PROBLEM NAJVEČJIH KLIK

Avtor: Tamara Eissa

Datum: 26. 3. 2022

Problem

Dan je neusmerjen graf z V vozlišči in P povezavami. Želimo najti največje (ang. maximal) klike v danem grafu.

Definicije

Klika

Klika je poln podgraf danega grafa.

Podgraf

Nek graf A je podgraf grafa B natanko tedaj, ko graf A vsebuje izključno nekatera izmed vozlišč grafa B ter graf A vsebuje izključno nekatere izmed povezav grafa B.

Poln graf

Poljuben graf je poln natanko tedaj, ko je vsako vozlišče danega grafa povezano z vsemi ostalimi vozlišči danega grafa (torej obstaja povezava med poljubnima dvema vozliščema danega grafa).

Predpostavke:

- prazen graf je klika poljubnega grafa,
- poljubna točka danega grafa je klika tega grafa.

Kaj je največja klika?

Največja klika je klika. Pojem "največja" pa ima dva pomena, ki izvirata iz angleščine.

1. Klika je največja (ang. maximum), kadar vsebuje največje število vozlišč. Torej je največja oziroma ena izmed največjih klik danega grafa glede na število vozlišč, vsebovanih v kliki.

2. Klika je največja (ang. maximal) natanko tedaj, ko ni podgraf druge klike. To pomeni, da je največja klika klika, ki ob vključitvi poljubnega vozlišča, ki ni del trenutne klike, ni več klika danega grafa.

V nadaljevanju se bomo osredotočili predvsem na največje klike iz druge točke.

Motivacija

Dano je socialno omrežje, katerega vozlišča predstavljajo ljudi, povezave pa poznanstva med njimi. Če želimo najti skupino ljudi, znotraj katere vsi poznajo vse, moramo najti kliko danega omrežja.

Kako se lotimo problema?

Dan je seznam vseh vozlišč danega grafa ter pripadajoča tabela sosednosti. Vemo, da bo potrebno za vsako vozlišče preveriti, ali je vsebovano v maksimalni kliki ali ne. Zato zaporedje izbire vozlišč tekom postopka ni relevantno. Algoritem na koncu vrne seznam vozlišč, ki tvorijo maksimalno kliko danega grafa. Pri iskanju slednje si pomagamo z Bron-Kerboschovim algoritmom.

Bron-Kerboschov algoritem

"Eden izmed algoritmov za iskanje vseh maksimalnih klik danega grafa je Bron-Kerboschov algoritem. Algoritem deluje na podlagi rekurzivnega sestopanja in tekom celotnega postopka obdeluje tri disjunktne nabore vozlišč `R`, `P` in `X`. Pri tem klika vsebuje *vsa vozlišča* iz `R`, *nekatera* iz `P` ter *nobenega* iz `X`. Ob inicializaciji algoritma sta zato množici `R` in `X` prazni, množica `P` pa vsebuje vsa vozlišča danega grafa. Vemo, da unija `P` in `X` na vsakem koraku algoritma vsebuje izključno vozlišča, ki so povezana z nekaterimi vozlišči iz `R`. Na vsakem koraku algoritma torej preverimo vsa vozlišča iz množice `P`:

1. če sta hkrati prazni množici `P` in `X`, vemo, da h kliki `R` ne moremo dodati novega vozlišča. Zato vemo, da trenutna množica `R` vsebuje maksimalno kliko danega grafa. Torej algoritem na tem mestu vrne množico `R`;

2. sicer posamezno vozlišče `v` iz `P` rekurzivno doda v `R`. Množici `P` in `X` prilagodi tako, da v njiju ostanejo le tista vozlišča iz posamezne množice, ki so sosednja vozlišču `v`. S tem pridobimo vse klike, ki vsebujejo vozlišča iz `R` (skupaj z vozliščem `v`). Ob koncu tega rekurzivnega klica se vozlišče `v` prestavi iz množice `P` v množico `X`. S tem lahko v nadaljevanju pridobimo še klike, ki ne vsebujejo vozlišča `v` [1]."

Ta algoritem poišče le netrivialne klike, torej klike, ki vsebujejo najmanj dve vozlišči.

Primer implementacije v Pythonu:

```
def Bron_Kerbosch(R, P, X, sl_vozlisc, samo_najvecje=False):
    '''
        Algoritem vrne seznam vseh maksimalnih klik danega omrežja.

        Vhodni podtki:
            R ... množica, katere vsi elementi so vsebovani v kliki;
            P ... množica, katere nekateri elementi so vsebovani v kliki;
            X ... množica, katere noben element ni vsebovan v kliki;
            sl_vozlisc ... slovar vozlišč danega omrežja, katerega elementi so
oblike
                `id_vozlišča: vozlišče`, pri čemer je `vozlišče` oblike
`Vozlisce(id, sosedi, ime, vsebina, st_sosedov)`
    '''
    st_vozlisc_v_najvecji_kliki = 0
    seznam_klik = list()
    if not P and not X:  # `P` in `X` sta prazni
        # k `R` ne moremo dodati novega vozlišča
        # torej `R` zagotovo vsebuje maksimalno kliko danega grafa
        if R and (R not in seznam_klik):
            st_vozlisc_v_najvecji_kliki, seznam_klik =
```

```
posodobi_seznam_maksimalnih_klik(R, seznam_klik, st_vozlisc_v_najvecji_kliki,
samo_najvecje)

    else:
        for v in P:
            sosedi_v = set(sl_vozlisc[v].sosedi)

            R_copy = R.copy()
            R_copy.add(v)
            P_copy = P.copy()
            P_copy = sosedi_v.intersection(P)
            X_copy = X.copy()
            X_copy = sosedi_v.intersection(X)

            # rekrzivno dodamo klike v seznam klik
            for klika in Bron_Kerbosch(R_copy, P_copy, X_copy, sl_vozlisc,
samo_najvecje):
                if klika and (klika not in seznam_klik):
                    st_vozlisc_v_najvecji_kliki, seznam_klik =
posodobi_seznam_maksimalnih_klik(klika, seznam_klik, st_vozlisc_v_najvecji_kliki,
samo_najvecje)

            P = P.difference({v})
            X.add(v)

    return seznam_klik


def posodobi_seznam_maksimalnih_klik(klika, sez_klik, st_vozlisc_v_najvecji_kliki,
samo_najvecje=False):
    '''
        Vrne število vozlišč v največji kliki izmed do sedaj odkritih klik in
seznam klik.

        Vhodni podatki:
            klika ... klika, ki jo želimo dodati v seznam klik
            sez_klik ... do sedaj ustvarjeni seznam "maksimalnih" klik
            st_vozlisc_v_najvecji_kliki ... število vozlišč v trenutno največji
kliki
    '''
    if samo_najvecje:
        st_eltov_v_kliki = len(klika)
        if st_eltov_v_kliki == st_vozlisc_v_najvecji_kliki:  # še ena klika, ki je
potencialno največja
            sez_klik.append(klika)
        elif st_eltov_v_kliki > st_vozlisc_v_najvecji_kliki:  # trenutna klika je
do sedaj največja odkrita
            sez_klik = [klika]
            st_vozlisc_v_najvecji_kliki = st_eltov_v_kliki
    else:  # vse maksimalne klike - ne samo največje glede na število vsebovanih
vozlišč
        sez_klik.append(klika)
    return st_vozlisc_v_najvecji_kliki, sez_klik
```

Časovna in prostorska zahtevnost Bron-Kerboschovega algoritma

Recimo, da je `e` število povezav, `v` število vozlišč in `s` količina prostora, potrebnega za shranjevanje enega vozlišča danega omrežja.

Časovna zahtevnost

Najboljša možna:

Če na primer dano omrežje ne vsebuje vozlišč, je časovna zahtevnost enaka `O(1)`.

Najslabša možna:

Vemo, da algoritem vedno pregleda vsa vozlišča danega omrežja, česar časovna zahtevnost je `O(v)`. Za vsako vozlišče nato preveri, ali je vsebovano v maksimalni kliki ali ne. Ker vemo, da za vsako vozlišče omrežja algoritem opravi rekurzivni klic `Bron_Kerbosch(R.add(v), sosedi_v.intersection(P), sosedi_v.intersection(X), sl_vozlisc)`, velja na tem mestu premisliti, kaj se pravzaprav zgodi. Torej:

1. inicializacija algoritma: `Bron_Kerbosch(set(), množica_vseh_vozlišč_danega_omrežja, set(), sl_vozlisc)`,
2. ob obravnavi prvega vozlišča omrežja se kliče: `Bron_Kerbosch(set(izbrano_vozlišče), množica_vseh_vozlišč_danega_omrežja.difference(izbrano_vozlišče), set(), sl_vozlisc)`.

Opazimo, da se zgodi `v` rekurzivnih klicov. Če vemo, da je na `i`-tem koraku rekurzije `|P| = v - i - 1`, vemo tudi, da se na vsakem koraku rekurzije opravi `O(v - i - 1)` operacij. Torej s pomočjo metode ostrega pogleda ugotovimo, da je najslabša časovna zahtevnost tega algoritma enaka `O(v) * O(v - 1) * O(v - 2) * ... O(1) * O(0) = O(v!)`

Prostorska zahtevnost

Najboljša možna:

Če na primer dano omrežje ne vsebuje vozlišč, je prostorska zahtevnost enaka `O(1)`.

Najslabša možna:

Če je graf omrežja poln, sta potrebni po dve tabeli (tj. za original in kopijo) za vsako od množic `P`, `R` in `X`, v katerih uniji so shranjena natanko vsa vozlišča. Torej potrebujemo najmanj `O(2 * v)` prostora. Poleg tega potrebujemo tudi prostor za tabelo sosednosti, kar znaša `O(v + 2 * e)` prostora, saj je vsaka povezava zabeležena na dveh koncih (tj. povezava `ab` je predstavljena tako: `{a: [b], b: [a]}`). Nato potrebujemo še prostor za slovar vozlišč, katerega potrebni prostor je močno odvisen od količine podatkov shranjenih v vozliščih. Torej lahko rečemo, da je potrebnega `O(e + v * s)` prostora. Upoštevati moramo še prostor, potreben za shranjevanje rezultata. Ker algoritem deluje na podlagi rekurzije, pri tem pa generira največ `3 ^ (v / 3)` največjih klik [1], torej je prostorska zahtevnost obravnavanega algoritma `O(e + v * s) + O(2 ^ (v / 3)) = O(2 ^ (v / 3))`.

Viri:

[1] *Bron–Kerbosch algorithm*, November 12, 2021. [Online], Avalible: https://en.wikipedia.org/wiki/Bron%E2%80%93Kerbosch_algorithm [Accessed Februar 12, 2022]