

# PageRank algoritam s visećim vrhovima (*dangling nodes*)

Marina Matešić<sup>1</sup>, Tomislav Novak<sup>1</sup>, and Timotej Repak<sup>1</sup>

<sup>1</sup>Prirodoslovno-matematički fakultet

U ovom radu obrađujemo PageRank algoritam za rangiranje web stranica po važnosti. Posebna pozornost dana je visećim vrhovima. Nakon teorijske analize priložen je i kod programa.

## 1 Uvod

PageRank algoritam i metode koje koristi razvili su Sergey Brin i Larry Page, tadašnji studenti na Sveučilištu Stanford 1998. godine. Tad datiraju njihovi prvi radovi na tu temu, i izlaganje na 7. međunarodnoj World Wide Web konferenciji. Iste godine osnuju i tvrtku Google. Na PageRanku i dan danas počivaju rezultati Google pretrage.

## 2 Definicije

Modeliramo strukturu mreže od  $n$  web stranica kao usmjereni graf gdje vrhovi predstavljaju web stranice, a brid iz vrha  $u$  u vrh  $v$  predstavlja poveznicu (link) na stranici  $u$  koja pokazuje na stranicu  $v$ . Graf opisujemo matricom  $G$ , koja ima 1 na mjestu  $(i, j)$  ako vrh  $i$  ima link na vrh  $j$ . No, željeli bismo vrhove rangirati po tome koliko su važni vrhovi koji na njih pokazuju, uz uvjet da na što više drugih vrhova određeni vrh, važnost istog se proporcionalno smanjuje. Ako s  $x_i$  označimo važnost vrha  $i$ , te s  $n_i$  broj vrhova na koje pokazuje vrh  $i$  te  $L_j$  skup vrhova koji pokazuju na vrh  $j$ , zapravo želimo da vrijedi

$$x_j = \sum_{i \in L_j} \frac{1}{n_i} x_i$$

Zatim definiramo matricu  $A$  koja to uzima u obzir: za  $[A_{ij}]$  stavimo  $1/n_i$  ako vrh  $i$  pokazuje na vrh  $j$ , a 0 inače. Tako vidimo da je problem zapravo naći vektor  $x$  t.d.  $Ax = x$ , tj. naći svojstveni vektor matrice  $A$  kojem pripada svojstvena vrijednost 1.

## 3 Markovljevi lanci i PageRank

Neka je skup svih stanja u kojima se neki proces može nalaziti konačan s  $n$  elemenata te svako stanje označimo brojem od 1 do  $n$ . Ako buduće stanje procesa ovisi samo o sadašnjem stanju, a ne i o prošlima (npr. koju ćemo sljedeću internetsku stranicu

posjetiti ovisi samo o stranici koju trenutno gledamo, ali ne i o stranicama koje smo nekoć gledali), tada se takav proces naziva Markovljev lanac. Ako se u sadašnjosti nalazimo u stanju  $i$ , označimo s  $a_{ij}$  vjerojatnost da ćemo se u sljedećem trenutku nalaziti u stanju  $j$ . Označimo s  $A$  matricu  $[a_{ij}]$ . Iz stanja  $i$  ćemo u sljedećem trenutku gotovo sigurno prijeći u neko stanje iz  $\{1, \dots, n\}$  pa je zbroj svakog retka u matrici  $A$  jednak 1, a svaki  $[a_{ij}]$  je nenegativan (jer predstavlja vjerojatnost prijelaza iz stanja  $i$  u stanje  $j$ ). Takve matrice zovu se stohastičkima.

Svaki proces ima neku svoju početnu distribuciju, a u ovom slučaju ona se reprezentira vektorom  $\lambda = (\lambda_1, \dots, \lambda_n)$  gdje  $\lambda_i$  predstavlja vjerojatnost da se nalazimo u stanju  $i$ . Stoga je to vektor čiji su svi elementi nenegativni i u zbroju daju 1 (jer ćemo se gotovo sigurno nalaziti u nekom stanju). Pretpostavimo da su na početku sva stanja jednaka, tj.  $\lambda_1 = \dots = \lambda_n = 1/n$ , tj. s jednakom vjerojatnošću gledamo svaku stranicu na našem internetu. U sljedećem trenutku vjerojatnost da gledamo stranicu  $j$  je  $\lambda_1 a_{1j} + \lambda_2 a_{2j} + \dots + \lambda_n a_{nj}$ , odnosno nova distribucija je upravo  $\lambda A$ . Distribuciju za koju vrijedi  $\lambda = \lambda A$  nazivamo stacionarna distribucija jer u svakom koraku vjerojatnost da se nalazimo u nekom stanju ostaje ista. Želimo naći distribuciju u beskonačnosti, tj. takav  $\lambda$  za kojeg je  $\lambda_i$  limes po  $n$  vjerojatnosti da se u  $n$ -tom trenutku nalazimo u stanju  $i$ . Lako se pokaže da ako takva distribucija postoji, ona nužno mora biti stacionarna.

Označimo s  $L(i)$  skup svih stranica na koje postoji link iz stranice  $i$ . Promatramo "slučajnog surfera" koji iz stranice  $i$  s nekom vjerojatnošću  $\alpha \in (0,1)$  prijeđe u slučajno odabranu stranicu  $j \in L(i)$ , a s vjerojatnošću  $1 - \alpha$  u slučajno odabranu stranicu (bilo koju). Neka je  $\pi_i$  asimptotski postotak vremena koje će surfer provesti na stanici  $i$ . Originalni PageRank određuje važnost stranice  $i$  kao  $\pi_i$ .

## 4 Viseći vrhovi

Visećim vrhovima (eng. *dangling nodes*) zovemo one vrhove  $i$  koji nemaju nijedan izlazni brid, tj. vrijedi  $n_i = 0$ . Tada je  $i$ -ti redak u  $A$  nul redak. Budući da daljnji algoritam počiva na tome da imamo stohastičku matricu, glavno je pitanje što napraviti s tim nul retcima odn. visećim vrhovima. Jedno moguće rješenje je dodati izmišljene veze s visećih vrhova, tj. dopuniti nul retke na način da definiramo matricu  $M_\alpha = (1 - \alpha)A + \alpha S$  za neki  $\alpha \in [0, 1]$ , gdje je  $[S_{ij}] = 1/n$ . Tada je  $M$  uistinu stohastička i to je algoritam obrađen u prez. s predavanja s priloženim kodom - svojstvenom vektoru konvergiramo primjenom matrice na neki stohastički vektor.

## 5 Sažimanje (*lumping*)

Kako je u praksi udio visećih vrhova izuzetno velik (po Ipsen and Selee (2007) on može doseći čak i 80% ukupnog broja vrhova), prirodno je zapitati se što s njima možemo efikasno učiniti. Grupirat ćemo matricu na način da imamo puno manju podmatricu samo s nevisećim vrhovima. Preuredimo prvo polaznu matricu tako da sortiramo vrhove. Neka je, od  $n$  vrhova,  $k$  broj nevisećih ( $1 \leq k < n$ ). Tih  $k$  vrhova fiksirajmo kao prvih  $k$ . Sada polazna  $n \times n$  matrica izgleda ovako:

$$H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & 0 \end{bmatrix}, \quad (1)$$

gdje je  $H_{11}$   $k \times k$  matrica linkova između nevisećih vrhova a  $H_{12}$  sadrži linkove nevisećih na viseće vrhove. Time smo izdvojili viseće vrhove u donjih  $n - k$  redova, a gornjih  $k$  redova su i dalje stohastički.

Ideja se sada svodi na sljedeće: o rezultatu (poretku) nevisećih  $k$  vrhova informaciju imamo iz matrice  $H_{11}$ , a na poredak  $n - k$  visećih vrhova zapravo samo utječu oni vrhovi koji pokazuju na njih, a ta informacija je u  $H_{12}$ .

Postupak će početi naizgled isto, tj. prvo ćemo odabrati stohastički vektor  $w$  (2) s kojim ćemo zamijeniti nul retke, a zatim napraviti i konveksnu kombinaciju sa stohastičkim vektorom prilagodbe  $v$  (3) kako bismo osigurali jedinstvenost stacionarne distribucije.

$$S := \begin{bmatrix} H_{11} & H_{12} \\ ew_1^T & ew_2^T \end{bmatrix}, \text{ gdje je } w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \text{ dimenzije } n \times 1 \quad (2)$$

$$G := \alpha S + (1 - \alpha)ev^T, \quad 0 \leq \alpha < 1 \quad (3)$$

Konačno, matrica  $G$  izgleda ovako:

$$G := \begin{bmatrix} G_{11} & G_{12} \\ eu_1^T & eu_2^T \end{bmatrix}, \quad (4)$$

gdje je  $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$  konveksna kombinacija  $v$  i  $w$ .

Za takvu matricu  $G$  rezultati iz Ipsen and Selee (2007) osiguravaju da ćemo moći sažeti višeće vrhove u jedan vrh i pritom očuvati točnost rješenja.

Međutim, u nekim situacijama ovaj pristup može biti nedostatan, primjerice kada bismo htjeli personalizirati pretragu prema različitim temama, jezicima ili domenama, ili kada bismo htjeli uzeti u obzir različite vrste stranica (npr. tekstualne datoteke, slike, videozapise). U takvim situacijama svakoj klasi dodjeljujemo jedinstveni vektor  $w_i$ , gdje je  $1 \leq i \leq m$ . To rezultira generaliziranim Google matricom  $F$ , gdje se  $n - k$  nul-redaka u matrici hiperlinkova  $H$  zamjenjuje s  $w_1, \dots, w_m$ .

Opća Google matrica ima oblik:

$$F = \begin{bmatrix} F_{11} & F_{12} & \cdots & F_{1,m+1} \\ u_1 & u_2 & \cdots & u_{m+1} \end{bmatrix},$$

gdje vrijedi:

$$u_i = \alpha w_i + (1 - \alpha)v, \quad 1 \leq i \leq m + 1.$$

Proces sažimanja uključuje transformacije sličnosti koje iterativno smanjuju veličinu matrice uz očuvanje njezinih stohastičkih svojstava i svojstvenih vrijednosti. Za dvije klase visećih vrhova, transformacija  $X_1$  sažima redove i stupce koji odgovaraju  $w_2$ , dok ostavlja nepromijenjen vodeći blok veličine  $k + k_1$ :

$$X_1 = \begin{bmatrix} I & 0 \\ 0 & L_1 \end{bmatrix}, \quad L_1 = I - \frac{1}{k_2}\hat{e}\hat{e}^T,$$

gdje je  $\hat{e} = e - e_1$ , a  $e_1$  jedinični vektor.

Primjenom  $X_1$  modificira se matrica:

$$F' = X_1 F X_1^{-1},$$

a postupak se ponavlja za sljedeću klasu visećih vrhova s transformacijom  $X_2$ .

Konačna sažeta matrica ima oblik:

$$F^{(1)} = \begin{bmatrix} F_{11} & F_{12}e & F_{13}e & \cdots \\ u_1e & u_2e & u_3e & \cdots \end{bmatrix}.$$

Stacionarna distribucija  $\pi$  sažete matrice zadovoljava:

$$\pi^T F = \pi^T, \quad \|\pi\|_1 = 1.$$

Korištenjem ove metode, generalizirana Google matrica zadržava svoja stohastička svojstva, čuva svojstvene vrijednosti i osigurava precizno izračunavanje PageRanka, omogućujući pritom učinkovito rukovanje s više klasa visećih vrhova.

## 6 Analiza složenosti

Složenost algoritma PageRank temeljenog na power metodi ovisi o broju iteracija potrebnih za konvergenciju i o broju operacija po iteraciji. U slučaju rijetko popunjenih matrica prijelaza  $S$ , broj operacija po iteraciji proporcionalan je broju nepraznih elemenata u matrici  $S$ , označenom kao  $\text{NNZ}(S)$ . Stoga je složenost jedne iteracije dana izrazom:

$$O(\text{NNZ}(S)).$$

Broj iteracija za konvergenciju ovisi o damping faktoru  $\alpha$  i zadanoj toleranciji  $\tau$ . Za damping faktor  $\alpha = 0.85$  i toleranciju  $\tau = 10^{-6}$ , algoritam zahtijeva približno  $k = O(\log(\tau) / \log(\alpha))$  iteracija. Dakle, ukupna složenost algoritma s visećim vrhovima je:

$$O(k \cdot \text{NNZ}(S)),$$

gdje je  $k$  broj iteracija, a  $\text{NNZ}(S)$  broj nepraznih elemenata u matrici  $S$ .

Implementacija sa sažimanjem dodatno smanjuje dimenzionalnost problema, jer se viseći vrhovi grupiraju u jedan čvor, čime se broj redaka i stupaca matrice prijelaza smanjuje s  $n \times n$  na  $(k+1) \times (k+1)$ , gdje je  $k$  broj nevisećih vrhova. To teoretski ne smanjuje puno složenost budući da u praksi  $k$  ne može biti puno manjeg reda veličine od  $n$ , no ipak, kako je spomenuto u 5, udio visećih čvorova može biti i 80%, što, kad primijenimo na kvadratnu dimenziju tablice, već možemo naslutiti neko ubrzanje u praksi.

## 7 Tehnički detalji algoritma i empirijski rezultati

Algoritam implementiramo u programskom jeziku *Python* koristeći module *numpy* i *scipy* za potrebne metode za efikasno baratanje velikim matricama. Izvorni kod priložen je radu u datoteci *PageRankWDanglingNodes.py*. U mapi podaci dano je više skupova podataka za testiranje preuzetih iz Sahu, a kod je testiran na primjeru *enron*. Neviseći vrhovi su poredani na početak, a viseći sažeti u jedan vrh.

Kod je komentiran i ispis daje potrebne ključne točke, pa navodimo samo nekoliko tehničkih detalja:

- Zbog prethodno implementiranog osnovnog algoritma za PageRank (datoteka *PageRank.py*) po nomenklaturi i oznakama s predavanja, u istom duhu (koristeći dijelove koda) implementiramo i ovo proširenje, a oznake se podudaraju do na transpoziciju - u kodu je matrica transponirana u odnosu na oznake u odlomcima 2 i 5 (pa tako imamo nul stupce u matrici  $H$ , transponiran cijeli izraz pridruživanja u petlji algoritma itd.).
- Sortiranje vrhova u matrici  $H$  radimo na način da pamtimo permutaciju vrhova iz originalnog poretka (zato nam koristi i funkcija za inverz permutacije).

- Za matricu  $A$  (s originalnim podacima) i zatim  $H$ , budući da su rijetko popunjene matrice, koristimo *sparse* paket iz *scipy* modula. Stvaramo ih kao objekt klase `coo_matrix` (*COOrdinate format*) budući da im dajemo poznate elemente s koordinatama, a zatim ih pretvaramo u `csc_matrix` format (*Compressed Sparse Column format*) za učinkovitije izvođenje operacija.
- Implementiran je algoritam sa sažimanjem, a nakon njega i algoritam bez sažimanja koji djeluje na cijelu matricu  $H$ . Za oba (kritična dijela) algoritma izmjereno je i ispisano vrijeme izvođenja koristeći *Pythonov time* modul.

U testnom skupu *enron* udio visećih vrhova je otprilike 75%. Zadan je  $\tau$  kao varijabla koja određuje što smatramo *konvergencijom* u željeni vektor (kad se norma razlike između dvije iteracije smanji ispod  $\tau$ ), i vidimo da za iste podatke metoda sa sažimanjem treba i manje iteracija od obične (uz to što je efikasnija po broju operacija).

Empirijski vidimo da se kod sa sažimanjem izvodi u prosjeku 3 do 4 puta brže od koda bez sažimanja za **isti broj iteracija**, što je potvrda slutnje analize složenosti. Ipak, u praksi nemamo potrebu za istim brojem iteracija (konvergenција ne ovisi isključivo o broju iteracija) pa, kad promatramo izvođenje do zadanog  $\tau$ , pr. kao u testnom primjeru  $10^{-9}$ , imamo oko 30% manje iteracija te isto toliko brži algoritam sa sažimanjem.

Također, primijetimo da se rankovi (gledajući nekoliko najviših rezultata) blago razlikuju. Na to utječe način na koji odaberemo vektore  $v$  i  $w$  - koliko će u algoritmu sa sažimanjem *pagerank* nevisećih vrhova utjecati na onaj od visećih vrhova.

## References

Ilse Ipsen and Teresa Selee. PageRank Computation, with Special Attention to Dangling Nodes. *SIAM Journal*, 2007. URL [https://www.researchgate.net/publication/220656288\\_PageRank\\_Computation\\_with\\_Special\\_Attention\\_to\\_Dangling\\_Nodes](https://www.researchgate.net/publication/220656288_PageRank_Computation_with_Special_Attention_to_Dangling_Nodes).

Subhajit Sahu. LAW Graphs Part 1 (A-U). URL <https://www.kaggle.com/datasets/wolfram77/graphs-law-01/>.