NTIN060 | Homework 05/5

Milan Wikarski (milan@wikarski.sk)

Zadanie

5.Pravděpodobnost.Každé spojení má pravděpodobnost úspěchu přenosu `0 ≤ p<sub>i</sub> ≤ 1`. Pravděpodobnost úspěchu pro cestu `v<sub>1</sub>, ..., v<sub>k</sub>` je `p<sub>(v1, v2)</sub> \* ... \* p<sub>(vk-1, vk)</sub>`. Chcete najít cestu, po které je nejvíce pravděpodobné, že uspějete. Jak ji najdete?

Riešenie a dôkaz správnosti

Majme graf `G = (V, E)`, zobrazenie `E -> p`, ktoré každej hrane priradí nejakú pravdepodobnosť `0 ≤ p<sub>i</sub> ≤ 1` a nejaké vrcholy `v<sub>0</sub>,u<sub>0</sub> ∈ V` Chcem zistiť cestu s najväčšou pravdepodobnosťou úspechu z `v<sub>0</sub>` do `u<sub>0</sub>`.

Vytvoríme si pole `S[v]`, kde budeme uchovávať pravdepodobnosť úspechu spojenia z vrcholu `v<sub>0</sub>` do vrcholu `v`. Začneme vo vrchole `v<sub>0</sub>`, pre ktorý bude platiť `S[v<sub>0</sub>] = 1`. Pre všetky ostatné vrcholy nastavíme túto hodnotu na 0.

Vytvoríme si frontu, do ktorej budeme ukladať otvorené vrcholy. Na začiatku cyklu z fronty vyberieme jeden vrchol, preskúmame jeho susedov, prepočítame pravdepodobnosti prenosu a otvoríme 0 alebo viac vrcholov, ktoré pridáme na koniec fronty. Toto sa bude opakovať, dokým sa fronta úplne nevyprázdni. Po skončení výpočtu budú v poli `S` správne hodnoty pravdepodobnosti úspechu prenosu pre každý vrchol.

Výpočet pravdepodobnosti spojenia

Rozoberme hlavný cyklus. Na jeho začiatku z fronty vyberieme vrchol `v`. Pozrieme sa na všetky jeho susedné vrcholy. Uvažujme susedný vrchol `u`. Chceme zistiť, či pravdepodobnosť úspechu prenosu po ceste `(v<sub>0</sub>, ..., v, u)` (označíme ju `p'`) je väčšia, ako hodnota `S[u]`. Ak platí `p' > S[u]`, potom sme objavili cestu z `v<sub>0</sub>` do `u`, ktorá má väčšiu pravdepodobnosť úspechu, než sme si doteraz mysleli. V takom prípade nastavíme `S[u] = p'` a vrchol `u` otvoríme (dáme ho na koniec fronty).

Predchodcovia vrcholov

Vytvoríme si pole `P[v]`, kde budeme uchovávať predchodcov vrcholu `v`. Takto budeme vedieť spätne zistiť cestu z `v<sub>0</sub>` do `u<sub>0</sub>`. Vždy, keď budeme nastavovať `S[u] = p'`, nastavíme aj `P[u] = v`.

Fázy výpočtu

Výpočet bude prebiehať vo fázach. Vo fáze `F<sub>0</sub>` otvoríme vrchol `v<sub>0</sub>`. Vo fáze `F<sub>i+1</sub>` zatvárame vrcholy otvorené vo fáze `F<sub>i</sub>`. Vo fáze `F<sub>i</sub>` budeme už definitívne poznať hodnoty `S[u]` a `P[u]` pre všetky vrcholy `u ∈ V`, pre ktoré platí `d(v<sub>0</sub>, u) ≤ i`, kde `d(v<sub>0</sub>, u)` je počet hrán medzi vrcholmi `v<sub>0</sub>` a `u` (analogicky z Bellman-Fordova algoritmu; dá sa dokázať indukciou). Tým pádom po najviac `O(n)` fázach budeme poznať definitívne hodnoty `S[v]` a `P[v]` pre všetky `v ∈ V`.

Algoritmus

```
ALGORITMUS: PravdepodobnostUspechuPrenosu
---------------------------------------------------------------
 INPUT: graf G = (V, E), zobrazenie E -> p a vrcholy v0, u0
OUTPUT: Cesta z v0 do u0 s najväčšou pravdepodobnosťou úspechu
```

```
 1. queue <- new Queue()                ◁ vytvoríme si prázdnu frontu
 2. for ∀ v ∈ V:
 3.   S[v] <- 0                         ◁ Pravdepodobnosť úspechu cesty (v0, ..., v)
 4.   P[v] <- null                      ◁ Predchodca vrcholu v
 5. S[v0] <- 1
 6. queue.enqueue(v0)                   ◁ Začneme vo v0
 7. while (queue is not empty):
 8.   v <- queue.dequeue()              ◁ Vyberieme v z fronty
 9.   for (∀ u; u is neighbour of v):   ◁ Každý vrchol u spojený hranou s vrcholom v
10.      if (S[u] < S[v] * p[(v, u)]):  ◁ Ak sme našli lepšiu cestu
11.        S[u] <- S[v] * p[(v, u)]
12.        P[u] <- v
13.        queue.enqueue(u)
14. path <- new List()
15. while (u0 != null):
16.   path.append(u0)
17.   u0 <- P[u0]
18. return path.reverse()
Poznámka: p[(v, u)] je pravdepodobnosť spojenia po hrane (v, u)
```

Časová a priestorová zložitosť

Označme si `n = |V|` počet vrcholov a `m = |E|` počet hrán.

Časová zložitosť

Rozdelme si algoritmus na 3 časti:

1. Inicializácia

Inicializácia prebehne v čase `O(n)`, pretože inicializujeme hodnoty `P[v]` a `S[v]` pre všetky `v ∈ V`.

2. Výpočet

Vieme, že výpočet skončí po najviac `O(n)` fázach. Behom jedej fázy algoritmnus relaxuje každý vrchol najviac raz, takže celá fáza trvá `O(m)`. Dokopy to je `O(nm)`.

3. Budovanie cesty

Výsledná cesta môže byť poskladaná z najviac `n` vrcholov, takže jej poskladanie bude trvať najviac `O(n)`.

Všetky tri časti spolu budú trvať `O(n + nm + n) = O(nm)`.

Priestorová zložitosť

Algoritmus pracuje s pomocnými poľami `P` a `D`, ktoré majú dĺžku `n`. Okrem toho potrebuje pamäť pre frontu, kde v jeden moment nemôže byť viac, ako `n` prvkov. Výslednú cestu vytvorí pomocou spojového zoznamu, ktorý bude mať najviac `n` prvkov. Spolu teda vyžaduje `O(4n) = O(n)` pamäte.

Alternatívny pohľad na problém

Alternatívne by sme sa na problém mohli dívať takto. Našou úlohou je násť cestu poskladanú z hrán `e<sub>1</sub>, e<sub>2</sub>, ..., e<sub>k</sub>`, kde platí, že `e<sub>1</sub> = (v<sub>0</sub>, w)`, `e<sub>k</sub> = (w', u<sub>0</sub>)`. Označme si `p<sub>i</sub>` pravdepodobnosť úspechu prenosu po hrane `e<sub>i</sub>`. Potom našou úlohou je nájsť:

```
argmax {e1, e2, ..., ek} (p1 * p2 * ... * pk)
```

Čo je ekvivalentné s

```
argmax {e1, e2, ..., ek} (log(p1) + log(p2) + ... + log(pk))
```

Hľadáme minimum súčtu logaritmov. Vynásobme celý súčet hodnoutou (-1) a budeme hľadať

```
argmin {e1, e2, ..., ek} (-log(p1) - log(p2) - ... - log(pk))
```

Vieme, že platí `0 ≤ p<sub>i</sub> ≤ 1` pre všetky `i`. To znamená, že určite platí aj `0 ≤ -log(p<sub>i</sub>)` pre všetky `i`. Spravme si substitúciu `W[i] = -log(p<sub>i</sub>)`. Dostávame alternatívne ohodnotenie hrán `W`, kde každá váha je nezáporná a my chcem nájsť najkratšiu cestu vzhľadom na toto ohodnotenie. Na to nám ale postačí ľubovoľný relaxačný algoritmus (ja som použil modifikovaný Bellmanov-Fordov).