SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 000

Primjena konstruktivnog optimizacijskog algoritma na problem rasporeda studenata

Martin Čekada

Zagreb, svibanj 2019.

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

SADRŽAJ

1. Uvod

Uvod rada. Nakon uvoda dolaze poglavlja u kojima se obrađuje tema.

2. Pregled područja

Problemi raspoređivanja su specijalizacija transportnih problema te su jedni od temeljnih optimizacijskih problema. Općenito gledano, problemi raspoređivanja bi se mogli definirati na sljedeći način:

> *Problem se sastoji od agenata i zadataka. Svakom agentu može biti dodijeljen bilo koji od zadataka uz određenu cijenu, a cijena može varirati ovisno o uparivanju agenta i zadatka. Potrebno je svakom agentu dodijeliti zadatak tako da ukupna cijena dodjeljivanja bude minimalna.*

Ovakvi problemi bi se mogli pokušati riješiti tako da se generira svaka kombinacija dodjeljivanja agenata i zadataka te da se odredi dodjeljivanje s najmanjom cijenom. Ukoliko bi se dodjeljivanje vršilo za n agenata i n zadataka, složenost dodjeljivanja bi bila $n!$. Porastom broja n vrijeme potrebno da se na ovaj način odredi dodjeljivanje s najmanjom cijenom vrlo brzo postaje preveliko da bi se izračunalo u realnom vremenu. Zbog toga kod problema raspoređivanja (i općenito optimizacijskih problema) ne tražimo iscrpno optimalno rješenje, nego različitim pristupima pokušavamo pronaći dovoljno dobro rješenje. U nastavku slijedi opis nekih specifičnih problema raspoređivanja.

Primjer 1.[1] Zadana je funkcija $g(x, y, z)$ nad domenom $[-300, 500] \times [-300, 500] \times [-300, 500] \subset \mathbb{R} \times \mathbb{R} \times \mathbb{R}$. Pronaći točku (x, y, z) za koju funkcija g poprima maksimalnu vrijednost.

Primjer 2. Satničaru su na raspolaganju popis kolegija koji se predaju, popis studenata i njihov izbor kolegija, popis slobodnih dvorana i termina, željeni tjedni broj predavanja za svaki od kolegija te popis nastavnika koji predaju određene kolegije. Satničar treba zadovoljiti sljedeće uvjete:

– svi studenti imaju zakazana sva predavanja i mogu ih slušati bez kolizija,

[1] Preuzet iz (**?**)

– niti jedan nastavnik ne drži više predavanja istovremeno,

– niti ujednu dvoranu ne smije biti smješteno više studenata nego što je kapacitet dvorane te

– niti ujednu prostoriju ne smiju biti smještena dva predavanja istovremeno.

Također, bilo bi poželjno kada bi satničar osigurao da:

– student u danu ima barem dva predavanja ili niti jedno,

– student ima minimalan broj rupa u danu,

– nastavnik ima minimalan broj rupa u danu te

– nastavnik ima minimalan broj promjena dvorana u danu.

Primjer 3. Primjer u nastavku je pojednostavljen i prilagođen organizaciji međuispita na Fakultetu elektrotehnike i računarstva. Prilikom izrade ispita dostupni su podaci o predmetima za koje treba održati ispit, podaci o slobodnim terminima te podaci o studentima i ispitima kojima oni mogu pristupiti. Potrebno je izraditi raspored međuispita tako da niti jedan student ne piše istovremeno dva ispita, da svaki student može pristupiti svakom svojem ispitu te tako da se niti u jednoj dvorani ne pišu istovremeno dva ispita. Bilo bi poželjno da student ima što ravnomjernije raspoređen broj slobodnih dana između ispita.

Na prethodnim primjerima se može primijetiti da se neki od uvjeta moraju ispuniti, dok su neka samo poželjna. To su tvrda (engl. *hard*) i meka (engl. *soft*) ograničenja (engl. *constraints*). Tvrda ograničenja moraju biti ispunjena kako bi rješenje bilo prihvatljivo (npr. u primjeru 2 ne može konačno rješenje biti ono u kojem nastavnik istovremeno predaje u dva termina). S druge strane, meka ograničenja nisu obavezna, ali što su ona ispunjenija, to je rješenje bolje.

Kako je u početku poglavlja ustanovljeno da se ovakvi problemi ne mogu rješavati tehnikom grube sile (engl. *brute force*), postavlja se pitanje na koje se načine u realnom vremenu može pronaći dovoljno dobro rješenje. Neki od optimizacijskih algoritama korišteni za rješavanje problema raspoređivanja su navedeni u nastavku.

2.1. Optimizacijski algoritmi

U ovom poglavlju su navedena samo dva primjera algoritama koji se mogu primijeniti na probleme raspoređivanja. Konstruktivni optimizacijski algoritam je detaljno opisan u narednim poglavljima, a postoji još mnoštvo drugih algoritama koji se mogu koristiti (algoritam roja čestica, algoritmi umjetnih imunoloških sustava, genetski algoritmi

itd.). Svaki od algoritama ima svoje prednosti i nedostatke te je prikladniji za određenu specijalizaciju problema, dok za drugu vrstu problema daje slabije rezultate. Ova tvrdnja je poznata kao *no-free-lunch* teorem kojeg su Wolpert i Macready dokazali u svojim radovima, a u originalu glasi:

> *All algorithms that search for an extremum of a cost function perform exactly the same, according to any performance measure, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A.* [2]

2.1.1. Mađarski algoritam

Algoritam je razvio i objavio Harold Kuhn 1955. godine, a nadjenuo mu je ime *mađarski* jer se algoritam velikom mjerom oslanja na rad dvaju mađarskih matematičara Dénes Kőnig i Jenő Egerváry. Algoritam je namijenjen raspoređivanju n poslova na n radnika pri čemu jedan radnik može obavljati samo jedan posao. Dakako, problem bi se mogao preslikati na prethodno naveden primjer 1. ili na raspoređivanje studenata u kojem je svakom studentu pridružen različit termin (ovo je češći slučaj kada se radi raspored za usmene ispite).

Vremenska složenost ovog algoritma je $\mathcal{O}(n^3)$, a algoritam koristi sljedeći teorem: *Ako svakom elementu bilo kojeg retka ili stupca matrice kazne dodamo ili oduzmemo neki broj, tada je optimalno raspoređivanje za rezultantnu matricu također optimalno i za prvotnu matricu.* Pritom se matricom kazne smatra matrica čiji redci predstavljaju radnike, stupci predstavljaju poslove, a vrijednost matrice na mjestu $C(i, j)$ predstavlja cijenu dodjeljivanja radniku i posao j. Koraci algoritma slijede.

1. Za svaki redak matrice, pronađi najmanji element i oduzmi ga od svakog elementa u njegovom retku.

2. Ponovi korak 1. za svaki stupac.

3. Prekrij sve nule u matrici koristeći minimalan broj horizontalnih i vertikalnih linija.

4. Test optimalnosti: ako je minimalan broj linija potrebnih da se prekriju sve nule u matrici jednak n, tada je optimalno rješenje moguće i završavamo s izvođenjem

algoritma. Ako je broj linija manji on n, tada optimalno rješenje nije pronađeno te nastavljamo na korak 5.

5. Pronađi najmanji element matrice koji nije prekriven niti jednom linijom. Oduzmi taj element od svakog neprekrivenog retka i potom ga dodaj svakom prekrivenom stupcu. Vrati se na korak 3.

Pogledajmo to sada na primjeru raspoređivanja poslova utovara robe, transporta robe te istovara robe među radnicima Petrom, Ivanom i Markom. Neka je cijena pridjeljivanja poslova između radnika određena sljedećom tablicom:

Tablica 2.1: Tablica rasporeda poslova

	Utovar	Transport	Istovar
Petar	25	40	35
Ivan	40	60	35
Marko	20	40	25

Da bi izvođenje algoritma započelo, potrebno je prvo 2.1 zapisati matrično:

$$\begin{bmatrix} 25 & 40 & 35 \\ 40 & 60 & 35 \\ 20 & 40 & 25 \end{bmatrix}.$$

Potom može započeti izvršavanje algoritma.

1. Oduzmi najmanju vrijednost svakog retka.

$$\begin{bmatrix} 0 & 15 & 10 \\ 5 & 25 & 0 \\ 0 & 20 & 5 \end{bmatrix}.$$

2. Prekrij sve nule minimalnim brojem linija.

$$\begin{bmatrix} \cancel{0} & \cancel{0} & \cancel{10} \\ \cancel{5} & \cancel{25} & \cancel{0} \\ \cancel{0} & 20 & 5 \end{bmatrix}$$

3. S obzirom na to da je broj linija potrebnih za prekrivanje svih nula jednak 3 optimalno rješenje je pronađeno.

Poslove treba raspodijeliti tako da Petar radi transport robe, Ivan istovar, a Marko utovar. Cijena takvog raspoređivanja je 95.

2.1.2. Algoritam kolonije mrava

Ovaj algoritam je inspiriran procesom kojim mravi pronalaze najkraći put između mravinjaka i hrane. Mravi se prilikom potrage za hranom ne služe svojim osjetilom vida, nego osjetilom feromona. Fermoni su kemijski tragovi koje mravi ostavljaju za sobom krećući se od mravinjaka do hrane i povratno. Intenzitet mirisa fermona slabi s vremenom ako tim putem ne prolaze mravi. Kako mravi na ovaj način pronalaze najkraći put između hrane i mravinjaka, intuitivno je jasno da je ovaj algoritam prikladan za pretraživanje grafova. Osim za pretraživanje grafova, algoritam kolonije mrava je uspješno primjenjivan na probleme izrade rasporeda [3] te je u velikoj mjeri sličan konstruktivnom optimizacijskom algoritmu koji je tema ovog rada [4]. Pseudo kod algoritma je: [5]

$ponavljajdoknijekrajponovizasvakogmravastvorirjeenjevrednujrjeenjekrajponoviisparifero$

Rješenje za svakog mrava se gradi tako da se odabire brid po brid kojima će mrav prolaziti (parovi bridova naravno moraju imati jednu zajedničku točku, odnosno mora se moći nakon prvog brida prijeći na drugi brid itd.). Odabir kojim bridom će mrav poći iz trenutnog čvora se određuje slučajnim proporcionalnim pravilom (engl. *random proportional rule*):

$$p_i^k j = \begin{cases} \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in N_k^i} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} & \text{ako } j \in N_k^i \\ 0 & \text{ako} j \notin N_k^i \end{cases}$$

pri čemu su α i β konstante, τ je intenzitet fermona na bridu, a η je heuristička informacija koliko je povoljno krenuti bridom. N_k^i predstavlja sve bridove kojima je početni čvor trenutni čvor u kojem se mrav nalazi. Prije prve iteracije, sve vrijednosti τ se postavljaju na vrijednost koja je nešto veća od očekivane vrijednosti koju će mravi ostavljati u svakoj iteraciji. Za izračun se koristi formula:

$$\tau_0 = \frac{m}{C^{nn}}$$

pri čemu je m broj mrava, a C^{nn} je procjena najkraćeg puta dobivena nekim jednostavnijim algoritmom.

[3] Na primjer `https://www.researchgate.net/publication/268047044_Evolucijsko_racunanje_i_problem_izrade_rasporeda`

[4] detaljnija usporedba algoritama je dana u narednim poglavljima

[5] Preuzeto s `https://www.researchgate.net/publication/268047044_Evolucijsko_racunanje_i_problem_izrade_rasporeda`

Isparavanje fermonskih tragova se vrši tako da se trenutna vrijednost pomnoži s konstantnim koeficijentom prema izrazu:

$$\tau_{ij} \leftarrow \tau_{ij} \cdot (1 - \rho)$$

Mravi ažuriraju feromonske tragove proporcionalno dobroti rješenja koje su izgradili prema izrazu:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

pri čemu je:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C^k} & \text{ako je } i\text{-}j \text{ na stazi mrava} \\ 0 & \text{ako nije na stazi} \end{cases}$$

3. Opis algoritma

Konstruktivni optimizacijski algoritmi su široko korišteni algoritmi za heurističku optimizaciju. Inačica konstruktivnog optimizacijskog algoritma koji je korišten u ovom radu se temelji na radu (**?**).

3.1. Opći matematički model algoritma

Promatramo problem s konačnim skupom izvodljivih rješenja S i funkcijom kazne $f : S \to \mathbb{R}$. Neka je skup S^* skup optimalnih rješenja te isključimo trivijalna rješenja, odnosno pretpostavim da $|S| > 1$ i da $S^* \neq S$. Neka se svako rješenje iz skupa S može zapisati kao konačan niz simbola $s = (s_1, ..., s_L)$ iz skupa abecede $\mathbb{A} := a_1, ..., a_K$. L je fiksna duljina rješenja. Model je generaliziran funkcijom izvodljivosti $C_i(y, a)$ koja dodjeljuje težinu svakom $a \in \mathbb{A}$ za svako moguće parcijalno rješenje y duljine i. Prema tome, ako je $C_i(y, a) = 0$, parcijalno rješenje y se ne može nastaviti tako da se na njega nadoda znak a. Što je vrijednost funkcije $C_i(y, a)$ veća, toliko je poželjnije da se niz nastavi znakom a pohlepno (engl. *greedy*) gledano. Pretpostavlja se da su vrijednosti funkcije normirane. Model gradi rješenja korak po korak dodavajući nove simbole na desni kraj niza sve dok rješenje nije potpuno. Formalnije rečeno, definiramo skup R_i izvodljivih parcijalnih rješenja rekurzivno kako slijedi: neka $\Diamond$ pretstavlja prazan string. Pretpostavljamo da je dano:

$$C_0(\Diamond, \cdot) : \mathbb{A} \to [0, 1], \sum_{a \in \mathbb{A}} C_0(\Diamond, a) = 1$$

što pretstavlja poželjnost i izvodljivost znaka a na prvom mjestu rješenja. Nadalje definiramo:

$$R_1 := \{a \in \mathbb{A} | C_0(\Diamond, a) > 0\}$$

Kao skup izvodljivih rješenja duljine 1. Pretpostavimo da je definiran skup R_i za neki $i \in 0, ..., L - 1$ i da je dano:

$$C_i(y, \cdot) : \mathbb{A} \to [0, 1], \sum_{a \in \mathbb{A}} C_i(y, a) = 1, \text{za svaki} y \in R_i$$

Neka (y, a) označava konkateniranje simabola $a \in \mathbb{A}$ na desni kraj parcijalnog rješenja y. Definiramo:

$$R_{i+1} := \{(y, a) | y \in R_i, a \in \mathbb{A}, C_i(y, a) > 0\}$$

i neka je $S := R_L$. Za svaki $y \in R_i$, $i \in \{0, ..., L-1\}$, neka je

$$C_i(y) := \{a \in \mathbb{A} | C_i(y, a) > 0\}$$

bude potpora $C_i(y, \cdot)$.

Ovakva generalizacija modela ne postavlja gotovo nikakva ograničenja na optimizacijski problem. Odnosno, odgovarajućim odabirom A, S i $C(y, a)$ svaki se problem može prilagoditi na ovaj model, no efikasnost može varirati. Model je razumnije koristiti u slučajevima kada je $|A| \ll |S|$. Ovo dodatno potvrđuje teorem s kraja prethodnog poglavlja koji tvrdi da svaki algoritam ne daje jednako dobre rezultate na svakom problemu.

3.2. Generalizirani algoritam

U suštini, algoritam razvija distribuciju nad skupom $S = R_L$ svih izvodljivih rješenja dajući pritom veliku vjerojatnost optimalnim rješenjima iz skupa S^*. Neka $\mathbb{P}(\mathbb{A})$ označeva skup svih vjerojatnosti nad skupom $\mathbb{A}$. Tada je $p \in \mathbb{P}(\mathbb{A})^L, p = (p(1), ..., p(L))$ vjerojatnost nad skupom $\mathbb{A}^L$ koja opisuje odabiranje rješenja $s = (s_1, ..., s_L) \in \mathbb{A}^L$ pri čemu je L simbola $s_1, ..., s_L$ odabrano neovisno. Pritom je $p(i) = p(a; i)_{a \in \mathbb{A}} \in \mathbb{P}(\mathbb{A})$ je distribucija vjerojatnosti za simbol na i-toj lokaciji. Ulazni podaci za algoritam su:

1. funkcija poželjnosti $C_i(\cdot, \cdot), i \in 0, ..., L-1$,
2. niz koeficijenata izglađivanja $(\varrho_t)_{t \geq 1}$ uz $\varrho_t \in (0, 1)$,
3. veličinu uzorka N i veličinu poduzorka N_b te
4. početnu distribuciju $p_0 \in \mathbb{P}(\mathbb{A})^L$

3.2.1. Početak

Za $t = 0$, postavi $p = p_0$. Iteriraj kroz korake $t = 1, 2, ...$ sve dok uvjet zaustavljanja nije zadovoljen.

3.2.2. Uzorkovanje (engl. *sampling*)

Ako je trenutna distribucija $p \in \mathbb{P}(\mathbb{A})^L$, tada je vjerojatnost uzrokovanja rješenje $s = (s1, ..., sL) \in S$ dana izrazom:

$$Q_p(s) := Q_p(s_1; 1, \Diamond) \cdot \prod_{i=2}^{L} Q_p(s_i; i, (s_1, ..., s_{i-1}))$$

pri čemu je:

$$Q_p(a; i, y) := \frac{p(a, i)C_{i-1}(y, a)}{\sum_{a' \in \mathbb{A}} p(a', i)C_{i-1}(y, a')}$$

vjerojatnost da se izvodljivi simbol $a \in \mathbb{A}$ nadoda na poziciju i izvodljivog parcijalnog rješenja $y \in R_{i-1}$. Koristi se konvencija $\frac{0}{0} = 0$. Na ovakav način, algoritam uzorkuje N rješenja $s^{(1)}, ..., s^{(N)}$ neovisno i podjednako distribuirano.

3.2.3. Ocjenjivanje (engl. *evaluation*)

Neka su uzorci $x := (s^{(1)}, ..., s^{(N)})$ poredani prema funkciji kazne f:

$$f(s^{n_1}) \leq f(s^{n_2}) \leq \cdots \leq f(s^{n_N})$$

te neka je odabrano najboljih N_b uzoraka $N_b := \{s^{(n_1)}, s^{(n_2)}, ..., s^{(n_{N_b})}\}$. Nakon toga određujemo relativnu frekvenciju simbola a na poziciji $i \in 1, ..., L$ u odabranom djelu uzorka

$$w(a; i, x) := \frac{1}{N_b} \sum_{s \in N_b} \mathbb{1}_{\{a\}}(s_i)$$

i prikupimo te frekvencije za svaki $a \in \mathbb{A}$ te kreiramo $w(i, x) = w(a; i, x)_{a \in \mathbb{A}}$ i

$$w(x) := (w(1, x), ..., w(L, x)).$$

Tada je $w(x)$ distribucija vjerojatnosti za $\mathbb{P}(\mathbb{A})^L$ koja daje relativne frekvencije simbola iz boljeg dijela uzorka x uzorkovanog s vjerojatnosti Q_p.

3.2.4. Ažuriranje (engl. *update*)

Trenutnu distribuciju p ažuriramo kao kombinaciju p i relativne frekvencije $w(x)$

$$p := (1 - \varrho_{t+1})p + \varrho_{t+1}w(x)$$

U idućem koraku se brojač t uvećava za 1 i korak uzorkovanja se obavlja s novom ditribucijom p.

4. Problem vodećih jedinica

Kako bi se bolje prikazalo ponašanje konstruktivnog optimizacijskog algoritma, u nastavku je razmatrana primjena algoritma na problem vodeće jedinice (engl. *LeadingOne*). Problem se sastoji od generiranja niza nula i jedinica s ciljem maksimiziranja broja početnih jedinica. Optimalno rješenje je ono u kojem se niz sastoji isključivo od jedinica. Prateći notaciju iz prethodnih poglavlja, problem se može formalizirati tako da je abeceda znakova $\mathbb{A} = \{0, 1\}$, skup svih rješenja $S = \{0, 1\}^L$ te funkcija kazne:

$$f(s) := L - \sum_{l=1}^{L} \prod_{i=1}^{l} s_i, \text{za } s = (s_1, ..., s_L)$$

Minimiziranjem kazne, broj početnih uzastopnih jedinica se maksimizira. Definirajmo dodatno τ kao prvu iteraciju u kojoj se pronalazi optimalno rješenje:

$$\tau := min\{t \geq 0 | X_t \cap S^* \neq \emptyset\}$$

pri čemu je $X_t := \{X_t^{(1)}, ..., X_t^{(N)}\}$ skup svih rješenja uzorkovanih u iteraciji t.

U poglavlju 3, teoremu 2 rada (**?**) se tvrdi da uz odabir konstantnog parametra izglađivanja $\varrho_t = \varrho$, veličine uzorka $N = L^{(2+\epsilon)}$, uz $\epsilon > 0$ i $Nb = \lfloor(\beta N)\rfloor$ za $0 < \beta < \frac{1}{3e}\prod_{m=1}^{\infty}(1-(1-\varrho)^m)$. Uz početnu distribuciju $\prod_0(1, i) \equiv \frac{1}{2}$, odnosno jednoliku distribuciju, za prethodno definirani problem vodeće jedinice vrijedi $\mathbb{P}(\tau < L) \to 1$ kada $L \to \infty$.

Ovaj teorem je ekspermintalno vrednovan, a rezultati su u tablici 4.1. Vidljivo je da broj iteracija potrebnih da rješenje konvengira u 1 manje od L za svaku testiranu duljinu rješenja. Prema dobivenim rezultatima se naslućuje da bi daljnjim rastom duljine L teorem i dalje bio zadovoljen.

Utjecaj parametra izglađivanja prikazan je u tablici 4.2. Iz mjerenja je vidljivo da opisani problem povećanjem parametra izglađivanja brže konvengira željenom rješenju. Najbolji rezultat je dobiven uz $\varrho_t = \varrho = 1$ što predstavlja slučaj u kojem trenutnu distribuciju p zamijenjujemo relativnom frekvencijom uzorkovanja, odnosno $p := w(x)$.

Tablica 4.1: Utjecaj duljine uzorka L na broj iteracija ($\epsilon = 0.5$, $\beta = 0.09$)

L	N	N_b	ϱ	i
10	316	28	0.8	6
20	1788	160	0.8	9
30	4929	443	0.8	13
40	10119	910	0.8	16
50	17677	1590	0.8	20
60	27885	2509	0.8	23
70	40996	3689	0.8	26
80	57243	5151	0.8	30
90	76843	6915	0.8	33
100	100000	9000	0.8	37

Tablica 4.2: Utjecaj parametra izglađivanja ϱ_t na broj iteracija ($\epsilon = 0.5$, $\beta = 0.09$)

L	N	N_b	ϱ	i
50	17677	6	0.2	43
50	17677	82	0.3	33
50	17677	279	0.4	29
50	17677	563	0.5	25
50	17677	881	0.6	23
50	17677	1195	0.7	21
50	17677	1483	0.8	19
50	17677	1736	0.9	18
50	17677	1950	1.0	15

Za svaki testirani skup parametara vršeno je 30 testova, a u tablicama je naveden medijan dobivenih vrijednosti. U svim navedenim testovima uvjet zaustavljanja je bio da vjerojatnost p da se na poziciji i uzorkuje 1 bude veća od 0.999:

$$p(1, i) > 0.999, \text{za svaki } i \in \{0, ..., L-1\}.$$

5. Problem izrade rasporeda laboratorijskih vježbi

Izrada rasporeda laboratorijskih vježbi je specijalizacija problema raspoređivanja. Cilj je svakom studentu pridružiti termin laboratorijskih vježbi s kojim on nema kolizije, a da pritom niti jedan termin ne bude niti prepunjen niti podpunjen. Glavna razlika izrade rasporeda laboratorijskih vježbi u odnosnu na izradu rasporeda ispita je ta što laboratorijsku vježbu treba smjestiti u postojeći raspored studenata, dok se prilikom izrade rasporeda ispita pretpostavlja da za vrijeme ispita studenti nemaju predavanja. Također, raspoređivanje za laboratorijske vježbe se vrši za svaki predmet neovisno, dok se prilikom izrade rasporeda ispita uzimaju svi ispiti studenta u obzir. Dodatno, prilikom izrade rasporeda laboratorijskih vježbi termini u kojima se laboratorijske vježbe mogu održati su unaprijed određene, dok se prilikom izrade rasporeda ispita trebaju odrediti i termini. Tvrda ograničenja izrade rasporeda laboratorijskih vježbi su:

- svakom studentu treba dodijeliti termin laboratorijske vježbe,
- student ne smije imati koliziju s dodijeljenim terminom laboratorijske vježbe,
- u termin ne smije biti smješteno više od maksimalnog broja studenata za termin te
- u termin ne smije biti smješteno manje od minimalnog broja studenata za termin.

Nakon što su navedena tvrda ograničenja zadovoljena, raspoređivanje se dodatno vrednuje mekim ograničenjima. Meka ograničenja dodatno kažnjavaju rasporede u kojima:

- je termin pridjeljen studentu na dan na koji nema predavanja i
- termin produžuje ukupno vrijeme studenta na fakultetu (laboratorijska vježba nije u rupi između drugih predavanja).

5.1. Formalna defincija problema

Konkretan problem raspoređivanja obrađen u ovom radu se odnosi na izradu rasporeda za laboratorijske vježbe iz predmeta Digitalna logika i Objektno orijentirano programiranje. Prema notaciji iz 3, skup abecede $\mathbb{A}$ je skup $Termini$ svih termina laboratorijskih vježbi. Duljina uzorka L je broj studenata u skupu $Studenti$ koje treba rasporediti. U distribuciji p, vrijednost na mjestu p_{ij} predstavlja vjerojatnost da studentu j bude pridjeljen termin i. Rješenje problema je niz $s = (s_1, ..., s_L)$ pri čemu je s_1 termin laboratorijske vježbe za prvog studenta itd. Nad svakim studentom iz skupa $Studenti$ je definirana metoda $imaKolizijuS(Termin) = \{\top, \bot\}$ koja vraća $\top$ ako student ima koliziju s predanim terminom, odnosno $\bot$ kada student nema koliziju s predanim terminom. Metoda $produljujeTrajanjeDana(Termin) = \{\top, \bot\}$ definirana nad elementima skupa $Studenti$ vraća $\top$ ako bi pridjeljivanje termina studentu produžilo trajanje dana na fakultetu, a $\bot$ ako ne bi. Odnosno, ova metoda vraća $\top$ kada predani termin nije u rupi između drugih predavanja studentu, a $\bot$ kada je. $naSlobodanDan(Termin) = \{\top, \bot\}$ je metoda definirana nad elementima skupa $Studenti$ koja vraća $\top$ ako je predani termin studentu u danu u kojem nema niti jedno predavanje, a $\bot$ ako student na dan termina ima predavanja.

5.2. Inicijalna distribucija

U problemu vodećih jedinica inicijalna distribucija je uniforma kako bi se demonstrirao rad algoritma. U stvarnom problemu kao što je problem izrade rasporeda uniformna početna distribucija može uvelike odužiti vrijeme potrebno da algoritam konvergira prema rješenju. Zato je korisno inicijalizirati početnu distribuciju u skladu s domenom problema kako bi se algoritam usmjerio prema rješenju. Kod problema raspoređivanja termina laboratorijskih vježbi studentima, u samom početku (prije početka rada algoritma) moguće je predvidjeti da će određena pridruživanja termina studentima biti nepovoljnija od drugih. U konačnom rješenju nije dopušteno da student ima koliziju s pridruženim terminom, stoga je jasno da u inicijalnoj distribuciji treba studentu pridružiti malu vjerojatnost da mu se pridruži termin s kojim ima koliziju. Također, poznato je da ako termin produljuje trajanje dana studentu ili je na studentov slobodan dan, da će to rješenje biti nepovoljnije od onih koja ne produljuju trajanje dana studentu. Pseudokod inicijalizacije početne distribucije koja je u skladu s prethodnim opažanjima prikazan je na 1. Pridruživanje vjerojatnosti se vrši na način da se svakom studentu za svaki termin pridjeli određena vrijednost proporcionalna tomu koliko je taj termin

povoljan za studenta. Nakon što su studentu tako pridružene vjerojatnosti potrebno je normalizirati dobivenu raspodjelu.

5.3. Funkcija dodjeljivanja kazne

Funkcija dodjeljivanja kazne dodjeljuje kaznu svakom pridruživanju studenata i termina. Ulaz u funkciju je lista pridruživanja studenata i termina. Ulazna lista je veličine broja studenata, a svaki element liste predstavlja indeks termina koji je pridjeljen studentu na odabranom indeksu liste. Na primjer, neka treba rasporediti pet studenata na tri termina laboratorijskih vježbi. Jedan od načina pridruživanja je:

$$[2, 2, 0, 1, 0].$$

Ovakvo pridruživanje označava da su $Student_0$ i $Student_1$ raspoređeni na $Termin_2$, $Student_2$ i $Student_4$ na $Termin_0$, a $Student_3$ na $Termin_1$. Izlaz iz funkcije je broj koji označava kaznu za predano rješenje.

Prilikom dodjeljivanja kazne uzeta su u obzir tvrda i meka ograničenja navedena na početku poglavlja. Kako bi bilo osigurano da algoritam svakako zadovolji tvrda ograničenja, a da na temelju mekih samo fino podešava (engl. $finetuning$), kazna koja se dodjeljuje za kršenje tvrdih ograničenja je dva reda veličina veća od kazne za kršenje mekih ograničenja. Kažnjavanje za prepunjavanje termina laboratorijskih vježbi se dodjeljuje kvadratno. Time je postignuto da algoritam ravnomjernije raspoređuje studente jer je kazna znatno manja ako su četiri termina prepunjena sa po jednim studentom, nego ako je jedan termin prepunjen s 4 studenta. Pseudokod funckije s konkretnim vrijednostima kazni je prikazan na 2.

5.4. Heuristike

Kako je problem izrade rasporeda daleko složeniji od problema vodećih jedinica, u algoritam je dodano nekoliko nadogradnji kako bi algoritam uspješno rješavao problem izrade rasporeda. Jedna od nadogradnji je dodavanje huristika kojima bi se nakon ažuriranja trenutne distribucije relativnom frekvencijom uzorkovanja, ta distribucija još dodatno poboljšala. Ideja počiva na tome da algoritam prilikom svoga rada rješenja vrednuje isključivo na temelju vrijednosti koje rješenju dodjeli funkcija kazne, a pritom ne zna ništa o domeni problema niti konkretnim razlozima zbog kojeg je rješenje dobilo dodjeljenu kaznu. Heuristikama se nastoji modificirati trenutnu distribuciju na temelju spoznaja o konkretnom problemu.

Algorithm 1 Izračun inicijalne distribucije

$inicijalneVrijednosti \leftarrow [][]$
$n \leftarrow duljina(Termini)$
$m \leftarrow duljina(Studenti)$
for $(i \leftarrow 0; i < n; i \leftarrow i + 1)$ **do**
  for $(j \leftarrow 0; j < m; j \leftarrow j + 1)$ **do**
    $inicijalneVrijednosti[Termin_i][Student_j] \leftarrow 10$
    if $Student_j.imaKolizijuS(Termin_i)$ **then**
      $inicijalneVrijednosti[Termin_i][Student_j] \leftarrow$
      $inicijalneVrijednosti[Termin_i][Student_j] - 8$
    end if
    if $Student_j.produljujeTrajanjeDana(Termin_i)$ **then**
      $inicijalneVrijednosti[Termin_i][Student_j] \leftarrow$
      $inicijalneVrijednosti[Termin_i][Student_j] - 1$
    end if
    if $Student_j.naSlobodanDan(Termin_i)$ **then**
      $inicijalneVrijednosti[Termin_i][Student_j] \leftarrow$
      $inicijalneVrijednosti[Termin_i][Student_j] - 1$
    end if
  end for
end for
$sumaStupaca \leftarrow []$
for $(i \leftarrow 0; i < n; i \leftarrow i + 1)$ **do**
  for $(j \leftarrow 0; j < m; j \leftarrow j + 1)$ **do**
    $sumaStupaca[Student_j] \leftarrow sumaStupaca[Student_j] +$
    $inicijalneVrijednosti[Termin_i][Student_j]$
  end for
end for
for $(i \leftarrow 0; i < n; i \leftarrow i + 1)$ **do**
  for $(j \leftarrow 0; j < m; j \leftarrow j + 1)$ **do**
    $distribucija[Termin_i][Student_j] \leftarrow inicijalneVrijednosti[Termin_i][Student_j]/sumaStup$
  end for
end for

Algorithm 2 Funkcija dodjeljivanja kazne

Ulaz: $uzorak$ – uzorak jednog rješenja.

Izlaz: kazna za predano rješenje.

$kazna \leftarrow 0$

$popunjenost \leftarrow []$

$n \leftarrow duljina(Studenti)$

for $(i \leftarrow 0; i < n; i \leftarrow i + 1)$ **do**

  if $Student_i.imaKolizijuS(uzorak_i)$ **then**

    $kazna \leftarrow kazna + 80000$

  end if

  if $Student_i.produljujeTrajanjeDana(uzorak_i)$ **then**

    $kazna \leftarrow kazna + 50$

  end if

  if $Student_i.naSlobodanDan(uzorak_i)$ **then**

    $kazna \leftarrow kazna + 50$

  end if

  $popunjenost[uzorak_i] \leftarrow popunjenost[uzorak_i] + 1$

end for

$n \leftarrow duljina(Termini)$

for $(i \rightarrow 0; i < n; i \rightarrow i + 1)$ **do**

  if $popunjenost[uzorak_i] > Termin_i.maksimalanBrojStudenata()$ **then**

    $kazna \leftarrow kazna + 5000 + koeficijent\_prepunjenja \cdot (popunjenost[uzorak_i] - Termin_i.maksimalanBrojStudenata())^2$

  end if

  if $popunjenost[uzorak_i] < termin_i.minimalanBrojStudenata()$ **then**

    $kazna \leftarrow kazna + 5000 + koeficijent\_prepunjenja \cdot (Termin_i.minimalanBrojStudenata() - popunjenost[uzorak_i])^2$

  end if

end for

return kazna

5.4.1. Rebalansiranje trenutne distirbucije

Kako je u problemu izrade rasporeda laboratorijskih vježbi broj studenata velik, a broj termina u kojima su prostorije za izvršavanje laboratorijskih vježbi slobodne znatno manji, čest je slučaj da se u jednom vremenskom terminu održavaju vježbe u dvije različite prostorije. U tim situacijama studentu oba termina jednako odgovaraju (jer je jedina razlika prostorija u kojoj će laboratorijska vježba biti održana). Iako se iz perspektive studenta dva istovremena termina ne razlikuju, algoritam bez heuristike ih smatra potpuno različitima.Zbog toga se može dogoditi da algoritam prepunjuje jedan termin, dok drugi termin u isto vrijeme može biti podpunjen.

Dodavanje heuristike koja ravnomjerno rebalansira vjerojatnosti između svih istovremenih termina je implementacijski jednostavna, a donosi poboljšanje u radu algoritma. Pseudokod opisane heuristike je prikazan u 3. $IstovremeniTermini$ je mapa kojoj su ključevi termini, a vrijednosti su liste svih istovremenih termina (uključujući i sam termin koji je ključ). Na ovaj način se jednostavno može doći do lista svih istovremenih termina (to su sve vrijednosti u mapi $IstovremeniTermini$). Izgradnja ovakve mape u programskom jeziku Java prikazano je u 5.1.

Algorithm 3 Heuristika rebalansiranja istovremenih termina

$ListeIstovremenihTermina \leftarrow IstovremeniTermini.vrijednosti()$
$n \leftarrow duljina(Studenti)$
for $(i \leftarrow 0; i < n; i \leftarrow i + 1)$ **do**
  for $(ListaIstovremenihTermina : ListeIstovremenihTermina)$ **do**
    $suma \leftarrow 0$
    for $(Termin : ListaIstovremenihTermina)$ **do**
      $suma \leftarrow suma + TrenutnaDistribucija[Termin][Student_i]$
    end for
    $balansirano \leftarrow suma/duljina(ListaIstovremenihTermina)$
    for $(Termin : ListaIstovremenihTermina)$ **do**
      $TrenutnaDistribucija[Termin][Student_i] \leftarrow balansirano$
    end for
  end for
end for

Listing 5.1: Izrada mape istovremenih termina u programskom jeziku Java

```
SimpleDateFormat timeFormat = new
   SimpleDateFormat("yyyy-MM-dd HH:mm");
```

```
for (int i=0, n=labs.size(); i<n; ++i) {
  String labTime =
     timeFormat.format(labs.get(i).getTime().getStart());

  if (!labsAtTime.containsKey(labTime)) {
      labsAtTime.put(labTime, new ArrayList<>());
  }

  labsAtTime.get(labTime).add(i);
}
```

5.4.2. Redistribucija vjerojatnosti prepunjenih termina

Kao što je prethodno navedeno, algoritam ne zna na temelju čega se pojedinom rješenju dodjeljuje kazna. Zato je korisno dodati heuristiku koja će znati na temelju čega su dodjeljene kazne i djelovati u smjeru da ublaži buduće dodjeljivanje kazni. Kako se kvalitetnom inicijalnom distribucijom može jako dobro riješiti problem stvaranja kolizija, algoritmu je dodana heuristika koja rješava problem prepunjavanja.

Ideja heuristike je da onim terminima koji su prepunjeni smanji vjerojatnost odabiranja prilikom idućeg uzorkovanja. Kako zbroj vjerojatnosti mora uvijek biti jednak jedan, potrebno je preostalim terminima koji nisu prepunjeni povećeti vjerojatnost odabiranja. Za koliko će se smanjiti vjerojatnost odabiranja pojedinom terminu se određuje tako da se svi prepunjeni termini proporcionalno rasporede na zadani interval. Time se postiže da se terminima koji su bili prepopunjeni u više uzoraka vjerojatnost odabiranja smanji intenzivnije nego onim terminima koji su bili prepunjeni u manjem broju uzoraka. Implementacija heuristike je prikazana u isječku koda 5.2. Varijabla $overFilledLabs$ je mapa koja kao ključeve ima indekse prepunjenih termina, a vrijednosti su broj uzoraka u kojima je odgovarajući termin bio prepunjen. U početku rada algoritma su sva generirana rješenja loša i mnogo je termina prepunjeno te se eksperimentalno pokazalo da je ovu heuristiku korisno početi koristiti u kasnijim fazama rada algoritma.

Listing 5.2: Redistribucija vjerojatnosti prepunjenih termina

```
double UPPER_BOUND = 0.05;
double LOWWER_BOUND = 0.01;
int maxi = overFilledLabs.values()
```

```
                                .stream()
                                .mapToInt(Integer::intValue)
                                .max()
                                .getAsInt();

for (int j = 0, m = students.size(); j < m; ++j) {
  double redistribute = 0;

  for (Map.Entry<Integer, Integer> overfiled :
     overFilledLabs.entrySet()) {
      double reduceCoff = (((double)
         overfiled.getValue()) / maxi) * (UPPER_BOUND -
         LOWWER_BOUND) + LOWWER_BOUND;

      redistribute +=
         currentDistribution[overfiled.getKey()][j] *
         reduceCoff;
      currentDistribution[overfiled.getKey()][j] *= (1 -
         reduceCoff);
  }
  double increment = redistribute / (labs.size() -
     overFilledLabs.size());
  for (int i = 0, n = labs.size(); i < n; ++i) {
      if (overFilledLabs.containsKey(i)) continue;
      currentDistribution[i][j] += increment;
  }
}
```

5.5. Nadogradnje algoritma prioritetnim redom

U orginalnom algortimu opisanom u LINK-NA-RAD trenutna distribucija se ažurira isključivo vjerojatnostima dobivenim relativnim uzorkovanjem najboljih uzoraka. Takav pristup je uspješan na problemima poput problema vodeće jedinice gdje je velika sličnost između distribucija koje su generirale rješenje s jednakom kaznom. Na primjer, ako dvije distribucije generiraju uzorke s početnih deset jedinica uzastopno,

velika je vjerojatnost da će obje distribucije na prvih deset mjesta imati puno veću vjerojatnost za generiranje jedinice, nego za generiranje nule. Zbog toga sva dobra rješenja usmjeruju distribuciju u istom smjeru, smjeru u kojem su vjerojatnosti za generiranje jedinice na svim mjestima veće nego za generiranje nule. Kod problema izrade rasporeda laboratorijskih vježbi to nije slučaj. Dvije distribucije mogu konzistentno generirati rješenja koja će imati isti iznos kazne, a da pritom distribucije budu vrlo različite. Tomu je tako jer kod izrade rasporeda postoji puno više rješenja koja su jednako dobra. Jednako su dobra jer prema funkciji kazne prikazanoj na 2 se u slučaju kada je studentu dodjeljen termin koji mu je unutar rupe u predavanjima sporedno koji je to od termina. Dokle god je termin laboratorijske vježbe u rupi između predavanja, studentu je podjednako dobro u koji je termin raspoređen. Orginalni algoritam se ne koristi ovom činjenicom, nego ima tendenciju da kada pronađe dobar termin za studenta da ga nastoji držati u tom terminu, a možda je povoljnije prebaciti studenta u neki termin koji mu jednako odgovara i drugog studenta staviti na njegovo mjesto.

Kako bi se povečala raznovrsnost rješenja koje algoritam uzima u obzir, algoritam je nadopunjen s prioritetnim redom koji pamti određen broj najboljih viđenih rješenja. Potom se prilikom ažuriranja distribucije uz relativnu frekvenciju novih uzoraka u obzir uzimaju i uzorci iz prioritetnog reda. Kako je red fiksne duljine, jednom kada dosegne svoju maksimalnu veličinu, novi element se ubacuje tek ako je bolji od najgoreg rješenja u redu, a najgore rješenje u redu se izbacuje. Baš zbog prethodno navedene potrebe da se često pristupa elementu s najvećom kaznom je odabrana struktura prioritetnog reda. Dodavanjem prioritetnog reda povečava se količina dobrih rješenja koja algoritam uzima u obzir prilikom ažuriranja trenutne distribucije. Da bi se spriječilo da red usporava napredak algoritma zadržavajući ga u lokalnim optimumima, red se periodički potpuno isprazni.

5.6. Rad algoritma

Osim prethodno navedenih nadogranja, sama srž algoritma je ostala ista kao i u LINK-NA-RAD. Nakon što se inicijalizira početna distribucija započinje petlja u kojoj se uzorkuju novi uzorci, zatim se oni ocjenjuje te se gradi distribucija relativnih frekvencija uzoraka s najmanjom kaznom. Nakon što se ternutna distribucija ažurira, provode se heuristike opisane u poglavlju 5.4. Pseudokod ovog algoritma je prikazan na 4. Algoritam nastavlja s radom sve dok se ne ispuni uvjet zaustavljanja. Uvjeti zaustavljanja mogu biti različiti. Jedan od načina da se zaustavi algoritam je da se u vršnom direktoriju projekta stvori datoteka s imenom $stop.txt$. Ovakav uvjet zaustavljanja je

praktičan kada postoji potreba da se prati kako algoritam napreduje te da ga se prekine jednom kada dosegne željenu kvalitetu rasporeda. Ovaj pristup je korišten za vrijeme razvijanja te bi bio pogodan za korištenje u produkcijskom okruženju. Za potrebe automatskog vrednovanja je prikladnije algoritam zaustaviti nakon određenog broja iteracija ili nakon što iznos kazne najboljeg rješenja dosegne zadanu vrijednost.

Algorithm 4 Rad algoritma

repeat
  $uzorci \leftarrow uzorkovanje()$
  $relativnaFrekvencija \leftarrow ocjeninjivanje(uzorci)$
  $auriranje(relativnaFrekvencija)$
  $pokreniHeuristike()$
until $uvjetZaustavljanja()$

6. Zaključak

Zaključak.

Primjena konstruktivnog optimizacijskog algoritma na problem rasporeda studenata

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.