Problem rasporeda ispita

Matija Halavanja

Veljača 2021.

Sadržaj

1 Uvod

Problemi rasporeda s ograničenjima spadaju u klasu poznatih problema u računalstvu koji su NP-teški. Za takve probleme ne postoje efikasni algoritmi pa se često koriste

metode pretraživanja skupa rješenja poput različitih heuristika. Jedna od jednostavnijih heuristika je i pretraživanje usponom na vrh (*hill climbing*) koju ćemo koristiti za rješavanje danog problema rasporeda. Druga metoda koju ćemo isprobati je metoda simuliranog kaljenja (*Simulated Annealing*). Na kraju ćemo usporediti rezultate jedne i druge metode.

Općenito, problem rasporeda se sastoji od nekih aktivnosti koje imaju zadana tvrda i meka ograničenja. Te aktivnosti treba rasporediti tako da su tvrda ograničenja ispunjena, a meka ograničenja da su što bliže optimumu. Odnosno, krajnje rješenje koje ne ispunjava tvrda ograničenja zapravo nije validno rješenje.

2 Kratki opis problema

Konkretan problem koji rješavamo u ovom seminaru je raspoređivanje kolegija (odnosno ispita iz danih kolegija) u moguće ponuđene termine. Tvrdo ograničenje u problemu je da ne postoji student s kolizijom, odnosno da ne postoji student s dva ispita u isto vrijeme. Meko ograničenje je da svaki student piše najviše jedan ispit u danu. Drugim riječima, želimo da ima što manje ispita kojih pišu studenti koji već imaju neki drugi ispit taj dan. Prije detaljnijeg opisa problema, uvest ćemo strukture koje se koriste pri rješavanju

3 Strukture podataka

Pri rješavanju problema koristimo tri strukture podataka, to su *Course*, *Student* i *Term*. Svaka od tih struktura ima varijablu *id* (tj. *courseId*, *courseId* i *termId*) koja jedinstveno označava instancu svake strukture. Id je ostvaren kao redni broj instance neke strukture pri stvaranju, tj. prva stvorena struktura Term ima id jednak 0, druga ima 1 itd. te id-evi za svaku pojedinu strukturu kreću od 0. Također, sve funkcije u daljnjim strukturama nemaju povratne vrijednosti već samo transformiraju unutarnje stanje instance strukture.

3.1 Term struktura

Struktura koja predstavlja termin pisanja ispita dana je sljedećim pseudokodom.

```
1 struct Term:
2     termId int
3     day int
```

Varijabla *day* predstavlja dan u kojem se odvija dani termin.

3.2 Course struktura

Struktura koja predstavlja kolegij dana je sljedećim pseudokodom.

```
1  struct COURSE:
2      courseId int
3      term TERM
4      students Student []
```

Varijabla *term* predstavlja termin u kojem se ispit piše. Lista *students* predstavlja studente koji pristupaju ispitu iz danog kolegija.

3.3 Student struktura

Struktura koja predstavlja studenta dana je sljedećim pseudokodom.

```
1  struct STUDENT:
2      studentId int
3      courses COURSE []
4      terms TERM []
5      numOfCourses int
6      numOfCollisionsForStudent int
7      numOfSameDayTerms int
8
9      function setTerms() → None
10     function calculateNumOfCollisionsForStudent() → None
11     function calculateNumOfSameDayTerms() → None
```

Lista *courses* predstavlja sve kolegije koje je student upisao. Lista *terms* predstavlja termine u kojima student ima ispite. Varijabla *numOfCourses* predstavlja broj upisanih kolegija studenta. Varijabla *numOfCollisionsForStudent* predstavlja broj kolizija u rasporedu studenta npr. ako student ima samo tri predmeta i svi su u istom terminu *numOfCollisionsForStudent* je jednak dva. Varijabla *numOfSameDayTerms* predstavlja broj ispita u istom danu npr. ako student ima četiri ispita, od toga dva u danu 0 i dva u danu 1, *numOfSameDayTerms* je jednak dva.

Funkcija *setTerms* prođe po listi *courses* i termin svakog kolegija kojeg je student upisao doda u listu *terms*, a zatim poziva sljedeće dvije funkcije. Funkcija *calculateNumOfCollisionsForStudent* izračuna broj kolizija za studenta i taj podatak spremi u varijablu *numOfCollisionsForStudent*. Funkcija *calculateNumOfSameDayTerms* izračuna broj ispita koje student piše u istom danu i spremi podatak u varijablu *numOfSameDayTerms*.

4 Programsko rješenje

4.1 Ulazni podaci

Obavezni ulazni podaci dani su sljedećom tablicom.

Tablica 1: Obavezni ulazni podaci i njihovo značenje

Naziv	**Značenje**
numOfStudents	broj studenata
numOfCourses	broj kolegija
numOfTerms	broj termina
numOfTermsPerDay	broj termina po danu
minNumOfCoursesPerStudent	min. broj kolegija svakog studenta
maxNumOfCoursesPerStudent	max. broj kolegija svakog studenta
mode	odabir metode koje koristimo
output	ime datoteke u koju ispisujemo izlaz

Predzadnji od obaveznih podataka, *mode*, služi za odabir metode kojom želimo rješavati problem. Moguće vrijednosti za *mode* su "CHC", "FHC" i "SA" (bez navodnika) gdje "CHC" i "FHC" redom označavaju klasičnu i brzu metodu uspona na vrh, a "SA" metodu simuliranog kaljenja. Metode ćemo objasniti u kasnijem dijelu seminara.

Osim obaveznih postoje i opcionalni ulazni podaci. Prvi takav je ***seed*** koji služi za inicijalizaciju generatora slučajnih brojeva. Ova opcija je korisna ako želimo usporediti metode na istim slučajno generiranim podacima. Ako pokrećemo program s u CHC ili FHC modu, do sad navedeni ulazni podaci su ujedno i svi mogući ulazni podaci.

Ako pokrećemo program u SA modu, moramo unijeti dodatne opcionalne parametre specifične za samu metodu simuliranog kaljenja. To su ***maxTemp*** odnosno maksimalna (početna) temperatura kaljenja, ***minTemp*** odnosno minimalna (završna) temperatura kaljenja te ***step*** odnosno korak kojim se smanjuje temperatura kaljenja. Minimalna temperatura ne može biti manja od 0, a ako se unese negativan broj, algoritam će raditi kao da je unesena temperatura jednaka 0.

4.2 Inicijalizacija podataka

Na početku programa inicijaliziraju se liste termina, kolegija i studenata koje se dalje koriste. Prilikom inicijalizacije termina, svakom terminu se osim id-a dodjeljuje i dan u kojem se piše. Dan u kojem se odvija termin t zadan je s $\lfloor \frac{id}{numOfTermsPerDay} \rfloor$, gdje varijabla *id*, pripada terminu t.

Nakon inicijalizacije termina, inicijaliziraju se kolegiji i svakom se kolegiju osim id-a dodaje i nasumično odabran termin pisanja ispita iz tog kolegija.

Nadalje, svaki student je upisan nasumično u neke kolegije, pri čemu je *maxNumOfCoursesPerStudent* najveći broj kolegija koje može upisati, a *minNumOfCoursesPerStudent* najmanji broj kolegija koje može upisati. Kolegijima postavljamo upisane studente, a studentima postavljamo upisane kolegije i pripadne termine. Prije početka pretraživanja prostora rješenja još izračunamo broj kolizija za svakog studenta.

4.3 Provjera korektnosti rješenja

Kako bi se ipak uvjerili da je krajnje rješenje korektno, na kraju svakog algoritma radimo određene provjere na manjoj razini apstrakcije kako bi se lakše uvjerili u korektnost. Nakon inicijalizacije napravimo rječnik oblika {studentId : coursesIdList}, odnosno za svakog studenta spremimo njegov id i listu id-eva kolegija koje je upisao. Po završetku algoritma, napravimo još jedan testni rječnik oblika {courseId : (term.termId, term.day)} odnosno, za svaki id kolegija, spremamo dan i termin u koji je smješten. Zatim, prolazeći po listi *students*, za svakog studenta provjerimo je li još uvijek ima iste kolegije upisane i jesu li broj kolizija i broj ispita u istom danu uistinu jednaki onima koje nam algoritam vrati.

4.4 Ispisi algoritma

Na kraju izvođenja algoritma, program ispiše krajnji rezultat u datoteku *out*, čije smo ime dali kao jedan od obaveznih parametara na početku izvršavanja programa. Ispis rezultata se sastoji od onoliko redaka koliko ima kolegija. Svaki pojedini redak je oblika "{id kolegija} {id termina}", dva id-a odvojena jednim razmakom, bez {} znakova. Tako spremljeno rješenje bi se dalje lako moglo koristiti u stvarnoj uporabi kada bi rješavali problem rasporeda ispita na nekom fakultetu. Tijekom izvođenja algoritma ispisuju se reci za svaku iteraciju algoritma u kojoj se postigne novo bolje rješenje od prethodnoga. Ti reci su oblika "{trenutni broj kolizija} {trenutni broj ispita na isti dan}", bez znakova. Ispis je ostavljen kako bi se pratilo napredovanje algoritma. Po završetku izvođenja algoritma, imamo još finalni ispis u konzoli koji kaže koliko je kolizija i ispita na isti dan te koliko je trajalo izvođenje algoritma.

5 Prikaz rješenja, susjedstvo i funkcija cilja

Prije nego što možemo reći nešto o samim metodama, moramo ustanoviti kako je prikazano rješenje, kako dobiti susjedstvo nekog rješenja i kako točno izgledaju funkcija cilja i funkcija cijene.

Rješenje iz skupa rješenja definirano je kao trenutni odabir termina svakog kolegija. Ako nekom kolegiju promijenimo termin pisanja ispita, dobivamo drugo rješenje. Rješenje ne ovisi o samim studentima jer tijekom pretrage skupa rješenja, studenti su upisani cijelo vrijeme u iste kolegije. Globalnu listu *students* ipak imamo radi jednostavnosti implementacije.

Susjedstvo danog rješenja je skup svih rješenja koja se mogu dobiti tako da jednom kolegiju u trenutnom rješenju postavimo neki novi termin pisanja, različit od trenutnog. Ako je n broj kolegija, a m broj termina, veličina prostora rješenja je m^n, a veličina susjedstva nekog specifičnog rješenja je $n \times (m - 1)$. Tako definirano susjedstvo je najintuitivnije i koristimo ga kod klasične metode uspona na vrh i kod metode simuliranog kaljenja, no kod brze metode uspona na vrh koristimo malo drugačije susjedstvo.

Kako bi dobili najbolje novo rješenje iz susjedstva danog rješenja, moramo proći po svim kolegijima i za svaki kolegij postaviti svaki mogući termin te izračunati cijenu toga

rješenja te na kraju odabrati rješenje s najnižom cijenom. Upravo tako i radi klasični uspon na vrh. Kod brzog uspona na vrh gledamo susjedstvo ovisno o danom rješenju i danom terminu. Znači da za svako dano rješenje imamo susjedstava koliko i kolegija. U svakom od tih manjih susjedstava tijekom jedne iteracije algoritma nađemo najbolji termin. To znači da se u jednoj iteraciji algoritma postavi puno više novih termina, dok se u klasičnom usponu na vrh postavi najviše jedan.

Funkcija cilja koju želimo minimizirati je vektorska funkcija s dvije komponente. Prva komponenta predstavlja tvrdo ograničenje i zadana se kao ukupan broj kolizija svih studenata. Druga komponenta funkcije cilja predstavlja meko ograničenje i zadana je kao ukupan broj ispita koje studenti pišu više od jednog u danu. Tj. prva i druga komponenta funkcije cijene $f(\omega)$ dane su redom sljedećim izrazima.

$$f_1(\omega) = \sum_{s \in students} s.numOfCollisionsForStudent$$

$$f_2(\omega) = \sum_{s \in students} s.numOfSameDayTerms$$

gdje je ω dano rješenja a *students* trenutno stanje te liste. Funkcija se dakle mijenja ovisno o tome koji je kolegij stavljen u koji termin pisanja što utječe na broj kolizija i broj ispita u istom danu za svakog studenta.

6 Metode pretraživanja prostora rješenja

6.1 Metode uspona na vrh

Klasična metoda uspona na vrh radi na jednostavnom principu. Krenemo od proizvoljnog stanja tj. nasumično odabranog rješenja iz prostora rješenja koje sasvim sigurno ne zadovoljava tvrda ograničenja. Zatim iz susjedstva toga rješenja uzmemo najbolje rješenje s obzirom na funkciju cijene. Tada postupak ponavljamo s boljim rješenjem kao početnim. Metoda traje dok ne dođemo do rješenja koje je najbolje rješenje u svojem susjedstvu.

Brza metoda uspona na vrh radi na istom principu uz prije spomenutu modifikaciju susjedstva. Iz takve definicije susjedstva, za očekivati je puno bržu konvergenciju što ćemo proučiti u sljedećem dijelu. Također, dvije metode uspona na vrh ne moraju konvergirati u isto rješenje i za očekivati je da će klasična metoda ipak dati bolje rješenje.

Nedostatak metodi uspona na vrh je što će skoro uvijek zaglaviti u lokalnom optimumu jer po samoj definiciji lokalnog optimuma, to je rješenje koje je najbolje u svome susjedstvu. Tako da vjerojatno nećemo naći globalni optimum ovim metodama. Iz tog razloga postoje i sofisticiranije metode poput metaheuristika. Metoda simuliranog kaljenja je također jedna od njih.

6.2 Metoda simuliranog kaljenja

Ova metaheuristika je inspirirana kaljenjem metala u metalurgiji. Kreće se od početne temperature koja je maksimalna temperatura i spušta se do minimalne temperature

svaku iteraciju i to za neki korak. U ovom projektu implementirana je najosnovnija metoda simuliranog kaljenja s fiksnim korakom smanjenja temperature. Metoda radi tako da u svakoj iteraciji odabere nasumično rješenje iz susjedstva trenutnog rješenja. Ako je rješenje bolje od trenutnog prihvaćamo ga, smanjujemo temperaturu i krećemo sa sljedećom iteracijom. Ako je rješenje lošije od trenutnog, postoji mogućnost da ga prihvatimo koja je jednaka $e^{\frac{f(\omega)-f(\omega')}{t}}$ gdje je ω trenutno rješenje, ω' novo, lošije rješenje, f funkcija cijene (želimo da bude što manja) i t temperatura u trenutnoj iteraciji.

7 Rezultati

U ovom dijelu seminara su izneseni rezultati koji uspoređuju sve tri metode. Način uspoređivanja je s fiksnim ulaznim parametrima koji bi mogli biti neke stvarne vrijednosti. Ovom usporedbom ćemo dobiti uvid koliko je svaka metoda dobra u nekom realnom scenariju. U stvarnosti, drugo kolokvijsko razdoblje u zimskom semestru 2020./2021. na matematičkom odsjeku zagrebačkog PMF-a, je provedeno u sličnim uvjetima. Broj kolegija, termina i termina po danu je jednak, a ostali parametri su odabrani kako bi donekle realno simulirali stvarno stanje. Vrijednosti ulaznih parametara za usporedbu algoritama dani su sljedećom tablicom.

Tablica 2: Vrijednosti ulaznih podataka

Značenje	**Vrijednost**
broj studenata	1600
broj kolegija	72
broj termina	42
broj termina po danu	3
min. broj kolegija svakog studenta	1
max. broj kolegija svakog studenta	5
minimalna temperatura	0
maksimalna temperatura	100000
korak smanjivanja temperature	1

Rezultati dobiveni ovim ulaznim parametrima za sve tri metode dani su sljedećom tablicom.

Tablica 3: Usporedba metoda

	FHC			CHC			SA		
seed	$f_1(\omega)$	$f_2(\omega)$	vrijeme/s	$f_1(\omega)$	$f_2(\omega)$	vrijeme/s	$f_1(\omega)$	$f_2(\omega)$	vrijeme/s
1	0	220	17.80	0	217	206.87	85	375	94.31
2	0	215	18.08	0	190	188.89	86	383	97.21
3	0	198	14.77	0	192	147.93	75	364	89.02
4	0	209	19.11	0	178	200.62	82	368	97.30
5	0	170	28.20	0	200	181.64	74	346	98.93
6	0	198	17.67	0	186	201.42	83	382	97.42
7	0	195	18.56	0	217	164.91	79	370	97.56
8	0	187	22.49	0	195	194.86	77	363	108.78
9	0	190	21.00	0	173	207.27	77	372	95.25
10	1	206	15.52	0	209	164.71	81	401	99.33

Svi testovi su izvođeni na laptopu s procesorom od 2.50GHz. Iz rezultata odmah iskače da SA daje daleko najlošije rezultate. SA korišten u ovom projektu je vrlo jednostavan i nije kvalitetno optimiziran po svojim parametrima, tako da je za očekivati da i taj algoritam može dati bolje rezultate od dobivenih, no i ne zadovoljavajuće rezultate. CHC i FHC daju zadovoljavajuće rezultate, s tim da CHC ipak daje bolje rezultate što je i za očekivati s obzirom na to kako rade oba algoritma. Velika prednost FHC je što bi se mogao koristiti i s većim parametrima i završavao bi u nekom razumnom danom vremenu.

8 Zaključak

U ovom projektnom zadatku smo rješavali problem rasporeda ispita pomoću tri metode od kojih su dvije slične metode uspona na vrh, a treća je metoda simuliranog kaljenja. Uspješnost metoda smo provjerili eksperimentalno. Metode uspona na vrh su dale puno bolje rezultate, a razlog tome je što je susjedstvo pojedinačnog rješenja preveliko. Uz tako veliko susjedstvo, metoda SA se svodi na nasumičan odabir nekog rješenja što nikako ne može biti dobro. S druge strane, veličina susjedstva pojedinačnog rješenja nije prevelika da se ne može cijela proći u relativno malom vremenu (barem za parametre s kojima smo mi radili). Zaključujemo da su metode uspona na vrh bolje za rješavanje ovog konkretnog problema, a između FHC i CHC, klasični pristup daje bolja rješenja tako da uz razumnu veličinu parametara, CHC je najbolji izbor, a ako su pak parametri preveliki za CHC da završi u razumnom vremenu, FHC je svakako bolji izbor.

Literatura

[1] prof. dr. sc. Bojana Dalbelo Bašić and doc. dr. sc. Jan Šnajder. *Umjetna inteligencija, 2. Pretraživanje prostora stanja*. Ak. god. 2013./2014.

[2] Marko Čupić. *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.* 2013. URL: http://java.zemris.fer.hr/nastava/pioa/knjiga-0.1.2013-12-30.pdf.

[3] Marko Čupić, Marin Golub, and Domagoj Jakobović. "Exam Timetabling Using Genetic Algorithm". In: (2009). URL: https://ferko.fer.hr/people/Marko.Cupic/files/2009-422047.iti2009.pdf.

[4] *Simulated annealing, Wikipedia.* URL: https://en.wikipedia.org/wiki/Simulated_annealing.