Rapport

Jan M. Rise og Jakob S. Vesterstrøm

17. januar 2003

Introduktion

Denne rapport beskriver vores arbejde og overvejelser i kurset “Algorithms for Web Indexing and Searching” i forbindelse med implementeringen af “PageRank”-algoritmen, der er en metode til at rangere en samling af indbyrdes forbundne webdokumenter.

Beskrivelse af PageRank

Idden bag PageRank er meget enkel: Jo flere sider der linker til en given side, desto mere vigtig er den givne side. Man kan altså angive en forsimplet PageRank for side i vha. udtrykket

$$\mathrm{Pr}(i) = |\mathrm{inlinks}(i)|$$

hvor $\mathrm{inlinks}(i)$ er mængden af sider, der linker til side i. PageRank er således defineret som antallet af sider, der peget på i, og dette mål er bla. kendt fra den videnskabelige literatur, hvor antal “citations” er et ofte benyttet mål for en artikels vigtighed.

Dette mål har imidlertid flere svagheder. Først og fremmest bliver der ikke taget højde for hvor mange links, der udgår fra side i; altså i’s udgrad, som vi angiver med $\mathrm{outdeg}(i)$. Man kan forsøge at løse dette problem ved at dividere bidraget fra en given side, kaldet i, til den side den linker til med $\mathrm{outdeg}(i)$. Det giver flg. udtryk for PageRank for side i:

$$\mathrm{Pr}(i) = \sum_{j \in \mathrm{inlinks}(i)} \frac{1}{\mathrm{outdeg}(j)}.$$

Der er dog stadig et væsentligt problem, som dette mål ikke gør noget ved: Det er muligt at lave en isoleret klynge af sider, som ikke er linket til fra resten af dokumentsamlingen (læs: internettet), således at denne klynge kan få kunstigt høje rankings. For at komme denne mulighed til livs kan man vedtage, at ikke alle siders links tæller med lige stor vægt. Den vægt de skal tælle med kan fx være givet ved deres PageRank. [1] Dermed bliver PageRank rekursivt defineret, og vi kan angive PageRank for side i, $\mathrm{Pr}(i)$:

$$\mathrm{Pr}(i) = \sum_{j \in \mathrm{inlinks}(i)} \frac{\mathrm{Pr}(j)}{\mathrm{outdeg}(j)}.$$

Man kan formulere overvejelserne omkring – og ikke mindst udregningen af – PageRank vha. lineær algebra. Set fra denne vinkel kan vi definere vektoren r af dimension N , hvor $r_i = \mathrm{Pr}(i)$. r er altså en PageRank vektor for samlingen af dokumenter. Tilsvarende kan vi definere matricen M af dimension $N \times N$. Indgang (i, j) i M er forskellig fra 0 hvis og kun hvis der er et link fra knude i til j. I den første simple version af PageRank vil $M_{i,j} = 1$, hvis der er et link fra i til j, i den næste er $M_{i,j} = \frac{1}{\mathrm{outdeg}(j)}$ og endelig i den sidst præsenterede version er $M_{i,j} = \sum_{j \in \mathrm{inlinks}(i)} \frac{\mathrm{Pr}(j)}{\mathrm{outdeg}(j)}$ i tilfælde af et link mellem i og j.

[1] Det var denne ide Sergey Brin og Lawrence Page fik for snart nogle år siden, og som lagde grundstenen til Google, den mest brugte søgemaskine på intetnettet den dag i dag.

Definitionen af PageRank vektoren r kan derfor formuleres som følger: PageRank for siderne $1 \ldots N$ er givet ved vektoren r, der opfylder

$$r = Mr = M^T r$$

hvor M er givet som ovenfor og PageRank for side i er r_i.

Hvis grafen er stærkt forbundet (dvs. at man kan komme fra en vilkårlig knude til at anden vilkårlig knude) kan vi beregne PageRank ved iterativt at udføre $r := Mr$, hvor r initielt er en tilfældig vektor. Denne metode kaldes "power iteration" og giver os egenvektoren for M.

Vi kan illustrere hvad der sker for en ikke stærkt forbundet graf ved anvendelse af power iteration med et lille eksempel: Lad os forestille os en graf med en stor mængde af stærkt forbundne knuder samt et enkelt link ud til en knude a, som linker til knude b, som igen linker til knude a. Så a og b linker kun til hinanden, og bliver kun linket til via det ene link, der linker til a.

Hvis PageRank skulle udregnes for en sådan graf ville der ske det, at "al ranken" ville forsvinde ned til knuderne a og b, idet rank der er hos enten a eller b aldrig kan komme tilbage til resten af grafen pga. manglende links. Således vil de to knuder ende med at dele al rank mellem sig. Et ligeså skræmmende scenarie ville opstå, hvis knude b (og de to links ind i og ud fra den) fjernes. I denne situation ville al rank før eller siden komme ned til a, hvorfra det ville forsvinde i et sort hul og alle knuders PageRanks ville med tiden gå mod 0.

Da internettet ikke er stærkt forbundet, kan vi ikke benytte power iteration til at beregne PageRank (i den ovenstående form) for internettet. En sidste modifikation af udtrykket for PageRank løser dog dette problem. Vi indfører en "decay factor" (eller "dampening factor") og ændrer udtrykket til

$$\mathrm{Pr}(i) = \frac{(1-d)}{N} + d \cdot \sum_{j \in \mathrm{inlinks}(i)} \frac{\mathrm{Pr}(j)}{\mathrm{outdeg}(j)}$$

hvor N er antallet af sider i samlingen. Således kan en knude aldrig få en PageRank under $\frac{(1-d)}{N}$ og dermed ikke gå mod nul.

En fortolkning af dette er at en knude giver $100 \cdot d$ procent af sin rank til de knuder den peger på, mens de resterende $100 \cdot (1-d)$ procent fordeles jævnt blandt samtlige sider. Sættes $d = 1$ fås i øvrigt den forrige version af PageRank.

Som en afsluttende bemærkning kan det nævnes at PageRank kan betragtes som en model af en surfende web-bruger: Denne bruger starter på en tilfældig side på nettet, og vælger nu tilfældigt at klikke på et link på siden. Dog bliver brugeren med sandsynlighed $1 - d$ træt af at bruge links til at komme frem med, og vælger derfor en helt ny og tilfældig side. Sandsynligheden for at brugeren til et givent tidspunkt er at finde på side i er (sjovt nok) lig med $\mathrm{Pr}(i)$.[2]

[2] Siderne er dog kun en sandsynlighedsfordeling (dvs. summer til 1) hvis de er stærkt forbundne – hvis ikke, bliver summen mindre end 1. Dette kan løses vha. et passende antal "pseudo-links", der gør grafen stærkt forbundet.

Udviklingsprocess

Vores udviklingsprocess kan meget passende beskrives som en 3-trins-raket:

- I første fase implementerede vi en simpel version af PageRank, hvor hele udregningen holdes i RAM. Denne fremgangsmåde er meget fordelagtig og hurtig hvis hele beregningen *kan* holdes i RAM. Ulempen ved metoden er selvfølgelig, at den bliver meget langsom (om overhovedet muligt), hvis data ikke kan holdes i RAM. I vores tilfælde kunne vi faktisk sagtens have klaret os udelukkende med denne simple metode, idet den kun bruger 140 Mb RAM. Dette går dog ikke for større datamængder – mere end 10 gange større ville blive besværligt på en alm. PC.
- En anden tilgang var at beregne delsummerne og skrive dem ud til fil en fil efter et forslag til forelæsningen. Denne metode virkede fint, omend langsommere end den første, men kan til gengæld klare store datamængder. Dog krævede denne metode at vi sorterede filen med delsummerne, og pga. dette minus valgte vi at implementere en tredie metode.
- Den tredie tilgang var en blok baseret udregning som beskrevet i artiklen "Efficient computation of PageRank". Denne tilgang beskrives i større detalje senere i rapporten.

Grafer

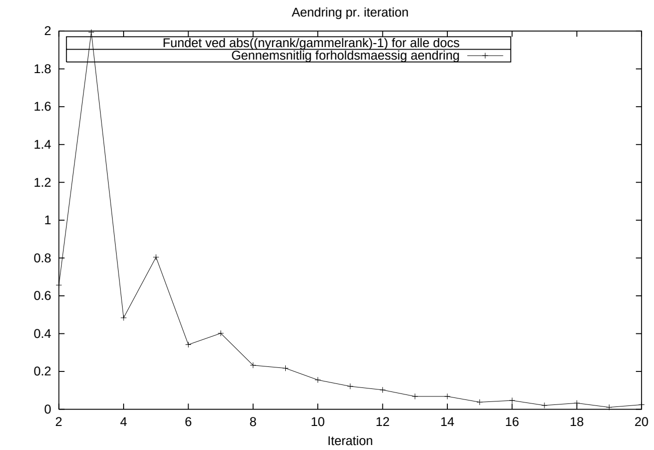
Nedenfor er vist 2 grafer, de er delt så den ene går fra iteration 2-20, mens den anden går fra iteration 20-140. Dette er gjort for bedre at kunne skelne detaljerne.

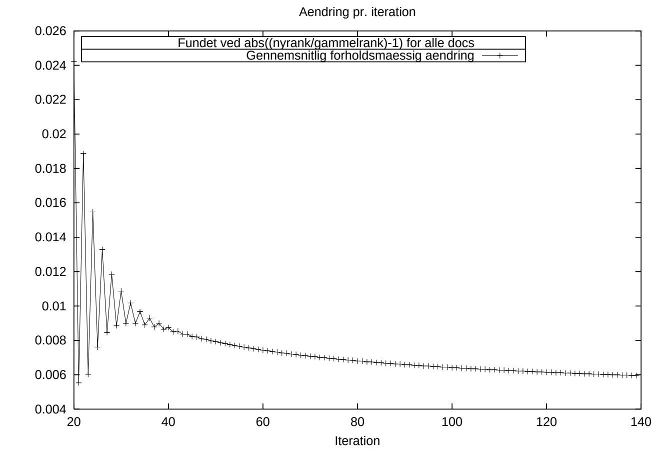
Beregningen er foretaget efter hver iteration og viser den gennemsnitlige forholdsvise ændring fra en iteration til den næste. Som det ses svinger den en del i starten. Ændringen fra iteration 0 til 1 er ikke medtaget idet den jo er initiel og ganske voldsom idet alle vektorerne ændres meget fra startværdien 1/(antal sider) til noget helt andet. Som det ses svinger ændringsraten en del indtil 40. iteration, hvor den begynder at flade ud for til sidst at nå ned på omkring 0,6% i snit.

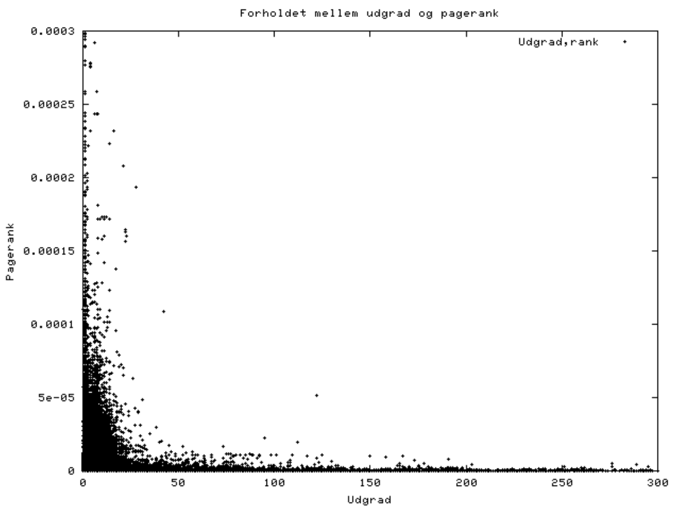
Vi havde en ide om at sider med høj udgrad typisk ikke ville have nogen høj pagerank. Uden at have noget belæg for det andet end en intuition, så er begrundelsen, at der typisk ikke linkes til sider med mange links, men til sider med indhold. Ser man på grafen ser det ud til at teorien holder stik. Den viser et udsnit, hvor udgrad og pagerank er plottet sammen. Den højeste udgrad i det endelige datasæt var 6809, men der er ganske få docs med udgrad over 300, og disse har ikke en høj rank. Ligeledes er der ganske få docs med en rank over 0.0003. Så for overskuelighedens skyld er disse grænser sat på grafen.

Som det også ses nedenfor har langt den overvejende del af siderne udgrad 0, mens sider med udgrad 1 og 2 følger lige efter.

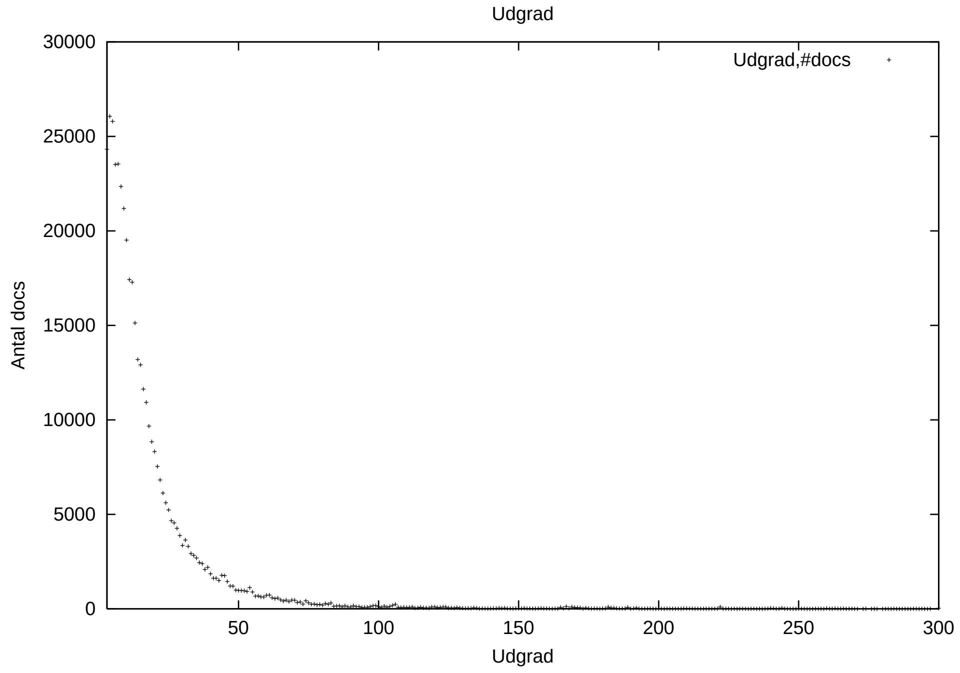
Det ses også på grafen at jo højere udgrad, jo færre docs er der med en udgrad. Vi troede først at dette kunne tilskrives crawleren. Hvis nu ikke crawlede links var medtaget i grafen,





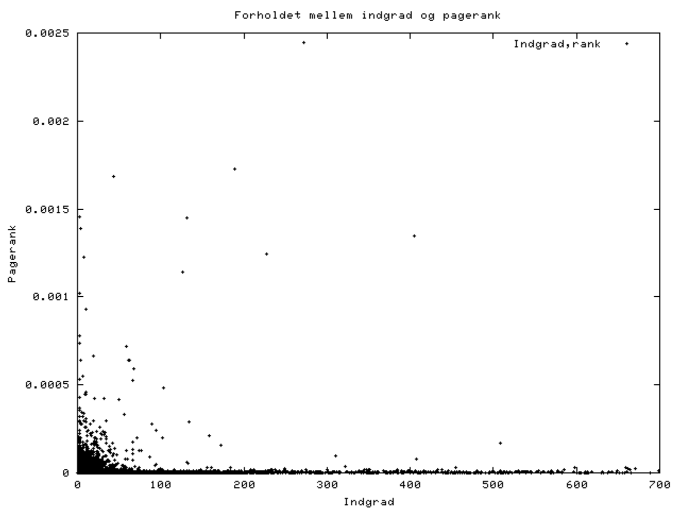


Udgrad	Antal
0	1.163.769
1	86.764
2	41.152



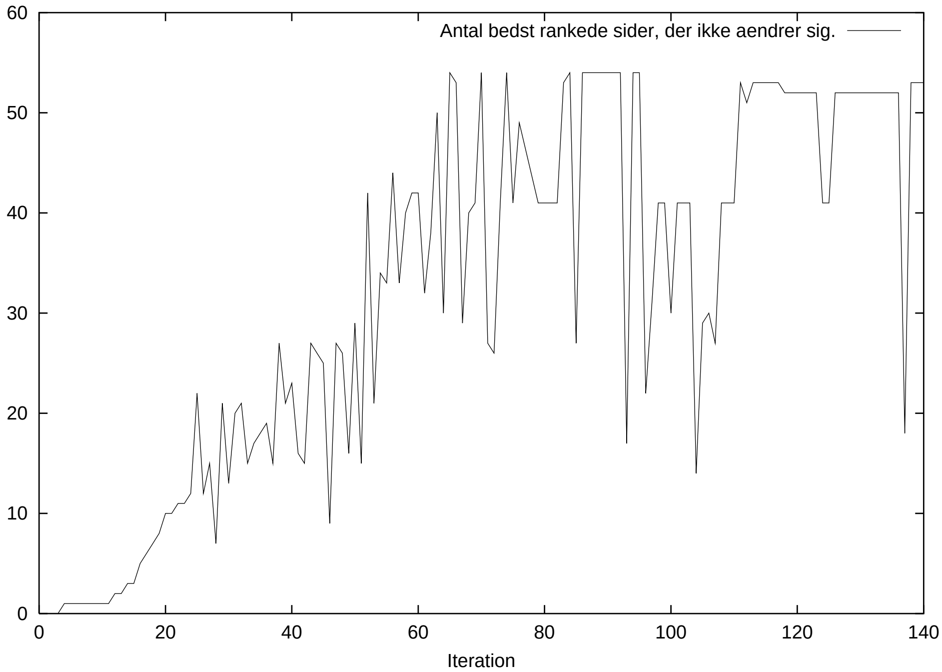
tilsyneladende var det ikke tilfældet. Men måske kunne det skyldes at mange links ikke kan parses , samt det faktum at links til udlandet ikke er med. Vi ved det ikke med sikkerhed.

I nedenstående graf har vi plottet sammenhængen mellem indgraden af sider og deres pagerank.



Umiddelbart ville man måske forvente at jo højere indgraden var jo højere ville pagerank være. Men det er jo netop det pagerank skal hindre, altså at de indgående links ikke har samme vægt. Derfor ser resultatet i grafen egentlig meget fornuftigt ud. Der er kun medtaget docs med indgrad fra 0-700, fordi der efter 700 er en enkelt side med høj pagerank, nemlig http://www.sol.dk og resten ligger meget lavt. Den højeste indgrad er 3520.

Som en sidste interessant observation har vi plottet antallet af de bedste sider der ikke ændrer sig fra iteration til iteration som funtion af antallet af iterationer. Dette tal er (som forventet stigende), men af og til falder det over en enkelt iteration, idet to højte rankede dokumenter trods alt pga. finjusteringer bytter pladser.



Samarbejde

Samarbejdsformen med at flere grupper skulle lave et stort koordineret projekt fungerede set fra vores synspunkt godt. Pagerank er et forholdsvis afgrænset og pænt område. Pænt i den forstand at den kan beregnes ud fra en graf, og der er en god matematisk model bag. Det var således også nemt for os at samarbejde med de andre grupper. Idet vi blot krævede grafstrukturen og leverede pagerankvektoren tilbage. Det giver en god variation, at man skal samarbejde, og at samarbejdet munder i noget større og mere brugbart end hvad en enkelt lille gruppe kunne lave. Ydermere er det altid mere spændende at se teori ført ud i praksis på data fra den virkelige verden, og det må man jo sige at der var tale om her. En sidste positiv ting er at man får gået i dybden med et specifikt område, som f.eks. vores pagerank. Men til gengæld er vi så ikke helt så godt inde i de områder de andre grupper har beskæftiget sig med. Skal man pege på en ulempe ved projektformen må det nok være det at ikke alle grupper arbejder i samme tempo. Da gruppernes opgaver ikke var helt uafhængige betød det, at der kunne forekomme ventetid på at en gruppe fik lavet noget færdigt som en anden gruppe skulle bruge.

Endelig løsning — detaljer + tids- og pladsforbrug

Den endelige løsning baserer sig som nævnt på artiklen "Efficient Computation of Pagerank". Vi besluttede os for at lave den idet sorteringen tog for lang tid i den tidligere nævnte algoritme. Først lidt notation:

- N er antallet af sider (knuder i grafen).
- P er størrelsen af hukommelsen.
- M er antallet af links (kanter i grafen).

Vi har lavet den blokbaseret så den skalerer op. Med det beskedne antal sider (ca. 1.7 millioner), der blev crawlet ville det være muligt at lave pagerank helt og aldeles i hukommelsen. Men for at lave det lidt realistisk valgte vi altså den blokbaserede strategi. Hovedmålet var at undgå sortering og fremme sekventiel læsning/skrivning af filer. Dette gøres ved at holde så meget som muligt af den nye pagerankvektor i hukommelsen.

Først scannes graffilen, som ankommer sorteret efter sourceId, for at finde udgrad og antallet af sider. Under scanningen dannes udgradsfilen, og bagefter genereres den initielle pagerankfil, hvor alle sider har rank $\frac{1}{N}$. Filerne er tekstfiler, og hver linie repræsenterer en side, så fx på linie 238 står pagerank for docId 238, og tilsvarende for udgraden i udgradsfilen.

Efter dette ligger udgradsfilen fast og bliver ikke ændret mere. Så laver vi opdelingen i blokke. Det eneste vi opbevarer i hukommelsen er dele af den nye pagerankvektor. Derudover afsætter vi lidt plads til andre variabler og diverse. Lad os f.eks. sige at $P = 4000000$ bytes. (4 Mb)

Vi bruger doubles til at repræsentere PageRank og en double fylder 8 bytes. Lad $N = 1.700.000$ sider. Så bliver

$$blocksize = \frac{4000000 - 25000}{8} = 496875.$$

Det ville så kræve 4 blokke givet en hukommelse på 4Mb for at dække de 1.7 mill. sider.

Så deler vi graffilen op 4 blokke; dette gøres ved at løbe igennem graffilen og smide hvert enkelt link i den rette blokfil. Dette gøres ved at destId i intervallet $[0, blocksize[$ kommer i blok 0. destId i intervallet $[blocksize, 2 * blocksize[$ kommer i blok 1 osv. De nye blokfiler fylder i forening det samme som den gamle graffil. Bemærk at det opretholder sorteringen efter sourceId. Såfremt der er plads til alle N sider i hukommelsen bliver blokstørrelsen blot sat til N, og der er således kun en blokfil, nemlig den oprindelige graf.

Nu kan vi så begynde beregningen af pagerank. Den køres i et vist antal iterationer, som man selv kan vælge. For hver iteration sker der flg.:

For hver fil udføres dette: Et array på størrelse blocksize initialeres til 0. (Pladsen til arrayet er allokeret da vi fandt blocksize). Derefter åbnes den gamle pagerankfil og den blokfil vi er i gang med samt udgradsfilen. Så køres en blokfilen igennem, og for hvert link i den lægges den delsum det link udgør til den rette indgang i arrayet. Samtidig scannes pagerankfilen og udgradsfilen sekventielt, så de følger med. Dvs. hvis fx blokfilen er ved linket (3556,4) og det foregående link var (3551,76) så spoles PageRank og udgrad 5 linier frem og værdierne aflæses og bruges i beregningen. Det smarte her – om man så må sige – er, at vi kan adressere direkte til arrayet og der skal ikke sorteres for at få den nye rank. Så med eksemplet (3551,76), tager vi pagerank og udgrad for sourceId 3551 og lægger til indgang [76-offset] i arrayet. 'offset' er lig nummeret på filen gange blocksize.

Når blokfilen er kørt igennem udskrives værdierne i arrayet til den nye pagerankfil, samtidig med udskrivningen får de lagt deres dampeningfaktor og bidrag fra 'fup'-siden til. Fup-siden er en knude i grafen som alle sider med udgrad 0 peger på. Dette gøres for at sikre at ingen knuder har udgrad 0. Fupsiden peger så til gengæld på alle N sider. Dette gøres fordi udregningen som tidligere nævnt opererer under den antagelse at alle knuder har en udgrad større end eller lig med 1.

I artiklen der ligger til grund for vores algorithme skriver de at de fjerner alle knuder med udgrad 0 en gang, og så gentager skridtet en gang til. Dette reducerer deres knudeantal fra 81 millioner til 19 millioner. Men vi mener ikke at det garanterer at der ikke stadig er knuder med udgrad 0. Så derfor har vi lavet denne fupknude, som så at sige fordeler pagerankbidraget fra alle knuder med udgrad 0 til alle N knuder. Knuden og dens tilhørende links er ikke med i graffilen, men får sin pagerank beregnet separat ved at løbe udgradsfilen igennem og lade hver knude med udgrad 0 bidrage med sin pagerank.

Efter udskrivningen af arrayet gentages beregningen for den næste blokfil. Når alle blokfilerne er løbet igennnem er den nye pagerankvektor beregnet og skrevet til en fil. I den næste iteration er det så den nye pagerankfil, der bruges og den nye 'fup'-knude.

Tid og plads

Som det fremgår af ovenstående bruger algoritmen oplagt blocksize hukommelse, mens den bruger filplads i størrelsesordenen

$$2M + 3N.$$

De $3N$ er til den nye og den gamle pagerankfil samt udgradsfilen. Mens de $2M$ er til graffilen og blokfilerne.

Mht. tid er den lineær i hver iteration i antallet af links plus antallet af blokke gange størrelsen af pagerank. For at understøtte det, bemærkes først at M i praksis altid vil være en del større end N, ellers ville der ikke være en nogenlunde sammenhængende graf at arbejde på.

Først scannes graffilen, det tager tid M. Bagefter genereres pagerank filen, det tager tid N. Så scannes grafen igen og splitte og i et antal filer, det tager tid M. Dette er starttidsforbruget før vi begynder at iterere, som jo klart er domineret af M.

For hver iteration køres graffilen igennem en gang idet hver blokfil køres igennem en gang. Ydermere initialiseres samt udskrives arrayet. Udgrads- og pagerankfilen køres igennem lige så mange gange, som der er blokfiler dvs. $\frac{N}{blocksize}$. Det er så dette tal, der kan blive afgørende for tidsforbruget. Når dette tal gange N overstiger M, er det dominerende. Ellers er det M, der bestemmer tidsforbruget.

Antag fx at hver side i snit har 7 i udgrad. Så er M 7 gange større end N. Antag ligeledes, at maskinen fx har 3Gb hukommelse. Så ville der være plads til ca. 375 millioner indgange i arrayet, og med fx 3 milliarder sider, ville det så kræve 8 blokfiler.

Kodereference

Koden kan findes i /users/jve/public_html/webalg
eller via internettet på http://www.daimi.au.dk/jve/webalg.