Izrada rasporeda ispita na PMF-U

Uroš Bojić

30 June 2023

Sadržaj

Uvod

Opis problema

Izrada rasporeda ispita na fakultetu je veoma složen zadatak koji zahtijeva pravilno raspoređivanje ispita, gdje pri tome imamo ograničene resurse. Cilj je stvoriti optimalan raspored koji zadovoljava različite zahtjeve i ograničenja kako bi se osiguralo učinkovito i ravnomjerno raspoređivanje ispita na obraznovnoj ustanovi.

Raspored ispita mora uzeti u obzir različite faktore kao što su dostupni termini i opterećenost studenta. Osim toga, treba uzeti u obzir i eventualne konflikte kao što su preklapanje termina. Ovaj problem je posebno izazovan jer je potrebno postići ravnotežu između različitih zahtjeva, dok pri tome treba uskladiti raspored da ne dolazi do preklapanja ili konflikta između različitih termina ispita.

Izrada rasporeda ispita na fakultetu je operacija koja može biti vremenski veoma zahtjevna i podložna greškama ako se obavlja ručno. Ručno raspoređivanje često zahtijeva mnogo vremena i truda, a moguće je i da se pojave greške kao rezultat greške osoblja fakulteta ili kao rezultat same složenosti zadatka. Stoga se koriste različiti algoritmi i metode raspoređivanja kako bi se olakšao proces i postigao optimalan raspored ispita.

Važno je napomenuti da je izrada rasporeda ispita na fakultetu NP-težak problem, što znači da nije moguće pronaći brzo i jednostavno optimalno rješenje. Različiti faktori i ograničenja doprinose složenosti problema, pa je potrebno primijeniti napredne algoritme i heuristike kako bi se pronašlo prihvatljivo rješenje u razumnom vremenskom okviru.

Problem izrade rasporeda ispita se može definisati na sljedeći način:
Imamo definisan skup disjunktnih termina T={ t_1, t_2, ..., t_m}. Za svaki termin poznat je datum kada se održava, vrijeme kada počinje, te kapacitet termina (koliko studenata termin da može prihvati). Razliku između termina definišemo kao razliku broja dana između dva termina. Ukoliko se termini održavaju na isti dan tada je razlika između njih jedan.

Definisan je skup predmeta P={p_1, p_2, ..., p_n}. Svaki predmet moguće je polagati samo u jednom terminu.

Definisana je relacija zajedno: P×P→{0,1}, na sljedeći način: predmeti pi i pj su u relaciji zajedno ako su oba biti smještena u isti termin.

Definisan je skup studenata S={ s_1, s_2, ..., s_k}.

Definisana je relacija sluša P×S→{0,1}, na sljedeći način: predmet p_i i student s_j su u relaciji sluša, ako student s_j sluša predmet p_i, i stoga treba polagati ispit iz tog predmeta. Definisana je relacija prihvatljivo s P×T→{0,1}, na sljedeći način: predmet p_i i termin t_j su u relaciji prihvatljivo ako je termin t_j prihvatljiv za održavanje ispita predmeta p_i.

Zadatak je izrada rasporeda ispita, gdje niti u jednom terminu ti ne postoje dva predmeta c_k i c_l dodijeljena tom terminu koji imaju zajedničkog studenta, te da ne postoji termin u kojem je broj smještenih studenata veći od kapaciteta termina.

Kvalitet potencijalnih rješenja se definiše na sljedeći način:

- jako je loše ako postoje studenti koji u istom danu imaju više ispita (ovo je moguće jer u danu postoji više termina kada se mogu polagati ispiti). Što je više ovakvih sukoba to je raspored gori.
- loše je ako postoje studenti koji imaju ispite u susjednim danima (npr. dva za redom) međutim ovo nije toliko loše, koliko je loše kao kad imaju ispite u istom danu).

Pregled radova drugih istraživača

U radu pod nazivom "*An integer programming approach to curriculum-based examination timetabling*" [1], objavljenom u novembru 2017. godine, primijenjena je tehnika Integer programming uz korištenje Python i R programskih jezika. Ovaj pristup ima prednost u većoj vjerovatnoći i efikasnosti u pronalaženju najboljih optimizacija u okviru rasporeda ispita. Međutim, nedostatak ovog pristupa je što ne rješava probleme rasipanja, odnosno ne uzima u obzir konflikte između ispita.

Rad "*Algorithms for optimizing fleet staging of air ambulances*" [2], objavljen je u septembru 2020. godine i u njemu je korišten je algoritam tabu pretrage uz korištenje okvira poput Tabu Search Framework-a. Ovaj pristup kombinuje prednosti kolonija mrava i imunološke teorije kako bi osigurao efikasnost u optimizaciji rasporeda flote vazdušnih ambulanti. Međutim, implementacija ovog pristupa je veoma kompleksna.

"*Elite Immune Ant Colony Optimization-Based Task Allocation for Maximizing Task Execution Efficiency in Agricultural Wireless Sensor Networks*" [3] je rad iz 2020. godine, te se u njemu se primjenjuje algoritam tabu pretrage i to u Matlab programskom jeziku. Ovaj pristup postepeno konvergira ka globalnom optimalnom rješenju kako bi izbjegao lokalna rješenja. Međutim, loše se ponaša u kontekstu nesavršene optimizacije.

U radu "*Intelligent timetable scheduler: A comparison of genetic, graph coloring, heuristic and iterated local search algorithms*"[4], predstavljenom na ICAC (International Conference on Advancements in Computing) 2019. godine, proučavane su četiri metode za izradu rasporeda: genetski algoritam, algoritam bojenja grafova, heuristički algoritam i algoritam iterativne lokalne pretrage.

- Genetski algoritam je pokazao fleksibilnost i sposobnost za globalnu optimizaciju, koristeći paralelnu pretragu i iterativno poboljšanje. Međutim, ovaj pristup je vremenski zahtjevan i efikasnost mu zavisi od izbora parametara i načina reprezentacije problema. Također postoji mogućnost konvergencije ka suboptimalnim rješenjima.
- Algoritam bojenja grafova se istakao po jednostavnoj implementaciji i brzini izvršavanja. On je posebno koristan za identifikovanje konflikata u rasporedu. Međutim, ova metoda je manje prilagodljiva za složenije instance problema raspoređivanja.
- Heuristički algoritam je jednostavan za implementaciju i brz u izvršavanju. Ova metoda može biti prilagodljiva i fleksibilna u odnosu na specifične zahtjeve rasporeda. Međutim, treba uzeti u obzir da heuristički algoritmi obično ne garantuju optimalno rješenje u poređenju s drugim pristupima.
- Algoritam iterativne lokalne pretrage postepeno poboljšava trenutno rješenje putem lokalnih promjena. On je efikasan u pronalaženju blizu-optimalnih rješenja. Međutim, ovaj pristup može zapeti u lokalnom minimumu i ne pruža garanciju globalne optimizacije.

Opis rješenja

Iterated Local Search

Opšti rad algoritma može se opisati kao:

1. Inicijalizacija: Algoritam započinje inicijalizacijom početnog rasporeda ispita. Početni raspored je generisan tako što prolazimo kroz svaki termin i svaki predmet i pokušavamo pronaći prihvatljivu kombinaciju. Također, se vrši početna procjena fitness-a rasporeda.

2. Iteriranje: Algoritam izvodi iteraciju određen broj puta (koji unapred definišemo). U svakoj iteraciji petlje, algoritam nastoji poboljšati trenutni raspored ispita.

3. Generiranje kandidata: U svakoj iteraciji petlje, algoritam generiše nasumičnog kandidata za novi raspored ispita. Novi kandidat je neki modifikirani oblik trenutnog rasporeda, koji dobijamo dodavanjem nasumičnog ispita umjesto postojećeg.

4. Lokalna pretraga: Algoritam primjenjuje lokalnu pretragu na generisanog kandidata kako bi pronašao bolji raspored ispita. Postavljamo ovog kandidata kao novi najbolji raspored. Vršimo unapred određen broj iteracija gdje tražimo susjede od kandidate, te biramo nasumičnog susjeda. Ako je on bolji od generisanog kandidate postavljamo njega kao novi najbolji raspored. Vraćamo novi najbolji raspored.

5. Procjena: Nakon lokalne pretrage, algoritam izračunava fitness novog rasporeda ispita. Ako je novi raspored bolji, ažuriraju se najbolji raspored i njegov fitness.

6. Povrat najboljeg rasporeda: Nakon što je glavna petlja završena, algoritam vraća najbolji raspored ispita koji je pronađen tokom izvođenja algoritma.

```
function find_schedule():
    best_schedule <- find_initial_schedule()
    best_fitness <- calculate_schedule_fitness(best_schedule)
    repeat number_of_iteration times:
        candidate_schedule <- generate_candidate(best_schedule)
        improved_schedule <- local_search(candidate_schedule)
        potential_best <- calculate_fitness(improved_schedule)
        if potential_best < best_fitness:
            best_fitness <- potential_best
            best_schedule <- improved_schedule
    return best_schedule
```

Pseudokod algoritma iterativne lokalne pretrage

Napomena: Fitness rasporeda računamo tako što za svaki ispit u rasporedu gledamo da li stvara konflikt, ako stvara konflikt kažnjavamo ga sa “velikom kaznom”, ako ne stvara konflikt gledamo opterećenost rasporeda, te ako je raspored opterećen kažnjavamo ovaj raspored “malom kaznom”. Velika i mala kazna su unapred definisane vrijednosti. Bolji rasporedi su oni koji imaju manji fitnes.

Genetski algoritam

Opšti rad genetskog algoritma može se opisati sljedećim koracima:

1. Inicijalizacija: Generisanje početne populacije kandidata rješenja. Svaki kandidat predstavlja raspored ispita. Početnu populaciju čine razne modifikacije prvog prihvatljivog rasporeda kojeg pronađemo.

2. Procjena fitness-a: Definišemo funkcijeu koja ocjenjuje kvalitet svakog kandidata. Funkcija uzima u obzir prilagođenost ispita, tako i ukupnu prilagođenost rasporeda. Procjena fitness-a kadidate se vrši na isti način kao u prethodno opisanom algoritmu.

3. Selekcija: Vrši odabir jedinki iz populacije za kreiranje nove populacije u sljedećoj generaciji. Selekcija uzima u obzir fitness jedinki, dajući veće vjerovatnoće odabira onima s boljim fitness-om. U ovom slučaju selekcija bira 20% jedinki s najboljim fitness-om i zadržava ih za sljedeću generaciju, dok se ostale odbacuju.

4. Ukršatanje (Crossover): Vršimo ukrštanje nad odabranim jedinkama kako bi se generisala potomstva. Ukrštanje podrazumeva kombinovanje genetskog materijala dva roditelja kako bi se stvorile nove jedinke. Mi koristimo jednostavno ukrštanje u jednoj tački (simple one-point crossover), a vjerovatnoća da će se izvršiti ukrštanja se proverava pomoću unapred definisanog parametra CROSSOVER_RATE.

5. Mutacija: Primjenjujemo operaciju mutacije radi uvođenja malih nasumičnih promjena u potomcima. Mutacija pomaže istraživanju novih regiona prostora rješenja. Hoće li se mutacija izvršiti se proverava osnovu unapred definisanog parametra MUTATION_RATE.

6. Kriterijum zaustavljanja: Algoritam se zaustavlja nakon određenog broja generacija. Broj generacija je unapred definisan parametar.

7. Iteracija: Ponavljanje koraka od 3 do 6 određeni broj puta (broj generacija) kako bi se postigao bolji kvalitet rešenja.

8. Najbolje rješenje: Nakon završetka iteracija dobijamo najbolji raspored. Najbolji raspored je jedinka iz posljednje populacije koja ima najbolju fitness vrijednost.

```
function genetic_algorithm():
    population <- initialize_population()
    best_schedule <- null
    repeat number_of_generations times:
        selected_population <- selection(population)
        offspring_population <- crossover(selected_population)
        mutated_population <- mutation(offspring_population)
        population <- mutated_population
    best_schedule <-  get_best_from_population(best_schedule)
    return best_schedule

function selection(population):
    elite <- select_top_percentage(population)
    return elite
```

```
function crossover(population):
    new_population <- population
    while new_population_size < population_size:
        parent1, parent2 <- randomly_select_parents(population)
        if should_do_crossover():
            offspring <- perform_crossover(parent1, parent2)
            new_population.add(offspring)
    return new_population

function mutation(population):
    repeat for every individual in population:
        if should_do_mutation():
            perform_mutation(individual)
```

Pseudokod genetskog algoritma

Eksperimentalni rezultati

Instanca	Iterated Local Search		Genetski algoritam		Solver	
Naziv	rezultat	t[ms]	rezultat	t[ms]	rezultat	t[ms]
small_1	0	64	0	75	0	7
small_2	200	57	200	76	200	5
small_3	200	75	200	122	200	7
small_4	100	63	100	112	100	6
small_5	100	88	500	146	100	6
big_1	0	246	200	396	0	11
big_2	0	226	200	987	0	16
big_3	0	267	100	429	0	12
big_4	0	248	0	383	0	17
big_5	200	238	100	451	0	11

Napomena: Eksperimentalno okruženje u kojem je testiran program:

Operativni sistem: MacOS

CPU: Apple M1

Broj niti (threads): 8 fizičkih jezgara, 8 logičkih procesora

Programski jezik: Python

Okruženje: Visual Studio Code

Instance

Generisano je ukupno 10 instanci. Male instance imaju 5 dostupnih termina kapaciteta između 25 i 50. One sadrže 3 ili 4 predmeta i 3 grupe studenata. Svaka grupa studenta sadrži između 10 i 20 studenata koji slušaju od jedan ili više predmeta. Velike instance imaju 20 dostupnih termina, 10 predmeta, te 3 ili 4 grupe studenata. Svaka grupa sastoji se od 10 do 20 studenata i oni slušaju do 5 kurseva. Sve instance su generisane nasumično.
Napomena: Svi termini počinju između 08:00 i 16:00, za manje instance svi termini su od 01. do 05. jula, dok su termini za velike instance od 01. do 07. jula. Svaki od ispita traje između 1h i 4h 45m.

Zaključak

Izrada rasporeda ispita na fakultetu je veoma složen zadatak koji zahtijeva pravilno raspoređivanje ispita uz ograničene resurse. U ovom radu smo predstavili tri pristupa, odnosno metode za rješavanje problema raspoređivanja ispita. Analizirani su radovi drugih istraživača koji su primijenili tehnike kao što su Integer programming, algoritam tabu pretrage, genetski algoritam, algoritam bojenja grafova i heuristički algoritam.
Iterated Local Search uglavnom ima bolje performanse nego genetski algoritam, međutim vidimo da u nekim slučajevima i ovaj pristup konvergira ka suboptimalnim rješenjima. I Iterated Local Search i genetski algoritam imaju relativno veliko vrijeme izvršavanja.

Dalje unapređenje rada može uključivati istraživanje i primjenu drugih heurističkih algoritama kao što je algoritam tabu pretrage, ali isto tako može uključivati i unapređenje iterativne lokalne pretrage i genetskog algoritma kako bi smanjili vrijeme izvršavanja i kako bi poboljšali efikasnost rješenja. Također, u ova dva pristupa moguće je dodati dodatne faktore i ograničenja u procjenu fitness-a, kao što su preferencije studenata, vremenski intervali između ispita ili maksimalna iskorišćenost prostora.

Literatura

- Michel Gendreau and Jean-Yves Potvin (2018). Handbook of Metaheuristics, Third Edition.
- Suman, B., Maniezzo, V., Ricciardelli, S. (2005). A Guided Tour of Combinatorial Optimization. Springer.
- Hayat A., Tahani A., Hosam A. and Abdullah B. (2020). A Review of Optimization Algorithms for University Timetable Scheduling
- Ender Özcan and Ersan Ersoy (2005). Final Exam Scheduler – FES
- Marko Čupić (2009). Prirodom inspirirani optimizacijski algoritmi

Reference:

[1] Alejandro Cataldo, Juan-Carlos Ferrer, Jaime Miranda, Pablo A. Rey & Antoine Sauré. (2017). An integer programming approach to curriculum-based examination timetabling

[2] Joseph Tassone a, Geoffrey Pond b, Salimur Choudhury (2020). Algorithms for optimizing fleet staging of air ambulances

[3] Zhenxing Zhang (2020). *Elite Immune Ant Colony Optimization-Based Task Allocation for Maximizing Task Execution Efficiency in Agricultural Wireless Sensor Networks*

[4] Tiny Wijerathna Ekanayake, Pavani Subasinghe, Shawn Ragel, Anjalie Gamage (2019). Intelligent timetable scheduler: A comparison of genetic, graph coloring, heuristic and iterated local search algorithms