UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Dušan Trtica

PRIMENA TEHNIKA ZADOVOLJENJA OGRANIČENJA I CELOBROJNOG PROGRAMIRANJA NA PROBLEM GENERISANJA RASPOREDA ČASOVA

master rad

Beograd, 2026.

Mentor:

dr Filip Marić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Aleksandar Kartelj, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Miljan Knežević, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mojoj porodici

Naslov master rada: Primena tehnika zadovoljenja ograničenja i celobrojnog programiranja na problem generisanja rasporeda časova

Rezime: U ovom radu razmatramo problem automatskog generisanja rasporeda časova na univerzitetskom nivou. Prikazujemo dve paradigme za rešavanje ovog problema: programiranje ograničenja (CP) i mešovito celobrojno programiranje (MIP). Oba pristupa su implementirana korišćenjem biblioteke Google OR-Tools i primenjena na podatke Matematičkog fakulteta Univerziteta u Beogradu. Sprovedena je eksperimentalna evaluacija na tri skale problema različite veličine. Rezultati pokazuju da CP-SAT rešavač značajno nadmašuje MIP/SCIP pristup u pogledu vremena rešavanja, kompaktnosti modela i memorijskog otiska, posebno kako veličina problema raste.

Ključne reči: raspored časova, programiranje ograničenja, celobrojno programiranje, CP-SAT, MIP, OR-Tools, optimizacija, Matematički fakultet

Sadržaj

Glava 1

Uvod

Generisanje rasporeda časova predstavlja jedan od najčešćih i najpraktičnijih kombinatornih problema sa kojim se suočavaju obrazovne institucije širom sveta. Svake godine, na početku akademskog semestra, univerziteti i fakulteti moraju da rasporede stotine predavanja, vežbi i laboratorijskih časova u ograničen broj učionica, uzimajući u obzir radne dane, raspoložive termine i različite vrste ograničenja.

Problem raspoređivanja časova (eng. *University Course Timetabling Problem* – UCTP) jeste NP-težak problem [8, 9], što znači da ne postoji poznat algoritam koji bi ga efikasno rešio u opštem slučaju. Tradicionalno, rasporedi se prave ručno, što zahteva veliku količinu ljudskog rada, vremena i stručnosti. Čak i iskusni raspoređivači mogu da propuste kolizije ili da stvore neoptimalan raspored.

U ovom radu istražujemo dva pristupa za automatsko rešavanje UCTP-a:

1. **Programiranje ograničenja** (eng. *Constraint Programming* – CP) – deklarativna paradigma u kojoj se problem definiše preko skupa promenljivih, njihovih domena i ograničenja koja moraju biti zadovoljena. Koristimo Google OR-Tools CP-SAT rešavač [10, 13].

2. **Mešovito celobrojno programiranje** (eng. *Mixed Integer Programming* – MIP) – matematička optimizacija sa binarnim promenljivama i linearnim ograničenjima. Koristimo SCIP rešavač [1] integrisan u OR-Tools.

Pored ove dve paradigme, u literaturi su na UCTP-u uspešno primenjeni i pristupi zasnovani na direktnom kodiranju u propozicionalnu logiku, pa rešavanju SAT, odnosno MaxSAT rešavačem [3]. Taj smer je srodan našem prvom pristupu, jer je i CP-SAT rešavač koji koristimo interno zasnovan na SAT tehnologiji (odeljak 2.2).

1.1 Cilj rada

Primarni cilj ovog rada je komparativna analiza CP i MIP paradigmi na problem generisanja rasporeda časova. Kroz eksperimentalnu evaluaciju na tri skale problema različite veličine (mali, srednji i veliki podskup realnih podataka), ispitujemo:

- brzinu konstrukcije modela i rešavanja,
- kompaktnost formulacije (broj promenljivih i ograničenja),
- memorijski otisak,
- skalabilnost pristupa pri rastu veličine problema.

Na osnovu rezultata ove analize, u narednoj fazi rada planira se nadogradnja pobedničke paradigme dodatnim ograničenjima (raspoređivanje nastavnog osoblja, lokacijska ograničenja, meka ograničenja za kvalitetniji raspored).

1.2 Struktura rada

Rad je organizovan na sledeći način:

- U glavi 2 izlažemo teorijske osnove: definiciju problema raspoređivanja časova, paradigmu programiranja ograničenja i paradigmu celobrojnog linearnog programiranja.
- U glavi 3 opisujemo korišćene alate i tehnologije: biblioteku Google OR-Tools, sistem za gradnju Bazel i pomoćne Python biblioteke.
- U glavi 4 detaljno izlažemo formulaciju problema, model ulaznih podataka i konkretne implementacije u obe paradigme.
- U glavi 5 predstavljamo eksperimentalnu evaluaciju osnovnog modela: metodologiju merenja, poređenje dve paradigme na tri skale problema i analizu dobijenih rezultata.
- U glavi 6 nadograđujemo izabrani CP model arhitekturom pravila, funkcijom cilja i trima novim ograničenjima kvaliteta rasporeda.
- U glavi 7 model proširujemo nastavnim osobljem: dodelom nastave, zajedničkim sesijama i pravilima koja štite raspored nastavnika.

- U glavi 8 dajemo završnu evaluaciju proširenog modela na tri skale, sa naglaskom na vrednost funkcije cilja u zavisnosti od proteklog vremena izračunavanja.
- U glavi 9 sumiramo zaključke i navodimo pravce daljeg rada.

Glava 2

Teorijske osnove

U ovoj glavi izlažemo teorijski okvir za razumevanje problema i pristupa koje koristimo u ovom radu. Najpre definišemo problem raspoređivanja časova, zatim opisujemo paradigmu programiranja ograničenja, i na kraju predstavljamo celobrojno linearno programiranje.

2.1 Problem raspoređivanja časova

Problem raspoređivanja univerzitetskih časova (UCTP) sastoji se od dodele časova predavanja, vežbi u raspoložive vremenske termine i učionice, tako da budu zadovoljena određena ograničenja [6, 5].

Formalno, problem se može opisati sledećim elementima:

- **Skup sesija** $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ – časovi koje treba rasporediti. Svaka sesija pripada određenom predmetu i grupi studenata.
- **Skup vremenskih termina** $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ – raspoloživi termini u toku radne nedelje, određeni kombinacijom radnog dana i sata.
- **Skup učionica** $\mathcal{R} = \{r_1, r_2, \ldots, r_k\}$ – prostorije u kojima se nastava može održati.
- **Skup ograničenja** $\mathcal{C}$ – pravila koja raspored mora da zadovolji.

Cilj je pronaći funkciju dodele $\sigma : \mathcal{S} \to \mathcal{T} \times \mathcal{R}$ takvu da su sva ograničenja iz $\mathcal{C}$ zadovoljena.

Vrste ograničenja

Ograničenja se u literaturi klasifikuju u dve kategorije [6, 3]:

1. **Tvrda ograničenja** (eng. *hard constraints*) – ograničenja koja moraju biti bezuslovno zadovoljena. Rešenje koje krši neko tvrdo ograničenje nije dopustivo. Primeri:
   - Nikoje dve sesije ne mogu se održavati u istoj učionici u isto vreme.
   - Jedna grupa studenata ne može istovremeno pohađati dva različita predavanja.
   - Sesija koja zahteva računare mora biti raspoređena u učionicu opremljenu računarima.
2. **Meka ograničenja** (eng. *soft constraints*) – ograničenja čije zadovoljenje je poželjno, ali ne i obavezno. Njihovo kršenje ne čini rešenje nedopustivim, već samo lošijim. Primeri:
   - Ravnomerna raspodela časova tokom nedelje.
   - Minimizacija broja promena lokacija za jednu grupu u toku dana.
   - Izbegavanje procepa (eng. *gap*) – slobodnih sati između dva časa – u rasporedu za studente.

U prvom delu ovog rada bavimo se isključivo tvrdim ograničenjima, tj. pronalaženjem dopustivog rasporeda (eng. *feasibility problem*). Proširenje mekim ograničenjima i optimizacija kvaliteta rasporeda predviđeni su za drugi deo rada.

NP-težina problema

Even, Itai i Shamir [8] su dokazali da je problem raspoređivanja NP-težak redukcijom sa problema bojenja grafova. Garey i Johnson [9] su ovaj rezultat uključili u svoju obimnu katalogizaciju NP-kompletnih problema. Ovo znači da je malo verovatno da postoji algoritam polinomijalne složenosti za rešavanje opšteg slučaja problema raspoređivanja, te se u praksi pribegava heuristikama ili egzaktnim metodama sa eksponencijalnom najgorom složenošću.

2.2 Programiranje ograničenja

Programiranje ograničenja (eng. *Constraint Programming* – CP) je deklarativna paradigma za rešavanje kombinatornih problema [15, 2]. Problem se formuliše kao *problem zadovoljenja ograničenja* (eng. *Constraint Satisfaction Problem* – CSP).

Problem zadovoljenja ograničenja (CSP)

CSP se formalno definiše trojkom $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ gde je:

- $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ – skup promenljivih,
- $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$ – skup domena, gde D_i sadrži moguće vrednosti promenljive x_i,
- $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ – skup ograničenja, gde svako ograničenje C_j specificira dopustive kombinacije vrednosti nad podskupom promenljivih.

Rešenje CSP-a je dodela vrednosti svim promenljivama iz njihovih domena tako da su sva ograničenja zadovoljena.

Tehnike rešavanja

CP rešavači kombinuju sistematsku pretragu sa tehnikama propagacije ograničenja [15]:

1. **Vraćanje unazad** (eng. *backtracking*) – pretraga prostora rešenja dodeljujući vrednosti promenljivama jednu po jednu. Kada se naiđe na ograničenje koje ne može biti zadovoljeno, vrši se vraćanje na poslednju tačku izbora.
2. **Propagacija ograničenja** (eng. *constraint propagation*) – smanjivanje domena promenljivih na osnovu ograničenja, pre i tokom pretrage. Ovo eliminiše nedopustive vrednosti rano i dramatično smanjuje prostor pretrage.
3. **Konzistentnost lukova** (eng. *arc consistency*) – tehnika propagacije koja obezbeđuje da za svaku vrednost u domenu jedne promenljive postoji kompatibilna vrednost u domenu svake povezane promenljive.

Globalna ograničenja

Ključna prednost CP-a je podrška za *globalna ograničenja* – specijalizovane strukture koje obuhvataju odnose između većeg broja promenljivih i koje imaju efikasne algoritme propagacije [14].

AllDifferent. Ograničenje `AllDifferent`$(x_1, x_2, \ldots, x_n)$ zahteva da sve promenljive $x_1, \ldots, x_n$ imaju međusobno različite vrednosti. Dok bi se isto moglo izraziti sa $\binom{n}{2}$ parnih nejednakosti, globalno ograničenje pruža značajno jaču propagaciju zahvaljujući algoritmima baziranim na sparivanju u bipartitnim grafovima [14].

Snaga propagacije ograničenja `AllDifferent` zavisi i od toga kako ono komunicira sa ostalim ograničenjima modela. Kada se u istom problemu javljaju i linearna ograničenja nad promenljivama koje su istovremeno obuhvaćene ograničenjem `AllDifferent`, propagacija svakog ograničenja zasebno gubi informaciju koju bi njihova zajednička analiza dala; postoje postupci koji tu vezu eksploatišu i zaključuju jače granice nad domenima [4]. Ovo je relevantno i za naš model, u kome se, uz ograničenja `AllDifferent` iz odeljka 4.3, javljaju i linearni izrazi nad istim vremenskim promenljivama.

AllowedAssignments. Ograničenje `AllowedAssignments` (poznato i kao *table constraint*) eksplicitno navodi dozvoljene kombinacije vrednosti promenljivih pomoću tabele (skupa torki). Ovo je korisno za ograničenja koja se ne mogu izraziti prostom aritmetikom, na primer ograničenje da sesija može biti raspoređena samo u podskup učionica.

Simetrije u prostoru rešenja

Prostor rešenja UCTP-a je izrazito simetričan. Ako jedan predmet ima više časova istog tipa za istu grupu studenata, ti časovi su međusobno *zamenljivi*: svaka njihova permutacija po terminima i učionicama daje raspored koji je za korisnika identičan. Ograničenja tipa `AllDifferent` iz odeljka 4.3 su simetrična po sesijama, pa ih ne razlikuju. Posledica je da svako rešenje ima veliki broj ekvivalentnih kopija, a rešavač ih pri pretrazi obilazi kao da su različite – isto važi i za delove drveta pretrage koji su simetrični već obiđenim delovima.

Standardni odgovor je dodavanje *ograničenja za razbijanje simetrija* (eng. *symmetry breaking constraints*), koja od svake klase ekvivalentnih rešenja dopuštaju samo jednog predstavnika, obično leksikografski najmanjeg. Postoje opšti postupci konstrukcije takvih ograničenja, primenljivi na simetrije nad promenljivama, nad vrednostima i nad oba istovremeno [16].

Kod rešavača sa lenjim generisanjem klauza situacija je nešto složenija, jer učenje konflikata i razbijanje simetrija delimično rade isti posao: naučene klauze same po sebi sprečavaju ponovno obilaženje simetričnih delova pretrage, pa dodatna ograničenja mogu doneti manju korist nego u klasičnom CP-u, ili čak odmoći ako otežaju propagaciju [7].

CP-SAT pristup

Moderna generacija CP rešavača, kakav je Google CP-SAT [13], kombinuje tehnike programiranja ograničenja sa tehnologijama SAT rešavača. Ovaj pristup se zasniva na *lenjom generisanju klauza* (eng. *lazy clause generation*) [12, 7], gde se ograničenja propagiraju kao u klasičnom CP, ali se konflikti pamte u obliku SAT klauza, što sprečava ponavljanje istih pogrešnih izbora tokom pretrage. Ovo kombinuje snagu globalne propagacije sa efikasnošću učenja konflikata poznatom iz SAT rešavača.

2.3 Celobrojno linearno programiranje

Mešovito celobrojno programiranje (eng. *Mixed Integer Programming* – MIP) je paradigma matematičke optimizacije u kojoj se problem formuliše pomoću linearne funkcije cilja i linearnih ograničenja nad promenljivama koje mogu biti celobrojne ili kontinualne [17, 11].

Definicija MIP problema

MIP problem u opštem obliku glasi:

$$\min \quad \mathbf{c}^T\mathbf{x} \tag{2.1}$$

$$\text{p.o.} \quad A\mathbf{x} \leq \mathbf{b} \tag{2.2}$$

$$x_i \in \mathbb{Z}, \quad i \in I \tag{2.3}$$

$$x_j \geq 0, \quad j \notin I \tag{2.4}$$

gde je $\mathbf{c}$ vektor koeficijenata funkcije cilja, A matrica koeficijenata ograničenja, $\mathbf{b}$ vektor desnih strana, a I skup indeksa celobrojnih promenljivih.

Kada su sve celobrojne promenljive binarne ($x_i \in \{0, 1\}$), govori se o *binarnom celobrojnom programu* (BIP). Problem raspoređivanja časova se prirodno formuliše kao BIP.

U našem radu ne koristimo funkciju cilja (jednačina 2.1), već tražimo dopustivo rešenje – bilo koju dodelu vrednosti koja zadovoljava sva linearna ograničenja.

Tehnike rešavanja

MIP rešavači koriste kombinaciju sledećih tehnika [17]:

1. **LP relaksacija** – opuštanje celobrojnih ograničenja i rešavanje kontinualnog linearnog programa. Ovo daje donju granicu za funkciju cilja (ili test nedopustivosti).

2. **Grananje i odsecanje** (eng. *branch and bound*) – sistematska pretraga koja deli prostor rešenja na manje podprobleme, koristeći LP relaksaciju za odsecanje grana koje ne mogu sadržati bolje rešenje.

3. **Ravni odsecanja** (eng. *cutting planes*) – dodavanje linearnih ograničenja koja eliminišu frakciona rešenja LP relaksacije, a ne eliminišu celobrojna rešenja.

SCIP rešavač

SCIP (Solving Constraint Integer Programs) [1] je jedan od najsnažnijih akademskih rešavača za MIP i mešovito celobrojno nelinearno programiranje. SCIP kombinuje tehnike grananja i odsecanja sa propagacijom ograničenja i heuristikama. U našem radu koristimo SCIP integrisanog u Google OR-Tools putem interfejsa `pywraplp`.

Glava 3

Korišćeni alati i tehnologije

U ovoj glavi opisujemo alate i tehnologije korišćene u implementaciji. Projekat je napisan u programskom jeziku Python[1] i koristi Bazel[2] kao sistem za gradnju i upravljanje zavisnostima. Izvorni kod je javno dostupan na adresi `https://github.com/dusantrtica/matf-master`.

3.1 Google OR-Tools

Google OR-Tools [10] je biblioteka otvorenog koda za kombinatornu optimizaciju razvijena od strane Google-a. Nudi rešavače za više tipova problema, uključujući:

- **CP-SAT** – rešavač za programiranje ograničenja sa SAT tehnikama, opisan u odeljku 2.2. U Pythonu se koristi putem modula `ortools.sat.python.cp_model`. Ključne klase su `CpModel` (za definisanje promenljivih i ograničenja) i `CpSolver` (za pokretanje pretrage).
- **Linear solver wrapper (pywraplp)** – uniformni interfejs za više MIP rešavača (SCIP, GLPK, Gurobi, CPLEX). U našem radu koristimo SCIP backend. Ključna klasa je `Solver`, sa metodama za kreiranje binarnih promenljivih (`BoolVar`), dodavanje linearnih ograničenja (`Add`) i pokretanje rešavanja (`Solve`).

OR-Tools je izabran zbog: otvorenog izvornog koda, aktivne zajednice, konkurentnih performansi u odnosu na komercijalna rešenja, i podrške za obe paradigme (CP i MIP) kroz jedinstven Python API.

[1] `https://www.python.org/`
[2] `https://bazel.build/`

3.2 Bazel

Bazel je sistem za gradnju i testiranje softvera razvijen od strane Google-a. Ključne prednosti Bazel-a su:

- **Reproducibilnost** – hermetičke gradnje garantuju da će isti izvorni kod uvek proizvesti isti rezultat, nezavisno od okruženja.
- **Inkrementalne gradnje** – Bazel gradi samo one delove koji su se promenili, što ubrzava razvoj.
- **Višeplatformska podrška** – isti konfiguracioni fajlovi funkcionišu na različitim operativnim sistemima.

3.3 Python biblioteke

Pydantic

Pydantic[3] je biblioteka za validaciju podataka korišćenjem Python anotacija tipova. U našem projektu, Pydantic se koristi za definisanje modela ulaznih podataka (videti odeljak 4.1). Svaka entitetska klasa (`Classroom`, `Course`, `Settings`, itd.) je definisana kao `pydantic.dataclass` sa jasno definisanim tipovima i alijas imenima za mapiranje JSON polja:

Listing 3.1: Pydantic modeli podešavanja rasporeda i učionice

```
@dataclass(config=_config)
class Settings:
    working_days: List[str] = Field(alias="workingDays")
    start_hour: int = Field(alias="startHour")
    end_hour: int = Field(alias="endHour")
    duration: int = 1

@dataclass(config=_config)
class Classroom:
    id: int
    name: str
    loc_id: int = Field(alias="locId")
```

[3] https://docs.pydantic.dev/

```
13      has_computers: bool = Field(alias="hasComputers")
14      capacity: int = 0
```

Prednosti korišćenja Pydantic-a:

- automatska validacija tipova pri učitavanju JSON podataka – anotacije `int`, `bool` i `List[str]` istovremeno su i specifikacija koju Pydantic proverava, pa neispravan ulaz otkazuje pri parsiranju, a ne u rešavaču,
- eksplicitna dokumentacija strukture podataka – definicija klase je jedini opis očekivanog oblika ulaza,
- podrška za alias nazive – deklaracija `Field(alias=...)` preslikava `camelCase` imena iz JSON-a (`startHour`, `hasComputers`) u `snake_case` imena u Python kodu (`start_hour`, `has_computers`), tako da oba jezika ostaju u svojoj konvenciji,
- podrazumevane vrednosti – polja kao `duration` i `capacity` mogu se izostaviti u ulaznom fajlu.

openpyxl

Biblioteka openpyxl[4] koristi se za generisanje Excel fajlova sa rasporedom. Za svaku grupu studenata kreira se poseban radni list (eng. *worksheet*) sa tabelom gde redovi predstavljaju radne dane, a kolone vremenske termine. Ćelije sadrže informacije o predmetu, tipu časa (predavanje/vežbe) i učionici.

PyFunctional

PyFunctional[5] je biblioteka koja omogućava funkcionalni stil programiranja u Pythonu. U projektu se koristi objekat `seq` za fluent filtriranje i pretraživanje listi (npr. filtriranje predmeta po smeru, pronalaženje učionice po identifikatoru i slično).

pytest

Za testiranje ispravnosti implementacije koristi se pytest[6]. Testovi pokrivaju parsiranje ulaznih podataka, formiranje grupa studenata, generisanje sesija, i verifika-

[4] `https://openpyxl.readthedocs.io/`
[5] `https://github.com/EntilZha/PyFunctional`
[6] `https://docs.pytest.org/`

ciju da rešavači nalaze dopustiva rešenja koja zadovoljavaju sva tvrda ograničenja. Testovi se pokreću komandom:

```
bazel test //src/algo:test_data
bazel test //src/algo:test_cp_solver
bazel test //src/algo:test_mip_solver
```

Glava 4

Formulacija problema i implementacija

U ovoj glavi detaljno izlažemo model ulaznih podataka, način na koji se formiraju sesije koje treba rasporediti, i konkretne formulacije problema u CP i MIP paradigmama.

4.1 Model ulaznih podataka

Ulazni podaci su definisani u JSON formatu i modelovani pomoću Pydantic klasa u modulu `model.py`, na način prikazan u listingu 3.1.

Struktura ulaznih podataka je sledeća:

1. **Settings** – podešavanja rasporeda:
   - `workingDays` – lista radnih dana (Ponedeljak, Utorak, ..., Petak),
   - `startHour`, `endHour` – početak i kraj radnog vremena (8–20),
   - `duration` – trajanje jednog časa predavanja i vežbi (1 sat).
2. **Location** – lokacije/zgrade fakulteta (Studentski trg, Svetog Nikole, Jagićeva).
3. **Classroom** – učionice sa atributima:
   - identifikator, naziv, lokacija,
   - da li poseduje računare (`hasComputers`),

- kapacitet.

4. **Smer** – smerovi studija (Profesor matematike i računarstva, Teorijska matematika, Informatika, itd.). U izvornom kodu odgovara klasi `StudyTrack`, odnosno polju `trackId` u JSON ulazu.

5. **Course** – predmeti sa:

   - identifikatorom, nazivom, semestrom, smerom,
   - kvotom (`Quota`) – koliko časova predavanja i vežbi nedeljno,
   - zahtevom za računarima (`needsComputers`), zadatim *odvojeno za predavanja i za vežbe*.

   Predmet je pri tome samo nosilac zajedničkog identiteta i kvote: predavanja i vežbe se dalje raspoređuju kao nezavisni časovi, o čemu detaljno govori sekcija 4.2.

6. **StudentsEnrolled** – broj upisanih studenata po smeru i semestru.

Sve navedene celine objedinjuje korenska klasa `SchedulingInput`, koja odgovara jednom ulaznom JSON fajlu:

Listing 4.1: Korenska klasa ulaznog modela

```
@dataclass(config=_config)
class SchedulingInput:
    settings: Settings
    locations: List[Location]
    classrooms: List[Classroom]
    courses: List[Course]
    tracks: List[StudyTrack] = Field(
        validation_alias=AliasChoices("tracks", "departments"))
    students_enrolled: List[StudentsEnrolled] = Field(
        alias="studentsEnrolled")
    rules: dict[str, RuleConfig] = Field(default_factory=dict)
    groups: List[GroupDef] = Field(default_factory=list)
```

Polja `rules` i `groups` imaju podrazumevano prazne vrednosti i uvode se u kasnijim glavama: `rules` nosi konfiguraciju mekih i tvrdih pravila (glava 6), a `groups` omogućava da se grupe studenata zadaju eksplicitno umesto da se izvode iz broja

upisanih. Ta mogućnost je zadržana za testove i starije ulazne fajlove – u realnom ulazu grupe su definisane u fajlu sa nastavnim osobljem (odeljak 7.2), pa `groups` sekcije tamo nema. Alias `departments` zadržan je kako bi stariji ulazni fajlovi, nastali pre preimenovanja katedri u smerove, ostali upotrebljivi. Model osoblja je izdvojen u odvojenu klasu `StaffInput` (odeljak 7.1).

Realni podaci

Za eksperimentalnu evaluaciju koristimo podatke koji reprezentuju jedan semestar nastave na Matematičkom fakultetu:

- 6 smerova,
- 129 predmeta raspoređenih po semestrima (1, 3, 5, 7),
- 29 učionica na 3 lokacije,
- 30 grupa studenata izvedenih iz broja upisanih (ovaj ulaz nema fajl sa osobljem),
- radna nedelja od 5 dana, po 12 sati dnevno (8:00–20:00).

4.2 Formiranje sesija

Pre nego što se pokrene rešavač, ulazni podaci se transformišu u listu *sesija* – atomskih jedinica raspoređivanja. Ovaj proces se odvija u modulu `data.py` i sastoji se od sledećih koraka:

1. **Formiranje grupa studenata.** Grupa je kohorta koja zajedno sluša nastavu i formira se na jedan od tri načina, po prioritetu (funkcija `build_groups` u modulu `data.py`):
   - ako je zadat fajl sa osobljem u kome dodele nastave imenuju grupe (polje `groupIds`), imena grupa se uzimaju odatle, a njihove veličine iz broja upisanih – tako radi realni ulaz i o tome detaljno govori odeljak 7.2;
   - inače, ako ulazni JSON ima eksplicitnu `groups` sekciju, grupe se koriste onako kako su zadate;

- inače se za svaki par (smer, semestar) upis od n studenata deli na $\lceil n/50 \rceil$ grupa, gde je 50 podrazumevana veličina grupe (konstanta `GROUP_SIZE`). Na primer, za Informatiku u 1. semestru sa 100 studenata formiraju se dve grupe.

2. **Dodeljivanje predmeta grupama.** Svaka grupa pohađa predmete koji odgovaraju njenom smeru i semestru.

3. **Generisanje sesija.** Za svaki predmet i svaku grupu, kreira se onoliko sesija koliko to nalaže kvota: `quota.theory` sesija tipa "predavanje" i `quota.practice` sesija tipa "vežbe".

Svaka sesija s je okarakterisana sledećim atributima:

- identifikator grupe (g_s),
- identifikator predmeta,
- da li zahteva računare (comp_s),
- tip časa (predavanje ili vežbe).

Za sesiju s, rešavač treba da odredi trojku (d_s, h_s, r_s) – dan, sat i učionicu u kojima će se čas održati.

Nezavisnost predavanja i vežbi

Naglašavamo da su *predavanja i vežbe nezavisne jedinice raspoređivanja.* Predmet u ulaznom modelu nije jedinica koja se raspoređuje – on samo povezuje časove koji pripadaju istoj nastavnoj celini i propisuje njihov nedeljni broj kroz kvotu. Sve odluke o raspoređivanju donose se na nivou pojedinačne sesije, pa se za predavanja i vežbe istog predmeta nezavisno određuju:

- **broj časova** – `quota.theory` i `quota.practice` su odvojena polja, tako da predmet može imati npr. 2 časa predavanja i 3 časova vežbi nedeljno ili ne mora uopšte imati vežbe.
- **nastavnik** – dodela nastave (`TeachingAssignment`, sekcija 7.1) vezuje se za par (predmet, tip nastave), pa predavanja i vežbe tipično drže različiti nastavnici,

- **sastav slušalaca** – kohorte se formiraju odvojeno po tipu (sekcija 7.3), što odgovara uobičajenoj praksi da više grupa sluša predavanje zajedno, a vežbe odvojeno,
- **termin i učionica** – promenljive (d_s, h_s, r_s) postoje po sesiji, pa ni jedno ograničenje ne primorava predavanje i vežbe istog predmeta na isti termin ili istu učionicu,
- **zahtev za računarima** – comp_s zavisi od tipa časa, o čemu govori nastavak.

Zahtev za računarima zaslužuje posebnu pažnju jer je u praksi najčešće vezan isključivo za vežbe: predavanje iz programiranja drži se u većim učionicama koje ne moraju imati računare, dok vežbe iz istog predmeta zahtevaju računarsku učionicu. Zbog toga se `needsComputers` ne zadaje kao jedna zastavica na nivou predmeta, nego strukturom `ComputerNeed` sa odvojenim poljima za dva tipa nastave:

Listing 4.2: Zahtev za računarima po tipu nastave

```
@dataclass(config=_config)
class ComputerNeed:
    theory: bool = False
    practice: bool = False

@dataclass(config=_config)
class Course:
    ...
    quota: Quota = Field(default_factory=Quota)
    needs_computers: ComputerNeedField = Field(
        default_factory=ComputerNeed, alias="needsComputers")

    def needs_computers_for(self, session_type: str) -> bool:
        if session_type == "theory":
            return self.needs_computers.theory
        if session_type == "practice":
            return self.needs_computers.practice
        raise ValueError(...)
```

Prilikom generisanja sesija, svaka sesija dobija vrednost koja odgovara *njenom* tipu, a ne vrednost celog predmeta:

Listing 4.3: Sesija nasleđuje zahtev za računarima svog tipa

```
Session(
    ...,
    course.needs_computers_for(session_type),   # a ne course.
       needs_computers
    session_type,
)
```

U JSON ulazu predmet čije vežbe traže računare a predavanja ne zapisuje se kao:

Listing 4.4: Predmet sa računarima samo za vežbe

```
{"id": 4, "name": "Programiranje 1", "semester": 1, "trackId":
   1,
 "quota": {"theory": 2, "practice": 3},
 "needsComputers": {"theory": false, "practice": true}}
```

Skalarni oblik („`needsComputers": true` ili `1`) i dalje je dopušten i tumači se kao "isti zahtev za oba tipa nastave", čime stariji ulazni fajlovi ostaju upotrebljivi bez izmena.

4.3 CP formulacija

CP formulacija je implementirana u klasi `SimpleCPSolver` u modulu `cp_solver.py`. Koristimo Google OR-Tools CP-SAT rešavač.

Promenljive

Za svaku sesiju $s \in \{0, 1, \ldots, |\mathcal{S}| - 1\}$ definišemo sledeće celobrojne promenljive:

$$\text{day}_s \in \{0, 1, \ldots, D - 1\} \tag{4.1}$$

$$\text{slot}_s \in \{0, 1, \ldots, H - 1\} \tag{4.2}$$

$$\text{room}_s \in \{0, 1, \ldots, R - 1\} \tag{4.3}$$

gde je D broj radnih dana, H broj raspoloživih sati u toku jednog dana i R broj učionica.

Dodatno, definišemo dve izvedene promenljive:

$$\text{flatTime}_s = \text{day}_s \cdot H + \text{slot}_s \tag{4.4}$$

$$\text{roomTime}_s = \text{room}_s \cdot (D \cdot H) + \text{flatTime}_s \tag{4.5}$$

Promenljiva flatTime_s linearizuje par (dan, sat) u jedinstven indeks unutar nedelje. Promenljiva roomTime_s linearizuje trojku (učionica, dan, sat) u jedinstven indeks preko svih učionica i termina. Izvedene promenljive nisu automatski vezane za osnovne: u CP-SAT modelu one se uvode kao nove celobrojne promenljive, a jednakosti (4.4) i (4.5) dodaju se eksplicitno kao ograničenja:

Listing 4.5: Linearizacija termina i para (učionica, termin)

```
self.flat_time_var[s] = self.model.NewIntVar(
    0, total_slots - 1, f"flat_time_{s}")
self.model.Add(
    self.flat_time_var[s]
    == self.day_var[s] * H + self.slot_var[s])

self.room_time_var[s] = self.model.NewIntVar(
    0, R * total_slots - 1, f"room_time_{s}")
self.model.Add(
    self.room_time_var[s]
    == self.room_var[s] * total_slots + self.flat_time_var[s])
```

Ukupan broj promenljivih: $5 \cdot |\mathcal{S}|$, što je linearna funkcija broja sesija. Za realne podatke MATF-a to iznosi reda stotina promenljivih.

Tvrda ograničenja

HC1: Jedinstvenost učionice u terminu. Nikoje dve sesije ne mogu se održavati u istoj učionici u isto vreme:

$$\texttt{AllDifferent}(\text{roomTime}_0, \text{roomTime}_1, \ldots, \text{roomTime}_{|\mathcal{S}|-1}) \tag{4.6}$$

Ovo globalno ograničenje obuhvata sve sesije i garantuje da su sve trojke (d_s, h_s, r_s) međusobno različite.

HC2: Bez kolizija unutar grupe. Za svaku grupu g, neka je $\mathcal{S}_g \subseteq \mathcal{S}$ skup sesija te grupe. Nikoje dve sesije iste grupe ne smeju biti u isto vreme:

$$\texttt{AllDifferent}(\{\text{flatTime}_s : s \in \mathcal{S}_g\}), \quad \forall g \tag{4.7}$$

U kodu se oba ograničenja svode na isti poziv, samo nad različitim skupovima promenljivih: HC1 nad svim vrednostima roomTime, a HC2 nad vrednostima flatTime jedne grupe (modul `cp_solver.py`):

Listing 4.6: Ograničenja HC1 i HC2 u CP-SAT modelu

```
# HC1: nikoje dve sesije ne dele istu (ucionicu, dan, sat)
self.model.AddAllDifferent(list(self.room_time_var.values()))

# HC2: nikoje dve sesije iste grupe nisu u istom terminu
groups = defaultdict(list)
for s, session in enumerate(self.sessions):
    for group_id in session.group_ids:
        groups[group_id].append(s)

for group_id, session_indices in groups.items():
    self.model.AddAllDifferent(
        [self.flat_time_var[s] for s in session_indices])
```

Sesija se u ograničenje HC2 uključuje za *svaku* grupu iz svoje liste `group_ids`, što je bitno za zajedničke sesije koje istovremeno pohađa više grupa (odeljak 7.3).

HC3: Ograničenje računarskih učionica. Ako sesija s zahteva računare, njen izbor učionice je ograničen na podskup učionica sa računarima:

$$\text{comp}_s \implies \text{room}_s \in \texttt{AllowedAssignments}(\{r : \text{hasComputers}(r)\}) \tag{4.8}$$

Bitno je da je comp_s atribut *sesije*, a ne predmeta (sekcija 4.2): ograničenje se dodaje samo onim sesijama čiji tip nastave traži računare, pa predavanja iz predmeta čije vežbe traže računare ostaju slobodna da idu u bilo koju učionicu. To se u kodu vidi po tome što se lista dozvoljenih učionica gradi po sesiji, a ograničenje se dodaje samo ako je ta lista uža od skupa svih učionica:

Listing 4.7: Dozvoljene učionice po sesiji

```
for s, session in enumerate(self.sessions):
    allowed = [
        i for i, room in enumerate(self.classrooms)
```

```
        if not session.needs_computers or room.has_computers
    ]
    if len(allowed) < len(self.classrooms):
        self.model.AddAllowedAssignments(
            [self.room_var[s]], [[idx] for idx in allowed])
```

Zahtev za računarima je pri tome *jedini* tvrdi uslov na izbor učionice. Kapacitet učionice ne ulazi u listu `allowed`: on je meko ograničenje, pa sesija sme da dobije i učionicu u koju svi njeni slušaoci ne staju, uz cenu u funkciji cilja (odeljak 6.6).

Funkcija cilja

U trenutnoj verziji, CP formulacija nema funkciju cilja – rešavač traži bilo koje dopustivo rešenje koje zadovoljava sva tvrda ograničenja (režim izvodljivosti).

4.4 MIP formulacija

MIP formulacija je implementirana u klasi `SimpleMIPSolver` u modulu `mip_solver.py`. Koristimo SCIP rešavač kroz OR-Tools `pywraplp` interfejs.

Promenljive

Za svaku sesiju s i svaku dopustivu kombinaciju (dan, sat, učionica) definišemo binarnu promenljivu:

$$x_{s,d,h,r} \in \{0,1\}, \quad \forall s, \forall d \in \{0,\ldots,D-1\}, \forall h \in \{0,\ldots,H-1\}, \forall r \in \text{Elig}(s) \quad (4.9)$$

gde $\text{Elig}(s)$ označava skup učionica u kojima sesija s može da se održi. Za sesije koje zahtevaju računare, $\text{Elig}(s)$ sadrži samo učionice sa računarima – promenljive za nedozvoljene učionice se uopšte ne kreiraju, čime se implicitno poštuje ograničenje računarskih učionica.

Ukupan broj promenljivih: $O(|\mathcal{S}| \cdot D \cdot H \cdot R)$. Za realne podatke MATF-a to može iznositi desetine hiljada binarnih promenljivih, što je za redove veličine više nego u CP formulaciji.

Tvrda ograničenja

HC1: Svaka sesija raspoređena tačno jednom.

$$\sum_{d,h,r} x_{s,d,h,r} = 1, \quad \forall s \in \mathcal{S} \tag{4.10}$$

HC2: Najviše jedna sesija u datoj učionici u datom terminu.

$$\sum_{s \in \mathcal{S}} x_{s,d,h,r} \leq 1, \quad \forall d, h, r \tag{4.11}$$

HC3: Bez kolizija unutar grupe. Za svaku grupu g i svaki termin (d, h):

$$\sum_{s \in \mathcal{S}_g} \sum_{r} x_{s,d,h,r} \leq 1, \quad \forall g, \forall d, \forall h \tag{4.12}$$

Ova tri ograničenja u potpunosti odgovaraju ograničenjima CP formulacije (odeljak 4.3): HC1 obezbeđuje da svaka sesija ima tačno jednu dodelu (u CP-u je to inherentno, jer svaka sesija ima po jednu promenljivu za dan, sat i učionicu), HC2 odgovara ograničenju `AllDifferent` nad *roomTime*, a HC3 ograničenju `AllDifferent` nad *flatTime* po grupi. Time obe formulacije opisuju *isti* skup dopustivih rešenja, pa je njihovo poređenje u glavi 5 korektno (poređenje „jednakih sa jednakim").

Funkcija cilja

Kao i CP formulacija, MIP formulacija radi u režimu izvodljivosti bez funkcije cilja.

4.5 Poređenje formulacija

Tabela 4.1 sumira ključne razlike između dve formulacije.

Tabela 4.1: Poređenje CP i MIP formulacija

Aspekt	**CP-SAT**	**MIP/SCIP**
Tip promenljivih	Celobrojne	Binarne
Broj promenljivih	$O(\lvert\mathcal{S}\rvert)$	$O(\lvert\mathcal{S}\rvert \cdot D \cdot H \cdot R)$
Tip ograničenja	Globalna	Linearne nejednakosti
Broj ograničenja	Kompaktan	$O(D \cdot H \cdot R + G \cdot D \cdot H)$
Pretraga	Propagacija + SAT	LP relaksacija + B&B

Ključna razlika je u kompaktnosti modela: CP formulacija koristi mali broj celobrojnih promenljivih sa globalnim ograničenjima, dok MIP formulacija eksplicitno

enumeriše sve moguće dodele kroz binarne promenljive. Ovo ima direktan uticaj na memorijski otisak i brzinu konstrukcije modela, posebno za veće instance problema.

Glava 5

Eksperimentalna evaluacija

U ovoj glavi opisujemo metodologiju merenja performansi, predstavljamo rezultate eksperimenata i analiziramo dobijene podatke.

5.1 Metodologija

Podskupovi realnih podataka

Eksperimenti su sprovedeni na tri podskupa realnih podataka Matematičkog fakulteta, filtrirane po semestrima i lokacijama:

1. **MATF-S** (mali) – 1. godina (semestar 1), lokacija Studentski trg. Ovo je najmanji podskup sa relativno malim brojem sesija.
2. **MATF-M** (srednji) – 1. i 2. godina (semestri 1 i 3), lokacije Studentski trg i Jagićeva. Srednje velik podskup sa većom raznolikošću predmeta i učionica.
3. **MATF-L** (veliki) – sve godine (semestri 1, 3, 5, 7), sve lokacije. Najbliži realnom scenariju raspoređivanja.

Metrike

Za svaki par (rešavač, skala) merimo sledeće metrike:

- **Broj sesija** – ukupan broj časova koje treba rasporediti.
- **Broj promenljivih** – veličina modela u terminima odlučivačkih promenljivih.
- **Broj ograničenja** – ukupan broj ograničenja u modelu.

- **Vreme konstrukcije** – vreme potrebno za kreiranje modela (promenljivih i ograničenja), mereno pomoću `time.perf_counter()`.
- **Vreme rešavanja** – vreme potrebno za pronalaženje dopustivog rešenja.
- **Ukupno vreme** – zbir vremena konstrukcije i rešavanja.
- **Memorija modela** – memorijski otisak modela, meren pomoću `tracemalloc` (razlika memorijskih snimaka pre i posle konstrukcije modela).
- **Maksimalna RAM** – vršna potrošnja memorije procesa, merena pomoću `resource.getrusage()`.
- **Status** – da li je rešavač našao dopustivo rešenje ili je isteklo vreme.
- **Validnost rešenja** – nezavisna provera da li dobijeno rešenje zadovoljava sva tvrda ograničenja (bez kolizija učionica, bez kolizija grupa, poštovanje računarskih zahteva).

Validacija rešenja

Rezultati svakog rešavača se nezavisno verifikuju funkcijom `validate_solution` koja proverava:

1. da nijedna dva časa ne dele istu trojku (dan, sat, učionica),
2. da nijedna dva časa iste grupe nisu u istom terminu (dan, sat),
3. da časovi koji zahtevaju računare budu u učionicama sa računarima (provera se vrši nad zahtevom sesije, koji zavisi od tipa nastave),
4. da broj dodeljenih termina odgovara broju sesija.

5.2 Rezultati

U tabelama 5.1, 5.2 i 5.3 prikazani su rezultati za svaku skalu.

Napomena: Prikazani rezultati su dobijeni pokretanjem benchmark-a komandom:

```
bazel run //src/algo:benchmark -- --json report.json
```

Tabela 5.1: Rezultati za MATF-S (1. godina, Studentski trg)

Metrika	**CP-SAT**	**MIP/SCIP**
Sesije	192	
Promenljive	1 152	125 760
Ograničenja	611	1 392
Vreme konstrukcije	0,014 s	2,001 s
Vreme rešavanja	0,021 s	1,384 s
Ukupno vreme	0,035 s	3,385 s
Memorija modela	0,21 MB	29,23 MB
Status	FEASIBLE	FEASIBLE
Rešenje validno	da	da

Tabela 5.2: Rezultati za MATF-M (1–2. godina, Studentski trg + Jagićeva)

Metrika	**CP-SAT**	**MIP/SCIP**
Sesije	410	
Promenljive	2 460	368 640
Ograničenja	1 299	2 330
Vreme konstrukcije	0,029 s	5,953 s
Vreme rešavanja	0,081 s	6,094 s
Ukupno vreme	0,110 s	12,047 s
Memorija modela	0,40 MB	81,20 MB
Status	FEASIBLE	FEASIBLE
Rešenje validno	da	da

5.3 Analiza rezultata

Na osnovu sprovedenih eksperimenata, možemo izvući sledeće zaključke:

Kompaktnost modela

CP-SAT formulacija koristi $5 \cdot |\mathcal{S}|$ celobrojnih promenljivih, dok MIP/SCIP formulacija koristi $O(|\mathcal{S}| \cdot D \cdot H \cdot R)$ binarnih promenljivih. Za MATF-L skalu sa $D = 5$ dana, $H = 12$ sati i $R = 29$ učionica, svaka sesija generiše do $5 \cdot 12 \cdot 29 = 1740$ binarnih promenljivih u MIP-u, naspram 5 celobrojnih u CP-u. Ova razlika od dva do tri reda veličine se direktno odražava na memorijski otisak i vreme konstrukcije modela.

Tabela 5.3: Rezultati za MATF-L (sve godine, sve lokacije)

Metrika	**CP-SAT**	**MIP/SCIP**
Sesije	762	
Promenljive	4 572	1 179 720
Ograničenja	2 433	4 302
Vreme konstrukcije	0,058 s	22,748 s
Vreme rešavanja	0,838 s	25,387 s
Ukupno vreme	0,896 s	48,135 s
Memorija modela	0,73 MB	263,84 MB
Status	FEASIBLE	FEASIBLE
Rešenje validno	da	da

Brzina rešavanja

CP-SAT rešavač koristi globalna ograničenja (`AllDifferent`, `AllowedAssignments`) sa specijalizovanim algoritmima propagacije koji rano eliminišu nedopustive vrednosti iz domena promenljivih. U kombinaciji sa SAT tehnologijama učenja konflikata, ovo omogućava znatno bržu pretragu prostora rešenja u poređenju sa LP relaksacijom i branch-and-bound pristupom MIP/SCIP rešavača.

Skalabilnost

Razlika u performansama između dva pristupa postaje izraženija sa rastom veličine problema. Dok na malim instancama (MATF-S) oba rešavača nalaze rešenje relativno brzo, na većim instancama (MATF-M, MATF-L) MIP/SCIP zahteva značajno više vremena, a u nekim slučajevima može da ne nađe rešenje u zadatom vremenskom limitu.

Teorijska kompleksnost

Tabela 5.4 sumira teorijsku kompleksnost oba pristupa.

Tabela 5.4: Teorijska kompleksnost formulacija

Aspekt	**CP-SAT**	**MIP/SCIP**
Promenljive	$O(\|\mathcal{S}\|)$	$O(\|\mathcal{S}\| \cdot D \cdot H \cdot R)$
Ograničenja	Kompaktna globalna	$O(D \cdot H \cdot R + G \cdot D \cdot H)$
Propagacija	Specijalizovani algoritmi	LP relaksacija
Učenje	Konflikti (SAT klauze)	Ravni odsecanja

Rezultati ove glave odnose se na osnovni model i služe izboru paradigme za dalji razvoj. Pošto se model u glavama 6 i 7 znatno proširuje, u glavi 8 dajemo završnu evaluaciju konačne verzije rešavača na istim trima skalama.

Glava 6

Nadogradnja CP rešavača dodatnim ograničenjima

Na osnovu rezultata iz glave 5, CP-SAT rešavač je izabran kao osnova za dalji razvoj sistema. U ovoj glavi opisujemo nadogradnju modela dodatnim ograničenjima koja raspored približavaju realnim potrebama Matematičkog fakulteta. Za razliku od prvog dela rada, gde smo tražili bilo koje dopustivo rešenje, ovde uvodimo i funkciju cilja koja optimizuje kvalitet rasporeda kroz meka ograničenja.

Implementirana su četiri nova ograničenja:

- **spajanje časova u blokove** – časovi istog predmeta, grupe i tipa grupišu se u uzastopne blokove (na primer dvočasi i tročasi);
- **jedna lokacija po danu za grupu** – grupa studenata ne menja zgradu u toku jednog radnog dana;
- **bez procepa u rasporedu** – izbegavanje slobodnih sati između dva zauzeta sata u istom danu;
- **kapacitet učionice** – prekoračenje kapaciteta se kažnjava brojem studenata bez mesta, ali nije zabranjeno.

6.1 Arhitektura pravila

Svako ograničenje je modelovano kao zaseban dodatak (eng. *plugin*) koji se priključuje rešavaču. Ovakva arhitektura omogućava da se pravila nezavisno uključuju, isključuju i konfigurišu, bez izmena u jezgru rešavača.

Osnovu čini apstraktna klasa `SchedulingRule` (modul `src/rules/base.py`):

Listing 6.1: Apstraktna klasa pravila

```
class SchedulingRule(ABC):
    def __init__(self, enabled=True, penalty=0, **kwargs):
        self.enabled = enabled
        self.penalty = penalty

    @property
    def is_hard(self) -> bool:
        return self.penalty == 0

    @abstractmethod
    def apply(self, solver) -> list[cp_model.IntVar]:
        ...
```

Svako pravilo ima dva parametra: `enabled` (da li je pravilo aktivno) i `penalty` (cena kršenja). Vrednost `penalty` istovremeno određuje i *tip* ograničenja:

- `penalty == 0` – **tvrdo** (eng. *hard*) ograničenje. Mora biti zadovoljeno; u suprotnom rešavač prijavljuje da je problem nedopustiv (`INFEASIBLE`). Metoda `apply` dodaje ograničenja direktno u model i vraća praznu listu.
- `penalty > 0` – **meko** (eng. *soft*) ograničenje. Rešavač pokušava da ga zadovolji, ali ga može prekršiti uz datu cenu. Metoda `apply` vraća listu binarnih promenljivih prekršaja (vrednost 1 označava prekršaj).

Konfiguracija pravila

Pravila se konfigurišu u ulaznom JSON fajlu pomoću sekcije `rules`, gde svaki ključ odgovara nazivu pravila:

Listing 6.2: Primer konfiguracije pravila u JSON-u

```
"rules": {
    "joinSameClasses":  { "enabled": true, "penalty": 0 },
    "noGapsInSchedule": { "enabled": true, "penalty": 5 }
}
```

Ova konfiguracija se učitava u Pydantic model `RuleConfig` (modul `model.py`). U gornjem primeru, pravilo `joinSameClasses` je tvrdo (penalty 0), dok je `noGapsInSchedule` meko sa cenom 5 po prekršaju.

Primena pravila i funkcija cilja

Rešavač održava registar pravila `RULE_REGISTRY` koji preslikava naziv pravila u odgovarajuću klasu. Metoda `apply_rules` instancira sva uključena pravila, prikuplja promenljive prekršaja i formira funkciju cilja:

Listing 6.3: Primena pravila i formiranje funkcije cilja

```
def apply_rules(self):
    self.penalty_vars = []
    for rule_name, config in self.scheduling_input.rules.items()
        :
        if not config.enabled:
            continue
        rule_class = RULE_REGISTRY.get(rule_name)
        rule = rule_class(enabled=config.enabled, penalty=config
            .penalty)
        violations = rule.apply(self)
        for v in violations:
            self.penalty_vars.append((config.penalty, v))

    if self.penalty_vars:
        self.model.Minimize(
            sum(weight * var for weight, var in self.
                penalty_vars)
        )
```

Funkcija cilja je dakle težinska suma svih prekršaja mekih ograničenja:

$$\min \sum_{r} \text{penalty}_r \cdot \sum_{v \in V_r} v \tag{6.1}$$

gde je V_r skup promenljivih prekršaja koje vraća pravilo r, a penalty_r njegova cena. Tvrda ograničenja ne učestvuju u funkciji cilja jer vraćaju praznu listu prekršaja.

6.2 CP-SAT tehnike modeliranja

Nova pravila koriste nekoliko naprednih tehnika modeliranja koje pruža CP-SAT [13]. U ovom odeljku ih ukratko objašnjavamo, jer se koriste u sve tri implementacije.

Reifikacija i `OnlyEnforceIf`

Reifikacija povezuje istinitost nekog ograničenja sa binarnom *indikatorskom* promenljivom (literalom). U CP-SAT, metoda `OnlyEnforceIf(b)` čini da ograničenje važi *samo ako* je literal b tačan:

Listing 6.4: Uslovno (reifikovano) ograničenje

```
# lit == 1  =>  flat_time[s] == t
model.Add(flat_time[s] == t).OnlyEnforceIf(lit)
# lit == 0  =>  flat_time[s] != t
model.Add(flat_time[s] != t).OnlyEnforceIf(lit.Not())
```

Navođenjem oba smera (za b i za $\neg b$, tj. `lit.Not()`) dobija se puna ekvivalencija $b \Leftrightarrow (\text{flat\_time}_s = t)$, čime literal b tačno odražava istinitost ograničenja.

Logičko ILI preko `AddMaxEquality`

Za binarne promenljive (vrednosti 0 ili 1), maksimum je jednak logičkom ILI. CP-SAT to izražava ograničenjem `AddMaxEquality`:

Listing 6.5: Logičko ILI nad binarnim promenljivama

```
# b == 1 ako je bar jedan od literala u "lits" jednak 1
model.AddMaxEquality(b, lits)   # b = max(lits) = OR(lits)
```

Ovo koristimo da utvrdimo da li je grupa zauzeta u nekom terminu (bar jedna sesija je u tom terminu).

`AddBoolAnd` i `AddBoolOr`

Ograničenja `AddBoolAnd` (konjunkcija) i `AddBoolOr` (disjunkcija) nameću da svi, odnosno bar jedan literal budu tačni. U kombinaciji sa `OnlyEnforceIf` koriste se za potpunu reifikaciju složenih logičkih izraza:

Listing 6.6: Reifikacija ekvivalencije logičkog izraza

```
# gap == 1  <=>  (has_before AND has_after AND NOT busy)
```

```
2 model.AddBoolAnd([has_before, has_after, busy.Not()]).
     OnlyEnforceIf(gap)
3 model.AddBoolOr([has_before.Not(), has_after.Not(), busy]).
     OnlyEnforceIf(gap.Not())
```

Prvi red obezbeđuje smer $gap \Rightarrow (\ldots)$, a drugi (primenom De Morganovih zakona) smer $\neg gap \Rightarrow \neg(\ldots)$.

Indeksiranje niza preko `AddElement`

Ograničenje `AddElement(index, array, target)` nameće target = array[index], gde je `index` promenljiva. Ovo omogućava "pretragu" kroz niz na osnovu vrednosti druge promenljive:

Listing 6.7: Indeksiranje niza promenljivom

```
1 # loc[s] == locations_of_classrooms[room_var[s]]
2 model.AddElement(room_var[s], locations_of_classrooms, loc[s])
```

6.3 Pravilo: spajanje časova u blokove

Pravilo `JoinSameClassesRule` (modul `rule_join_same_classes.py`) grupiše časove istog predmeta, grupe i tipa (predavanje/vežbe) u uzastopne blokove. Cilj je izbeći rascepkane rasporede gde su, na primer, četiri časa istog predmeta razbacana kroz nedelju.

Formiranje blokova

Sesije se najpre grupišu u *familije* po ključu (predmet, grupa, tip). Za svaku familiju, broj sesija se deli na blokove funkcijom `split_into_blocks`:

$$\text{split}(n) = \begin{cases} [n] & \text{ako } n \leq 3 \\ [2, 2] & \text{ako } n = 4 \end{cases} \tag{6.2}$$

Tako se tročasovni blok drži zajedno, dok se četiri časa dele na dva dvočasa. Unutar svakog bloka, prva sesija se bira kao *sidro* (eng. *anchor*), a ostale se vezuju za nju.

Tvrda varijanta

U tvrdoj varijanti, svaka sesija s_i u bloku (sa rednim brojem i u odnosu na sidro a) mora biti istog dana, u istoj učionici i u uzastopnom satu:

$$\text{day}_{s_i} = \text{day}_a \tag{6.3}$$
$$\text{slot}_{s_i} = \text{slot}_a + i \tag{6.4}$$
$$\text{room}_{s_i} = \text{room}_a \tag{6.5}$$
$$\text{slot}_a + |B| - 1 \leq H - 1 \tag{6.6}$$

gde je $|B|$ veličina bloka, a poslednji uslov obezbeđuje da blok stane unutar radnog vremena. U kodu:

Listing 6.8: Tvrdo spajanje bloka oko sidra

```
for i, s in enumerate(block[1:], start=1):
    solver.model.Add(solver.day_var[s]  == solver.day_var[anchor
        ])
    solver.model.Add(solver.slot_var[s] == solver.slot_var[
        anchor] + i)
    solver.model.Add(solver.room_var[s] == solver.room_var[
        anchor])
solver.model.Add(solver.slot_var[anchor] + len(block) - 1 <= H -
    1)
```

Meka varijanta

U mekoj varijanti, ista ograničenja se uvode uslovno, preko literala b ("blok je spojen"). Ako blok ne može biti spojen, b je netačno i uvodi se promenljiva prekršaja:

Listing 6.9: Meko spajanje sa promenljivom prekršaja

```
b = solver.model.NewBoolVar(f"joined_{block[0]}_{block[-1]}")
for i, s in enumerate(block[1:], start=1):
    solver.model.Add(solver.day_var[s]  == solver.day_var[anchor
        ]).OnlyEnforceIf(b)
    solver.model.Add(solver.slot_var[s] == solver.slot_var[
        anchor] + i).OnlyEnforceIf(b)
    solver.model.Add(solver.room_var[s] == solver.room_var[
        anchor]).OnlyEnforceIf(b)
```

```
6
7 violation = solver.model.NewBoolVar(f"violation_{block[0]}_{
      block[-1]}")
8 solver.model.Add(violation == 1).OnlyEnforceIf(b.Not())
9 solver.model.Add(violation == 0).OnlyEnforceIf(b)
```

Promenljiva `violation` ulazi u funkciju cilja (jednačina 6.1), pa rešavač spaja blokove kad god je to moguće, a krši pravilo samo kada je neophodno.

6.4 Pravilo: jedna lokacija po danu za grupu

Pravilo `SingleLocationInDayForGroupRule` (modul `rule_single_location_in_day_for_group.py`) sprečava da grupa studenata menja zgradu (lokaciju) tokom jednog radnog dana. Matematički fakultet izvodi nastavu na više lokacija (Studentski trg, zgrada Sv. Nikole, Jagićeva), pa bi promena zgrade između dva uzastopna časa bila nepraktična. Ovo je tvrdo ograničenje.

Mehanizam

Ključ implementacije je dvostruka primena ograničenja `AddElement`. Najpre, za svaku sesiju s uvodimo promenljivu loc_s koja predstavlja lokaciju dodeljene učionice:

$$\text{loc}_s = \text{locOfClassroom}[\text{room}_s] \tag{6.7}$$

Pošto je room_s promenljiva, koristimo `AddElement` da indeksiramo niz lokacija učionica:

Listing 6.10: Kanalisanje učionice u lokaciju

```
1 loc[s] = solver.model.NewIntVar(0, max_loc, f"loc_{s}")
2 solver.model.AddElement(
3     solver.room_var[s], self.locations_of_classrooms, loc[s]
4 )
```

Zatim, za svaku grupu g i dan d uvodimo promenljivu $\text{groupDayLoc}_{g,d}$ – lokaciju te grupe tog dana. Za svaku sesiju s grupe g namećemo da njena lokacija bude jednaka lokaciji grupe na dan kada se sesija održava:

$$\text{loc}_s = \text{groupDayLoc}_g[\text{day}_s] \tag{6.8}$$

Ponovo koristimo `AddElement`, ovog puta indeksirajući niz dnevnih lokacija promenljivom day_s:

Listing 6.11: Vezivanje lokacije sesije za lokaciju grupe po danu

```
for g, indices in group_session_indices.items():
    day_locs = [group_day_loc[g, d] for d in range(D)]
    for s in indices:
        # loc[s] == group_day_loc[g][day_var[s]]
        solver.model.AddElement(solver.day_var[s], day_locs, loc
            [s])
```

Pošto sve sesije grupe g u danu d moraju imati istu vrednost loc_s (jednaku $\text{groupDayLoc}_{g,d}$), rešavač je prinuđen da ih sve smesti u učionice na istoj lokaciji.

6.5 Pravilo: bez procepa u rasporedu

Pravilo `NoGapsInScheduleRule` (modul `rule_no_gaps_in_schedule.py`) cilja na kvalitet rasporeda iz ugla studenata: izbegava *procepe* – slobodne sate koji se nalaze između dva zauzeta sata u istom danu za istu grupu. Na primer, ako grupa ima čas u 8h i u 11h, a 9h i 10h su slobodni, to su dva procepa.

Matrica zauzetosti

Najpre se za svaku grupu gradi matrica zauzetosti $\text{busy}_{d,h}$ koja je tačna ako grupa ima neki čas u danu d, satu h. Za svaku sesiju i svaki termin $t = d \cdot H + h$ uvodimo reifikovani literal koji je tačan kada je sesija u tom terminu, a zatim ih agregiramo logičkim ILI preko `AddMaxEquality`:

Listing 6.12: Gradnja matrice zauzetosti

```
for s in sess:
    lit = solver.model.NewBoolVar(f"at_{s}_{t}")
    solver.model.Add(solver.flat_time_var[s] == t).OnlyEnforceIf
        (lit)
    solver.model.Add(solver.flat_time_var[s] != t).OnlyEnforceIf
        (lit.Not())
    lits.append(lit)
b = solver.model.NewBoolVar(f"busy_{g}_{d}_{h}")
```

```
7 solver.model.AddMaxEquality(b, lits)   # OR preko svih sesija
    grupe
```

Detekcija procepa

Sat h je procep ako je slobodan ($\neg \text{busy}_{d,h}$), a postoji zauzet sat i pre i posle njega u istom danu. Pomoćni literali hasBefore i hasAfter se računaju kao logičko ILI nad prefiksom, odnosno sufiksom reda zauzetosti. Sama promenljiva gap se reifikuje kao konjunkcija:

$$\text{gap}_{d,h} \Leftrightarrow \text{hasBefore}_{d,h} \wedge \text{hasAfter}_{d,h} \wedge \neg \text{busy}_{d,h} \tag{6.9}$$

Listing 6.13: Detekcija unutrašnjeg procepa

```
1  has_before = solver.model.NewBoolVar(f"hb_{g}_{d}_{h}")
2  solver.model.AddMaxEquality(has_before, busy_day[:h])
3  has_after = solver.model.NewBoolVar(f"ha_{g}_{d}_{h}")
4  solver.model.AddMaxEquality(has_after, busy_day[h + 1:])
5
6  gap = solver.model.NewBoolVar(f"gap_{g}_{d}_{h}")
7  solver.model.AddBoolAnd(
8      [has_before, has_after, busy_day[h].Not()]
9  ).OnlyEnforceIf(gap)
10 solver.model.AddBoolOr(
11     [has_before.Not(), has_after.Not(), busy_day[h]]
12 ).OnlyEnforceIf(gap.Not())
```

Rubni sati (prvi i poslednji u danu) ne mogu biti unutrašnji procepi, pa se preskaču.

Tvrda i meka varijanta

U tvrdoj varijanti svaki procep se zabranjuje ograničenjem $\text{gap}_{d,h} = 0$. U mekoj varijanti, promenljive $\text{gap}_{d,h}$ se vraćaju kao prekršaji i ulaze u funkciju cilja, pa rešavač minimizuje ukupan broj procepa:

Listing 6.14: Tvrda naspram meke varijante

```
1 if self.is_hard:
2     for gap in gap_vars:
```

```
            solver.model.Add(gap == 0)    # zabrana svih procepa
        return []
    return gap_vars                        # meko: minimizuj broj procepa
```

Ispravnost pravila proverava se testom `test_no_gaps_in_schedule.py`, koji potvrđuje da rešenje na malom ulazu nema nijedan unutrašnji procep.

6.6 Pravilo: kapacitet učionice kao meko ograničenje

Svaka učionica ima kapacitet, a svaka sesija poznat broj slušalaca size_s (zbir grupa koje je pohađaju, odeljak 7.3). Prirodno je zahtevati da svi slušaoci stanu u učionicu, $\text{size}_s \leq \text{capacity}_{r_s}$, i model je u ranijoj verziji to nametao kao tvrd uslov – učionice nedovoljnog kapaciteta prosto nisu ulazile u listu dozvoljenih učionica sesije (odeljak 4.3).

Takav tvrd uslov se, međutim, pokazao i previše krut i previše osetljiv na kvalitet ulaznih podataka:

- brojevi upisanih studenata su procene, a stvarna prisutnost je po pravilu manja od upisa; nerealno je da jedan student preko kapaciteta učini ceo raspored nedopustivim,
- u kombinaciji sa zahtevom za računarima kapacitet drastično sužava izbor učionica (računarskih učionica je malo i one su male), pa rešavač lako prijavi `INFEASIBLE` i ne vrati *nikakav* raspored – a raspored sa nekoliko prepunjenih učionica u praksi je daleko korisniji od nijednog,
- svako sužavanje dopustivog skupa gubi informaciju: umesto poruke „nema rešenja“ korisnije je videti *gde* i *koliko* raspored odstupa od željenog.

Zbog toga kapacitet prelazi iz tvrdih ograničenja u pravilo `rule_room_capacity.py` (`roomCapacity`). U mekoj varijanti pravilo ne sužava dopustiv skup – sesija sme u bilo koju učionicu koja zadovoljava tvrda ograničenja – ali plaća cenu jednaku broju studenata koji u toj učionici ostaju bez mesta:

$$\text{cost}_s = \max\left(0,\ \text{size}_s - \text{capacity}_{r_s}\right) \tag{6.10}$$

Bitno je da se kažnjava *samo* prekoračenje. Sve učionice u koje cela kohorta staje su međusobno jednako dobre, pa pravilo ne pravi nepotrebnu razliku između njih i ne troši prostor rešenja na to da najveću salu dobije najveća grupa kad to nikome ne pomaže. Kazna je nula uvek kada raspored nikoga ne ostavlja bez mesta, što je i uobičajen ishod: prekoračenje se javlja samo tamo gde učionica odgovarajuće veličine (ili odgovarajuće opremljena računarima) više nije slobodna.

Za razliku od procepa, prekršaj ovde nije „jedan događaj“ nego *količina* – pravilo je prvo koje kao prekršaj vraća celobrojne, a ne binarne promenljive. Funkcija cilja (6.1) je težinska suma vraćenih promenljivih, pa se takva cena uklapa bez izmena u jezgru rešavača. Cena po prekršaju je pri tome namerno niska, a niska težina drži ovaj sabirak podređenim ostalim ciljevima čak i kada se prekoračenje meri desetinama studenata.

U implementaciji se za svaku sesiju gradi tabela prekoračenja po učionicama (konstante, jer su i veličina sesije i kapaciteti poznati unapred), a promenljiva prekoračenja se vezuje za izabranu učionicu preko `AddElement` (odeljak 6.2). Sesije koje staju u svaku učionicu ne dobijaju nikakvu promenljivu:

Listing 6.15: Prekoračenje kapaciteta po sesiji

```
overflows = [max(0, session.size - cap) for cap in capacities]
if not any(overflows):
    continue                        # sesija staje u svaku ucionicu

if self.is_hard:                    # tvrdo: samo ucionice u koje staje
    fitting = [i for i, over in enumerate(overflows) if over == 0]
    solver.model.AddAllowedAssignments(
        [solver.room_var[s]], [[i] for i in fitting])
    continue

overflow = solver.model.NewIntVar(0, max(overflows), f"room_overflow_{s}")
solver.model.AddElement(solver.room_var[s], overflows, overflow)
overflow_vars.append(overflow)
```

Tvrda varijanta (`penalty` 0) svodi se na klasično ograničenje kapaciteta: izbor učionica se sužava na one u koje kohorta staje, isto kao što se za računare radi u

jezgru rešavača. Ako za neku sesiju takve učionice nema, greška se prijavljuje odmah pri konstrukciji modela, umesto da se pojavi kao neinformativan status `INFEASIBLE`.

Ispravnost obe varijante proverava test `test_room_capacity.py`: meko pravilo bira učionicu u koju grupa staje, a kada takve učionice nema ipak vraća raspored sa najmanjim prekoračenjem; tvrdo pravilo takve učionice zabranjuje.

Glava 7

Raspoređivanje nastavnog osoblja

Do sada je model raspoređivao samo grupe studenata u učionice i termine. U ovoj glavi model proširujemo nastavnim osobljem: svakoj sesiji se dodeljuje nastavnik, uvode se ograničenja vezana za nastavnike, i uvodi se pojam *kohorte* – skupa grupa koje zajedno slušaju isti čas. Ovaj deo predstavlja poslednji korak u približavanju modela realnom scenariju na Matematičkom fakultetu.

Nastavno osoblje se zadaje u *odvojenom* ulaznom fajlu, tako da osnovni ulaz (predmeti, učionice, broj upisanih) ostaje nepromenjen. Time se podaci o osoblju mogu menjati nezavisno, a raspoređivanje bez osoblja i dalje radi kao ranije. Pokazaće se da je taj fajl i prirodno mesto za definisanje grupa studenata (odeljak 7.2): grupa postoji tek onda kada neko drži nastavu za nju.

7.1 Model osoblja i dodela nastave

Podaci o osoblju modelovani su klasom `StaffInput` (modul `model.py`) koja sadrži tri celine: listu nastavnika (`teachers`), listu dodela nastave (`assignments`) i konfiguraciju pravila (`rules`). Nastavnik (`Teacher`) ima identifikator, ime i opciono lično ograničenje najvećeg broja radnih dana:

Listing 7.1: Modeli nastavnika i dodele nastave

```
@dataclass(config=_config)
class Teacher:
    id: int
    name: str
    max_working_days: Optional[int] = Field(default=None, alias=
        "maxWorkingDays")
```

```
6
7  @dataclass(config=_config)
8  class TeachingAssignment:
9      teacher_id: int = Field(alias="teacherId")
10     course_id: int = Field(alias="courseId")
11     session_type: str = Field(alias="sessionType")   # "theory"
           | "practice"
12     group_ids: Optional[List[str]] = Field(default=None, alias="
           groupIds")
```

Dodela (`TeachingAssignment`) povezuje par (predmet, tip nastave) sa nastavnikom, i opciono navodi konkretne grupe. Prilikom generisanja sesija, svaka sesija dobija polje `teacher_id`, a rešavač gradi preslikavanje `teacher_sessions` (nastavnik → indeksi njegovih sesija). Sesije bez dodeljenog nastavnika (`teacher_id` jednako `None`) ne učestvuju u pravilima osoblja.

Nastavnik ne može držati dva časa istovremeno

Prvo i osnovno ograničenje osoblja je tvrdo i dodaje se u jezgru rešavača: nijedan nastavnik ne može držati dve sesije u istom terminu. Analogno ograničenju za grupe (HC2, listing 4.6), koristimo `AllDifferent` nad apsolutnim vremenima svih sesija jednog nastavnika:

$$\texttt{AllDifferent}(\{\text{flatTime}_s : s \in \mathcal{S}_t\}), \quad \forall t \tag{7.1}$$

gde je $\mathcal{S}_t$ skup sesija nastavnika t. U kodu (modul `cp_solver.py`):

Listing 7.2: Nastavnik nije dvostruko zauzet

```
1  for teacher_id, session_indices in self.teacher_sessions.items()
       :
2      if len(session_indices) > 1:
3          self.model.AddAllDifferent(
4              [self.flat_time_var[s] for s in session_indices]
5          )
```

7.2 Grupe studenata izvedene iz dodela nastave

Dodela nastave imenuje grupe kojima nastavnik drži čas, na primer `„groupIds": [„1i1a", „1i1b", „1i1v"]`. Time je u fajlu sa osobljem već zapisano *koje* grupe postoje, pa bi njihovo ponovno navođenje u osnovnom ulazu bilo redudantno.

Postupak je implementiran u funkciji `build_groups` (modul `data.py`) i odvija se u dva koraka.

Imena i pripadnost. Prolazi se kroz sve dodele koje navode `groupIds`; za svaki naziv grupe smer i semestar se *ne* zadaju posebno, nego se čitaju sa predmeta na koji se dodela odnosi (`courseId`). Grupa `1i1a` tako pripada smeru i semestru predmeta koji sluša. Ako se isti naziv grupe pojavi na predmetima iz dva različita para (smer, semestar), to je nekonzistentan ulaz i prijavljuje se kao greška:

Listing 7.3: Imena grupa iz dodela nastave

```
for a in staff_input.assignments:
    if not a.group_ids:
        continue
    course = courses_by_id.get(a.course_id)
    key = (course.track_id, course.semester)
    for gid in a.group_ids:
        existing = meta.get(gid)
        if existing is not None and existing != key:
            raise ValueError(f"Group '{gid}' is used for both {
                existing} and {key}")
        meta[gid] = key
```

Veličine. Broj studenata se i dalje uzima iz sekcije `studentsEnrolled` osnovnog ulaza, koja je zadata po paru (smer, semestar). Upis se ravnomerno deli na sve grupe tog para, a ostatak se raspoređuje na prve grupe po abecednom redu naziva, tako da se veličine razlikuju najviše za jednog studenta:

Listing 7.4: Deljenje upisanih studenata na imenovane grupe

```
base, rem = divmod(enrollment.count, len(group_ids))
for i, gid in enumerate(sorted(group_ids)):
    count = base + (1 if i < rem else 0)
    groups.append(Group(gid, enrollment.track_id, count,
        enrollment.semester))
```

Na primer, 100 upisanih studenata prvog semestra Informatike i šest grupa navedenih u dodelama (`1i1a`, `1i1b`, `1i1v`, `1i2a`, `1i2b`, `1i2v`) daju četiri grupe od 17 i dve od 16 studenata.

Dakle, fajl sa osobljem određuje *koje* grupe postoje i *ko* im drži nastavu, a osnovni ulaz *koliko* ih je i *šta* slušaju. Ako neki par (smer, semestar) nema ni jednu imenovanu grupu u dodelama, za njega se koristi rezervni postupak deljenja po veličini grupe iz odeljka 4.2, pa delimično popunjen fajl sa osobljem ne obara raspoređivanje ostatka fakulteta.

7.3 Kohorte i zajedničke sesije

Bez dodatne pažnje, ako dve grupe slušaju isto predavanje, model bi napravio dve odvojene sesije i time primorao nastavnika da isti čas drži dva puta. Da bismo to izbegli, uvodimo pojam *kohorte* (`Cohort`) – skupa grupa koje zajedno pohađaju sesije jednog kursa i tipa, sa (opcionim) zajedničkim nastavnikom. Veličina kohorte je zbir broja studenata svih njenih grupa, što je bitno za kapacitet učionice.

Kohorte se formiraju funkcijom `build_cohorts` (modul `data.py`) na osnovu dodela nastave, po sledećim pravilima:

- **Zajednička (joint) dodela** – ako dodela navodi više grupa u `groupIds`, te grupe čine *jednu* kohortu i dobijaju *jednu zajedničku sesiju*. Tako nastavnik drži čas *jednom*, a prisustvuju mu sve grupe (npr. oba toka informatike slušaju zajedno predavanje iz predmeta “Programske paradigme”).
- **Dodela za jednu grupu** – `groupIds` sa jednom grupom važi samo za tu grupu.
- **Generička dodela** – ako `groupIds` nije naveden, dodela važi za svaku *preostalu* grupu kursa pojedinačno (svaka grupa dobija svoju sesiju sa tim nastavnikom).
- Grupe bez ijedne dodele postaju pojedinačne kohorte bez nastavnika.

Listing 7.5: Formiranje kohorti iz dodela nastave

```
1 for a in assignments:
```

```
    if a.course_id != course.id or a.session_type !=
       session_type:
        continue
    if a.group_ids is None:
        generic_teacher = a.teacher_id          # primenjuje se
           na sve preostale grupe
        continue
    cohorts.append(Cohort([groups_by_id[g] for g in a.group_ids
       ], a.teacher_id))
    for g in a.group_ids:
        remaining.pop(g, None)                  # ove grupe su
           pokrivene

for g in remaining.values():                    # grupe bez
   eksplicitne dodele
    cohorts.append(Cohort([g], generic_teacher))
```

Za svaku kohortu i svaki traženi čas iz kvote kreira se tačno jedna sesija, čija je lista grupa `group_ids` jednaka grupama kohorte. Zajednička sesija tako ulazi u ograničenje o koliziji *svake* grupe koja je pohađa (tvrdo ograničenje HC2 prolazi kroz sve `session.group_ids`). Veličina kohorte ne ograničava izbor učionice tvrdo, ali kroz meko pravilo iz odeljka 6.6 usmerava sesiju ka dovoljno velikoj sali.

Primer. Dve grupe `ga` (30 studenata) i `gb` (25 studenata) slušaju predavanja zajedno, a vežbe odvojeno. Za predmet sa 2 predavanja i 2 vežbe nedeljno dobija se: 2 zajedničke teorijske sesije (kohorta `ga+gb`, veličine 55) i po 2 vežbe za svaku grupu – ukupno 6 sesija umesto 8. Zajedničkom predavanju je za 55 slušalaca potrebna sala kapaciteta najmanje 55, dok vežbama od 30, odnosno 25 studenata odgovara i znatno manja učionica.

7.4 Pravila nastavnog osoblja

Pravila osoblja koriste istu *plugin* arhitekturu i CP-SAT tehnike (reifikacija, `AddMaxEquality`, `AddBoolAnd`/`AddBoolOr`, `AddElement`) uvedene u glavi 6. Konfigurišu se u `rules` sekciji staff fajla i mogu biti tvrda (`penalty` 0) ili meka (`penalty` > 0).

Jedna lokacija po danu za nastavnika

Pravilo `rule_staff_single_location_in_day.py` (`staffSingleLocationInDay`) sprečava da nastavnik menja zgradu tokom jednog dana – fizički bi bilo neizvodljivo držati dva uzastopna časa na različitim lokacijama. Mehanizam je isti kao za grupe: za svakog nastavnika i dan uvodi se promenljiva lokacije, a sve njegove sesije tog dana vezuju se za nju pomoću `AddElement`:

Listing 7.6: Nastavnik na jednoj lokaciji dnevno

```
day_locs = [model.NewIntVar(0, max_loc, f"teacher_day_loc_{
    teacher_id}_{d}")
            for d in range(D)]
for s in session_indices:
    # loc_var[s] == day_locs[day_var[s]]
    solver.model.AddElement(solver.day_var[s], day_locs, solver.
        loc_var[s])
```

Ovo je tvrdo ograničenje i vraća praznu listu prekršaja.

Najveći broj radnih dana nastavnika

Pravilo `rule_staff_max_working_days.py` (`staffMaxWorkingDays`) ograničava koliko različitih dana u nedelji nastavnik dolazi na fakultet. Za svaki dan d uvodi se indikator $\text{dayUsed}_{t,d}$ (logičko ILI reifikovanih jednakosti $\text{day}_s = d$ preko sesija nastavnika), a broj radnih dana je njihov zbir:

$$\text{worked}_t = \sum_{d=0}^{D-1} \text{dayUsed}_{t,d} \tag{7.2}$$

Tvrda varijanta nameće $\text{worked}_t \leq \text{limit}_t$, gde je limit_t lični limit nastavnika (`maxWorkingDays`) ili podrazumevana vrednost pravila (`maxDays`). Meka varijanta uvodi celobrojnu promenljivu viška i minimizuje ga:

$$\min \sum_t \text{penalty} \cdot \max(0,\ \text{worked}_t - \text{limit}_t) \tag{7.3}$$

Listing 7.7: Ograničenje broja radnih dana

```
worked_days = sum(day_used)                    # day_used[d] = OR
    preko sesija
if self.is_hard:
```

```
    solver.model.Add(worked_days <= limit)
else:
    excess = solver.model.NewIntVar(0, D, f"staff_days_excess_{
        teacher_id}")
    solver.model.Add(excess >= worked_days - limit)
    violations.append(excess)                # ulazi u funkciju
        cilja
```

Nedeljni budžet procepa (broj sati pauze) nastavnika

Pravilo `rule_staff_no_gap_greater_than.py` (`staffMaxGapHoursPerWeek`) ograničava ukupan broj procepa (pauza) nastavnika *tokom cele nedelje*. Procep se detektuje isto kao u pravilu Pravilo: bez procepa u rasporedu: matrica zauzetosti se gradi reifikovanim jednakostima i `AddMaxEquality`, a procep je konjunkcija "ima čas pre", "ima čas posle" i "slobodan sat". Za razliku od pravila za grupe, ovde se procepi *sabiraju kroz sve dane*:

$$\text{weekGaps}_t = \sum_{d=0}^{D-1} \sum_{h} \text{gap}_{t,d,h} \tag{7.4}$$

Tvrda varijanta nameće $\text{weekGaps}_t \leq B$ (budžet $B =$ `maxGapHours`, podrazumevano 1). Meka varijanta minimizuje prekoračenje budžeta:

$$\min \sum_{t} \text{penalty} \cdot \max(0,\ \text{weekGaps}_t - B) \tag{7.5}$$

Listing 7.8: Nedeljni budžet procepa

```
total_gaps = sum(week_gaps)
if self.is_hard:
    solver.model.Add(total_gaps <= self.max_gap_hours)
else:
    excess = solver.model.NewIntVar(0, D * H, f"
        staff_gap_excess_{teacher_id}")
    solver.model.Add(excess >= total_gaps - self.max_gap_hours)
    violations.append(excess)
```

Svi prekršaji mekih pravila osoblja ulaze u istu globalnu funkciju cilja iz glave 6: $\min \sum_r \text{penalty}_r \cdot \sum_{v \in V_r} v$.

7.5 Konfiguracija i provera

Osoblje se zadaje u zasebnom JSON fajlu (na primer `staff_2_semester.json`). Sekcija `rules` bira koja pravila su aktivna i sa kojom cenom, `teachers` navodi nastavnike, a `assignments` povezuje predmete sa nastavnicima i istovremeno definiše grupe studenata (odeljak 7.2):

Listing 7.9: Isečak staff ulaznog fajla

```
{
  "rules": {
    "staffMaxWorkingDays":     {"enabled": true, "penalty": 3, "
       params": {"maxDays": 4}},
    "staffMaxGapHoursPerWeek": {"enabled": true, "penalty": 0, "
       params": {"maxGapHours": 1}},
    "staffSingleLocationInDay":{"enabled": true, "penalty": 0}
  },
  "teachers": [
    {"id": 4, "name": "milan.mitreski", "maxWorkingDays": 2}
  ],
  "assignments": [
    {"teacherId": 43, "courseId": 54, "sessionType": "theory", "
       groupIds": ["2la", "2lb"]}
  ]
}
```

Rešavač se pokreće sa dodatnim argumentom `--staff`:

```
bazel run //src/algo:run_cp_solver -- \
    --input input_full_2_semester.json \
    --staff staff_2_semester.json --time-limit 600
```

Ispravnost pravila osoblja proverava se testovima u `test_staff_rules.py`: da nastavnik nije dvostruko zauzet, da se poštuje najveći broj radnih dana (uključujući lične limite), da je nedeljni budžet procepa ispunjen, da nastavnik ne menja lokaciju u toku dana, i da se zajednička sesija dve grupe rešava kao jedan čas bez kolizija u rasporedu obe grupe. Izvođenje grupa iz dodela pokriveno je testovima u `test_data.py`.

Glava 8

Završna evaluacija proširenog modela

Evaluacija iz glave 5 poredila je dve paradigme na osnovnom modelu, u režimu traženja bilo kog dopustivog rasporeda. Pošto je model u glavama 6 i 7 proširen mekim ograničenjima, funkcijom cilja i raspoređivanjem nastavnog osoblja, u ovoj glavi merimo kako se tako prošireni CP-SAT model ponaša na tri skale problema. Za razliku od prve evaluacije, ovde nas ne zanima samo da li rešenje postoji, nego i *kvalitet* rešenja i brzina kojom se on popravlja, pa pratimo vrednost funkcije cilja u zavisnosti od proteklog vremena izračunavanja.

Merenja su sprovedena u dva okruženja. Prvo je realno okruženje, sa podacima onakvim kakvi jesu, i ono odgovara na pitanje da li je prošireni model upotrebljiv u praksi. Drugo je zaoštreno okruženje (odeljak 8.4), u kome je broj raspoloživih termina smanjen tako da meka ograničenja dođu u stvarni sukob, i ono pokazuje kako se rešavač ponaša na granici rešivosti.

8.1 Metodologija

Instance po godinama studija

Prošireni model radi nad realnim podacima letnjeg semestra (`input_full_2_semester.json`) i dodelama nastave iz `staff_2_semester.json`, iz kojih se izvode grupe studenata (odeljak 7.2). Svaka grupa nasleđuje semestar predmeta koji sluša (2, 4, 6 ili 8), što odgovara godini studija, te se instanca može seći po godinama:

1. **FINAL-S** – 1. godina (semestar 2),

2. **FINAL-M** – 1. i 2. godina (semestri 2 i 4),
3. **FINAL-L** – sve četiri godine (semestri 2, 4, 6 i 8).

Za razliku od podskupova iz glave 5, ovde se skalira samo potražnja: sve tri instance zadržavaju svih 29 učionica i sve tri lokacije, pa se razlikuju jedino po količini nastave koju treba smestiti. Filtriraju se samo predmeti i upisi; grupe se povlače same, jer se izvode iz dodela za predmete koji su ostali, a dodele za izbačene predmete se prosto ne aktiviraju – ulaz sa osobljem tako ostaje nepromenjen.

Konfiguracije i funkcija cilja

Svaka skala se rešava u dve konfiguracije:

- **bez osoblja** – pravila iz glave 6: spajanje časova u blokove kao tvrdo ograničenje, izbegavanje procepa kao meko ograničenje sa cenom 2 i kapacitet učionica kao meko ograničenje sa cenom 1;
- **sa osobljem** – dodatno i ceo model iz glave 7: kohorte i zajedničke sesije, tvrdo ograničenje da nastavnik nije dvostruko zauzet, tvrda pravila za jednu lokaciju po danu i nedeljni budžet procepa nastavnika, i meko pravilo najvećeg broja radnih dana sa cenom 3.

Funkcija cilja je, u skladu sa izrazom (6.1) iz odeljka 6.1, težinska suma prekršaja mekih pravila:

$$\min \; 2 \cdot \underbrace{\sum_{g,d,h} \mathrm{gap}_{g,d,h}}_{\text{procepi grupa}} + 3 \cdot \underbrace{\sum_{t} \mathrm{excess}_{t}}_{\text{dani preko limita nastavnika}} + 1 \cdot \underbrace{\sum_{s} \mathrm{overflow}_{s}}_{\text{studenti bez mesta}},$$

pri čemu u konfiguraciji bez osoblja drugi sabirak ne postoji. Treći sabirak je meko ograničenje kapaciteta iz odeljka 6.6 i u obe konfiguracije nosi najmanju težinu. Vrednost 0 znači da nijedno meko ograničenje nije prekršeno, dakle raspored bez ijednog procepa, bez ijednog nastavnika koji radi više dana nego što mu je dozvoljeno i bez ijednog studenta koji ostaje bez mesta u učionici.

Uslovi merenja

Merenja su izvršena na računaru sa procesorom Apple M4 (10 jezgara) i 24 GB radne memorije, pod operativnim sistemom macOS 26.5, sa bibliotekom OR-Tools 9.15. Rešavač koristi 8 niti pretrage i vremenski limit od 600 sekundi po pokretanju.

Vrednost funkcije cilja kroz vreme prati se povratnim pozivom (*callback*) koji CP-SAT poziva pri svakom pronađenom poboljšanju. Za svako poboljšanje beleže se proteklo vreme pretrage, vrednost cilja i najbolja donja granica, iz čega se dobija stepenasta funkcija: vrednost u trenutku t je cilj poslednjeg rešenja nađenog do tog trenutka. Rešenja se, kao i u prvoj evaluaciji, nezavisno proveravaju funkcijom `validate_solution`, koja je za potrebe proširenog modela dopunjena i proverom da nastavnik nema dva časa u istom terminu.

8.2 Veličina modela

Tabela 8.1 prikazuje veličinu modela po skalama i konfiguracijama.

Tabela 8.1: Veličina proširenog CP modela po skalama

	FINAL-S		**FINAL-M**		**FINAL-L**	
Metrika	bez	sa	bez	sa	bez	sa
Sesije	204	216	398	441	722	839
Nastavnici	0	39	0	81	0	112
Promenljive	15 348	41 036	30 026	88 223	54 652	144 685
Ograničenja	27 857	72 082	54 424	155 803	98 974	257 191
Vreme konstrukcije	0,09 s	0,24 s	0,18 s	0,52 s	0,35 s	0,94 s

Kolone označene sa „bez“ odnose se na konfiguraciju bez osoblja, a kolone „sa“ na konfiguraciju sa raspoređivanjem nastavnog osoblja.

Tabela 8.2 sumira tok rešavanja: kada je nađeno prvo rešenje, kolika je bila njegova vrednost cilja, kada je dostignut najbolji rezultat i koliko je poboljšanja usput pronađeno.

8.3 Funkcija cilja u zavisnosti od vremena

Očitavanje funkcije cilja u unapred zadatim vremenskim presecima (5 s, 10 s, 20 s, 30 s, 60 s, 120 s, 300 s i 600 s) ovde ne govori mnogo: u skoro svakom preseku rešavač ili još nema nijedno dopustivo rešenje, ili je već dostigao optimum, a najkasnije od preseka na 300 s sve konfiguracije imaju cilj jednak nuli. Pošto se sva poboljšanja dešavaju u uskom intervalu odmah po nalaženju prvog rešenja, gruba mreža preseka ne pokazuje dinamiku pretrage. Zato je na slici 8.1 data cela putanja poboljšanja za konfiguraciju sa osobljem, sa po jednim panelom za svaku skalu i vremenskom

Tabela 8.2: Tok rešavanja proširenog modela

	FINAL-S		FINAL-M		FINAL-L	
Metrika	bez	sa	bez	sa	bez	sa
Prvo rešenje	1,59 s	2,69 s	4,86 s	9,06 s	23,92 s	30,03 s
Cilj prvog rešenja	1 355	585	1 009	1 849	2 440	2 973
Broj nađenih rešenja	23	10	15	38	23	56
Vreme do najboljeg	2,35 s	3,79 s	6,94 s	16,37 s	40,84 s	120,44 s
Ukupno vreme	2,46 s	4,05 s	7,15 s	16,95 s	41,23 s	121,76 s
Najbolji cilj	0	0	0	0	0	0
Donja granica	0	0	0	0	0	0
Status	OPTIMAL					
Rešenje validno	da					

osom ograničenom na interval u kome se pretraga zaista odvija. Kriva je stepenasta jer vrednost u trenutku t ostaje jednaka cilju poslednjeg do tada nađenog rešenja, a svaka tačka na krivoj označava jedno poboljšanje. Konfiguracija bez osoblja prolazi kroz istu vrstu putanje (23, 15 i 23 pronađena rešenja po skalama), samo je kraća i celom dužinom se odvija u prvih nekoliko sekundi, pa je ne prikazujemo posebno.

Napomena: prikazani rezultati dobijeni su komandom:

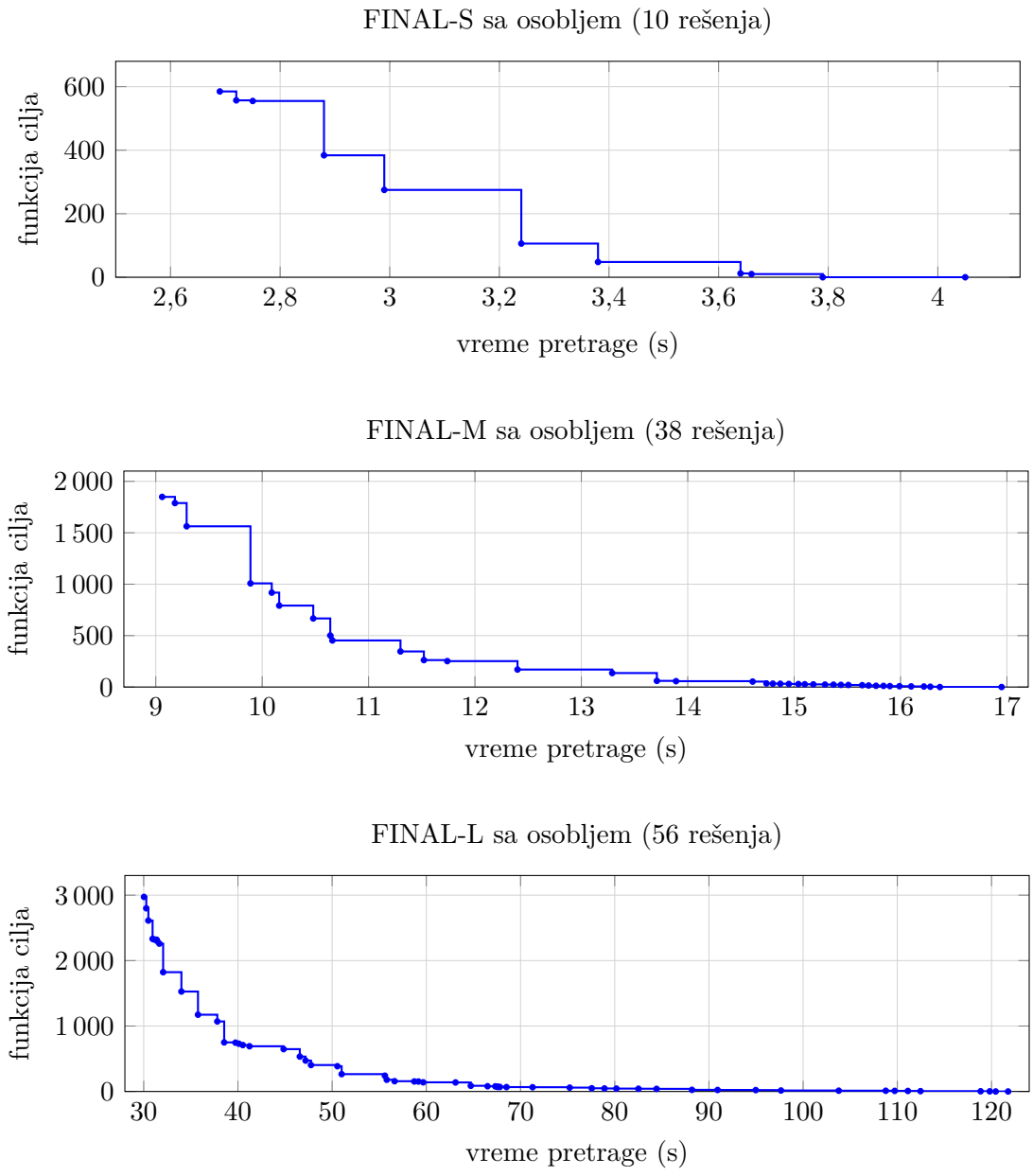
```
bazel run //src/algo:benchmark -- --mode final --max-time 600 \
    --final-json report_final.json
```

8.4 Zaoštreno okruženje

Rezultati iz prethodnog odeljka pokazuju da rešavač na svim skalama dostigne cilj 0, i to za nekoliko desetina sekundi, najviše dva minuta. Razlog je što realna instanca ima dovoljno slobodnog prostora: 29 učionica puta 60 termina nedeljno daje 1740 mesta za najviše 839 časova, dakle popunjenost ispod 50%. U tako labavom rasporedu meka ograničenja retko dolaze u stvarni sukob, pa kriva funkcije cilja sigurno padne na nulu i o granicama modela govori malo.

Da bismo videli kako se model ponaša kada resursi postanu oskudni, uvodimo *zaoštreno okruženje* – istu instancu sa dve izmene:

- nastava traje od 8 do 15 časova umesto do 20, čime broj termina pada sa 1740 na 1015, a popunjenost najveće instance raste na 83%;



Slika 8.1: Putanja poboljšanja funkcije cilja u realnom okruženju, po skalama, u konfiguraciji sa osobljem. Kriva počinje u trenutku prvog dopustivog rešenja i završava se u nuli, kada je optimalnost dokazana.

- podrazumevani najveći broj radnih dana nastavnika smanjen je sa 4 na 2 (lični limiti pojedinih nastavnika, koji su stroži, ostaju nepromenjeni).

Dužina dana je birana tako da problem ostane jedva rešiv. Skraćenje za još jedan čas (8–14, dakle 870 termina i 96% popunjenosti) dovodi najveću instancu sa osobljem preko granice rešivosti: rešavač za 600 sekundi ne nađe *nijedno* dopustivo rešenje i vraća status UNKNOWN. Zona u kojoj je ovaj problem istovremeno rešiv i netrivijalan je, dakle, uska – meri se jednim časom nastave dnevno.

Tabela 8.3 prikazuje tok rešavanja u tako zaoštrenom okruženju.

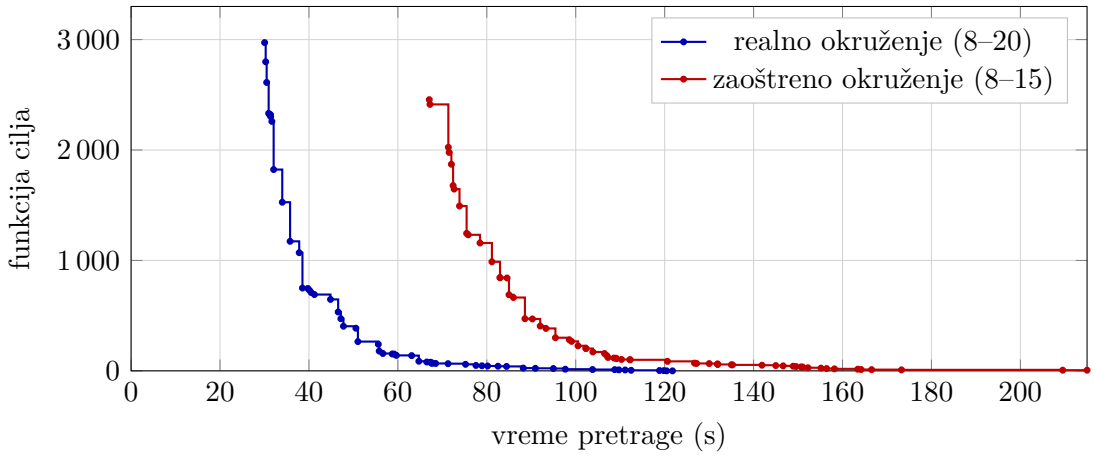
Tabela 8.3: Tok rešavanja u zaoštrenom okruženju (8–15 časova)

	FINAL-S		**FINAL-M**		**FINAL-L**	
Metrika	bez	sa	bez	sa	bez	sa
Promenljive	9 448	24 511	18 476	52 648	33 602	86 685
Ograničenja	16 657	42 782	32 524	92 453	59 124	152 891
Prvo rešenje	1,01 s	1,63 s	4,01 s	7,18 s	43,26 s	67,06 s
Cilj prvog rešenja	744	684	1 837	1 677	2 545	2 456
Broj nađenih rešenja	18	16	14	23	24	62
Vreme do najboljeg	1,35 s	2,73 s	6,51 s	14,47 s	80,31 s	209,53 s
Najbolji cilj	0	3	0	6	0	6
Donja granica	0	3	0	6	0	6
Status			OPTIMAL			
Rešenje validno			da			

Za razliku od realnog okruženja, ovde najbolji cilj *nije* nula: na najmanjoj skali ostaje 3, a na srednjoj i najvećoj 6. Pošto se donja granica poklapa sa nađenim rešenjem, reč je o dokazano najboljem mogućem kompromisu – u zbijenoj nedelji prosto nije moguće da svi nastavnici stanu u dva radna dana bez ijednog procepa kod grupa.

I u zaoštrenom okruženju cilj u vremenskim presecima ostaje gotovo konstantan: male i srednje skale već u prvih dvadesetak sekundi dođu na svoju optimalnu vrednost i tu ostanu do isteka vremena. Izuzetak je pokretanje FINAL-L sa osobljem, koje je ono što se u ovakvim merenjima obično i očekuje: prvo rešenje se pojavi tek posle nešto više od minuta, a cilj zatim pada sa 472 na 100 i konačno na 6, kroz gotovo tri i po minuta pretrage. Slika 8.2 poredi putanju poboljšanja za to pokretanje sa putanjom istog modela u realnom okruženju.

Zaoštreno okruženje se dobija dodavanjem opcije `--profiletight`:



Slika 8.2: Putanja poboljšanja za FINAL-L sa osobljem u dva okruženja. U zaoštrenom okruženju prvo dopustivo rešenje se pojavljuje više od dva puta kasnije, a pretraga se zaustavlja na dokazano optimalnoj vrednosti 6 umesto na nuli.

```
bazel run //src/algo:benchmark -- --mode final --profile tight \
    --max-time 600 --final-json report_final_tight.json
```

8.5 Analiza rezultata

Cena proširenja modela

Uvođenje nastavnog osoblja model uvećava oko dva i po puta: broj promenljivih raste 2,6 do 2,9 puta, a broj ograničenja 2,6 do 2,9 puta na svim skalama. Deo tog rasta dolazi od pomoćnih promenljivih kojima se izražavaju pravila osoblja, a deo od toga što instanca sa osobljem opisuje *više* nastave: na najvećoj skali 839 naspram 722 časa.

Razlog za veći broj časova je to što se grupe izvode iz dodela nastave (odeljak 7.2). Bez fajla sa osobljem upis se deli na grupe od najviše 50 studenata, pa najveća skala ima 30 grupa; dodele nastave, međutim, opisuju stvarnu podelu na 44 grupe (na primer, prva godina Informatike ima šest grupa vežbi umesto dve). Više grupa znači i više časova vežbi.

Pojam kohorte iz odeljka 7.3 radi u suprotnom smeru: dodele oblika `„groupIds": [„1i1a", „1i1b", „1i1v"]` spajaju više grupa u jedan zajednički čas, pa nastavnik isto predavanje ne drži više puta. Na najvećoj skali 139 od 839 časova su takve

zajedničke sesije. Zbog toga broj časova raste 16% iako broj grupa raste 47% – spajanje predavanja pojede veći deo razlike.

Ponašanje funkcije cilja kroz vreme

U obe konfiguracije prvo pronađeno rešenje je daleko od optimuma, i u obe rešavač cilj spusti do nule. Bez osoblja početna vrednost je 1355, 1009 i 2440 na skalama S, M i L, a do nule se stiže kroz 23, 15 i 23 pronađena rešenja; sa osobljem početne vrednosti su 585, 1849 i 2973, a poboljšanja ima 10, 38 i 56.

Sastav početne kazne se pri tome menja sa konfiguracijom. Bez osoblja u njoj su samo procepi u rasporedu grupa i studenti bez mesta u učionici; sa osobljem se na to dodaju i nastavnici koji rade više dana nego što im je dozvoljeno, pa je početna kazna na srednjoj i najvećoj skali osetno veća. Na najmanjoj skali je obrnuto (585 naspram 1355), što pokazuje da te dve konfiguracije nisu ni ista instanca: sa osobljem se raspoređuje 216 časova podeljenih na 11 grupa, bez osoblja 204 časa na 8 grupa.

Poboljšanja su gusta i neravnomerna – najveći skokovi dešavaju se rano (na primer, sa 2259 na 1823 u svega 0,4 sekunde na skali L sa osobljem), dok poslednji koraci do nule traju duže jer je preostale prekršaje sve teže ukloniti: poslednjih 65 poena cilja na najvećoj skali oduzima gotovo pola minuta pretrage.

Skalabilnost

Vreme do prvog dopustivog rešenja raste sa veličinom instance (1,59 s, 4,86 s i 23,92 s bez osoblja, odnosno 2,69 s, 9,06 s i 30,03 s sa osobljem), što je očekivano jer raste i broj sesija koje treba smestiti u isti broj termina i učionica. Ukupno vreme do dokazano optimalnog rešenja u konfiguraciji sa osobljem iznosi 4,05 s, 16,95 s i 121,76 s – rast je nadlinearan, pa se vidi da najveća skala već traži znatan deo posla u fazi optimizacije, a ne samo u traženju prvog rešenja.

Bitno je da su svi rezultati dobijeni sa statusom OPTIMAL, dakle donja granica se poklopila sa nađenim rešenjem. Vremenski limit od 600 sekundi ni na jednoj skali nije ni približno iskorišćen: najsporije pokretanje završeno je za oko dva minuta, dakle unutar 21% raspoloživog vremena. Prošireni model, dakle, ne samo da ostaje rešiv posle svih dodatih ograničenja, nego i dalje ima rezervu za buduća proširenja navedena u glavi 9.

Uticaj oskudice resursa

Poređenje dva okruženja pokazuje da težina ovog problema mnogo više zavisi od količine raspoloživih termina nego od broja pravila. Skraćenje radnog dana sa dvanaest na sedam časova ne menja nijedno ograničenje modela, a vreme rešavanja najveće instance sa osobljem raste sa 122 na 210 sekundi, dok cilj prestaje da bude dostižan u nuli.

Najviše se, međutim, produži traženje *prvog* dopustivog rešenja: na najvećoj skali sa osobljem sa 30 na 67 sekundi, a bez osoblja sa 24 na 43 sekunde. To je i suština razlike između dva okruženja – u realnom je težak deo posla optimizacija, a u zaoštrenom sama dopustivost. Na manjim skalama je efekat blaži (FINAL-S sa osobljem: 2,69 s naspram 1,63 s), jer prva godina i u kratkom danu ima dovoljno prostora.

Vredi primetiti i da je početna kazna u zaoštrenom okruženju *manja* nego u realnom (2456 naspram 2973 na najvećoj skali), iako je okruženje strože. U kratkom danu jednostavno ima manje prilika za procepe. Razlika je u tome što se ta manja kazna ne može svesti na nulu: u realnom okruženju cilj je velik ali potpuno uklonjiv, a u zaoštrenom je manji ali sadrži nesvodiv ostatak od 3 poena na najmanjoj i 6 na srednjoj i najvećoj skali.

Konačno, granica rešivosti je bliža nego što se iz ovih brojeva vidi. Dan od 8 do 15 časova daje 1015 termina za 839 časova, dakle 83% popunjenosti, i rešavač tu još dokazuje optimalnost. Jedan čas kraći dan (870 termina, 96% popunjenosti) menja sliku iz temelja: rešavač za 600 sekundi ne nađe nijedno dopustivo rešenje. Prelaz od dokazano optimalnog rešenja do nemogućnosti da se nađe bilo kakvo mera se, dakle, jednim časom nastave dnevno.

Validnost

Na svih dvanaest pokretanja, u oba okruženja, nezavisna provera potvrđuje da su tvrda ograničenja zadovoljena: nema dva časa u istoj učionici u istom terminu, nijedna grupa nema dva časa istovremeno, časovi koji zahtevaju računare su u odgovarajućim učionicama, a nijedan nastavnik nije dvostruko zauzet.

Glava 9

Zaključak i dalji rad

9.1 Zaključak

U ovom radu smo razmatrali problem automatskog generisanja rasporeda časova na primeru Matematičkog fakulteta Univerziteta u Beogradu. Implementirana su dva pristupa: programiranje ograničenja (CP-SAT) i mešovito celobrojno programiranje (MIP/SCIP), oba korišćenjem biblioteke Google OR-Tools.

Eksperimentalna evaluacija na tri skale realnih podataka pokazala je sledeće:

1. **CP-SAT je značajno kompaktniji.** Sa svega 5 celobrojnih promenljivih po sesiji naspram hiljada binarnih promenljivih u MIP-u, CP model zauzima znatno manje memorije i brže se konstruiše.

2. **CP-SAT je brži.** Zahvaljujući globalnim ograničenjima sa specijalizovanom propagacijom i tehnikama učenja konflikata iz SAT rešavača, CP-SAT nalazi dopustivo rešenje za redove veličine brže od MIP/SCIP rešavača.

3. **CP-SAT bolje skalira.** Na većim instancama problema, prednost CP-SAT pristupa postaje izraženija, dok MIP/SCIP može da ne nađe rešenje u razumnom vremenskom roku.

4. **Oba rešavača daju validna rešenja.** Nezavisna validacija potvrđuje da oba pristupa, kada nađu rešenje, zadovoljavaju sva tvrda ograničenja.

Na osnovu ovih rezultata, CP-SAT je izabran kao osnova za dalju nadogradnju sistema za raspoređivanje časova. Nadogradnja je sprovedena u dva koraka. U glavi 6 uvedena je arhitektura pravila (eng. *plugin*) sa podrškom za tvrda i meka ograničenja

i funkcija cilja koja minimizuje težinsku sumu prekršaja, zajedno sa četiri nova ograničenja kvaliteta – spajanje časova u blokove, jedna lokacija po danu za grupu, izbegavanje procepa u rasporedu i kapacitet učionice kao meko ograničenje. U glavi 7 model je proširen nastavnim osobljem: uveden je pojam kohorte i zajedničkih sesija (kojima se izbegava da nastavnik drži isti čas više puta), tvrdo ograničenje da nastavnik nije dvostruko zauzet, i tri pravila osoblja – jedna lokacija po danu, najveći broj radnih dana i nedeljni budžet procepa. Time je pokazano da se kompaktni CP-SAT model lako i modularno proširuje ka realnim potrebama Matematičkog fakulteta, uz kapacitet učionica i ograničenja nastavnika.

Završna evaluacija iz glave 8 pokazala je da prošireni model ostaje praktično upotrebljiv. Iako raspoređivanje nastavnog osoblja otprilike udvostručuje veličinu modela, na najvećoj instanci (sve četiri godine studija, 685 časova) rešavač nalazi dokazano optimalan raspored – bez ijednog procepa u rasporedu grupa i bez ijednog nastavnika preko dozvoljenog broja radnih dana – za oko 31 sekundu, dakle unutar svega nekoliko procenata zadatog limita od 600 sekundi. Praćenje funkcije cilja kroz vreme pokazuje i da se najveći deo kvaliteta rasporeda dobije u prvih nekoliko sekundi nakon prvog dopustivog rešenja, što ostavlja prostora za dalja proširenja modela.

Merenje u zaoštrenom okruženju, u kome je nastava skraćena na šest časova dnevno, pokazalo je da težina problema mnogo više zavisi od oskudice termina nego od broja pravila: isti model tada do najboljeg rešenja stiže za 174 sekunde umesto za 31, a najbolji cilj više nije nula nego dokazano najmanja moguća vrednost 6. Zona u kojoj je problem istovremeno rešiv i netrivijalan pritom je uska – pri smanjenju broja termina za još 17% rešavač ne nalazi nijedno dopustivo rešenje ni za 600 sekundi.

9.2 Dalji rad

Kroz glave 6 i 7 implementirani su meka ograničenja sa funkcijom cilja, lokacijska ograničenja za grupe i nastavnike, grupisanje časova u blokove, raspoređivanje nastavnog osoblja i kapacitet učionica kao meko ograničenje. Preostali pravci daljeg rada su:

1. **Raspoloživost i preferencije nastavnika.** Uključivanje termina u kojima nastavnik ne može ili ne želi da drži nastavu, kao i mekih preferencija (na primer, prepodnevni termini).

2. **Dodatna meka ograničenja kvaliteta.** Na primer, ravnomerna raspodela časova kroz nedelju i izbegavanje kasnih termina, uz fino podešavanje težina u funkciji cilja i analizu njihovog međusobnog odnosa.

3. **Veb interfejs.** Izgradnja veb aplikacije koja omogućava unos podataka, pokretanje rešavača i vizuelni prikaz rasporeda, korišćenjem FastAPI okruženja za serversku stranu.

Literatura

[1] Tobias Achterberg. "SCIP: Solving Constraint Integer Programs". English. In: *Mathematical Programming Computation* 1.1 (2009), pp. 1–41. DOI: 10.1007/s12532-008-0001-1.

[2] Krzysztof R. Apt. *Principles of Constraint Programming*. English. Cambridge University Press, 2003.

[3] Roberto Asín Achá and Robert Nieuwenhuis. "Curriculum-based Course Timetabling with SAT and MaxSAT". English. In: *Annals of Operations Research* 218.1 (2014), pp. 71–91. DOI: 10.1007/s10479-012-1081-x.

[4] Milan Banković. "Solving Finite-Domain Linear Constraints in Presence of the `alldifferent`". English. In: *Logical Methods in Computer Science* 12.3 (2016), pp. 1–28. DOI: 10.2168/LMCS-12(3:5)2016.

[5] Roman Barták, Miguel A. Salido, and Francesca Rossi. "Constraint Satisfaction Techniques in Planning and Scheduling". English. In: *Journal of Intelligent Manufacturing* 21.1 (2010), pp. 5–15. DOI: 10.1007/s10845-008-0203-4.

[6] Edmund K. Burke and Sanja Petrovic. "Recent Research Directions in Automated Timetabling". English. In: *European Journal of Operational Research* 140.2 (2002), pp. 266–280. DOI: 10.1016/S0377-2217(02)00069-3.

[7] Geoffrey Chu. "Symmetries and Lazy Clause Generation". English. In: *CP'10 Doctoral Programme Proceedings, 16th International Conference on Principles and Practice of Constraint Programming*. St Andrews, UK, 2010, pp. 43–48.

[8] S. Even, A. Itai, and A. Shamir. "On the Complexity of Timetable and Multicommodity Flow Problems". English. In: *SIAM Journal on Computing* 5.4 (1976), pp. 691–703. DOI: 10.1137/0205048.

[9] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. English. W. H. Freeman, 1979.

[10] Google LLC. *OR-Tools – Google Optimization Tools*. English. on-line at: `https://developers.google.com/optimization`. 2024.

[11] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. English. John Wiley & Sons, 1988.

[12] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. "Propagation via Lazy Clause Generation". English. In: *Constraints* 14.3 (2009), pp. 357–391. DOI: `10.1007/s10601-008-9064-x`.

[13] Laurent Perron and Frédéric Didier. *CP-SAT Solver*. English. on-line at: `https://developers.google.com/optimization/cp/cp_solver`. 2023.

[14] Jean-Charles Régin. "A Filtering Algorithm for Constraints of Difference in CSPs". English. In: *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press, 1994, pp. 362–367.

[15] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. English. Elsevier, 2006.

[16] Toby Walsh. "General Symmetry Breaking Constraints". English. In: *Principles and Practice of Constraint Programming – CP 2006*. Vol. 4204. Lecture Notes in Computer Science. Springer, 2006, pp. 650–664. DOI: `10.1007/11889205_46`.

[17] Laurence A. Wolsey. *Integer Programming*. English. John Wiley & Sons, 1998.

Biografija autora

Dušan Trtica rođen je 1983. godine u Beogradu. Završio je Devetu beogradsku gimnaziju 2002. godine. Matematički fakultet Univerziteta u Beogradu, smer informatika i računarstvo, završio je 2009. godine. Oblasti interesovanja uključuju razvoj softvera, funkcionalno programiranje, kao i algoritme veštačke inteligencije.