Univerza v Ljubljani
Fakulteta za matematiko in fiziko
Finančna matematika - 1. stopnja

Matej Škerlep

Problem največje neodvisne množice

Projekt pri predmetu finančni praktikum

Mentorja: prof. dr. Riste Škrekovski in asist. dr. Janoš Vidali

Ljubljana, 2020

Kazalo

Slike

1 Predstavitev problema

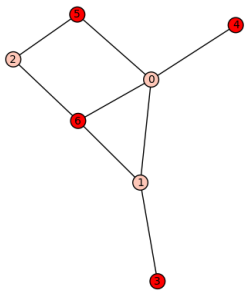
1.1 Navodila za delo

Definirajte problem največje množice nesosednjih vozlišč kot CLP in ga rešite za nekaj primerov. Eksperimentalno primerjajte rezultate CLP in njegove relaksacije na LP in ugotovite, za koliko se lahko razlikujejo med sabo po velikosti. Napišite algoritem za lokalno iskanje po grafu in njegov rezultate primerjajte s prejšnjimi. Ugotovite za kako velike grafe je posamezen izmed primerov rešljiv.

1.2 Definicije pojmov

Definicija. Naj bo $G = (V, E)$ graf in $I \subseteq V$. Množica vozlišč I je ***neodvisna***, če ne vsebuje sosednjih vozlišč.
Formalno, če za $\forall v, u \in V$, $uv \in E$ velja $v \in I \Leftrightarrow u \notin I$.

Primer. Oglejmo si preprost primer največje neodvisne množice na grafu s 7 vozlišči. Množica vozlišč $\{3, 4, 5, 6\}$ predstavlja največjo množico neodvisnih vozlišč.



Slika 1: Primer največje neodvisne množice na grafu s sedmimi vozlišči

1.3 Celoštevilski linearen program

Tako kot zgoraj v definiciji, bo v celotni nalogi največja neodvisna množica označevana z I. Za začetek si oglejmo celoštevilski linearen program za opisani problem. Vsakemu vozlišču $v \in V$ priredimo spremenljivko x_v z vrednostmi 0 ali 1. Pri tem je

$$x_v = \begin{cases} 1; & v \in I \\ 0; & v \notin I \end{cases}$$

Pogoj $x_u + x_v \leq 1$ nam tako zagotovalja, da ima vsaka povezava $uv \in E$ največ eno vozlišče v množici I. Vrednost kriterijske funkcije spodnjega CLP je enaka moči množice I, v njej pa so vsa vozlišča v za katera je $x_v = 1$.

$$\begin{array}{ll} \max & \sum_{v \in V} x_v \\ \text{p.p.} & x_u + x_v \leq 1, \quad \text{za vsak par } uv \in E \\ & x_v \in \{0,1\}, \quad \forall v \in V \end{array}$$

1.4 Relaksacija celoštevilskega linearnega programa

V primeru relaksacije LP pogoj $x_v \in \{0,1\}$ zamenjamo s pogojem $0 \leq x_v \leq 1$:

$$\begin{array}{ll} \max & \sum_{v \in V} x_v \\ \text{p.p.} & x_u + x_v \leq 1, \quad \text{za vsak par} uv \in E \\ & 0 \leq x_v \leq 1, \quad \forall v \in V. \end{array}$$

V tem primeru dobimo trivialno rešitev $x_v = \frac{1}{2}$ za $\forall v \in V$, ki pa nam o tem katera vozlišča so v množici I ne pove ničesar. Zato bomo v množico I dodajali zgolj tista vozlišča, ki bodo imela vrednost $x_v > \frac{1}{2}$. S tem zagotovimo, da bo v I nastopalo največ eno vozlišče vsake povezave, s čemer bomo dobili neodvisno množico in torej dopustno rešitev. V nadaljevanju bomo videli, da se pogosto zgodi, da je prav rešitev $x_v = \frac{1}{2}$ za $\forall v \in V$ optimalna in posledično (glede na pogoj $x_v \in I \iff x_v > \frac{1}{2}$) bo množica I prazna.

2 Algoritmi

2.1 Izbira programskega jezika in knjižnic

Algoritmi so napisani v programskem jeziku *Sage*, ki temelji na programskem jeziku *Python*. Uporabil sem module *time* za merjenje časa za izvedbo posameznega algoritma, *random* za generiranje naključnega nabor vozlišč v algoritmu lokalnega iskanja in *csv* za shranjevanje rezulatov poskusov v csv obliki.

V kodi sem najprej napisal vse tri funkcije `clp`, `rclp` in `lokalno_iskanje`. Nato sem za namen testiranj napisal funkcijo `testiraj`, ki na naključnem grafu (kot argumente grafa določimo število vozlišč in verjetnost povezave med dvema vozliščema) izvede vse tri zgoraj omenjene funkcije in kot rezultat vrne seznam izhodnih podatkov vseh treh funkcij. Na koncu pa je še funkcija `izpis_csv`, ki rezultate testiranj zapiše v csv datoteteko. Posamezna vrstica v csv datoteki predstavlja izhodne podatke vseh treh algoritmov za graf na določenem številu vozlišč.

Za namene analize časovne zahtevnosti, so v vseh treh funkcijah uporabljeni klici `time.time()`, čas celotne izvedbe posameznega algoritma pa nato dobimo kot razliko med končnim in začetnim časom (t.j. `cas = kon - zac`).

2.2 Celoštevilski linearni program

```
def clp(graf):
    zac = time.time()

    CLP = MixedIntegerLinearProgram(maximization = True)
    vozlisca = CLP.new_variable(binary = True)
    CLP.set_objective(sum([vozlisca[v] for v in graf.vertices()]))
    for u,v in graf.edges(labels = False):
        CLP.add_constraint(vozlisca[u] + vozlisca[v] <= 1)

    moc_max = CLP.solve()
    vozlisca = CLP.get_values(vozlisca)

    sez = [k for k, v in vozlisca.items() if v == 1.0]

    kon = time.time()
    cas = kon - zac
    return [int(moc_max), sez, cas]
```

Funkcija clp kot vhodni argument vzame nek graf. Celoštevilski linearni

program najprej definiramo s funkcijo `MixedIntegerLinearProgram`. Nato za vsako vozlišče priredimo binarno spremenljivko `vozlisca` z vrednostmi 0 ali 1. Kriterijsko funkcijo definiramo s klicem funkcije `set_objective`, omejitev za vsaki dve povezani vozlišči pa z `add_constraint`. Optimalno vrednost kriterijske funkcije dobimo s `CLP.solve()`, kar je hkrati tudi moč največje neodvisne množice. Vrednost x_v posameznega vozlišča pa dobimo z `CLP.get_values(vozlisca)`. V iskano največjo množico neodvisnih vozlišč `sez` torej dodamo vozlišča z vrednostjo x_v enako 1. Kot izhodni podatek na koncu dobimo seznam oblike moc najvecje neodvisne mnozice, seznam vozlisc v tej množici in čas izvedbe algoritma. Funkcija `int()` nam število s plavajočo vejico zapiše v celoštevilski obliki, kar sem uporabil Zaradi težav Excela pri branju podatkov, sem uporabil funkcijo `int()`, ki nam število s plavajočo vejico zapiše v celoštevilski obliki.

2.3 Relaksacija celoštevilskega linearna programa

```
def rclp(graf):
    zac = time.time()

    RCLP = MixedIntegerLinearProgram(maximization = True)
    vozlisca = RCLP.new_variable(real = True)
    RCLP.set_max(vozlisca,1)
    RCLP.set_min(vozlisca,0)
    RCLP.set_objective(sum([vozlisca[v] for v in graf.vertices()]))
    for u,v in graf.edges(labels = False):
        RCLP.add_constraint(vozlisca[u] + vozlisca[v] <= 1)

    RCLP.solve()
    vozlisca = RCLP.get_values(vozlisca)

    sez = [k for k, v in vozlisca.items() if v > 0.5]
    moc_max = len(sez)

    kon = time.time()
    cas = kon - zac
    return [int(moc_max), sez, cas]
```

Večina komentarjev je podobnih kot pod algoritmom za CLP v razdelku 2.2, zato bom tu obrazložil le tisto kar se razlikuje. Pri relaksaciji CLP vrednosti x_v omejimo z `set_max` oz. `set_min`. Vrednost kriterijske funkcije nam o moči množice ne pove prav dosti, zato z argumenti iz razdelka 1.4, dodajamo v

največjo množico neodvisnih vozlišč (v algoritmu označena s `sez`) le tista vozlišča z $x_v > \frac{1}{2}$. Funkcija sprejme in vrne enake argumente kot `clp`.

2.4 Lokalno iskanje

```
def lokalno_iskanje(graf, st_ponovitev):
    moc_max = 0
    max_mnozica_neod = set()
    zac = time.time()
    for i in range(st_ponovitev):
        mnozica_neod = set()
        mnozica_neod.add(graf.random_vertex())
        for v in random.sample(graf.vertices(), len(graf.vertices())):
            if not any(u in mnozica_neod for u in graf[v]):
                    mnozica_neod.add(v)
        if len(mnozica_neod) > moc_max:
            max_mnozica_neod = mnozica_neod
            moc_max = len(mnozica_neod)
    kon = time.time()
    cas = kon - zac
    return[moc_max, max_mnozica_neod, cas]
```

Ideja zgornjega algoritma je sledeča. Začnemo s prazno množico I v katero dodamo naključno izbrano vozlišče. Nato po naključnem naboru vozlišč, v I dodajamo tista vozlišča, katerih sosedje niso v množici I. To zanko ponovimo poljubno mnogokrat in nato vzamemo tisto I, ki ima največjo moč.

Opomba. Za razliko od zgornjih dveh algoritmov, kjer sem delal s seznamom, sem pri lokalnem iskanju uporabil množico, saj je s tem zmanjšana časovna zahtevnost algortima (pričakovano konstantna namesto linearne).

3 Rezultati

V namen analize algoritmov sem vse tri zgornje funkcije izvedel na istih naključnih grafih. Kot argumente generiranja grafa sem vzel vse možne kombinacije števila vozlišč (1 do 100) in verjetnosti povezave dveh vozlišč ($\frac{1}{10}, \frac{3}{10}, \frac{5}{10}, \frac{7}{10}$ in $\frac{9}{10}$). Pri tem sem število ponovitev lokalnega iskanja nastavil na 50.

Primerjal sem moči največjih neodvisnih množic vseh treh algoritmov. Podrobnejša analiza je dostopna v datoteki *analiza_testov.xlsx*, tu pa bom opisal ugotovitve. Generalno gledano za vse rezultate testiranj velja, da je

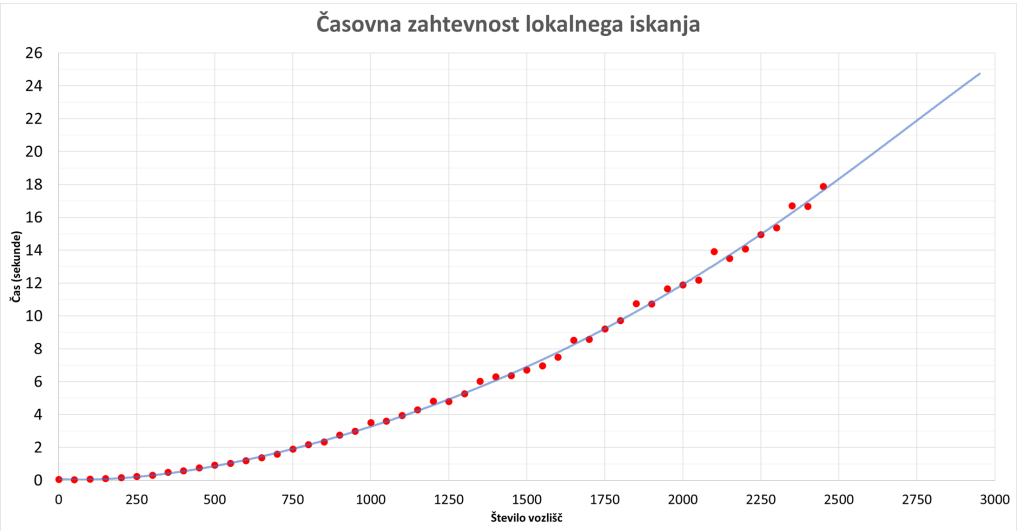
moč množice dobljene s CLP vedno večja ali enaka moči množice dobljene z lokalnim iskanjem.

Zelo opazna je neučinkovitost relaksacije CLP. Slednja je razmeroma učinkovita v primeru zelo nepovezanih grafov. V primeru verjetnosti 0,1 vsi trije algoritmi vse do grafa z 18 vozlišči vračajo enako moč, nato se enakost moči CLP in njegove relaksacije le ponekod ujemata, od grafa z več kot 34 vozlišči, pa se moči dobljenih množic ne ujemata več. Pri tej majhni verjetnosti, algoritem relaksacije CLP od grafa s 74 vozlišči vrača prazno množico.
Pri

4 Časovna zahtevnost

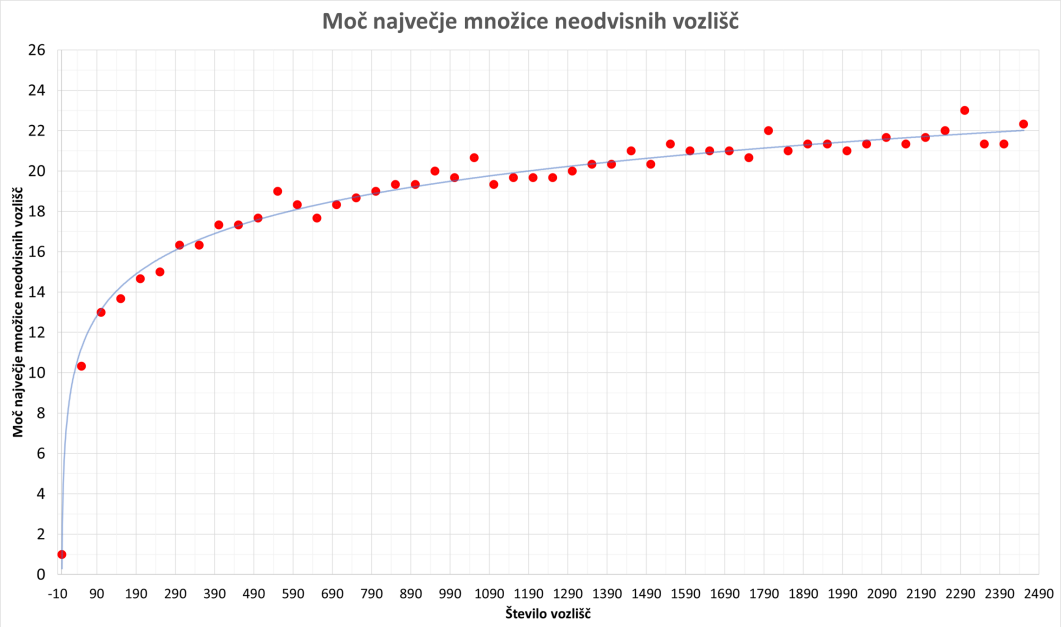
Pri analizi časovne zahtevnosti sem se omejil zgolj na algoritem za lokalno iskanje. Časovne zahtevnosti CLP nisem meril, saj je izjemno neučinkovit, prav tako ne relaksacije CLP, saj nam za velike grafe skoraj vedno vrača prazno množico.

V namen analize časovne zahtevnosti algoritma lokalnega iskanja sem na grafu z verjetnostjo povezave dveh vozlišč 0,3 in 50 ponovitvami zunanje zanke izvedel algoritem. Neodvisna spremenljivka je bila torej število vozlišč, ki sem jo povečeval s korakom 50 do končne velikosti grafa z 2451 vozlišči. Meritve časa sem izvedel trikrat in nato vzel povprečje vseh treh časov pri posamezni velikosti grafa. Na grafu je prikazan tudi aproksimacijski polinom šeste stopnje in napoved za povečevanje časovne zahtevnosti za grafe do velikosti 3000 vozlišč.



Slika 2: Časovna zahtevnost lokalnega iskanja v odvisnosti od števila vozlišč

Na enakih podatkih kot zgoraj sem poleg časovne zahtevnosti pogledal tudi moč največje neodvisne množice vozlišč v odvisnosti od števila vozlišč.



Slika 3: Velikost največje množice neodvisnih vozlišč v odvisnosti od števila vozlišč

5 Zaključek