IZVJEŠTAJ PROJEKTA IZ PREDMETA HMO 2018./2019. Autori : Marin Krešo i Stjepan Petruša

OPIS PROBLEMA

Studenti *Fakulteta elektrotehnike i računarstva* (FER) imaju mogućnost promijeniti grupu u kojoj slušaju određenu aktivnost na nekom predmetu (npr. mogu promijeniti grupu predavanja za Matematiku 1, ili možda grupu za neku laboratorijsku vježbu iz Digitalne Logike). Budući da su resursi (predavaonice) ograničeni i zbog moguće velikog broja zahtjeva za promjenom grupe, zadatak je kompleksan i potrebno ga je optimizirati kako bi ostvarili što veći broj zamjena uz zadana ograničenja.

PODATKOVNI SKUP

Podaci na ulaz dolaze kroz 4 datoteke :

- student-file - Ova datoteka sadrži inicijalne podatke o inicijalnim grupama studenata - koji student (student_id) je u kojoj grupi (group_id) na kojoj aktivnosti (activity_id).
- requests-file - Ova datoteka sadrži popis zahtjeva za zamjenom grupe.
- overlaps-file - Ova datoteka sadrži popis svih zabranjenih kombinacija grupa. Riječ je o grupama koje imaju, na primjer, preklapanje u rasporedu i slično.
- limits-file - Ova datoteka sadrži popis svih ograničenja o broju studenata u određenoj grupi

OPIS ALGORITMA

Opisani algoritmi su implementirani u programskom jeziku Python 3.6.5

Prikaz rješenja

Svako potencijalno rješenje tokom izvođenja algoritma je predstavljeno klasom *Solution* u programskom jeziku Python. Rješenje sadrži sljedeće atribute :

- listu obavljenih zamjena prikazanih kao uređene trojke u obliku (id studenta, id aktivnosti, id željene grupe).
- limite grupa ažuriranih sa obavljenim zamjenama.
- listu studenata skupa s njihovim trenutnim grupama na aktivnostima koje pohađaju.

Funkcija cilja

Kao funkciju cilja jedan od logičnih izbora bi bio broj obavljenih zamjena. Pozivi takvih evaluacija nisu vremenski zahtjevni i funkcija nam vrednuje više rješenja kod kojih je veći broj obavljenih zamjena - što i želimo. No, izborom takve funkcije cilja imamo određenih problema. Iako su limiti neke grupe/predavaonice zadovoljeni, možda bi svima bilo ugodnije kada bi broj studenata u toj grupi/predavaonici bio manji i raspoređen po ostalim grupama kako bi kvaliteta predavanja bila veća. Također, algoritmu je svejedno da li je obavio pet zamjena za jednog studenta ili po jednu zamjenu za pet studenata. Možda bi htjeli veću slobodu i mogućnost optimiranja takvih odluka. Zato sam se odlučio za drugu funkciju cilja, koja nam daje više mogućnosti (ali je vremenski zahtjevnija za izračunavanje). Funkcija je predstavljena idućim izrazom : BodoviA + BodoviB + BodoviC – BodoviD – BodoviE. Bodovi se računaju na idući način:

- BodoviA - ukupni zbroj svih težišnih faktora koje smo dodijelili studentima koji traže za zamjenu
- BodoviB - računaju se kao zbroj svih nagradnih faktora za studenta. Nagradni faktor se računa temeljem broja aktivnosti za koje je napravljena zamjena za jednog studenta (ovime možemo razriješiti prethodno navedenu dvojbu vrednovanja istog broja zamjena s različitim brojem studenata kojima je odobrena zamjena).
- BodoviC - računaju se za studente kod kojih se uspješno provedu zamjene za sve aktivnosti na kojima su tražili zamjene prema umnošku broja takvih studenata i nagradnog faktora za potpune zamjene
- BodoviD - računaju se kao suma bodova svih grupa kod kojih je broj studenata manji od minimalnog preferiranog broja studenata u grupi.
- BodoviE - računaju se kao suma bodova svih grupa kod kojih je broj studenata veći od maksimalnog preferiranog broja studenata u grupi.

Algoritam

Algoritam koji se koristi za zamjenu grupa je kombinacija više različitih heuristika.

Pohlepni algoritam (početno rješenje)

Ovaj algoritam koristimo kako bi dobili početno rješenje. Njegova implementacija je jednostavna. Algoritam rješava zahtjeve onim redom koje zaprima. Ako u tom trenutnu može obaviti zahtjev, to i učini te pređe na idući zahtjev. Ako nije moguće odobriti zahtjev preskačemo ga i idemo na idući. Važno je napomenuti da kako bi zahtjev bio odobren on treba zadovoljiti i soft ograničenja (mora se zadovoljiti minimalni i maksimalni preferirani broj studenata u grupi). To činimo kako ne bi zapeli u lokalnom minimumu i kako bi idući algoritmi imali veći manevarski prostor. Obavlja se više prolaza kroz zahtjeve (ovo ne predstavlja multistart pohlepni algoritam), jer se može dogoditi da se zbog neke zamjene koja je obavljena i u redu se nalazi iza zamjene koja nije obavljena (promatrajmo zamjene kao red), u idućem prolazu zbog obavljanja navedene zamjene se ovaj put može obaviti i ta neobavljena zamjena. Također, zahtjeve prije početka rada algoritma promiješamo slučajnim odabirom kako ne bi preferirali zamjene koje se nalaze pri početku skupa podataka. Izmjenom redoslijeda praktički imamo rad gdje na slučajan način izvlačimo jednu po jednu zamjenu i gledamo da li ju možemo obaviti. Za diskusiju je važno napomenuti da ako želimo preferirati zamjene koje su zatražene ranije od drugih, to možemo učiniti ovdje sortirajući zamjene uzlazno po vremenu zatraženja zamjene. Također je važno dodati da ako je student zatražio zahtjev za grupom koja je u istom terminu kad i trenutna (time je automatski i u datoteci koja nam daje preklapanja) odobravamo takav zahtjev ukoliko su ostala ograničenja zadovoljena

Hibridni algoritam (kombinacija tabu pretraživanja i simuliranog hlađenja)

Nakon što generiramo populaciju sa pohlepnim algoritmom, ta populacija je ulaz u idući algoritam. Taj algoritam je većinom implementacija algoritma simuliranog hlađenja uz dodatak tabu pretraživanja pri generiranju susjedstva. Ideja je sljedeća - poništavanjem neke zamjene u trenutačnom rješenju možemo osloboditi mjesta za jednu ili više drugih zamjena nakon kojih ćemo imati bolju vrijednost funkcije evaluacije. Poništeni zahtjev ignoriramo pri traženju drugih zamjena (ovo nam predstavlja tabu pretraživanje). Ako smo dobili bolje rješenje njega prihvaćamo, inače stohastički gledamo da li ćemo prihvatiti lošije rješenje (simulirano hlađenje). Ukoliko smo prihvatili lošije rješenje, pamtimo zahtjev koji ne smijemo obaviti i dodajemo mu novi zahtjev (dakle, sada imamo listu zabranjenih zahtjeva). Poništavanjem i ignoriranjem možemo pobjeći van lokalnog optimuma u područja s kvalitetnijim vrijednostima funkcije cilja. Konstantno poboljšavanje rješenja kroz vrijeme ide u prilog ovoj tezi.

Direktne zamjene

Jedan dio algoritma je i traženje direktnih zamjena. Ukoliko se student A ne može prebaciti u traženu grupu jer je popunjena, možda postoji student B koji želi izaći iz te grupe. Ako student B nije uspio otići u neku drugu grupu i ima zahtjev za grupom kojoj pripada student A možemo obaviti direktnu zamjenu.

Kriterij zaustavljanja

Kriterij zaustavljanja je vremensko ograničenje. Implementirani algoritam prati situaciju s rješenjem i ukoliko se ono nije uspjelo poboljšati velik broj iteracija, "resetirati" ćemo algoritam i krenuti pretraživati rješenje van lokalnog optimuma u kojem smo "zapeli". Time se može dogoditi i da nakon puno iteracija koje nam nisu donijele poboljšanje, ipak nađemo neku promjenu koja nas dovodi do poboljšanja. Ostali kriteriji zaustavljanja poput broja iteracija se mogu lagano dodati unutar algoritma.

PSEUDOKOD

Kako bi bolje razumjeli gore navedene algoritme, pokazati ćemo pseudokod algoritma.

```
def pohlepni_algoritam():
        for i in broj_prolaza:
            for zahtjev in zahtjevi:
                ako_zadovoljava_ogranicenja:
                    odobri_zahtjev()

def tabu_i_simulirano_hladenje(pocetno_rjesenje):
        inicijaliziraj_parametre()
        best = najbolje_rjesenje()
        trenutno = best
        while not uvjet_zaustavljanja:
            temp = izbaci_i_zabrani_zahtjev(slucajan_odabir)
            ako temp >= trenutno:
                trenutno = temp
                isprazni_zabranjena()
            inace ako p > p_prihvati_losije:
                trenutno = temp
            inace:
                isprazni_zabranjena()

            ako trenutno > best:
                best = trenutno


def main():
  studenti = ucitaj_studente()
  limiti = ucitaj_limite()
  preklapanja = ucitaj_preklapanja()
  zahtjevi = ucitaj_zahtjeve

  promijesaj_zahtjeve(zahtjevi)
  pocetno_rjesenje = pohlepni_algoritam() + direktne_zamjene()
  najbolje_rjesenje = Tabu_i_Simulirano_hladenje(pocetno_rjesenje)
```

```
    vrati najbolje_rjesenje
```

UTJECAJ PARAMETARA

Jako je važno izabrati dobre parametre za navedene algoritme kako bi došli do boljih rješenja.
Eksperimentiranjem smo izveli iduće zaključke za odabrane parametre:

- Većim brojem prolaza kroz zahtjeve pohlepnim algoritmom dolazimo do većeg broja zamjena. Ovim pristupom se ne zapada u lokalne optimume jer ih izbjegavamo simuliranim hlađenjem i tabu pretraživanjem kojima istražujemo rješenja van lokalne optimalnosti.
- Većim početnim iznosom temperature hlađenja dobivamo bolja rješenja. To je lako objasniti time što ipak povećavamo prostor rješenja i možemo naći na bolje rješenje od trenutnog.
- Algoritam je elitistički koje god parametre odabrali.
- Izradom liste zabranjenih zahtjeva (ne samo da pamtimo jednu poništenu zamjenu) pronalazimo kvalitetnija rješenja
- Dužim vremenom pretraživanja pronalazimo bolja rješenja

DISKUSIJA O "FAIRNESS-u" ZAMJENE GRUPA

Pitanje je treba li algoritam prioritizirati studente koji traže samo jednu zamjenu ili studente koji traže više zamjena. Pitanje je vrlo delikatno i ovisi o puno čimbenika. Studenti koji traže puno zamjena možda žele u potpunosti promijeniti raspored jer im sadašnji ne odgovara (žele imati predavanja ujutro, a ne popodne). Time bi oni imali možda prednost nad studentima kojima ne odgovara jedan predmet, jer se prvom studentu trebaju obaviti sve zamjene (par zamjena mu ne donosi veliku korist), dok student s jednom zamjenom samo želi slušati predmet kod drugog profesora (nije mu toliko bitna zamjena). U suprotnom je situacija jasna: ako odobrimo jednom studentu 5 zamjena, drugome nijednu, teško je poreći favoriziranje prvog. Nije svaka situacija ista i zato je po meni situaciji potrebno pristupiti s druge strane : Treba nas samo zanimati kako obaviti što više zamjena i gledati pri time popunjenost dvorana. Time imamo "najpravedniji" algoritam koji ne favorizira nikoga, nego ovisi o slučajnosti, odnosno o situaciji koja donosi najviše zamjena. Također treba uzeti u obzir da je postupak iterativan, jer neće svi studenti odmah zatražiti zamjenu, studenti često dodaju nove zahtjeve itd. Što bržim obavljanjem što većeg broja zamjena student ima bolji uvid u situaciju i može ažurirati svoje zahtjeve.

ZAKLJUČAK (What's next ?)

Pametnim korištenjem heuristika i algoritama možemo dobiti vrlo dobru optimizaciju (u razumnom vremenu) problema koji nam je zadan. Pomnijim odabirom parametara možemo dobiti i bolja rješenja. No, možda bi neke druge heuristike i algoritmi donijeli bolja rješenja. Moguće poboljšanje bi bili algoritmi koji su zasnovani na populaciji rješenja (genetski algoritam, imunološki algoritam...). Uz pametan odabir operatora bi ti algoritmi možda mogli donijeti i bolja rješenja. Međutim, ovdje smo pokazali da i korištenjem "jednostavnijih" algoritama možemo dobiti rješenja zadovoljavajuće kvalitete u razumnom vremenu.