala da treća metoda, algoritam temeljen na ciklusima mrava, daje najbolje rezultate i u ovom radu će se samo ona koristiti.

Algoritam temeljen na ciklusima mrava se može podijeliti na dva različita pristupa [6]. Recimo da mrav konstruira rješenje s_1 i to rješenje se pomoću lokalne pretrage unaprijedi i nastane rješenje s_2. Ukoliko obavljamo polaganje feromona koristeći rješenje s_2 (nakon lokalne pretrage), radi se o *Lamarckovom modelu*. Ukoliko se koristi rješenje s_1, radi se o *Darwinovom modelu*.

Testiranja su pokazala da *Darwinov model* nauči konstruirati dobra početna rješenja, ali *Lamarckov model* daje bolja rješenja u konačnici. Pošto želimo da se u rješenjima vidi efekt lokalne pretrage i želimo bolja rješenja u konačnici, koristi ćemo *Lamarckov model*.

Vizualizacija feromonskih tragova

Ukoliko je riječ o jednostavnom feromonskom modelu, kao u ovom radu, moguće ga je lako vizualizirati. Pošto je naš feromonski model zapravo tablica čiji su stupci predavanja a redovi termini. Vrijednosti tablice predstavljene su nijansama sive boje. Vrijednosti se prije pretvaranja u odgovarajuću nijansu skaliraju na interval $[0, 1]$ tako da se sve vrijednosti podijele s najvećom dostupnom vrijednosti:

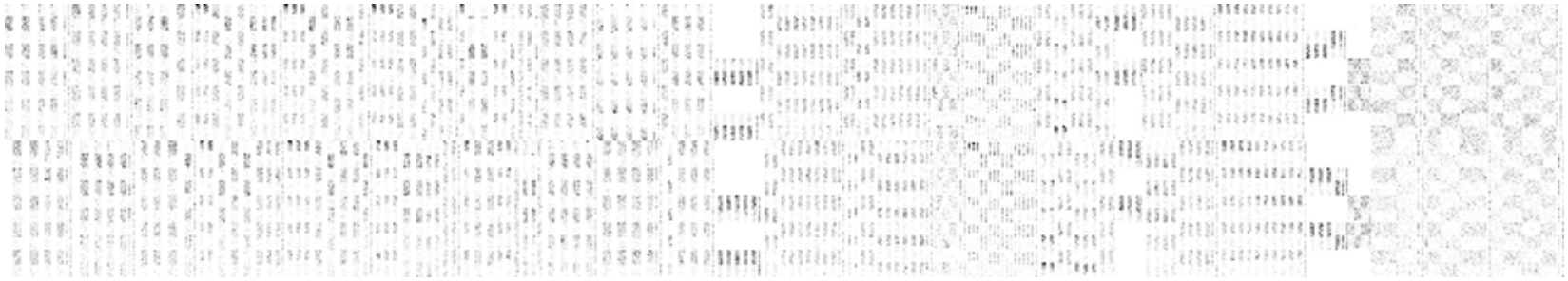
$$x \leftarrow \tau_{ij}/argmax(\tau_{ij}). \tag{3.2}$$

Što je veća vrijednost feromona to će biti tamnije polje koje mu odgovara. Rad je opisan algoritmom 3.3.

[3] Kod problema trgovačkog putnika bi se ovdje radilo o duljini puta.

Algoritam 3.3 Algoritam vizualizacije feromonskih tragova

$slika \leftarrow$ stvori novu sliku veličine $brojPredavanja * brojTermina$
za od 1 do $brojPredavanja$ **radi**
    za od 1 do $brojTermina$ **radi**
        $feromon \leftarrow$ dohvati vrijednost feromona na poziciji i, j
        $x \leftarrow$ normaliziraj vrijednost feromona
        oboji feromon na poziciji i, j za vrijednost x
    kraj za
kraj za



Slika 3.4: Vizualizacija feromonskih tragova

3.3. Mravlji sustav

Mravlji sustav (engl. *ant system*) je prva i najjednostavnija inačica algoritma optimizacije kolonijom mrava. Rad algoritma prati korake opisane u pseudokodu 3.1. Feromonski model odgovara prethodno opisanom modelu uz manje prilagodbe. Vrijednosti se inicijaliziraju prema sljedećoj jednadžbi:

$$\tau_{ij} = \tau_0 = \frac{n}{C^p} \tag{3.3}$$

gdje je n broj mrava, C^p vrijednost početnog rješenja[4], τ_{ij} vrijednost feromona na poziciji i, j, a τ_0 je početna vrijednost feromona.

Nakon konstrukcije mravljih rješenja, isparavaju se sve vrijednosti feromona na sljedeći način:

$$\tau_{ij} = (1 - \rho)\tau_{ij}, \forall i, j \in M \tag{3.4}$$

gdje je ρ neki parametar < 1 za koji isparavaju feromoni, a L označava feromonski model.

Nakon isparavanja se osvježavaju vrijednosti feromona za sva novonastala rješenja:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \forall i, j \in M, \tag{3.5}$$

$$\Delta\tau_{ij} = \begin{cases} 1/C^k & \text{ako } (i,j) \text{ pripada } T^k \\ 0 & \text{inače.} \end{cases} \tag{3.6}$$

pri čemu je T^k rješenje k-tog mrava, a C^k vrijednost njegovog rješenja. Ostale oznake su iste kao u prethodnim jednadžbama.

3.4. Elitistički mravlji sustav

Elitistički mravlji sustav (engl. *elitist ant system*) je nadogradnja *mravljeg sustava*. Jedina razlika je u načinu polaganja feromonskih tragova jer *elitistički mravlji sustav* daje prednost boljim rješenjima tako da se u svakoj iteraciji dodatno ojačaju feromoni koji pripadaju najboljem-do-sada rezultatu. Taj rezultat označavamo sa T^{bs} (engl. *best-so-far*) i mrav čije je to rješenje ne mora obavezno pripadati trenutnoj koloniji mrava.

Isparavanje feromona nije promijenjeno u odnosu na obični *mravlji sustav*, jednadžba (3.4).Pri polaganju feromona se obavlja postupak polaganja kao kod običnog

[4]Vrednovanje rješenja objašnjeno je u poglavlju 5.2.

mravljeg sustava, ali se na feromone koji pripadaju i najboljem-do-sada rješenju polaže dodatnu vrijednost feromona u iznosu od

$$\frac{e}{C^{bs}} \tag{3.7}$$

gdje je e parametar koji daje težinu, a C^{bs} je vrijednost najboljeg-do-sad rješenja. Funkcija koja polaže nove feromone izgleda ovako:

$$\tau_{ij} = (1-\rho)\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs}, \tag{3.8}$$

pri čemu vrijedi:

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & \text{ako } (i,j) \text{ pripada } T^{bs} \\ 0 & \text{inače.} \end{cases} \tag{3.9}$$

Elitistički mravlji sustav, uz odgovarajući parametar e, postiže puno bolje rezultate od običnog *mravljeg sustava*.

3.5. Poredani mravlji sustav

Poredani mravlji sustav (engl. *rank-based ant system*) [2] funkcionira kao i običan *mravlji sustav* samo što je vrijednost novih feromona koju polažu mravi proporcionalna rangu mrava unutar kolonije. Mravi se rangiraju prema kvaliteti njihova rješenja od najboljeg prema najlošijem.

U svakoj iteraciji samo $(w-1)$ najboljih mrava i najbolji-do-sada mrav polaže feromone. Ukoliko su neka dva mrava izjednačena, rangovi između ta dva mrava će biti slučajno raspoređeni. Najbolji mrav polaže najviše feromona Vrijednost feromona koji će mrav položiti jednaka je:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs} \tag{3.10}$$

pri čemu je $\Delta\tau_{ij}^{r} = 1/C^{r}$ i $\Delta\tau_{ij}^{bs} = 1/C^{bs}$. *Poredani mravlji sustav* postiže malo bolje rezultate od *elitističkog mravljeg sustava*.

3.6. $\mathcal{MAX}-\mathcal{MIN}$ mravlji sustav

$\mathcal{MAX}-\mathcal{MIN}$ mravlj sustav (engl. $\mathcal{MAX}-\mathcal{MIN}$ *ant system*) se temelji na *mravljem sustavu*, ali puno bolje iskorištava najbolja pronađena rješenja. U nastavku se nalaze četiri glavne razlike:

1. Samo najbolji mrav iz svih dosadašnjih kolonija ili samo najbolji mrav iz trenutne kolonije smije polagati feromone. Ovo može dovesti do stagnacije jer će svi mravi početi slijediti isti put zbog previsokih vrijednosti feromona.

2. Vrijednosti feromona su ograničene na interval $[\tau_{min}, \tau_{max}]$.

3. Feromoni se inicijaliziraju na maksimalnu vrijednost τ_{max} što uz sporije isparavanje feromona omogućava da se u početku istražuju manje vjerojatna rješenja.

4. Ukoliko algoritam počne stagnirati, tj. ne bude pronađeno bolje rješenje određen broj iteracija, vrijednosti feromona se ponovno inicijaliziraju na τ_{max}.

Isparavanje se vrši kao u *mravljem sustavu* (jednadžba (3.4)), a polaganje feromona vrši samo najbolji mrav. Najbolji mrav može biti ili najbolji-do-sada mrav ili najbolji-u-koloniji mrav. Odabir mrava ovisi o broju iteracija, a time i o veličini problema. Preporuka je za manje probleme koristiti najboljeg-u-koloniji mrava, a za veće probleme, zavisno koliko želimo da algoritam bude pohlepan, da se izmjenjuju najbolji-do-sada i najbolji-u-koloniji mrav.

Polaganje feromona se vrši prema sljedećoj formuli:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best} \tag{3.11}$$

pri čemu je $\Delta\tau_{ij}^{best} = 1/C^{best}$ vrijednost najboljeg-do-sada ili najbolji-u-koloniji mrava.

Nakon svakog polaganja feromona se vrši provjera granica vrijednosti feromona koje moraju biti u intervalu $[\tau_{min}, \tau_{max}]$. Time svaki termin ima vjerojatnost $0 \leq p_{min} \leq p_{max} \leq 1$ da bude odabran. Vrijednosti iz intervala se mogu izračunati pomoću ovih jednadžbi:

$$\tau_{max} \leftarrow \frac{1}{\rho C^{bs}}, \tag{3.12}$$

$$\tau_{min} \leftarrow \frac{\tau_{max}}{a} \tag{3.13}$$

gdje C^{bs} odgovara vrijednosti najboljeg-do-sad rješenja, a a je parametar. Pri inicijalizaciji će početno rješenje generirano pomoću jednostavnog konstrukcijskog algoritma predstavljati vrijednost C^{bs} rješenja. Gornja granica će se osvježavati pri svakom pronalasku novog najboljeg-do-sada rješenja.

Nakon što algoritam određeni broj iteracija nije pronašao bolje rješenja, vrijednosti feromona se ponovno inicijaliziraju na gornju vrijednost. Inicijalizacija na gornju vrijednost omogućava polagano povećavanje razlika između vrijednosti feromona, a time i veću mogućnost za istraživanje do sada neotkrivenih rješenja.

3.7. Sustav kolonije mrava

Sustav kolonije mrava (engl. *ant colony system*) je građen nad *mravljim sustavom*, ali ne uvođenjem manjih izmjena, već uvođenjem novih ideja. Novosti možemo svesti na tri stvari:

1. Bolje iskorištava iskustvo koje su stekli mravi novim načinom odabira termina za predavanje.
2. Isparavanje i polaganje novih feromona se obavlja samo na feromonima koji pripadaju najboljem-do-sada mravu.
3. Svaki put kada mrav odabere neki termin za neko predavanje, odgovarajući feromon se malo ispari kako bi se povećala vjerojatnost za druge termine

U nastavku su promjene detaljno opisane.

Sustav kolonije mrava je povezan i s $\mathcal{MAX}-\mathcal{MIN}$*mravljim sustavom*. $\mathcal{MAX}-\mathcal{MIN}$*mravlji sustav* ima eksplicitno zadane granice τ_{min} i τ_{max} dok *sustav kolonije mrava* ima implicitno zadane granice.

3.7.1. Pseudoslučajni proporcionalni odabir

Pravilo *slučajnog proporcionalnog odabira* se nadograđuje i uvodi se pravilo *psedoslučajnog proporcionalnog odabira* (engl. *pseudorandom proportional rule*) koje je definirano sljedećom jednadžbom:

$$j = \begin{cases} \max_{l \in N_i^k}\{\tau_{il}[\eta_{il}]^\beta\}, & \text{ako } q \leq q_0; \\ J & \text{inače} \end{cases} \tag{3.14}$$

gdje je q uniformno razdijeljena varijabla iz intervala $[0, 1]$, a q_0 je unaprijed zadani parametar $(0 \leq q_0 \leq 1)$. J predstavlja slučajno odabrani termin pomoću jednadžbe (3.1)($\alpha = 1$). Ovo pravilo zapravo s vjerojatnošću q_0 odabire najbolji mogući termin ili se ponaša po pravilu slučajnog proporcionalnog odabira. Što je manji parametar q_0 veća je vjerojatnost da će algoritam istraživati nova rješenja nezavisno od heurističkih informacija.

3.7.2. Globalno osvježavanje feromona

Osvježavanje feromona se sastoji od isparavanja feromona i polaganja novih feromona, ali pošto se u *sustavu kolonije mrava* samo feromoni najboljeg-do-sada mrava osvje-

žavaju, funkciju možemo objediniti u:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs}, \forall(i,j) \in T^{bs}. \tag{3.15}$$

U prethodnoj jednadžbi parametar ρ predstavlja isparavanje fermona, a $\Delta\tau_{ij}^{bs} = 1/C^{bs}$. Dodatna razklika u odnosu na *mravlji sustav* jest u umanjivanju novih feromonskih tragova za ρ. Time je nova vrijednost feromona bliža postojećoj vrijednosti feromona i τ_{min} nikada neće biti manji od τ_0.

3.7.3. Lokalno osvježavanje feromona

Mravi nakon svakog koraka osvježavaju vrijednost feromona koji odgovara odabranom koraku za:

$$\tau_{ij} \leftarrow (1-\xi)\tau_{ij} + \xi\tau_0 \in T^{bs}. \tag{3.16}$$

gdje su $\xi, 0 \leq \xi \leq 1$ i τ_0 predstavljaju parametre. τ_0 je namješten na inicijalnu vrijednost svih feromona, a za ξ se pokazalo da je 0.1 povoljna vrijednost.

Lokalno osvježavanje feromona koje se provodi odmah po odabiru novog koraka svakog mrava omogućava trenutno smanjenje feromonskih vrijednosti već odabranih termina, a time povećava vjerojatnost da će se koristiti drugi manje istraženi termini već u ovoj iteraciji.

4. Algoritmi lokalne pretrage

Lokalna pretraga je vrlo često korištena vrsta heurističkih algoritama. Počinje od nekog početnog rješenja na kojem u svakoj iteraciji radi niz lokalnih promjena koje poboljšavaju kvalitetu rješenja u svakoj iteraciji dok se ne pronađe lokalni optimum. Primjeri lokalnih pretraga su simulirano kaljenje, tabu pretraga, pohlepni slučajni prilagodljivi algoritam pretrage (GRASP), pretraživanje promjenjivom širinom i slični.

Metode lokalne pretrage rade niz lokalnih promjena nad početnim rješenjem čime mu u svakoj iteraciji poboljšavaju vrijednost evaluacijske funkcije dok nije postignut lokalni optimum. Točnije, u svakoj iteraciji stvara se poboljšano rješenje x' u susjedstvu $\mathcal{N}(x)$ trenutnog rješenja x sve dok se više ne može pronaći bolje rješenje. To najčešće znači da je algoritam zapeo u lokalnom optimumu. [10]

U algoritmima lokalne pretrage možemo uočiti tri glavna načina za pomak na novo rješenja:

Slučajno poboljšanje (engl. *random improvement*) Novo rješenje se bira slučajnim odabirom boljeg rješenja iz susjedstva.

Prvo poboljšanje (engl. *first improvement*) Iterira se nekim redoslijedom kroz susjedstvo i prvo bolje rješenje postaje novo rješenje.

Najbolje poboljšanje (engl. *best improvement*) Iterira se nekim redoslijedom kroz susjedstvo i najbolje rješenje u cijelom susjedstvu postaje novo rješenje.

Zavisno od veličine prostora koji pretražujemo i očekivane kvalitete rješenja koristi se jedno od spomenutih pravila.

4.1. Pretraga promjenjivih susjedstva

Lokalna pretraga s više susjedstva koja se izmjenjuju je bila vrlo neuobičajena prije pojave ovog algoritma. Koristeći više susjedstva koja se izmjenjuju i jednostavnu lo-

kalnu pretragu dobili smo novi algoritam. Ovaj pristup se zove pretraga promjenjivih susjedstva (engl. *Variable Neibghorhood Search, VNS*). [8]

VNS se zasniva na sljedeće tri pretpostavke:

1. Lokalni optimum unutar jednog susjedstva ne mora biti isti kao i u drugom susjedstvu.

2. Globalni optimum je lokalni optimum unutar svih mogućih susjedstva.

3. Lokalni optimumi su vrlo blizu jedan drugoga, a mogu se nalaziti u istome ili različitim susjedstvima.

VNS ne prati putanju, već istražuje sve udaljenija susjedstva u odnosu na trenutno rješenje i odabire novo rješenje ako i samo ako je pronađeno rješenje bolje od trenutnog. Na ovaj način će mnoge poželjne karakteristike trenutnog rješenja biti prenesene u novo rješenje. Nad novim rješenjem se dodatno provodi lokalna pretraga kako bi se došlo do lokalnog optimuma. Susjedstvo je okolina trenutnog rješenja x, a susjedstva se međusobno razlikuju po načinu na koji stvaraju nova rješenja. Za rad sa susjedstvima je potrebno definirati funkcije koja će moći mjeriti udaljenost između dva rješenja.

Postoji više različitih inačica algoritma, ali sve inačice zahtijevaju funkciju inicijalizacije opisanu pseudokodom 4.1. Ova funkcija prije početka rada algoritma definira evaluacijsku funkciju f, prostor rješenja X, konačni skup susjedstva $\mathcal{N}_k, (k = 1, \ldots, k_{max})$, a s $\mathcal{N}_k(x)$ skup rješenja oko x u k-tom susjedstvu.

Algoritam 4.1 Inicijalizacija pretrage promjenjivih susjedstva

funkcija INICIJALIZIRAJ
  $f \leftarrow$ evaluacijska funkcija
  $X \leftarrow$ prostor rjesenja
  definiraj skup susjedstva $\mathcal{N}_k, (k = 1, \ldots, k_{max})$
  $x \leftarrow$ početno rješenje
kraj funkcija

Kako bi algoritam mogao izaći iz lokalnih optimuma, unija svih susjedstva oko nekog mogućeg rješenja x bi trebala sadržavati cijeli skup mogućih rješenja:

$$X \subseteq \mathcal{N}_1(x) \cup \mathcal{N}_2(x) \cup \ldots \cup \mathcal{N}_{k_{max}}(x), \forall x \in X, \tag{4.1}$$

pri čemu je X skup svih mogućih rješenja. Ovi skupovi (susjedstva) ne moraju dijeliti skup X na dijelove već mogu činiti ugniježđene cjeline tako da vrijedi:

$$\mathcal{N}_1(x) \subset \mathcal{N}_2(x) \subset \ldots \subset \mathcal{N}_{k_{max}}(x), X \subset \mathcal{N}_{k_{max}}(x), \forall x \in X, \tag{4.2}$$

čime odmah vrijedi i $|\mathcal{N}_1| < |\mathcal{N}_2| < \ldots < |\mathcal{N}_{k_{max}}|$. Ukoliko ne vrijedi barem jedno od pravila (4.1) i (4.2) još uvijek postoji mogućnost da ćemo biti u mogućnosti potpuno istražiti skup svih mogućih rješenja, ali nije garantirano. [9]

4.1.1. Spust po promjenjivim susjedstvima

Algoritam spusta po promjenjivim susjedstvima (engl. *variable neighborhood descent, VND*) je vrlo jednostavan i gotovo potpuno deterministički algoritam. Algoritam funkcionira tako da pronađe najboljeg susjeda x' unutar susjedstva $\mathcal{N}_k(x)$, ukoliko je novo rješenje x' bolje od trenutnog rješenja x, novo rješenje postaje trenutno i nastavlja potragu od prvog susjedstva $\mathcal{N}_k(k \leftarrow 1)$, a u suprotnom se pomiče u novo susjedstvo $\mathcal{N}_k(k \leftarrow k+1), k \leq k_{max}$. Cijeli postupak je opisan pseudokodom 4.2.

Algoritam 4.2 Algoritam spusta po promjenjivim susjedstvima, VND

```
k ← 1
dok k ≤ k_max radi
    pronađi najboljeg susjeda x' iz susjedstva N_k(x)
    ako f(x') < f(x) onda
        x ← x'
        k ← 1
    inače
        k ← k + 1
    kraj ako
kraj dok
```

Istraživanje susjedstva završava pronalaskom najboljeg rješenja iz susjedstva $\mathcal{N}_k(x)$. Ukoliko je pretraga cijelog susjedstva dugotrajna i zahtjevna, a zadovoljni smo skoro optimalnim rješenjem, možemo koristiti jedan od sljedećih algoritama.

4.1.2. Osnovna pretraga promjenjivih susjedstva

Algoritam osnovne pretrage promjenjivih susjedstva (engl. *basic variable neighborhood search, BVNS*) je osnovni algoritam pretrage promjenjivih susjedstva nad kojim

su građene daljnje inačice algoritma koje uglavnom razlikuju u algoritmu *pretresa* susjedstva i algoritmu lokalne pretrage.

Rad algoritma je opisan pseudokodom 4.3. U trenutnom susjedstvu *pretresom* nalazi novo rješenje i nad njim provodi lokalnu pretragu. Ukoliko je novo rješenje bolje od trenutnog, novo rješenje postaje trenutno i nastavlja potragu u istom susjedstvu, a inače prelazi u sljedeće susjedstvo. [9] *Pretres* (engl. *shaking*) predstavlja pretragu susjedstva. Ovdje se radi o slučajnom odabiru novog rješenja iz trenutnog susjedstva. Izbjegava se korištenje nekog deterministčkog pravila kojim bi od trenutnog rješenja određenim nizom koraka došli do novog rješenja kako ne bi zapeli u petlji.

Algoritam 4.3 Algoritam osnovne pretrage promjenjivih susjedstva, BVNS

$k \leftarrow 1$
dok $k \leq k_{max}$ **radi**
  PRETRES: $x' \leftarrow$ slučajno rješenje iz susjedstva $\mathcal{N}_k(x)$
  $x'' \leftarrow$ obavi lokalnu pretragu nad x'
  ako $f(x'') < f(x)$ **onda**
    $x \leftarrow x''$
    $k \leftarrow 1$
  inače
    $k \leftarrow k + 1$
  kraj ako
kraj dok

Ostale inačice algoritma se uglavnom razlikuju po načinu pretrage susjedstva. Važno je napomenuti da se unutar glavne petlje (pseudokod 4.1) može definirati neki drugi kriterij za zaustavljanje poput:

- najveći broj iteracija,
- najveći broj iteracija bez poboljšanja ili
- najdulje trajanje izvođenja.

4.1.3. Reducirana pretraga promjenjivih susjedstva

Algoritam reducirane pretrage promjenjivih susjedstva (engl. *Reduced Variable Neighborhood Search, RVNS*) se razlikuje od BVNS-a samo po tome što ne provodi lokalnu pretragu nad novo izabranim rješenjem. Rad algoritma opisan je pseudokodom 4.4.

Algoritam 4.4 Algoritam reducirane pretrage promjenjivih susjedstva, RVNS

$k \leftarrow 1$
dok $k \leq k_{max}$ **radi**
  PRETRES: $x' \leftarrow$ slučajno rješenje iz susjedstva $\mathcal{N}_k(x)$
  ako $f(x') < f(x)$ **onda**
    $x \leftarrow x'$
    $k \leftarrow 1$
  inače
    $k \leftarrow k + 1$
  kraj ako
kraj dok

4.1.4. Sveobuhvatna pretraga promjenjivih susjedstva

Algoritam sveobuhvatne pretrage promjenjivih susjedstva (engl. *general variable neighborhood search, GVNS*) koristi spoj BVNS i VND algoritma. Umjesto lokalne pretrage iz BVNS algoritma ubačen je VND algoritam i koriste se dvije strukture susjedstva, $\mathcal{N}_k$ i $\mathcal{N}_l$ za zasebne dijelove algoritma.

Rad algoritma počinje *pretresom* susjedstva koje vraća moguće rješenje x' iz susjedstva $\mathcal{N}_k(x))$ koje se koristi za lokalnu pretragu. VND algoritam počinje pretragu od prvog susjedstva iz VND strukture susjedstva. Pretražuje susjedstvo $\mathcal{N}_l(x'')$ za najboljeg susjeda x'''. Ukoliko vrijedi $f(x''') < f(x'')$ postavlja x''' umjesto x'' i nastavlja potragu od prvog susjedstva $\mathcal{N}_l(l \leftarrow 1)$, a u suprotnom prelazi u slijedeće susjedstvo $\mathcal{N}_l(l \leftarrow l + 1), l \leq l_{max}$.

Po završetku VND algoritma ukoliko je vrijednost evaluacijske funkcije rješenja lokalne pretrage x'' manja od vrijednosti trenutnog rješenja x, prihvaća se novo rješenje i kreće se u novu iteraciju od prvog susjedstva $\mathcal{N}_k(k \leftarrow 1)$. U suprotnom se prelazi u sljedeće susjedstvo $\mathcal{N}_k(k \leftarrow k + 1), k \leq k_{max}$ i počinje nova iteracija. Cijeli postupak je detaljno opisan algoritmom 4.5.

4.1.5. Dekompozicijska pretraga promjenjivih susjedstva

Dekompozicijska pretraga promjenjivih susjedstva (engl. *variable neighborhood decomposition search, VNDS*) poboljšava rješenje iz susjedstva $\mathcal{N}_k(x)$ na temelju rastava problema koji se rješava. Jedina razlika u odnosu na BVNS jest da umjesto lokalne pretrage se traži rješenje nekog podproblema unutar nekog podprostora $V_k \subseteq \mathcal{N}_k(x)$ pri čemu vrijedi $x' \in V_k$. Za rješavanje ovog problema se također može koristiti VNS.

Algoritam 4.5 Algoritam sveobuhvatne pretrage promjenjivih susjedstva, GVNS

$k \leftarrow 1$
dok $k \leq k_{max}$ **radi**
    PRETRES: $x' \leftarrow$ slučajno rješenje iz susjedstva $\mathcal{N}_k(x)$
    $x'' \leftarrow$ obavi lokalnu pretragu nad x'
    $l \leftarrow 1$
    dok $l \leq l_{max}$ **radi**
        pronađi najboljeg susjeda x''' iz susjedstva $\mathcal{N}_l(x'')$
        ako $f(x''') < f(x'')$ **onda**
            $x'' \leftarrow x'''$
            $l \leftarrow 1$
        inače
            $l \leftarrow l + 1$
        kraj ako
    kraj dok
    ako $f(x'') < f(x)$ **onda**
        $x \leftarrow x''$
        $k \leftarrow 1$
    inače
        $k \leftarrow k + 1$
    kraj ako
kraj dok

Rad algoritma opisan je pseudokodom 4.6.

Algoritam 4.6 Algoritam dekompozicije promjenjivih susjedstva, VNDS

$k \leftarrow 1$
dok $k \leq k_{max}$ **radi**
  PRETRES: $x' \leftarrow$ slučajno rješenje iz susjedstva $\mathcal{N}_k(x)$
  $y \leftarrow$ skup k parametara prisutnih u x', ali ne i u $x(y = x' \backslash x)$
  LOKALNA PRETRAGA: $y' \leftarrow V_k(y)$, $x'' \leftarrow (x' \backslash y) \cup y')$
  ako $f(x'') < f(x)$ **onda**
    $x \leftarrow x''$
    $k \leftarrow 1$
  inače
    $k \leftarrow k + 1$
  kraj ako
kraj dok

U y je pohranjen skup k parametara prisutnih u x', ali ne i u $x(y = x' \backslash x)$ koji predstavljaju neki podproblem. *Lokalna pretraga* traži lokalni optimum u prostoru oko y pretraživanjem ili heuristikom. Rješenje pohranjuje u y' te x'' koji sadrži odgovarajuće rješenje ($x'' = (x' \backslash y) \cup y'$).

4.1.6. Iskrivljena pretraga promjenjivih susjedstva

VNS daje bolje rezultate kada postoji mnogo dolina s lokalnim optimumima. Mnogi problemi imaju lokalne optimume zajedno grupirane i tada je lako "skakati" iz jednog u drugi optimum dok se ne pronađe globalni optimum. U slučajevima kada su doline s lokalnim optimumima vrlo udaljene jedna od druge, a proučavamo sve šira i šira susjedstva, može se dogoditi da algoritam počne stagnirati. Prihvaćanje nekog lošijeg rješenja u blizini ili potpuno nasumičnog (boljeg) rješenja jako udaljenog od trenutnog rješenja nam neće reći jesmo li našli kvalitetniji skup rješenja. Ovaj problem možemo riješiti tako da kada provjeravamo hoće li novo rješenje biti prihvaćeno umjesto evaluacijske funkcije ubacimo funkciju koja analizira i udaljenost dvaju rješenja. Time smo dobili iskrivljenu pretragu promjenjivih susjedstva (engl. *skewed variable neighborhod descent, SVNS*).

SVNS koristi funkciju $\rho(x, x'')$ koja mjeri udaljenost između dva moguća rješenja. Parametar α mora biti odabran tako da omogući istraživanje udaljenih rješenja koja su malo lošija od trenutnog rješenja. Drugim riječima, vrijednost evaluacijske funkcije

Algoritam 4.7 Algoritam iskrivljene pretrage promjenjivih susjedstva, SVNS

$k \leftarrow 1$
dok $k \leq k_{max}$ **radi**
  PRETRES: $x' \leftarrow$ slučajno rješenje iz susjedstva $\mathcal{N}_k(x)$
  $x'' \leftarrow$ obavi lokalnu pretragu nad x'
  ako $f(x'') < f(x)$ **onda**
    $f_{opt} \leftarrow f(x'')$
    $x_{opt} \leftarrow x''$
  kraj ako
  ako $f(x'') - \alpha\rho(x, x'') < f(x)$ **onda**
    $x \leftarrow x''$
    $k \leftarrow 1$
  inače
    $k \leftarrow k + 1$
  kraj ako
kraj dok

udaljenog rješenja $f(x'')$ mora biti dovoljno umanjena da rješenje bude prihvaćeno iako je lošije od trenutnog rješenja x.

5. Implementacija

Ovim poglavljem je opisana implementacija *Sustava* ostvarena u okviru ovog rada. U prvom dijelu su opisani pojedini moduli na koje je podijeljen *Sustav*, njihova funkcija i razlozi za takvu podjelu. U sljedećim poglavljima se govori o implementaciji algoritama opisanih u prethodnim poglavljima. U drugom dijelu se objašnjava način implementacije algoritma optimizacije mravljom kolonijom, a u trećem dijelu implementacija algoritma lokalne pretrage. Točnije, radi se o algoritmu pretrage promjenjivih susjedstva.

Implementacija u okviru ovog rada je ostvarena nad dijelom postojećeg rješenja ostvarenog u [12].

5.1. Implementacija rješenja

Iako su obavljeni razgovori u pet osnovnih i srednjih škola, ostaje velika mogućnost da postoje neki zahtjevi koji nisu pokriveni zahtjevima opisanim u [14]. Upravo iz tog razloga *Sustav* mora biti fleksibilan i lako nadogradiv. Cijeli *Sustav* je implementiran u Java programskom jeziku koristeći načela objektno-orijentiranog programiranja.

5.1.1. Osnovni dijelovi *Sustava*

Sustav je zbog fleksibilnosti i bolje apstrakcije podijeljen na četiri dijela:

- modul *Core*,
- modul *Data*,
- modul *Engine* i
- modul *GUI*.

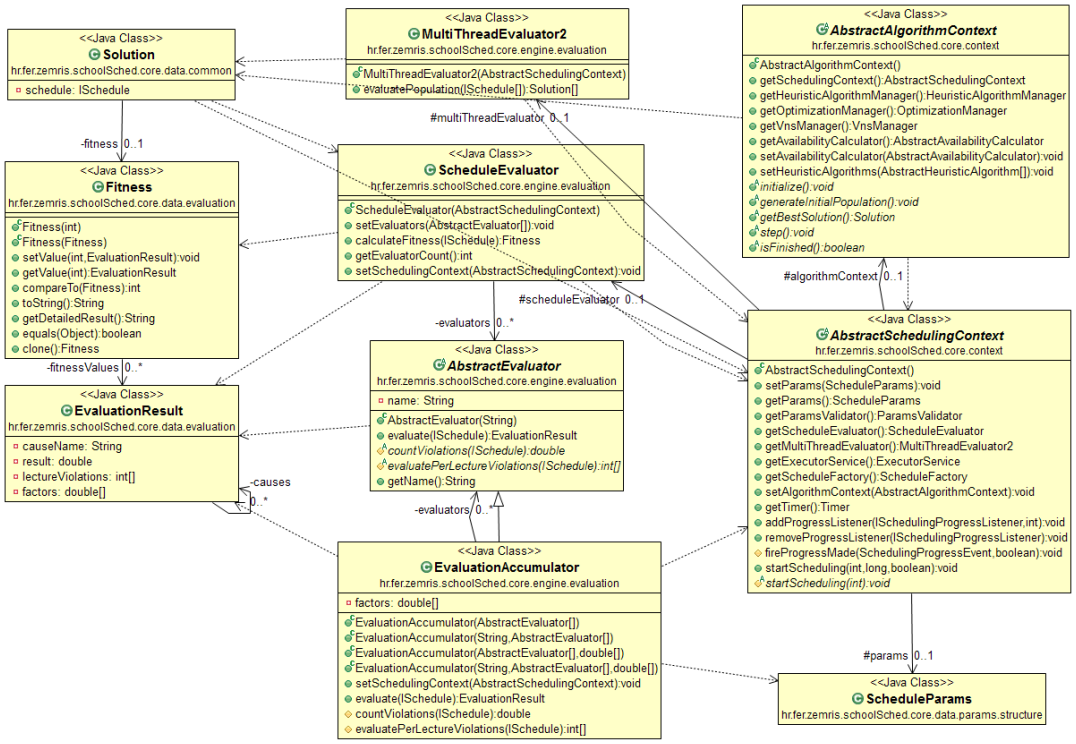
Dva osnovna modula su *Core* i *Data*, a nad njima se gradi konkretno rješenje problema koje pokreće modul *Engine*. Modul *GUI* je korisničko sučelje koje nudi mogućnost

unosa ulaznih vrijednosti i zadavanja ograničenja te pokretanje izrade i pregled rješenja. Modularnost *Sustava* je velika prednost jer se izmjenom ili kombiniranjem različitih *Engine* modula može doći do novih (boljih) rješenja.

U nastavku se nalazi pojednostavljeni pregled spomenutih modula jer za razumijevanje rada *Sustava* nije potreban njihov detaljan opis. U nastavku je opisano nekoliko osnovnih sučelja i razreda s kojima ćemo se često susretati. Detaljniji opis je dostupan u [12].

Modul *Core*

Core modul sadrži osnovne podatkovne strukture, sučelja i apstraktne razrede za različite algoritme te okolinu koja pokreće i vodi izradu rasporeda sati. U nastavku je ponuđen pregled osnovnih razreda i sučelja koja čine ovaj modul. Pregled najvažnijih dijelova ovog modula dan je i dijagramom razreda na slici 5.1.



Slika 5.1: Dijagram razreda modula *Core*

Unutar *Core* modula nalaze se strukture podataka koje predstavljaju resurse, a neki primjeri takvih razreda su:

- `Classroom` predstavlja prostoriju,
- `Lecture` predstavlja predavanje,

- `Period` predstavlja školski sat,
- `Subject` predstavlja školski predmet,
- `Teacher` predstavlja predavača,
- `Availability` predstavlja dostupnost pojedinog resursa,
- `Connection` predstavlja vezu i odnos između dva predavanja
- i drugi razredi.

Kako bi se stvorio precizan okvir za izgradnju *Engine* modula, unutar *Core* modula je definiran niz sučelja i apstraktnih razreda koji definiraju izgled algoritama i postupaka koji će biti implementirani. Ovo je nužno kako bi okolina za izradu rasporeda mogla pokrenuti izradu rasporeda, pratiti napredak, spremati rješenja i zaustaviti izradu kada je gotova. Neki od apstraktnih razreda su:

- `AbstractAlgorithm` za običan algoritam,
- `AbstractHeuristicAlgorithm` za heuristički algoritam,
- `AbstractMetaheuristicAlgorithm` za metaheuristički algoritam,
- `AbstractLocalSearchAlgorithm` za algoritam lokalne pretrage
- i drugi razredi.

Okolina za izradu rasporeda je predstavljena apstraktnim razredom koji je potrebno nadograditi kako bi sadržavala sve informacije potrebne za rad odabrane vrste algoritma. Spomenuta okolina brine o sljedećem:

- učitavanju podataka iz ulazne datoteke,
- inicijalizaciji algoritama i struktura podataka,
- davanju pristupa potrebnim podacima i algoritmima,
- pokretanju izrade rasporeda,
- pokretanju novih iteracija algoritama,
- praćenju i bilježenju rada,
- pohrani međurješenja i konačnog rješenja te
- zaustavljanju rada i gašenju *Sustava*.

Za vrijeme rada su svim algoritmima dostupne sve informacije o dostupnosti resursa te čvrstim i mekim ograničenjima kao i sve ostale informacije sadržane u ulaznoj datoteci. Neki od osnovnih konkretnih razreda koji predstavljaju okolinu za pokretanje i izradu rasporeda su:

`ParamsLoader` služi za učitavanje parametara iz ulazne *xml* datoteke u odgovarajuće strukture podataka.

`SchedulingContext` služi kao okruženje koje učitava parametre, provjerava ih, pokreće izradu rasporeda, prati napredak i zapisuje rezultat.

`AbstractAlgorithmContext` apstraktni razred koja služi kao osnova za izgradnju okruženja za pokretanje algoritama izrade rasporeda i njihove optimizacije. Daje algoritmima na raspolaganje sve potrebne informacije i pazi na redoslijed izvođenja.

`ProgressListener` razred za vrijeme rada *Sustava* bilježi sve događaje poput novih rješenja i pogrešaka.

Raspored je predstavljen `ISchedule` sučeljem koje definira osnovne metode koje mora podržavati svaki raspored. Konačna implementacija ovog sučelja u sebi sadrži sve podatke o terminima i prostorijama za predavanja. Neke od metoda sučelja koje moraju biti implementirane su:

- `reserveLecture` Metoda za rezervaciju predavanja u traženom terminu i prostorijama.
- `freeLecture` Metoda za oslobađanje termina i prostorija koje su bile rezervirane za definirano predavanje.
- `getSlotForLecture` Dohvat termina za odabrano predavanje.
- `getLocationsForLecture` Dohvat prostorija za odabrano predavanje.
- `getLecturesInSlot` Dohvat svih predavanja za odabrani termin.

Sučeljem su definirane i druge metode za dohvat informacija o rasporedu.

Modul *Data*

Modul *Data* u sebi sadrži osnovne metode za rad s ulaznim *xml* te izlaznim *xml* i *xls* datotekama te definira njihov osnovni oblik i ponašanja kroz sučelja i apstraktne razrede. U ovom modulu se nalaze i konkretne implementacije `ISchedule` sučelja.

Podaci potrebni za rad *Sustava* su definirani u ulaznoj *xml* datoteci[1] koja sadrži popis razreda, nastavnika, predmeta, predavanja, prostorija i drugih resursa te su zadana sva čvrsta i meka ograničenja. Neki od parametara potrebnih za izradu rasporeda su:

- broj tjedana u kojem se održava nastava

[1] Primjer ulazne datoteke se nalazi na kraju ovog rada.

– broj dana u tjednu u kojima se održava nastava

– popis svih termina predavanja

– popis prostorija[2]

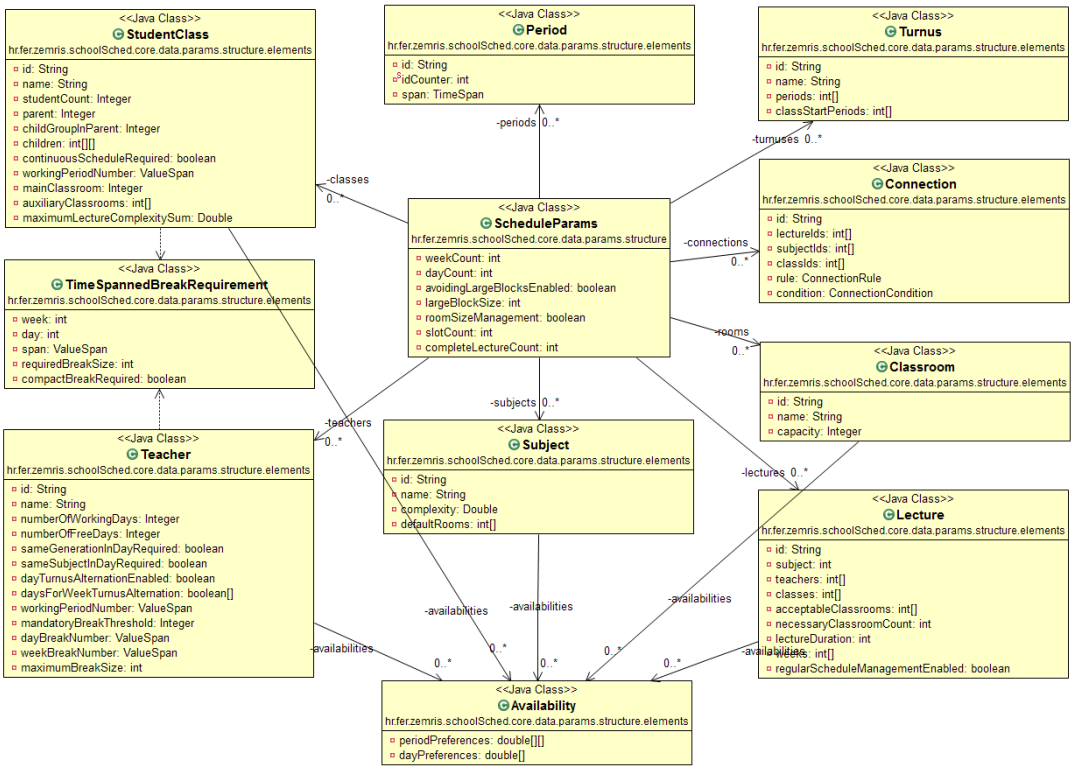
– popis predmeta

– popis nastavnika

– popis predavanja

– popis veza među predavanjima

Pregled strukture parametara vidljiv je i na dijagramu razreda na slici 5.2.



Slika 5.2: Dijagram razreda strukture parametara

Struktura izlazne *xml* datoteke u koju se zapisuje rezultat je jednostavna, sadrži popis predavanja sa pridruženim terminom i prostorijom[3] dok izlazna *xls* datoteka sadrži tablični prikaz rasporeda nastave za profesore, razrede i prostorije.

[2]Nije obavezan.

[3]Ukoliko je moguće definirati prostoriju.

Modul *Engine*

Modul *Engine* predstavlja konkretnu implementaciju algoritama koji brinu o izradi rasporeda. Ovdje se javljaju konkretne implementacije prethodno spomenutih apstraktnih razreda i njihov opis se nalazi u poglavljima 5.3 i 5.4. Osnovna zamisao *Sustava* jest da ovaj modul bude lako zamijenjiv.

5.1.2. Višedretvenost

Višedretvenost se koristi za rad mrava, stvaranje novih susjeda, i vrednovanje više rasporeda odjednom. U sljedećim poglavljima će biti više govora o načinu rada tih algoritama.

Za ostvarivanje višedretvenosti koristi se Java implementacija `ExecutorService` a. Taj razred sam upravlja stvaranjem, pokretanjem i zaustavljanjem dretvi. Pomoću metode `newFixedThreadPool(n)` se kreira novi `ExecutorService` koji odjednom dozvoljava da bude najviše `n` dretvi pokrenuto. Zatim se pomoću metode `execute( Runnable task)` zadaju zadaci koje je potrebno pokrenuti.

Broj dretvi je ograničen na broj dostupnih jezgri, dostupan pomoću metode `Runtime.getRuntime().availableProcessors()` koja vraća broj procesora koji su na raspolaganju Java virtualnom stroju.

5.2. Vrednovanje rješenja

Raspored se vrednuje po tome koliko je „prekršaja“ napravljeno takvim rasporedom predavanja. Neki primjeri prekršaja su: nedostupan profesor, razred ili prostorija u tom terminu, preljev bloka predavanja u drugi turnus ili dan, više predavanja istog predmeta u istome danu i slično. Ukratko, to su sva prekršena čvrsta i meka ograničenja.

Procjena se vrši pomoću razreda koji nasljeđuju apstraktni razred `AbstractEvaluator`. Osnovna metoda rada, koja se pokreće metodom `evaluate()` jest da konkretni procjenitelj iterira po predavanjima, profesorima, razredima i slično tražeći prekršaj za koji je zadužen. Ako ga je našao povećava brojač prekršaja koji će vratiti po završetku. Paralelno uz ovakav brojač u većini procjenitelja[4] je implementirana i metoda koja vraća polje predavanja s brojačem prekršaja za pojedino predavanje. Na taj način je moguće izolirati predavanja koja su najproblematičnija i pomoću

[4]Kod nekih procjenitelja to nije moguće jer se vrednovanje odnosi na grupu predavanja ili rupe između predavanja.

lokalne pretrage popraviti raspored. U trenutnoj implementaciji se koriste sljedeći procjenitelji:

- `IllegalLectureLocation` Predavanje u krivoj prostoriji.
- `DaySplittedLecture` Blok s preljevom u drugi dan.
- `IllegalLectureWeek` Predavanje u krivom tjednu predavanja.
- `ConnectionViolation` Nepoštivanje veze između predavanja.
- `TeacherAvailabilities` Profesor nije dostupan u tom terminu.
- `ClassAvailabilities` Razred nije dostupan u tom terminu.
- `RoomAvailabilities` Prostorija nije dostupna u tom terminu.
- `LectureAvailabilities` Predavanje nije dostupno u tom terminu.
- `ClassContinuousSchedule` Rupe u rasporedu razreda.
- `TeacherContinuousSchedule` Rupe u rasporedu profesora.
- `MultipleLecturesOfSameSubjectInDay` Više predavanja istog predmeta u danu.
- `TeacherWorkingPeriods` Previše ili premalo predavanja za jednog profesora u danu.
- `ClassWorkingPeriods` Previše ili premalo predavanja za razred u danu.

Nakon vrednovanja rasporeda vraća se `Solution` razred koji u sebi sadrži razred `EvaluationResult` i raspored koji je upravo evaluiran. Razred `EvaluationResult` predstavlja rezultat vrednovanja i u sebi sadrži niz razreda `Fitness` koje vraćaju spomenuti procjenitelji.

Za primjenu nekoliko procjenitelja odjednom se koristi `EvaluationAccumulator` koji nasljeđuje `AbstractEvaluator` tako da za algoritam on predstavlja obični procjenitelj, a istovremeno omogućuje ugnježđivanje procjenitelja. Razred `AbstractEvaluator` prima u konstruktoru niz `AbstractEvaluator`a i dozvoljava postavljanje koeficijenata koji predstavljaju težine i odgovaraju pojedinom procjenitelju. Pri pozivu metode `evaluate()`, iterativno poziva `evaluate()` metodu procjenitelja koje sadrži. Kao rezultat vraća `EvaluationResult` koji u sebi sadrži pojedinačne rezultate svih procjenitelja. Ti rezultati se množe s težinama i zbrajaju te daju konačnu vrijednost za taj procjenitelj, težinsku sumu pojedinačnih procjenitelja.

Svakom procjenitelju je dodijeljen i naziv tako da se pri detaljnom ispisu lako može procijeniti gdje je najveći problem.

Izvorni kod 5.1: Primjer ispisa procjenitelja

```
[37348.0]   Final accumulator
    10.0*[132.0]    Illegal lecture location
    100.0*[0.0] Day splitted lecture
    50.0*[0.0]  Illegal lecture week
    30.0*[29.0] Connection violation
    50.0*[63.0] Availabilities accumulator
        1.0*[0.0]   Teacher availabilities
        1.0*[36.0]  Class availabilities
        1.0*[0.0]   Room availabilities
        1.0*[27.0]  Lecture availabilities
    35.0*[60.0] Class continuous schedule
    35.0*[719.0]    Teacher continuous schedule
    25.0*[167.0]    Multiple lectures of same subject in day
    8.0*[71.0]  Period count accumulator
        1.0*[43.0]  Teacher working periods
        1.0*[28.0]  Class working periods
```

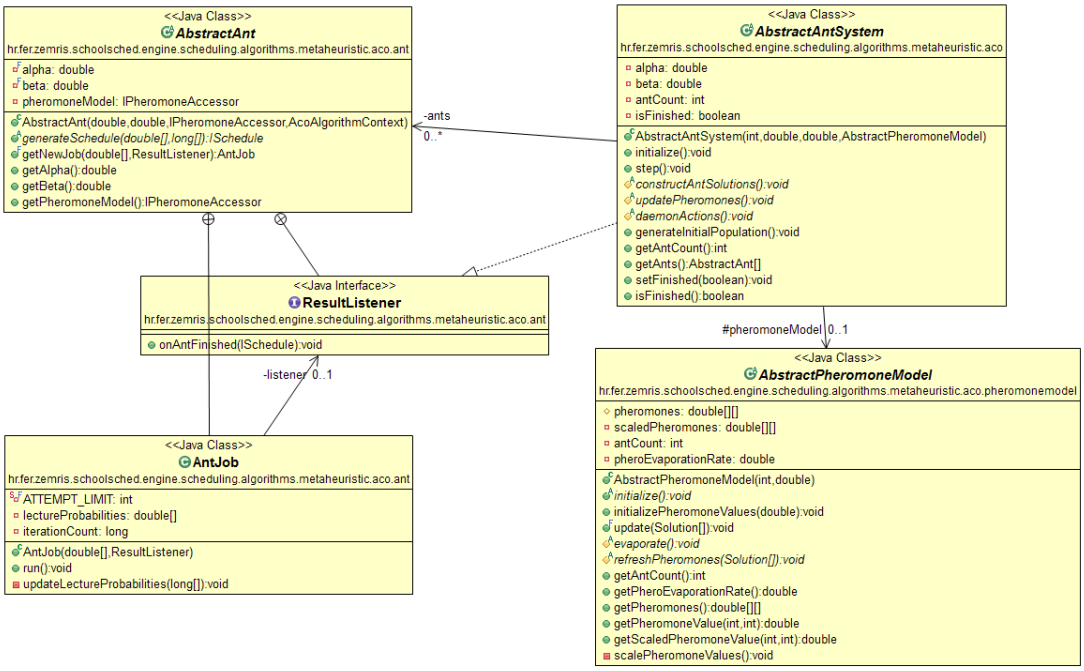
5.3. Algoritam optimizacije mravljom kolonijom

U ovom odjeljku su opisani algoritmi optimizacije mravljom kolonijom. Na početku se opisuje jednostavan konstrukcijski algoritam koji služi za stvaranje početnih rješenja i nad kojim je izgrađen algoritam za stvaranje mravljih rješenja. U nastavku su opisane implementacije pojedinih inačica algoritma. Slika 5.3 nudi dijagram razreda osnovnih razreda implementacije mravljih algoritama.

5.3.1. Konstrukcijski algoritam

Osnovni heuristički konstrukcijski algoritam se koristi za stvaranje početnih rješenja prilikom pokretanja *Sustava* i inicijalizacije algoritama. Rad ovog algoritma može se prikazati pseudokodom 5.1.

Algoritam počinje s radom tako da inicijalizira vrijednosti jednodimenzionalnog polja $slozenostPredavanja$ veličine l, gdje l odgovara ukupnom broju predavanja. Vrijednost $slozenostPredavanja_i$ predstavlja brojač neuspjelih pokušaja raspoređivanja i-tog predavanja. Početna vrijednost polja $slozenostPredavanja$ je 1. Svaki put kada se predavanje l_i ne uspije rasporediti u neki termin, povećava se vrijednost $slozenostPredavanja_i$ za jedan.



Slika 5.3: Diagram razreda za algoritam optimizacije mravljom kolonijom

Nakon inicijalizacije, poziva se metoda za izgradnju rasporeda sve dok ne vrati potpuni raspored. Polje $slozenostPredavanja$ kopiramo u polje $slozenostPredavanjaK$ kako bismo mogli eliminirati pojedina predavanja postavljanjem vrijednosti njihova polja na 0. Polje $dostupnostTermina$ veličine s gdje s odgovara ukupnom broju termina se računa pomoću izračuna dostupnosti koji će biti opisan u idućem odjeljku. Odabir predavanja p i termina t se vrši *pravilom slučajno-proporcionalnog odabira* u odnosu na polja $slozenostPredavanjaK$ i $dostupnostTermina$. Ukoliko više nema termina za odabir, povećava se vrijednost $slozenostPredavanja_p$ za jedan i vraća se prazan raspored.

Rezervacija termina t za predavanje p se obavlja ukoliko su svi potrebni resursi u danom trenutku dostupni. Ako predavanje p nije moguće smjestiti u termin t postavlja se vrijednost $dostupnostTermina$ na 0. U suprotnom postavljamo vrijednost $slozenostPredavanjaK_p$ na 0 i biramo novo predavanje sve dok ima još predavanja za rasporediti.

Kažemo da eliminiramo predavanja ili termine postavljanjem pojedinih elemenata polja $slozenostPredavanjaK$ i $dostupnostTermina$ na 0 jer se ti elementi preskaču kada se koristi *pravilo slučajno-proporcionalnog odabira*.

Cilj ovog algoritma nije konstrukcija konačnog rješenja koje bi se negdje koristilo, već dobivanje početnog rješenja koje nam nudi neke općenite informacije o rješenjima

Algoritam 5.1 Osnovni konstrukcijski algoritam

```
funkcija MAIN
    slozenostPredavanja[] ← 1
    raspored ← null
    dok r == null radi
        raspored ←generirajRaspored(slozenostPredavanja)
    kraj dok
kraj funkcija
funkcija GENERIRAJRASPORED(slozenostPredavanja)
    r ←noviRaspored()
    slozenostPredavanjaK ← slozenostPredavanja.kopiraj()
    dok ima predavanja za rasporediti radi
        p ← odaberiPredavanje(slozenostPredavanjaK)
        dostupnostTermina ← izracunajDostupnosti(p)
        dok radi beskonačno radi
            t ←odaberiTermin(dostupnostTermina)
            ako nema vise termina onda
                slozenostPredavanja[p]++ vrati null
            kraj ako
            ako r.rezerviraj(p,t) onda
                slozenostPredavanjaK[p] ← 0
                prekini petlju
            inače
                dostupnostTermina ← 0
            kraj ako
        kraj dok
    kraj dok
vrati r
kraj funkcija
```

koja možemo očekivati. Rad algoritma jako sliči radu mrava koji je prethodno opisan, samo što nema informaciju o feromonskim tragovima i koristi jednostavnije (neprecizније) heurističke informacije.

5.3.2. Izračun dostupnosti

Izračun dostupnosti (engl. *availability calculator*) je jedinstven način za izračun dostupnosti svih resursa uključenih u odabrano predavanje. Dostupnost resursa predstavlja heurističku informaciju u algoritmu optimizacije mravljom kolonijom. Apstraktni razred `AbstractAvailabilityCalculator` čiji izvorni kod je vidljiv u 5.3.2 definira izgled i ponašanje implementacija izračuna dostupnosti.

Metoda `initializeForLecture(int LectureId)` priprema kontekst za odabrano predavanje. Kontekst se sastoji od:

- dostupnosti predavanja,
- dostupnosti razreda koji slušaju to predavanje,
- dostupnosti predavača koji održava to predavanje i
- dostupnosti prostorija u kojima je poželjno održati to predavanje.

Pozivom metode `calculateForAllSlots(int lectureId)` priprema se kontekst za odabrano predavanje i potom iterira po svim terminima predavanja i izračunava dostupnost za sve kombinacije odabranog predavanja i svih termina. Rezultat izračuna je jednodimenzionalno polje A veličine s koja odgovara broju termina. Na poziciji s_i se nalazi kombinirana vrijednost navedenih dostupnosti koja se računa pozivom metode `calculateForSlot(int slotId)`.

Kombinirana dostupnost za pojedini termin se izračunava prethodnim izračunom svake od navedenih dostupnosti. Vrijednost pojedine dostupnosti će uvijek biti veća od 0, nalaziti će se u intervalu $< 0, 1 >$ ukoliko je neka od dostupnosti za navedeni termin 0, a inače je uvijek veća od 1. Dobivene dostupnosti se međusobno množe i skaliraju na interval $< 0, 1]$.

Izvorni kod razreda za izračun dostupnosti

```
1 public abstract class AbstractAvailabilityCalculator extends
    SchedulingContextAccessor{
2
3     protected abstract void initializeForLecture(int lectureId);
4     protected abstract double calculateForSlot(int slotId);
5
6     public synchronized double[] calculateForAllSlots(int lectureId)
          {
7         double[] availabilities = new double[schedulingContext.
              getParams().getSlotCount()];
8
9         initializeForLecture(lectureId);
```

```
10
11          for (int slotId = 0; slotId < slotCount; slotId++)
12              availabilities[slotId] = calculateForSlot(slotId);
13
14          return availabilities;
15      }
16  }
```

Klasa `CachingAvailabilityCalculator` omotava konkretnu implemntaciju klasu dane izvornim kodom 5.3.2 i pamti izračunate vrijednosti dostupnosti. Na zahtjev umjesto ponovnog izračuna vraća spremljene vrijednosti.

5.3.3. Izgradnja mravljeg rješenja

Izgradnja mravljeg rješenja se temelji na radu jednostavnog konstrukcijskog algoritma. Osnovna razlika je u izračunu dostupnosti termina. Pri izračunu dostupnosti termina koriste se iste informacije kao i u jednostavnom konstrukcijskom algoritmu uz dodatak feromonskih tragova.

5.4. Algoritmi lokalne pretrage

Ovo poglavlje ukratko opisuje implementirane algoritme lokalne pretrage.

5.4.1. Jednostavna optimizacija

Jednostavna optimizacija se provodi nad svim rješenjima koja se stvore za vrijeme rada *sustava* jer je vrlo jednostavna. Optimizacija kreće dohvatom sume prekršaja po predavanju kojom se utvrđuje koja su najproblematičnija predavanja u danom rasporedu. Potom se jedno po jedno predavanja pokušava smjestiti u povoljniji termin, samo koristeći dostupnosti kao heurističke informacije. Algoritam se zaustavlja kada dođe do predavanja koje nema prekršaja.

5.4.2. Algoritmi pretrage promjenjivih susjedstva

Implementirani su sljedeći algoritmi pretrage promjenjivih susjedstva:

- RVNS
- BVNS
- SVNS

Implementacija pojedinog algoritam točno prati pseudokod kojim su opisani algoritmi u poglavlju 5.4.

Za potrebe rada algoritma implementirana su sljedeća susjedstva:

Susjedstvo slučajne zamjene predavanja u rasporedu razreda Ovo susjedstvo odabire dva slučajna predavanja u rasporedu slučajno odabranog razreda i međusobno im zamijeni termine.

Susjedstvo slučajne zamjene predavanja u rasporedu profesora Ovo susjdestvo radi isti postupak kao i prethodno, ali u rasporedu slučajno odabranog profesora.

Susjedstvo kontinuiranog rasporeda za razrede Ovo susjedstvo traži rupe u rasporedu slučajno odabranog razreda i pokušava predavanjima s početka i kraja dana popuniti rupe u rasporedu.

Susjedstvo kontinuiranog rasporeda za profesore U ovom susjedstvu, kao i u prethodnom se popunjavaju rupe u rasporedu, ali u rasporedu profesora.

Susjedstvo previše ili premalo radnih sati za razrede Za svaki razred je definiran minimalan i maksimalan broj sati u danu. Ukoliko je taj kriterij prekršen, ovo susjedstvo pokušava u rasporedu odabranog razreda iz dana koji su preopterećeni prebaciti dovoljno sati u dane u kojima ima premalo sati.

Susjedstvo previše ili premalo radnih sati za profesore Ovo susjedstvo obavlja isti postupak kao i prethodno, ali u rasporedu odabranog profesora.

Susjedstvo više istih predmeta u danu Ukoliko u rasporedu slučajnog odabranog razreda postoji više od jednog predavanja istog predmeta unutar jednog dana, ova predavanja se pokušavaju izdvojiti u zasebne dane.

Susjedstvo optimizacije prostorija Ovo susjedstvo provjerava jesu li prostorije slučajno odabranog predavanja odgovarajuće. Ukoliko nisu pokušava predavanje premjestiti u odgovarajuće prostorije.

6. Rezultati i rasprava

Ovo poglavlje opisuje način testiranja i analizu rezultata. U prvom odjeljku će biti objašnjeni parametri koji su odabrani za testiranje i vrijednosti koje ćemo testirati. Drugi odjeljak opisuje metode i način provođenja testiranja, a treći odjeljak prezentira rezultate i kratku raspravu. Tabličnim prikazom i grafovima su predstavljeni najvažniji rezultati, a na kraju rada se nalazi tablica sa svim rezultatima. Za testiranje je korišten problem izrade rasporeda u *Osnovnoj školi Voltino* iz [12]. Prije opisanog skupa testova proveden je niz testova koji ovdje nisu opisani, a služili su za određivanje okvirnih vrijednosti parametara i ispitivanje stabilnosti *Sustava*.

6.1. Testni parametri

Implementacijom *Sustava* spojena su dva algoritma kako bi radila zajedno. Algoritam optimizacije mravljom kolonijom i algoritam pretrage promjenjivih susjedstva. Za oba algoritma bilo je potrebno odabrati koje parametre ćemo iskoristiti za testiranje, a koje ćemo postaviti na neku unaprijed zadanu vrijednost.

Odabrana je $\mathcal{MAX}-\mathcal{MIN}$ inačica algoritma optimizacije mravljom kolonijom za testiranje jer prema [6] daje najbolje rezultate i prvotnim testiranjem se to pokazalo istinitim. Ova inačica algoritma ima mogućnost mijenjanja sljedećih parametara:

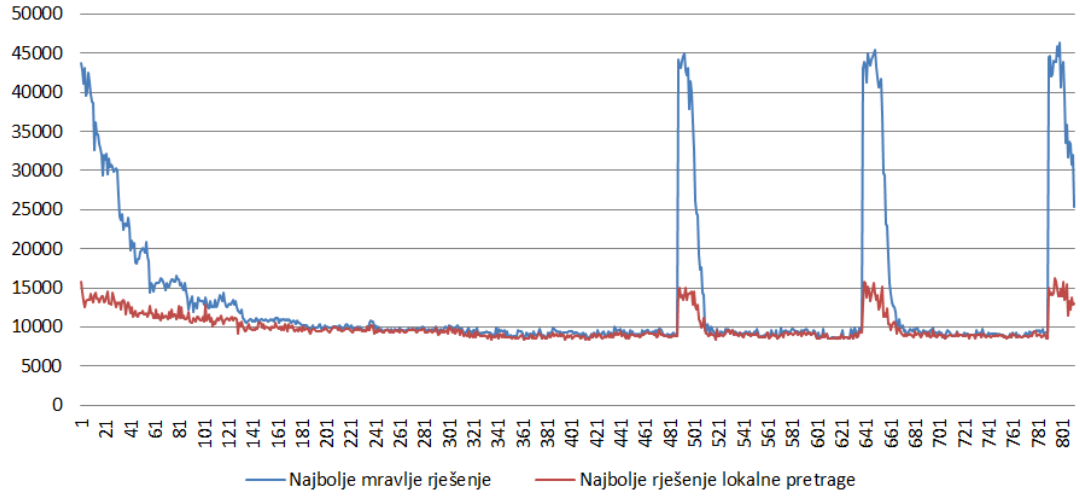
- m (broj mrava),
- α i β (odnos feromona i heurističke vrijednosti pri odabiru slijedećeg koraka),
- ρ (koeficijent isparavanja feromona),
- broj iteracija bez poboljšanja do ponovne inicijalizacije feromona i
- vjerojatnost za korištenje najboljeg-do-sad rješenja.

Za potrebe testiranja korišteni su parametri n, α i β te ρ.

U [6] preporučaju da *broj iteracija bez poboljšanja do ponovne inicjalizacije feromona* bude neka vrijednost nakon koje je kartakteristično za odabrani problem da

algoritam uđe u stanje stagnacije. Prethodnim testiranjem rada *Sustava* s početnim vrijednostima kao u *testu 1-1* (slika 6.1) uočeno je da kada algoritam uđe u stanje stagnacije da najčešće nakon otprilike 100 iteracija pronađe bolje rješenje i nastavi s radom. Zato je ovaj parametar fiksiran na vrijednost od 150 iteracija.

Za parametar *vjerojatnosti korištenja najboljeg-do-sad rješenja* predlaže se da se za manje probleme uvijek koristi najbolji-u-koloniji mrav. Za veće probleme je predloženo da se uvijek koristi najbolje-do-sada rješenje ukoliko želimo da algoritam bude pohlepan ili da se izmjenjuju ta dva rješenja. Kako se uz algoritam optimizacije mravljom kolonijom koristi i algoritam lokalne pretrage nije poželjno da algoritam bude pohlepan tako da je odabrana mala vrijednost, 0.2. Odabrana vrijednost se pokazala dobrom jer nakon ponovne inicijalizacije feromonskih vrijednosti algoritam nije prebrzo težio prema boljim rješenjima i nije odmah zapeo u lokalnom optimumu.



Slika 6.1: Odnos najboljeg mravljeg rješenja i najboljeg rješenja lokalne pretrage

Jedan od prvotnih testova je prikazan na slici 6.1. Apscisa prikazuje vrijednosti rješenja, a na ordinati se nalazi broj iteracije. Plava krivulja predstavlja najbolja mravlja rješenja, a crvena krivulja predstavlja najbolja rješenja lokalne pretrage. Iz plave krivulje na grafu možemo razaznati poželjno ponašanje mravljeg algoritma. Algoritam postepeno, uz povremena lošija rješenja (što je isto poželjno), teži k nekim boljim rješenjima i postepeno dolazi do nekog lokalnog optimuma. Nakon unaprijed određenog broja iteracija, vrijednosti rješenja se naglo povećavaju zbog ponovne inicijalizacije feromonskih tragova $\mathcal{MAX}-\mathcal{MIN}$ algoritma.

Pokusi vezani uz algoritam optimizacije mravljom kolonijom su prikazani u tablici 6.1.

Tablica 6.1: Parametri za pokuse s $\mathcal{MAX}-\mathcal{MIN}$

Test br.	α	β	ρ	m	n
1.1	2	1	0.3	10	250
1.2	3	1	0.3	10	250
1.3	3	2	0.3	10	250
1.4	5	1	0.3	10	250
1.5	10	1	0.3	10	250
1.6	1	5	0.3	10	250
2.1	2	1	0.1	10	250
2.2	2	1	0.3	10	250
2.3	2	1	0.5	10	250
2.4	2	1	0.7	10	250
3.1	2	1	0.3	5	250
3.2	2	1	0.3	10	250
3.3	2	1	0.3	50	250
3.4	2	1	0.3	100	250

Algoritam pretrage promjenjivih susjedstva nema puno parametara koje je potrebno ispitati. Za potrebe testiranja odabrana je inačica *osnovne pretrage promjenjivih susjedstva*, BVNS. Za odabrani algoritam lokalne pretrage moguće je odabrati tri parametra:

- korištena susjedstva i njihov poredak,
- veličina susjedstva i
- način odabira novog rješenja.

Odabir dobrih susjedstva je puno važniji od njihova poretka. Za vrijeme implementacije te postepenog testiranja i dodavanja novih susjedstva bilo je očigledno kako se poboljšava rezultat čak i za loše vrijednosti parametara. Ovo se događalo jer se širio prostor rješenja koja su nam dohvatljiva kroz pretragu susjedstva. Ukoliko su susjedstva dobro odabrana, biti će obuhvaćen cijeli prostor rješenja, ali to nije moguće garantirati. Ovdje postaje važan odabir redoslijeda susjedstva. U [10] je preporučeno da susjedstva budu ugniježđena. Ovime se može postići redoslijed koraka kojima će ipak i globalni optimum biti uključen u neko od susjedstva. Pošto se s većim brojem[1]

[1]Ukoliko se zadržimo unutar nekog razumnog broja susjedstva.

susjedstva gotovo sigurno poboljšava konačno rješenje, korištena su sva implementirana susjedstva poredana na sljedeći način:

1. Susjedstvo slučajne zamjene predavanja u rasporedu razreda,
2. susjedstvo slučajne zamjene predavanja u rasporedu profesora,
3. susjedstvo kontinuiranog rasporeda za razrede,
4. susjedstvo kontinuiranog rasporeda za profesore,
5. susjedstvo previše ili premalo radnih sati za razrede,
6. susjedstvo previše ili premalo radnih sati za profesore,
7. susjedstvo više istih predmeta u danu i
8. susjedstvo optimizacije prostorija.

Iz ovog poretka možemo vidjeti da prva dva susjedstva rade nasumične pomake i tako obuhvaćaju vrlo velik, ako ne i cijeli prostor rješenja čime su ostala susjedstva ugniježđena u njih. Redoslijed ostalih susjedstva je takav da popravljaju neki prekršaj za koji su zadužena, ali da slijedeće susjedstvo ne pokvari ono što je prethodno susjedstvo popravilo. To nije uvijek moguće tako da je iznimka *susjedstvo više istih predmeta u danu* koje može pokvariti neke od prethodnih optimizacija jer se ovaj prekršaj pokazao kao posebno velik, tako da je ovo uključeno kao pretposljednje susjedstvo. Poredak susjedstva također može biti važan za brži rad algoritma, jer će se time smanjiti broj prelazaka u novo susjedstvo, a time i broj novih susjeda koje je potrebno stvoriti i vrednovati.

Veličina susjedstva nema preporučenu vrijednost i potrebno ju je eksperimentalno odrediti jer ovisi o problemu koji se pokušava riješiti. Ovo će biti još jedan parametar koji ćemo pokusima pokušati optimizirati.

Novo rješenje se u lokalnoj pretrazi može odabrati na jedan od tri spomenuta načina:

- slučajno poboljšanje,
- prvo poboljšanje i
- najbolje poboljšanje.

Pošto se generiranje novih susjeda i njihovo vrednovanje prvotnim testovima pokazalo kao relativno brz i jednostavan postupak, ovaj parametar će biti fiksiran na *najbolje*

poboljšanje. Kratkoročno, ovaj odabir će sigurno ponuditi bolja rješenja, ali u konačnici može prouzrokovati vrlo skoru stagnaciju algoritma u lokalnom optimumu jer algoritam postaje pohlepan. Rezultati su pokazali da ovo nije slučaj.

Pokus vezan uz algoritam pretrage promjenjivih susjedstva je prikazan u tablici 6.2.

Tablica 6.2: Parametri za pokuse s VNS

Test br.	α	β	ρ	m	n
4.1	2	1	0.3	10	50
4.2	2	1	0.3	10	150
4.3	2	1	0.3	10	250
4.4	2	1	0.3	10	500
4.5	2	1	0.3	10	1000

Za vrednovanje rješenja korišteni su svi implementirani procjenitelji. Za svakog procjenitelja korištene su težinske vrijednosti koje su prikazane u tablici 6.3.

Tablica 6.3: Težinske vrijednosti procjenitelja

Procjenitelj	Težinska vrijednost
Brojač nepoštivanja dostupnosti razreda	50
Brojač nepoštivanja dostupnosti predavanja	50
Brojač nepoštivanja dostupnosti učionica	50
Brojač nepoštivanja dostupnosti nastavnika	50
Brojač nepoštivanja veza između predavanja	30
Brojač nepoštivanja neprekidnog rasporeda za razrede	35
Brojač nepoštivanja neprekidnog rasporeda za nastavnike	35
Brojač nepoštivanja radnih sati (u danu) za razrede	8
Brojač nepoštivanja radnih sati (u danu i tjednu) za nastavnike	8
Brojač predavanja raspoređenih u pogrešan tjedan	50
Brojač predavanja raspoređenih u nepoželjnu učionicu	10
Brojač blok predavanja s preljevom u drugi dan	100
Brojač više predavanja istog predmeta u istom danu	25

6.2. Način testiranja

Svaki od četiri vrste pokusa je imao skup vrijednosti koje je potrebno ispitati. Pokusi su izvođeni slijedno jer su najbolje vrijednosti iz prethodnog pokusa prenesene u idući pokus. Pokus je za svaku vrijednost ponovljen pet puta i trajao je najviše sat vremena ili 1000 iteracija nakon čega je prekinut. Jedna iteracija predstavlja konstrukciju jednog rješenja[2] (rasporeda).

Prvotni testovi kojima se utvrdilo okvirne vrijednosti za parametre i provjera stabilnosti *Sustava* su provedeni na računalu s procesorom *Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz* koji ima 8 jezgri i 12 GB radne memorije. Testovi čiji su rezultati prikazani u sljedećem odjeljku su provedeni na računalu s procesorom *Intel(R) Core(TM) i7 CPU 920 @ 2.67GHz* koji ima 8 jezgri i 6 GB radne memorije. Na testnom računalu se u vrijeme izvođenja pokusa nije izvodio niti jedan drugi zahtjevniji proces.

6.3. Rezultati

Iako je prethodno u tekstu rečeno da konačni raspored ne smije kršiti nijedno strogo ograničenje, poput dostupnosti resursa, u konačnici se to pokazalo kao prestrogo pravilo. Odluka o valjanosti konačnog rasporeda je prepuštena korisniku s time da su mu na uvid dani svi prekršaji koje dani raspored sadrži.

Najbolji rezultati za pojedini pokus su dani u tablici 6.4 i poredani su prema najboljem rješenju. Prosječan broj iteracija nam može poslužiti kao orijentir koliko je neki test bio zahtjevan. Pregledom tablice sa svim rezultatima C.1 možemo vidjeti kako su neki testovi iz prvog pokusa bili iznimno složeni jer je broj iteracija vrlo malen. Prosjek rješenja predstavlja aritmetičku sredinu ponavljanja.

Tablica 6.4: Najbolji rezultati pojedinog pokusa

Test	Min VNS	Medijan	Prosjek	Std. devijacija	Br. iteracija
3-3	7828	8647.205042	8086	627.70961	595
4-4	8584	9603.615108	9323.5	500.124231	556
2-4	9388	10419.16346	10174.5	259.2232484	520
1-1	9688	11053.20438	10627	611.856856	274

Iznimku od načina testiranje koja kaže da se parametri najboljeg rezultata prenose u sljedeći pokus se javlja kod drugog pokusa gdje su preneseni parametri drugog naj-

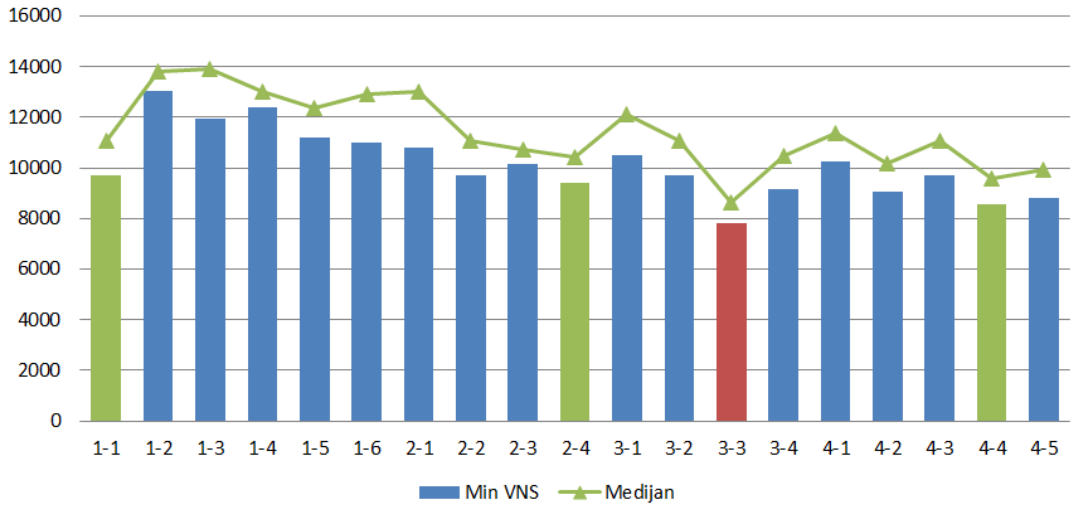
[2]Rješenja stvorena u okviru lokalne pretrage se ovdje ne ubrajaju.

boljeg rezultata. Razlog su bili vrlo loši rezultati trećeg pokusa. Mravlji algoritam je usprkos dobrim rezultatima lokalne pretrage davao lošija rješenja nego u drugom pokusu. Koeficijent isparavanja feromonskih tragova je očigledno bio previsok i mravi nisu mogli stabilno konvergirati k boljim rješenjima, već su počeli nasumce "lutati" po prostoru rješenja. Rezultati drugog pokusa, zajedno s parametrima su dani u tablici 6.4.

Tablica 6.5: Rezultati drugog pokusa

Test	α	β	ρ	m	n	Min VNS	Medijan	Prosjek	Std. devijacija
2-1	2	1	0.1	10	250	10817	13000.75	12956	664.920382
2-2	2	1	0.3	10	250	9688	11053.20438	10627	611.856856
2-3	2	1	0.5	10	250	10140	10749.03865	10328	362.3779468
2-4	**2**	**1**	**0.7**	**10**	**250**	**9388**	**10419.16346**	**10174.5**	**259.2232484**

Rezultati svih pokusa su dani za usporednu analizu na slici 6.2. Apscisa prikazuje vrijednosti rješenja, a pojedini stupac predstavlja pojedini test čije je ime navedeno na ordinati. Zeleni stupci predstavljaju najbolje rješenje za pokus, a crveni stupac označava najbolje rješenje svih testova. Zelena linija predstavlja medijan testova. Iz grafa je vidljiv očigledan pad vrijednosti rješenja iz pokusa u pokus.



Slika 6.2: Najbolja rješenja i medijani pokusa

U tablici 6.6 je dana usporedba najboljeg dobivenog rješenja u sklopu ovog rada, rješenja iz [12] i stvarnog rasporeda korištenog u *OŠ Voltino* tokom školske godine 2011/2012. Rezultati su zadovoljavajući, ali nažalost još uvijek nedovoljno dobri za korištenje u školi. Mogu poslužiti kao početna točka za izradu rasporeda, ali nažalost

još uvijek vrijedi da čovjek može napraviti bolji raspored od računala. Usporedbom rješenja dobivenog u [12] pomoću pomoću genetskog algoritma i lokalne pretrage te rezultata dobivenog u ovom radu dolazimo do zaključka da algoritmi imaju poteškoća u istim područjima. Dodavanjem dodatnih susjedstva i daljnjom optimizacijom parametara bilo bi moguće dostići razinu rješenja genetskog algoritma.

Tablica 6.6: Usporedba rješenja

Vrsta prekršaja	Rješenja		
	Mravlje	Genetsko	Stvarno
Broj nepoštivanja dostupnosti razreda	3	3	0
Broj nepoštivanja dostupnosti predavanja	2	2	0
Broj nepoštivanja dostupnosti učionica	0	0	0
Broj nepoštivanja dostupnosti nastavnika	1	1	0
Broj nepoštivanih veza između predavanja	20	19	6
Broj nepoštivanja neprekidnosti rasporeda, razredi	20	2	0
Broj nepoštivanja neprekidnosti rasporeda, nastavnici	34	10	26
Broj nepoštivanja broja radnih sati za nastavnike	60	32	0
Broj nepoštivanja broja radnih sati za razrede	1	7	1
Broj predavanja raspoređenih u pogrešan tjedan	0	0	0
Broj predavanja u manje poželjnim učionicama	110	70	110
Broj predavanja s preljevom u drugi dan	0	0	0
Broj više predavanja istog predmeta u istom danu	150	139	97
Težinski zbroj	7828	4777	4623

7. Zaključak

U ovom radu se bavimo problemom izrade rasporeda za nastavu u osnovnim i srednjim školama u Republici Hrvatskoj. Riječ je o procesu koji do sada nije bio automatiziran zbog nedostatka prikladnog sustava koji bi zadovoljavao sve potrebe hrvatskih škola. Satničari pet osnovnih i srednjih škola su izašli u susret i iznijeli svoje zahtjeve i dosadašnje metode izrade rasporeda. Pri izradi ovog rada je stečeno znanje uzeto u obzir i dijelom implementirano u konačno rješenje.

Na početku rada opisan je problem izrade rasporeda s kojim se susrećemo i ponuđen pristup za njegovo rješavanje. Opisano je pet inačica algoritma za optimizaciju mravljom kolonijom, kao i pet inačica algoritma za pretragu promjenjivih susjedstva. Pri opisivanju pojedinog algoritma opisan je osnovni algoritam i iznesen prijedlog njegove prilagodbe na problem izrade rasporeda. Kao dio rada izrađen je i novi *Sustav* za izradu rasporeda. Implementirano rješenje koristi algoritam optimizacije mravljom kolonijom za izradu početnih rješenja koja se zatim lokalnom pretragom popravljaju. Lokalna pretraga se obavlja algoritmom pretrage promjenjivih susjedstva. Implementirane su tri inačice algoritma optimizacije mravljom kolonijom i dvije inačice algoritma pretrage promjenjivih susjedstva.

Implementirani *Sustav* je testiran i izneseni su rezultati. Za testiranje je korišten $\mathcal{MAX}-\mathcal{MIN}$*mravlji sustav* i algoritam *osnovne pretrage promjenjivih susjedstva*. Testiranjem su utvrđene optimalne vrijednosti odabranih parametara za implementirane algoritme. Njihova anliza je pokazala zadovoljavajuće rezultate i smjernice za daljnji rad.

Za vrijeme izrade implementacije bilo je puno pokušaja implementacije različitih inačica algoritama, susjedstva, optimizacija i slično. Neki pokušaji poput pamćenja broja neuspjelih dodjela termina predavanju i pamćenja izračunatih vrijednosti dostupnosti su postali dijelom konačnog *Sustava*. Ostali pokušaji su ostali čekati daljnje dorade i razvoj. U njih možemo nabrojati sljedeće stavke.

- Pametni mrav koji bira među samo najboljih 20% mogućih koraka.
- Implementacija liste već posjećenih rješenja (tabu liste) u $\mathcal{MAX}-\mathcal{MIN}$*mravlji*

sustav.

- SVNS inačica pretrage promjenjivih susjedstva.
- Susjedstvo za optimizaciju veza između predavanja.
- Koristiti potpuno slučajno rješenje iz nekog susjedstva za bijeg iz lokalnog optimuma u algoritmu pretrage promjenjivih susjedstva.

Kao sljedeće korake bih spomenuo tri stavke:

1. daljnja optimizacija postojećeg rješenja i dodavanje novih susjedstva,
2. spajanje postojećih sustava u jedan veliki i
3. distribuiranje rada sustava.

Trenutnu implementaciju je moguće uz manje nadogradnje dodatno pospješiti i dobiti bolja rješenja. Za to je dovoljno dodati nova susjedstva i provesti dodatne testove koji će pokazati optimalne vrijednosti parametara koji nisu pokriveni ovim radom. Spajanjem postojećih implementacija da rade zajedno ili jednostavno korištenjem pojedinih dijelova postojećih sustava bilo bi moguće stvoriti novi sustav koji bi mogao dati nova rješenja. Osim navedenoga, još jedna nadogradnja bi bila distribucija sustava. Testiranja su provedena na novim računalima s procesorima koji su brži i imaju više jezgri nego prosječna računala koja možemo pronaći u školama. Pošto je *Sustav* namijenjen školama, a današnje prosječno računalo u školi ima najviše dvije jezgre i neusporedivo sporiji i stariji procesor, ovo je realan problem.

Nažalost pokazalo se točno da računalo ne može sastaviti bolji raspored od čovjeka, barem zasad. Satničar ipak poznaje školu, ljude koji u njoj rade i najvažnije, ima iskustva. Razgovorom sa satničarima pokušalo se prenijeti to iskustvo na sustav, koristiti iste ili slične metode, ali to nije dovoljno.

LITERATURA

[1] T. Bronić. Diplomski rad. Diplomski rad, Fakultet elektrotehnike i računarstva, 2012.

[2] B. Bullnheimer, R.F. Hartl, i C. Strauss. A new rank based version of the ant system. a computational study. 1997.

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

[4] J.L. Deneubourg, S. Aron, S. Goss, i J.M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 1990.

[5] M. Dorigo i L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[6] M. Dorigo i T. Stützle. *Ant Colony Optimization*. Bradford Books. MIT Press, 2004. ISBN 9780262042192.

[7] 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.

[8] P. Hansen i N. Mladenović. Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449–467, 2001.

[9] P. Hansen, N. Mladenovic, i Québec) Groupe d'études et de recherche en analyse des décisions (Montréal. A tutorial on variable neighborhood search, 2003.

[10] N. Mladenovic i P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[11] S. Pribil. Izrada rasporeda nastave za osnovne i srednje škole. Diplomski seminar, Fakultet elektrotehnike i računarstva, 2011.

[12] S. Pribil. Diplomski rad. Diplomski rad, Fakultet elektrotehnike i računarstva, 2012.

[13] C. Schauer i B. Hu. Heuristic optimisation techniques. A lecture held on University of Technology Vienna WS 2011.

[14] A. Tus. Analiza zahtjeva i postojedih rješenja u izradi rasporeda sati za škole. Diplomski seminar, Fakultet elektrotehnike i računarstva, 2011.

Dodatak A
Izvorni kodovi

U nastavku se nalaze izvorni kodovi implementacija nekoliko najvažnijih funkcija i algoritama.

Izvorni kod main metode

```
1  public class Main {
2
3  private static String paramsFile = "res/voltinoParamsFix4.xml";
4
5  private static int alpha = 3;
6  private static int beta = 1;
7  private static int antCount = 20;
8
9  private static double pheroEvaporationRate = 0.6;
10
11 private static double elitistValue = 10;
12
13 private static int iterationsToReset = 100;
14 private static double useBestProbability = 0.0;
15
16 private static int iterations = 1000;
17
18 private static int onScreenInterval = 1;
19 private static int progressInterval = 1;
20 private static int exportInterval = 1;
21
22 private static String progressFolder = "progress";
23 private static String progressFile = "schedulingProgress.txt";
24
25 private static double optimizationPercent = 0.5;
26 private static int neighborhoodSize = 150;
27
```

```
28 private static double notRecommendedValue = 0.4;
29
30 private static int tabuListLength = 50;
31
32
33 public static void main(String[] args) throws FileNotFoundException,
       ParamsConvertingException,
34         InputFileLoadingException, IllegalParametersStateException {
35
36     final AcoSchedulingContext schedulingContext = new
           AcoSchedulingContext();
37
38     IParamsHandler paramsLoader = new ParamsXMLHandler();
39     schedulingContext.setParams(paramsLoader.fromExternal(new
           FileInputStream(paramsFile)));
40     schedulingContext.getParamsValidator().setValidators(new
           IValidator[] { new SlotCounter() });
41
42     // EVALUATORS
43     EvaluationAccumulator availabilitiesAccumulator = new
           EvaluationAccumulator("Availabilities␣accumulator",
44             new AbstractEvaluator[] { new
                   TeacherAvailabilitiesViolationCounter(),
45                     new ClassAvailabilitiesViolationCounter(), new
                           RoomAvailabilitiesViolationCounter(),
46                     new LectureAvailabilitiesViolationCounter() });
47
48     EvaluationAccumulator periodCountAccumulator = new
           EvaluationAccumulator("Period␣count␣accumulator",
49             new AbstractEvaluator[] { new
                   TeacherWorkingPeriodsViolationCounter(),
50                     new ClassWorkingPeriodsViolationCounter() });
51
52     double[] weightVector = new double[] { 10, 100, 50, 30, 50, 35,
           35, 25, 8 };
53     AbstractEvaluator[] finalAccomulator = { new
           EvaluationAccumulator("Final␣accumulator",
54             new AbstractEvaluator[] { new
                   IllegalLectureLocationCounter(), new
                   DaySplittedLectureCounter(),
55                     new IllegalLectureWeekCounter(), new
                           ConnectionViolationCounter(),
                           availabilitiesAccumulator,
```

```
56                      new ClassContinuousScheduleViolationCounter(),
                            new TeacherContinuousScheduleViolationCounter
                            (),
57                      new MultipleLecturesOfSameSubjectInDayCounter(),
                            periodCountAccumulator }, weightVector) };
58      schedulingContext.getScheduleEvaluator().setEvaluators(
            finalAccomulator);
59
60      // SCHEDULE FACTORY
61      schedulingContext.getScheduleFactory().setCreator(new
            IScheduleCreator() {
62          @Override
63          public ISchedule getNewSchedule() {
64              return new SmarterSchedule(schedulingContext);
65          }
66      });
67
68      // ALGORITHM CONTEXT
69      AcoAlgorithmContext algorithmContext = new AcoAlgorithmContext()
            ;
70
71      // ANT COLONY OPTIMISATION
72      algorithmContext.setHeuristicAlgorithms(new
            AbstractHeuristicAlgorithm[] { new SimpleHeuristicAlgorithm()
            });
73  //      algorithmContext.setMetaheuristicAlgorithm(new AntSystem(
        antCount, alpha, beta, pheroEvaporationRate));
74  //      algorithmContext.setMetaheuristicAlgorithm(new
        ElitistAntSystem(antCount, alpha, beta, pheroEvaporationRate,
        elitistValue));
75      algorithmContext.setMetaheuristicAlgorithm(new MaxMinAntSystem(
            antCount, alpha, beta, pheroEvaporationRate,
76              iterationsToReset, useBestProbability));
77  //      algorithmContext.setMetaheuristicAlgorithm(new
        MaxMinTabuAntSystem(antCount, alpha, beta, pheroEvaporationRate,
78  //              iterationsToReset, useBestProbability,
        tabuListLength));
79      algorithmContext.setAvailabilityCalculator(new
            CachingAvailabilityCalculator(schedulingContext,
            notRecommendedValue ));
80
81      // OPTIMISATION
```

```
82      algorithmContext.getOptimizationManager().
           setInvokeOptimizationPercent(optimizationPercent);
83      algorithmContext.getOptimizationManager().setWorkers(
84              new AbstractOptimizationWorker[] { new
                   AdvancedAvailabilitiesOptimizaton() });
85
86      // LOCAL SEARCH
87      algorithmContext.getVnsManager().setNeighborhoodGenerators(
88              new AbstractNeighborGeneratorWorker[] {
89                      new TeacherLectureSwapNeighborhood(0.5),
90                      new TeacherContinuousScheduleNeighborhood(),
91                      new ClassContinuousScheduleNeighborhood(),
92                      new MultipleLecturesOfSameSubjectNeighborhood(),
93
94              });
95      algorithmContext.getVnsManager().setWorkers(new
           AbstractLocalSearchWorker[] { new RVNS(neighborhoodSize) });
96
97      schedulingContext.setAlgorithmContext(algorithmContext);
98      schedulingContext.addProgressListener(new ProgressListener(
           progressFolder, progressFile, progressInterval,
99              exportInterval, schedulingContext), onScreenInterval);
100     schedulingContext.startScheduling(iterations);
101
102 }
```

Izvorni kod algoritma optimizacije mravljom kolonijom

```
1  public abstract class AbstractAntSystem extends
       AbstractMetaheuristicAlgorithm<AcoAlgorithmContext> implements
       ResultListener {
2
3
4       private double alpha;
5       private double beta;
6       private int antCount;
7
8       private AbstractAnt[] ants;
9
10      protected AbstractPheromoneModel pheromoneModel;
11
12      private boolean isFinished = false;
13
```

```
14      public AbstractAntSystem(int antCount, double alpha, double beta
            , AbstractPheromoneModel pheromoneModel) {
15          this.pheromoneModel = pheromoneModel;
16          this.antCount = antCount;
17          this.alpha = alpha;
18          this.beta = beta;
19      }
20
21      @Override
22      public void initialize() {
23          pheromoneModel.setAlgorithmContext(algorithmContext);
24          pheromoneModel.initialize();
25
26          this.ants = new AbstractAnt[antCount];
27          for (int i = 0; i < antCount; i++)
28              ants[i] = new Ant(alpha, beta, pheromoneModel,
                    algorithmContext);
29      }
30
31      @Override
32      public void step() {
33          System.out.print("Constructing...␣");
34          constructAntSolutions();
35
36          if(isFinished)
37              return;
38
39          daemonActions();
40
41          updatePheromones();
42      }
43
44      protected abstract void constructAntSolutions();
45
46      protected abstract void updatePheromones();
47
48      protected abstract void daemonActions();
49
50      @Override
51      public void generateInitialPopulation() {
52          // Nothing to do here
53      }
54
```

```
55      public int getAntCount() {
56          return antCount;
57      }
58
59      public AbstractAnt[] getAnts() {
60          return ants;
61      }
62
63      public void setFinished(boolean isFinished) {
64          this.isFinished = isFinished;
65      }
66
67      @Override
68      public boolean isFinished() {
69          return isFinished ;
70      }
71
72  }
```

Izvorni kod BVNS algoritma

```
1  public Solution improveSolution(Solution initialSolution) {
2      double bestSolutionValue = Double.MAX_VALUE;
3      Solution bestSolution = initialSolution;
4
5      AbstractNeighborGeneratorWorker[] neighborhoods =
           algorithmContext.getVnsManager().getNeighborhoodGenerators();
6      MultiThreadEvaluator2 evaluator = algorithmContext.
           getSchedulingContext().getMultiThreadEvaluator();
7
8      int k = 0;
9      while (k < neighborhoods.length) {
10         List<ISchedule> newNeighbors = new ArrayList<ISchedule>();
11         do {
12             ISchedule newNeighbor = bestSolution.getSchedule().clone
                   ();
13             if (neighborhoods[k].generateNewNeighbor(newNeighbor))
14                 newNeighbors.add(newNeighbor);
15         } while (newNeighbors.size() < neighborhoodSize);
16
17         Solution[] solutions = evaluator.evaluatePopulation(
               newNeighbors.toArray(new ISchedule[newNeighbors.size()]))
               ;
18
```

```
19          boolean improvement = false;
20          for (Solution solution : solutions) {
21              double fitness = solution.getFitness().getValue(0).
                    getResult();
22              if (fitness < bestSolutionValue) {
23                  bestSolutionValue = fitness;
24                  bestSolution = solution;
25                  improvement = true;
26              }
27          }
28          if (!improvement)
29              k++;
30          else
31              k = 0;
32      }
33      algorithmContext.getOptimizationManager().performOptimization(
            bestSolution.getSchedule());
34      return bestSolution;
35  }
```

Dodatak B
Primjer ulazne i izlazne datoteke

U nastavku se nalazi skraćeni primjer ulazne i izlazne datoteke.

Primjer ulazne XML datoteke

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <xmlScheduleParams xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance">
3      <weekCount>2</weekCount>
4      <dayCount>5</dayCount>
5      <avoidingLargeBlocksEnabled>false</avoidingLargeBlocksEnabled>
6      <largeBlockSize>2</largeBlockSize>
7      <roomSizeManagementEnabled>false</roomSizeManagementEnabled>
8      <periods>
9          <period id="j1">08:00 - 08:45</period>
10         ...
11     </periods>
12     <turnuses>
13         <turnus id="jut">
14             <name>Jutarnji turnus</name>
15             <periods>j1,j2,j3,j4,j5,j6</periods>
16             <classStartPeriods>j1,j2</classStartPeriods>
17         </turnus>
18         ...
19     </turnuses>
20     ...
21     <rooms>
22         <room id="hrv1">
23             <name>Hrvatski 1</name>
24             <availabilities>
25                 <weekAvailability>
26                     <periodPreferences>
```

```
                        <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="njemRed">
            <name>Njemacki jezik (redovni)</name>
            <defaultRooms>njem</defaultRooms>
            <availabilities>
                <weekAvailability>
                    <periodPreferences>
                        <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>
                        ...
                    </periodPreferences>
                    <dayPreferences>1,1,1,1,1</dayPreferences>
                </weekAvailability>
                ...
            </availabilities>
```

```
67              <sameGenerationInDayRequired>false</
                    sameGenerationInDayRequired>
68              <sameSubjectInDayRequired>false</
                    sameSubjectInDayRequired>
69              <dayTurnusAlternationEnabled>false</
                    dayTurnusAlternationEnabled>
70              <daysForWeekTurnusAlternation>false,false,false,false,
                    true</daysForWeekTurnusAlternation>
71              <workingPeriodNumber>2 - 7</workingPeriodNumber>
72              <mandatoryBreakThreshold>7</mandatoryBreakThreshold>
73              <dayBreakNumber>0 - 1</dayBreakNumber>
74              <weekBreakNumber>1 - 2</weekBreakNumber>
75              <maximumBreakSize>1</maximumBreakSize>
76          </teacher>
77          ...
78      </teachers>
79      <classes>
80          <class id="4b">
81              <name>4.b</name>
82              <availabilities>
83                  <weekAvailability>
84                      <periodPreferences>
85                          <preference>1,1,1,1,1,1,1,1,1,1,1,1,1</
                                preference>
86                          ...
87                      </periodPreferences>
88                      <dayPreferences>1,1,1,1,1</dayPreferences>
89                  </weekAvailability>
90                  ...
91              </availabilities>
92              <continuousScheduleRequired>false</
                    continuousScheduleRequired>
93              <workingPeriodNumber>0 - 4</workingPeriodNumber>
94              <mainClassroom>4b</mainClassroom>
95          </class>
96          ...
97      </classes>
98      <lectures>
99          <!-- rpapic, 6b, hrv -->
100         <lecture id="hrv:rpapic:6b:1">
101             <subject>hrv</subject>
102             <teachers>rpapic</teachers>
103             <classes>6b</classes>
```

```
104             <classrooms>hrv1,hrv2,eng,povGeo,mat,njem,teh,bioKem,fiz
                    </classrooms>
105             <necessaryClassroomCount>1</necessaryClassroomCount>
106             <lectureDuration>2</lectureDuration>
107             <weeks>1</weeks>
108             <availabilities>
109                 <weekAvailability>
110                     <periodPreferences>
111                         <preference>0,0,0,0,0,0,1,1,1,1,1,1,1</
                                preference>
112                         ...
113                     </periodPreferences>
114                     <dayPreferences>1,1,1,1,1</dayPreferences>
115                 </weekAvailability>
116             </availabilities>
117             <regularScheduleManagementEnabled>true</
                    regularScheduleManagementEnabled>
118         </lecture>
119         ...
120     </lectures>
121     <connections>
122         <connection id="5a:lik-teh:1">
123             <lectureIds>lik:ngolubic:5a:1,teh:ssostaric:5a:1</
                    lectureIds>
124             <rule>MUST_NOT_BE</rule>
125             <condition>SAME_WEEK</condition>
126         </connection>
127         ...
128     </connections>
129 </xmlScheduleParams>
```

Primjer izlazne XML datoteke

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <xmlSchedule>
3      <lectures>
4          <lecture>
5              <lectureId>hrv:rpapic:6a:8</lectureId>
6              <slot>80</slot>
7              <locations>hrv2</locations>
8          </lecture>
9          <lecture>
10             <lectureId>engRed:mstrahija:7b:1</lectureId>
11             <slot>1</slot>
```

```
12              <locations>eng</locations>
13          </lecture>
14          <lecture>
15              <lectureId>mat:ggojmerac:8b:1</lectureId>
16              <slot>47</slot>
17              <locations>hrv1</locations>
18          </lecture>
19          ...
20      </lectures>
21  </xmlSchedule>
```

Dodatak C
Rezultati testiranja

Na sljedećoj stranici se nalazi tablica s pregledom svih rezultata.

Tablica C.1: Rezultati analize

Test	α	β	ρ	m	n	Max mrav	Min mrav	Max VNS	Min VNS	Medijan	Prosjek	Std. devijacija	Broj iteracija
1-1	**2**	**1**	**0.3**	**10**	**250**	**44848**	**10099**	**15778**	**9688**	**11053.20438**	**10627**	**611.856856**	**274**
1-2	3	1	0.3	10	250	42415	23633	14966	13040	13835.125	13670	455.2336456	81
1-3	3	2	0.3	10	250	47023	33681	15329	11968	13890.5	14132.5	626.488585	47
1-4	5	1	0.3	10	250	44753	16900	13615	12380	13015	12880	128.6975624	52
1-5	10	1	0.3	10	250	45295	11313	15619	11175	12381.48301	12254	343.1723323	559
1-6	1	5	0.3	10	250	45023	23458	15146	10999	12917.23944	12980	382.9239861	71
2-1	2	1	0.1	10	250	44289	21840	16206	10817	13000.75	12956	664.920382	52
2-2	2	1	0.3	10	250	44848	10099	15778	9688	11053.20438	10627	611.856856	274
2-3	2	1	0.5	10	250	46499	10142	17622	10140	10749.03865	10328	362.3779468	828
2-4	**2**	**1**	**0.7**	**10**	**250**	**46729**	**9408**	**15717**	**9388**	**10419.16346**	**10174.5**	**259.2232484**	**520**
3-1	2	1	0.3	5	250	47846	10517	16547	10507	12137.72646	11653	653.334684	223
3-2	2	1	0.3	10	250	44848	10099	15778	9688	11053.20438	10627	611.856856	274
3-3	**2**	**1**	**0.3**	**50**	**250**	**42733**	**7870**	**14724**	**7828**	**8647.205042**	**8086**	**627.70961**	**595**
3-4	2	1	0.3	100	250	42839	9183	15488	9163	10459.14844	9915	714.317397	256
4-1	2	1	0.3	50	50	46352	10267	16235	10252	11348.79455	10782.5	734.839897	404
4-2	2	1	0.3	50	150	43694	9285	14899	9055	10163.54042	10079	315.0389754	668
4-3	2	1	0.3	50	250	44848	10099	15778	9688	11053.20438	10627	611.856856	274
4-4	**2**	**1**	**0.3**	**50**	**500**	**46338**	**8670**	**16947**	**8584**	**9603.615108**	**9323.5**	**500.124231**	**556**
4-5	2	1	0.3	50	1000	41618	8874	14391	8834	9954.63	9661	482.684967	400

Heurističke metode lokalne pretrage primijenjene na problem izrade rasporeda sati za škole

Sažetak

U ovom radu je predložen pristup rješavanju problema automatizacije izrade rasporeda u hrvatskim osnovnim i srednjim školama. Pristup se sastoji od spoja algoritma optimizacije mravljom kolonijom i algoritma lokalne pretrage promjenjivih susjedstva. Opisani su različiti oblici spomenutih algoritama i način njihove primjene na problem izrade školskog rasporeda. U sklopu rada je izrađen i *Sustav* koji predstavlja implementaciju predloženog rješenja te je ispitana njegova efikasnost. Rezultati su opisani na kraju rada.

Ključne riječi: izrada rasporeda sati, školski raspored, optimizacija kolonijom mrava, pretraga po promjenjivim susjedstvima, lokalna pretraga, ACO, VNS

Heuristic local-search methods applied on school timetabling

Abstract

This thesis offers a solution for the problem of automating the creation of school timetables in Croatian elementary and high schools. The offered approach is a combination of ant colony optimization and variable neighborhood search. Different types of the mentioned algorithms and a possible approach to applying them to the given problem is described in the thesis. Also, a software system was built to support the claims stated in this thesis. In the end tests were carried out to test the performance and the results are provided.

Keywords: scheduling, timetabling, school timetable, ant colony optimization, variable neighborhood search, local search, ACO, VNS