Iskanje kromatičnega števila grafa: Projekt pri predmetu operacijske raziskave

Lonja Tjaša Tavčar in Žan Šifrer

21. februar 2017

Najina naloga

Naloga najinega projekta je generirati grafe, ki so k-barvni za $k = 2, 4, ...$ in poiskati njihova kromatična števila z uporabo celoštevilskega linearnega programa. Analizirati pa morava tudi število grafov, ki jih bova generirala kot k-barve, a so v resnici $(k-1)$-barvni.

Splošno o barvanju grafov

Barvanje grafa je preslikava, ki vsakemu vozlišču grafa priredi neko barvo. Vsako barvo pa označimo kar z naravnim številom. Dobro ali pravilno barvanje vozlišč je tako, da so sosednja vozlišča pobarvana z različnimi barvami. Najmanjše število k, za katerega obstaja dobro k-barvanje (minimalno barvanje) vozlišč grafa imenujemo kromatično število grafa. Označimo ga z $\chi(G)$. Barvanje je minimalno, ko je graf pobarvan s številom barv, ki je enako kromatičnemu številu grafa.
Za nakatere grafe kromatična števila poznamo:

- Polni graf z n vozlišči ima kromatično število n
- Ciklični graf ima kromatično število 2, če ima sodo število vozlišč ali 3, če ima liho število vozlišč
- Kolo ima kromatično število 3, če ima liho število vozlišč ali 4, če ima sodo število vozlišč

Pri iskanju kromatičnega števila upoštevamo omejitve:

- $1 \leq \chi(G) \leq n$ (pri čemer je n število vozlišč grafa)

- edini grafi, ki so lahko 1-barvni so točke
- če je H podgraf grafa G je $\chi(H) \leq \chi(G)$
- če graf vsebuje kliko velikosti k, potem je potrebnih vsaj k barv, da pobarvamo to kliko, torej je $\chi(G) \geq k$.
- Če graf vsebuje zanko (torej povezavo, ki izstopa iz in vstopa v isto vozlišče), ne more imeti pravilnega barvanja.

Potek projekta

algoritmi in psevdokode in ILP

Algoritem za generiranje polnih obarljivih grafov

Kombinacije vozlišč za k-obarvljiv graf z n vozlišči.

```
def kombinacije(k,n):
    z=n-k
    A=matrix(1,k)
    komb=[]
    for i in range(k):
        A[0,i]=1
    if z>k:
        a=k
    else:
        a=z
    for j in range(a):
        x=matrix(1,k)
        for m in range(j+1):
            if m==0:
                x[0,m]=(z-j)
            else:
                x[0,m]=1
        komb.append(x+A)
        ali=True
        while ali:
            t=0
            desno=True
```

```
            while desno:
                if x[0,0]==x[0,t+1]:
                    t=t+1
                    if t==k-1:
                        desno=False
                else:
                    desno=False
            spr2=False
            spr1=True
            for u in range(t+1,k):
                if (spr1)and not(spr2):
                    if x[0,u]==0:
                        ali=False
                        spr1=False
                    else:
                        if (x[0,u]+2)<=x[0,t]:
                            x[0,t]=x[0,t]-1
                            x[0,u]=x[0,u]+1
                            spr2=True
            if spr2:
                komb.append(x+A)
            else:
                ali=False
    return komb
```

Za primer: kombinacije(4,11) dobimo: [[8 1 1 1], [7 2 1 1], [6 3 1 1], [5 4 1 1], [6 2 2 1], [5 3 2 1], [4 4 2 1], [4 3 3 1], [5 2 2 2], [4 3 2 2], [3 3 3 2]]

```
def pretvorivA(G):
    z = 1
    for P in G:
        for j in P:
            if j > z:
                z=j
    A=matrix(z)
    for (i,j) in G:
        A[(i-1),(j-1)]=1
        A[(j-1),(i-1)]=1
    return A

def pretvorivG(A):
    x=1
```

```
    G=[]
    for z in A:
        if x==1:
            m=len(z)
            x+=1
    for i in range(m):
        for j in range(i,m):
            if A[i,j]==1:
                G.append(((i+1),(j+1)))
    return G

def generiraj_polni(M):
    n=0
    k=0
    for i in M:
        k = k + 1
        n = n + i
    X=matrix(k,n)
    s=0
    for j in range(k):
        for z in range(M[j]):
            X[j,s]=1
            s=s+1
    G=[]
    for d in range(k):
        for e in range(n):
            if X[d,e]==1:
                for f in range(n):
                    if X[d,f]==0:
                        G=G+[((e+1),(f+1))]
    A=pretvorivA(G)
    return(A)
```

Linearni program:

Za iskanje kromatičnega števila sva napisala celoštevilski linearni program. Najprej sva definirala binarni spremenljivki z naslednjima pomenoma:

$$y_k = \begin{cases} 1, & \text{če uporabimo barvo } k \\ 0, & \text{sicer} \end{cases}$$

in še:

$$x_{i,k} = \begin{cases} 1, & \text{če je vozlišče } i \text{ barve } k \\ 0, & \text{sicer} \end{cases}$$

V najinem primeru sta k in i števili od 1 do n, saj i predstavlja število vozlišč, ki jih je n; k pa predstavlja število barv, ki jih uporabimo. V najslabšem primeru je vsako vozlišče svoje barve, torej uporabimo n barv in v tem primeru je $k = n$.
Ker iščeva najmanjše število barv za graf, mora najin program iskati minimum.

$$min \sum_{k=1}^{n} y_k$$

p.p.

$$\sum_{k=1}^{n} x_{ik} = 1, \qquad \text{za} \qquad i = 1, ..., n \tag{1}$$

$$x_{i,k} - y_k \leq 0, \qquad \text{za} \qquad i, k = 1, ..., n \tag{2}$$

$$x_{i,k} + x_{j,k} \leq 1, \qquad \text{kjer} \qquad (i,j) \in E(G), k = 1, ..., n \tag{3}$$

$$0 \leq x_{i,k}, y_k \leq 1 \tag{4}$$

$$x_{i,k}, y_k \in \mathbb{Z} \tag{5}$$

Pogoj (1) pomeni, da je vsako vozlišče neke barve, in da je lahko samo ene barve. Pogoj (2) pomeni, da vozlišče i dobi barvo k samo, če je barva k sploh uporabljena (torej v primeru, ko je $y_k = 1$. Naslednji pogoj (3) zagotovi, da sosednji vozlišči ne smeta biti iste barve. Z zadnjima pogojema (4) in (5) pa si zagotovimo, da sta x in y binarni spremenljivki in celi števili.

Časovna zahtevnost algoritma

Ker so y spremenljivke odvisne od x, bi časovno zahtevnost algoritma ocenila na podlagi x spremenljivk.
Denimo, da imamo graf z n vozlišči. Ko skonstruiramo spremenljivke $x_{i,k}$ dobimo n^n spremenljivk in ker so spremenljivke enake 1 ali 0 imamo za vsako 2 možnosti, torej je vseh možnih kombinacij za x 2^{n^2}, če pa bi gledali vse

možne kombinacije bi morali za vsako preveriti $O(n)$ omejitev. Torej, če bi recimo sestavili celoštevilski program, ki preveri vse možnosti bi bila časovna zahtevnost $O(n * 2^{n^2})$. To pomeni, da bi se pri takem algoritmu izogibali grafov z velikim številom vozlišč. Sicer pa je časovna zahtevnost, seveda odvisna od sestave celoštevilskega linearnega programa in je dosti manjša, vendar še vedno narašča eksponentno.

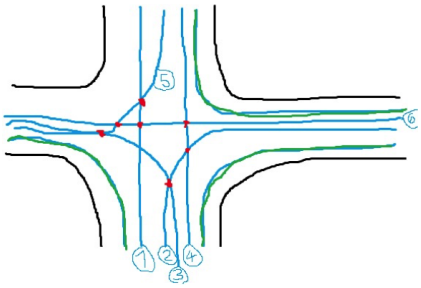
Uporaba algoritmov za preverjanje kromatičnega števila grafa v praksi

V resničnem življenju lahko veliko problemov prevedemo na iskanje kromatičnega števila grafa, kar nam pomaga pri iskanju optimalne rešitve problema. Nekateri primeri so:

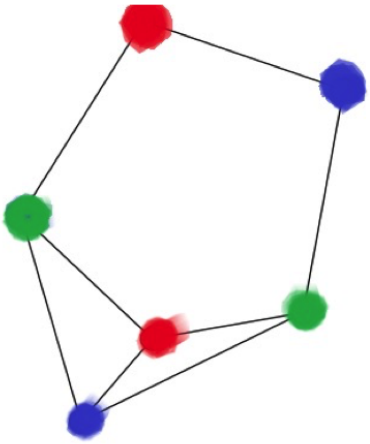
- Denimo, da imamo več radijskih postaj, ki se med sabo sporazumevajo. Postaje imamo razporejene po zemljevidu, in če sta 2 radijski postaji dovolj blizu ena drugi, se lahko motita. Torej za vozlišča uporabimo radijske postaje, povezave pa naredimo med tistimi potajami, ki so si dovolj blizu. Zdaj iščemo vsaj koliko različnih radijskih frekvenc potrebujemo, da se naprave med seboj ne bodo motile. Različne radijske frekvence so v tem primeru barve.

- Malce bolj nenavaden primer, pri katerem morda sprva ne bi pomislili, da se rešuje z iskanjem kromatičnega števila, pa najdemo pri cestah. Recimo, da imamo križišče na cesti in gledamo koliko najmanj intervalov za semaforje potrebujemo, da bodo vsi avtomobili prevozili križišče brez trčenja. Narišemo krivulje, po katerih avtomobili potujejo-te označimo za vozlišča. Povezave v grafu dobimo tako, da smeri povežemo, če se v križišču sekata. Barve pa so faze semaforskega režima. Tiste krivulje, ki ne sekajo nobene poti lahko zanemarimo. Postopek lahko vidimo na zgornjih slikah (1a, 1b, 1c, 1d)

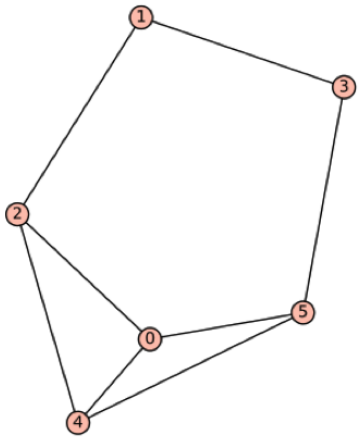
Viri



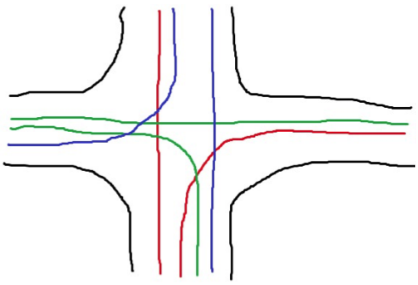
(a) Poti avtomobilov



(b) Graf



(c) Pobarvan graf



(d) Prevoz križišča