

Tópicos em Recuperação de Informação

Nivio Ziviani

Conjunto de transparências elaborado por Nivio Ziviani, Patrícia Correia e Fabiano C. Botelho.

Tipos de Consultas

Existem dois tipos de consultas:

- ▶ Lógica (*boolean*)
 - ▶ lista de termos que são combinada através dos operadores lógicos AND, OR e NOT.
- ▶ Ordenada (*ranked*)
 - ▶ envolve uma heurística que é aplicada para medir a similaridade de cada documento com a consulta.

Consulta Lógica

- ▶ A resposta à consulta é formada pelos documentos que satisfazem a condição estipulada.

Ex.: text AND compression AND retrieval

- ▶ Busca ampla $\Rightarrow$ Aumenta revocação (*Recall*).

Ex.:

(text OR data OR image)
AND
(compression OR compaction OR decompression)
AND
(archiving OR retrieval OR storage OR indexing)

- ▶ () indicam a ordem das operações.

Consulta Ordenada

- ▶ Esforço para obter:
 - medidas de similaridade.
 - estratégias de ordenação.
 - ▶ Com o objetivo de manter:
 - revocação alta.
 - precisão alta.
1. Técnica simples:
 - contagem do número de termos da consulta que aparecem no documento.
 2. Técnica sofisticada:
 - atribui um peso numérico a cada termo.
 - medida do cosseno.
 - modelo vetorial.

Consulta Ordenada

3. Muitas outras técnicas...

4. Coleção TREC:

- 50 consultas.
- medida do cosseno consegue precisão acima de 40% na média (difícil de conseguir com modelo booleano).

Processamento de Consultas Lógicas

1. Vocabulário é pesquisado para cada termo.
 2. Cada entrada do arquivo invertido é recuperada (e decodificada).
 3. As entradas são intercaladas (interseção, união ou complemento).
 4. Documentos são recuperados de acordo com a lista final.
- ▶ Tipicamente, consultas com 5 a 10 termos levam cerca de 1 segundo para o processamento das listas (para coleção e computador especificado no livro Witten, Moffat e Bell, 1999).
 - Visualização depende do tamanho da resposta.

Tipos de Consultas Lógicas

Conjuntivas:

text AND compression AND retrieval

- ▶ Termos são ordenadas pela frequência do termo na coleção f_t .
- ▶ Lista de documentos candidatos é inicializada a partir do termo t de menor f_t : $C \leftarrow I_t$

Algoritmo para Consultas Conjuntivas

1. **Para cada** termo t da consulta **faça**
 - (a) Obtenha t .
 - (b) Pesquise no vocabulário.
 - (c) Armazene f_t e o endereço de I_t (entrada do arquivo para t).
2. Identifique o termo t da consulta com o menor f_t .
3. Leia a entrada I_t correspondente do arquivo invertido.
Faça $C \leftarrow I_t$, sendo C a lista de *candidatos*.
4. **Para cada** termo t restante **faça**
 - (a) Leia a entrada I_t do arquivo invertido.
 - (b) **Para cada** $d \in C$ **faça**
Se $d \notin I_t$ **então** $C \leftarrow C - \{d\}$.
 - (c) **Se** $|C| = 0$ **então**
Retorne *não existem respostas*.
5. **Para cada** $d \in C$ **faça**
 - (a) Procure o endereço do documento d .
 - (b) Recupere o documento d e o apresente ao usuário.

Ordem de Processamento dos Termos

Razões para iniciar C com termo de menor f_t :

1. Minimizar espaço temporário de memória
 - ▶ Tamanho de C é não crescente.
 - ▶ Maior tamanho é na inicialização.
 - ▶ C pode ser reduzido rapidamente, algumas vezes a 0.

Ordem de Processamento dos Termos

Razões para iniciar C com termo de menor f_t :

2. Minimizar tempo de resposta

Processamento envolve consulta a membros da lista e não intercalação.

- ▶ Intercalação envolvendo $|C|$ candidatos e f_t apontadores: $|C| + f_t$ passos.
- ▶ Se $|C| = 60$ e $f_t = 60000 \Rightarrow 60060$ passos.
- ▶ Como as listas estão ordenadas: busca binária $\Rightarrow |C| \log 60060$.
- ▶ Para $j > 2$ tamanho da menor lista é $O(1)$.
- ▶ Para $j = 2$ tamanho da menor lista é $O(n^{\beta(1-2\beta)} \log n)$.
- ▶ Veja Navarro, Moura, Neubert, Ziviani, Baeza-Yates, *Information Retrieval* 3(1), 2000, 49-77.

Tipos de Consultas Lógicas

Não-Conjuntivas:

*(text OR data OR image) AND
(compression OR compaction) AND
(retrieval OR indexing OR archiving)*

- ▶ Cada termo entre parêntesis pode ser processado simultaneamente:
 - ▶ candidatos ficam no conjunto se aparecem em qualquer membro do grupo OR.
- ▶ C deve ser inicializado com o menor conjunto resultante da união dos membros.
- ▶ Aproximação para tamanho de cada conjunto:
 - ▶ soma dos f_t de cada termo constituintes ignorando possibilidade de repetição no OR (aproximação pessimista).

Processamento de Consultas Ordenadas

Coordinate matching:

Técnica simples: conta o número de termos da consulta que aparece em cada documento.

- ▶ Quanto maior o número de termos da consulta em um documento, mais provável que ele seja relevante.
- ▶ Consulta se torna híbrida, intermediária entre a consulta conjuntiva AND e a consulta disjuntiva OR:
 - ▶ documento que contenha qualquer dos termos é resposta potencial.
 - ▶ preferência é para documentos que contenham todos ou a maioria dos termos.
- ▶ Toda informação necessária está no arquivo invertido.

Processamento de Consultas Ordenadas

d	Documento D_d
1	Pease porridge hot, pease porridge cold,
2	Pease porridge in the pot,
3	Nine days old.
4	In the pot cold, in the pot hot,
5	Pease porridge, pease porridge,
6	Eat the lot.

- Coleção com 6 documentos e 10 termos.

$Q = \text{"eat"} \Rightarrow D_6$

$Q = \text{"hot porridge"}$

- Conjuntiva: D_1

- Coordinate matching: $D_1 > D_2 = D_4 = D_5 > D_3 = D_6$

Similaridade do Produto Interno

- ▶ Produto interno entre vetor da consulta e vetores de docs.

(a)	d	Vetores de documentos ($w_{d,t}$)									
		<i>col</i>	<i>day</i>	<i>eat</i>	<i>hot</i>	<i>lot</i>	<i>nin</i>	<i>old</i>	<i>pea</i>	<i>por</i>	<i>pot</i>
	1	1	0	0	1	0	0	0	1	1	0
	2	0	0	0	0	0	0	0	1	1	1
	3	0	1	0	0	0	1	1	0	0	0
	4	1	0	0	1	0	0	0	0	0	1
	5	0	0	0	0	0	0	0	1	1	0
	6	0	0	1	0	1	0	0	0	0	0

(b)											
	<i>eat</i>	0	0	1	0	0	0	0	0	0	0
	<i>hot porridge</i>	0	0	0	1	0	0	0	0	1	0

- ▶ Similaridade entre consulta Q com documento D_d :

$$M(Q, D_d) = Q \cdot D_d$$

onde,

$$X \cdot Y = \sum_{i=1}^n x_i \cdot y_i$$

$$\text{Ex.: } M(\text{"hot, porridge"}, D_1) = 2$$

Problemas

1. Não considera frequência do termo.

Ex.: *porridge* aparece 2 vezes em D_1 e 1 vez em D_2

$Q = \text{"porridge"}$: os 2 documentos são ordenados igualmente.

2. A fórmula não considera a raridade do termo.

Ex.: *eat* aparece somente em D_6 e *porridge* aparece em D_1 , D_2 e D_5 .

3. Longos documentos são favorecidos (talvez indevidamente pela diversidade do vocabulário, devido ao tamanho).

Soluções

1º Problema:

- ▶ Substituir bit por inteiro indicando o número de vezes que o termo aparece no documento.

Ex.: $M(\text{"hot, porridge"}, D_1)$

$(0,0,0,1,0,0,0,0,1,0).(1,0,0,1,0,0,0,2,2,0) = 3$

- ▶ Esta idéia pode ser generalizada:

$w_{d,t}$: document-term weight e $w_{q,t}$: query-term weight

$$M(Q, D_d) = Q \cdot D_d = \sum_{i=1}^n w_{q,t} \cdot w_{d,t}$$

- ▶ Quando t não ocorre em Q , $w_{q,t} = 0$, logo:

$$M(Q, D_d) = \sum_{t \in Q} w_{q,t} \cdot w_{d,t}$$

2º Problema:

- ▶ Dar peso aos termos de acordo com o inverso da frequência em documentos - idf (*inverse document frequency*).

$$w_t = \log_e \left(1 + \frac{N}{f_t} \right)$$

onde,

N : número de documentos na coleção.

f_t : número de documentos que contém termo t .

$$w_{d,t} = (1 + \log_e f_{d,t}) \cdot w_t$$

- ▶ Conhecida como regra TF x IDF (*term frequency x inverse document frequency*)

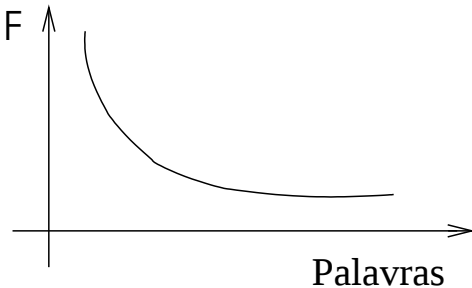
Soluções

Lei de Zipf, 1949:

Frequência do termo é inversamente proporcional ao seu “rank”.

Zipf, “Human Behavior and the Principle of Least Effort”, 1949

Frequência da i -ésima palavra mais frequente é $1/i^\Theta$ vezes a da palavra mais frequente, $\Theta \geq 1$



Soluções

w_t : **Peso do termo t**

$$w_t = \frac{1}{f_t}$$

$$w_t = \log_e(1 + \frac{N}{f_t})$$

$$w_t = \log_2 \frac{N}{f_t} \quad \text{Persin, Zobel \& Sacks_Davis: JASIS, 1996}$$

- Logaritmo não deixa um termo com $f_t = 1$ ser duas vezes mais importante que um termo com $f_t = 2$

Soluções

$r_{d,t}$: **Relative term frequency**

- Existem muitas maneiras para calcular $r_{d,t}$

$$r_{d,t} = 1 + \log_e f_{d,t}$$

$$r_{d,t} = 1$$

$$r_{d,t} = f_{d,t}$$

$$r_{d,t} = \log_2 f_{d,t} \quad \text{Persin, Zobel \& Sacks_Davis: JASIS, 1996}$$

- Crescimento menor quando $f_{d,t}$ cresce com o uso do logaritmo (crescimento monotônico)

Valores de documentos podem ser calculados como:

$$W_{d,t} = r_{d,t} \text{ ou}$$

$$W_{d,t} = r_{d,t} \cdot w_t \quad (\text{TF} \times \text{IDF})$$

$r_{d,t}$: crescimento monotônico

w_t : decrescimento monotônico

Soluções

- ▶ A direção de dois vetores (documento e consulta), ou mais precisamente a *diferença da direção de cada vetor*, é o que importa.
- ▶ Ângulo Θ entre dois vetores satisfaz:

$$X.Y = |X||Y| \cos \Theta$$

- $X.Y$: produto interno
- $|X| = \sqrt{\sum_{i=1}^n x_i^2}$: distância Euclidiana do vetor X
- X : vetor n -dimensional $\langle x_i \rangle$

$$\cos \Theta = \frac{X.Y}{|X||Y|}$$

3º Problema:

Documentos longos são favorecidos porque contêm mais termos e o valor do produto interno cresce.

- ▶ Introduzir a normalização:

$$M(Q, D_d) = \frac{\sum_{t \in Q} w_{q,t} \cdot w_{d,t}}{|D_d|}$$

onde,

$$|D_d| = \sum_i f_{d,t} \text{ (comprimento do documento } D_d)$$

Implicações:

1. Normalização descrita anteriormente é a distribuição Euclidiana do documento – i.e., comprimento no espaço n -dimensional dos pesos documento-termo que descrevem o documento
2. Fórmula dá uma visualização clara do que a regra de ranking significa:
 - ▶ Documentos são pontos na região positiva do espaço n -dimensional, onde pequenos documentos estão mais próximos da origem e *consultas são raios a partir da origem*

Modelo de Espaço Vetorial

- ▶ Documentos e consultas são representados por vetores em um espaço n -dimensional.
- ▶ Da álgebra vetorial:

$$\cos \Theta = \frac{X \cdot Y}{|X| \cdot |Y|} = \frac{\sum_{i=1}^n x_i \cdot y_i}{(\sqrt{\sum_{i=1}^n x_i^2})(\sqrt{\sum_{i=1}^n y_i^2})}$$

- ▶ Normalização: distância Euclidiana do documento (comprimento no espaço n -dimensional do conjunto de pesos doc-termo).
- ▶ Similaridade expressa pelo $\cos \Theta$: quanto maior o cosseno maior a similaridade.

$$\cos \Theta = 1 \text{ para } \Theta = 0$$

$$\cos \Theta = 0 \text{ para } \Theta = 90^\circ$$

Modelo de Espaço Vetorial

$$\cos(Q, D_d) = \frac{Q \cdot D_d}{|Q| |D_d|} = \frac{\sum_{i=1}^n w_{q,t} \cdot w_{d,t}}{W_q W_d}$$

onde,

$$W_d = \sqrt{\sum_{i=1}^n w_{d,t}^2} \quad \text{e} \quad W_q = \sqrt{\sum_{i=1}^n w_{q,t}^2}$$

- ▶ Pode-se usar qualquer método “*term-weighting*”.
- ▶ Um bom método (simples): TF x IDF

$$\cos(Q, D_d) = \frac{1}{W_d W_q} \sum_{t \in Q \cap D_d} (1 + \log_e f_{d,t}) \cdot \log_e \left(1 + \frac{N}{f_t}\right)$$

- ▶ Como W_q é constante para uma dada consulta ele pode ser desprezado sem alterar o “*ranking*”.

Modelo de Espaço Vetorial

d	Vetores de documentos ($w_{d,t}$)										W_d
	<i>col</i>	<i>day</i>	<i>eat</i>	<i>hot</i>	<i>lot</i>	<i>nin</i>	<i>old</i>	<i>pea</i>	<i>por</i>	<i>pot</i>	
1	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.7	1.7	0.0	2.78
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.73
3	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.73
4	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.7	2.21
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.7	1.7	0.0	2.39
6	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.41
f_t	2	1	1	2	1	1	1	3	3	2	
w_t	1.39	1.95	1.95	1.39	1.95	1.95	1.95	1.10	1.10	1.39	

d	Consulta			
	<i>eat</i>	<i>porridge</i>	<i>hot porridge</i>	<i>eat nine day old porridge</i>
	$W_q = 1.95$	$W_q = 1.10$	$W_q = 1.77$	$W_q = 3.55$
1	0.00	0.61	0.66	0.19
2	0.00	0.58	0.36	0.18
3	0.00	0.00	0.00	0.63
4	0.00	0.00	0.36	0.00
5	0.00	0.71	0.44	0.22
6	0.71	0.00	0.00	0.39
Top	6	5	1	3

Medida do Cosseno

- ▶ Exige mais informação do que o processamento de consultas booleanas
- ▶ Decisões importantes sobre como estruturar a informação para fazer o *processamento do ranking eficiente*:
 1. Precisamos da $f_{d,t}$ no índice
 2. Como avaliar a medida do cosseno
 - ▶ Para grandes coleções é potencialmente caro
 - tempo CPU
 - largura de banda de disco
 - demanda por memória
 - ▶ Soluções
 - limitar o número de candidatos ao ranking
 - ordenar por $f_{d,t}$
 - representação eficiente de w_d

Implementação da Medida do Cosseno

Frequência de t em $d(f_{d,t})$:

- ▶ $f_{d,t}$ tem que ser adicionado ao índice.
- ▶ $f_{d,t}$ é necessário para a maioria dos métodos de atribuição de peso aos termos (*term weight assignment*).
- ▶ Entrada para “porridge” no arquivo invertido:
 $\langle 3; [1, 2, 5] \rangle$ passa a ser $\langle 3; [(1, 2)(2, 1), 5, 2] \rangle$
onde,

$f_{porridge} = 3$ (*porridge* ocorre em 3 documentos: 1, 2, 5)

$f_{1,porridge} = 2$ (*porridge* ocorre 2 vezes no documento 1)



$$\langle f_t; [(d_1, f_{1,t}), (d_2, f_{2,t}), \dots, (d_{f_t}, f_{d_{f_t},t})] \rangle$$

Implementação da Medida do Cosseno

- ▶ Como resolver problema de espaço extra necessário para armazenar $f_{d,t}$?
- ▶ Código unário:

1	0
2	10
3	110

- ▶ Logo, a entrada para o termo t no arquivo invertido cresce de um número de bits igual ao número total de ocorrências de t na coleção.

$$F_t = \sum_{d=1}^N f_{d,t} \quad (\text{total de ocorrências de } t)$$

$$F = \sum_{i=1}^n F_t \quad (\text{total de palavras na coleção})$$

- ▶ Logo, o arquivo invertido cresce de F bits.

Implementação da Medida do Cosseno

Fato:

É possível indexar toda palavra do texto usando menos de 1 byte por apontador (mesmo usando $f_{d,t}$)

Elias- γ :

- ▶ γ nunca é muito pior que unário, mas pode ser muito melhor!
 - ▶ Para documentos pequenos ocupa 1 bit a mais do que o unário
 - ▶ Muito melhor quando existem muitos termos com $f_{d,t}$ alto

Implementação da Medida do Cosseno

Método	Bits por apontador			
	<i>Bíblia</i>	<i>GNUbib</i>	<i>Comact</i>	<i>TREC</i>
Unário	1.27	1.16	1.74	2.49
γ	1.38	1.23	1.88	2.13
δ	1.53	1.32	2.15	2.41
Observed frequency	1.27	1.15	1.72	2.02
Interpolative	0.76	0.45	1.41	1.77

- ▶ Coleções *Bible*, *GNUbib*, *Comact*: código unário é perto do ótimo. Isto ocorre porque a distribuição das probabilidades é tipo: $Pr[x] = (\frac{1}{2})^x$ (probabilidades são múltiplos de 1/2).
- ▶ Usando compressão, mesmo para índice aumentando de $f_{d,t}$, para TREC o índice ocupa 6.3% do texto.
- ▶ Elias- γ : melhor opção para coleções grandes em linguagem natural. Consome $1 + 2 \log x$ bits (menor do que o unário para termos com $f_{d,t}$ alto).

Cálculo do Valor do Cosseno

$$\cos(Q, D_d) = \frac{1}{W_d W_q} \sum_{t \in Q \cap D_d} (1 + \log_e f_{d,t}) \cdot \log_e \left(1 + \frac{N}{f_t}\right)$$

- ▶ $W_q = \text{constante}$
- ▶ $W_d = \sqrt{\sum_{i=1}^n w_{d,t}^2}$
- ▶ $w_{d,t} = (1 + \log_e f_{d,t}) \cdot w_t$
- ▶ $w_t = \log_e \left(1 + \frac{N}{f_t}\right)$

Estruturas de dados:

1. Arranjo para W_d (pode ser pré-computado).
2. Acumuladores indexados por d para acumular $\cos(Q, D_d)$ (qualquer doc d contendo algum termo de Q faz $A_d > 0$).

Cálculo do Valor do Cosseno para r Docs

1. $A \leftarrow \{\}$. A é o conjunto de acumuladores.
2. **Para cada** termo na consulta $t \in Q$ **faça**
 - (a) Obtenha t .
 - (b) Pesquise no vocabulário.
 - (c) Armazene f_t e o endereço de I_t , a entrada do arquivo invertido para t .
 - (d) $w_t \leftarrow 1 + \log_e (N/f_t)$.
 - (e) Leia a entrada do arquivo invertido I_t .
 - (f) **Para cada** par $(d, f_{d,t})$ em I_t **faça**
 - i. **Se** $A_d \notin A$ **então**
 $A_d \leftarrow 0$,
 $A \leftarrow A + \{A_d\}$.
 - ii. $A_d \leftarrow A_d + \log_e (1 + f_{d,t}) * w_t$
3. **Para cada** $A_d \in A$ **faça**
 $A_d \leftarrow A_d / W_d$.
 A_d agora é proporcional ao valor $\text{cosseno}(Q, D_d)$.
4. **Para** $1 \leq i \leq r$ **faça**
 - (a) Escolha d tal que $A_d = \max\{A\}$.
 - (b) Procure o endereço do documento d .
 - (c) Recupere o documento d e o apresente para o usuário.
 - (d) $A \leftarrow A - \{A_d\}$.

Cálculo do Valor do Cosseno

Pontos importantes :

1. W_q e $f_{q,t}$ são ignorados
 W_q é constante e não afeta o *ranking*.
 $f_{q,t} = 1$ para palavras presentes.
 $f_{q,t} = 0$ para palavras ausentes.
2. Muita memória é necessária
 W_d : 4 bytes por documento.
 A_d : 4 bytes por documento.
3. $r \ll N$
Passo 4 não precisa ordenar completamente A_d , basta extrair os r maiores valores.

Cálculo do Valor do Cosseno

Memória para W_d :

TREC 2: 740.000 documentos

1. Guardar W em disco e acessar seqüencialmente.
2. Guardar $f_{d,t}/W_d$ ao invés de $f_{d,t}$
 - Problema: se o arquivo invertido é mantido comprimido esta opção é muito cara!

Logo, passa-se menos de 2 bits por $f_{d,t}$ para 16 ou 32 bits.

Tratamento dos Acumuladores A_d

- ▶ Heurísticas baseadas no fato de que entradas do arquivo invertido são processadas na ordem crescente de f_t (ou decrescente de w_t), como nas consultas booleanas.
- ▶ Heurística: *ranking* será uma aproximação da regra do cosseno.

Regra simples:

- ▶ Limite “a priori” no número de acumuladores > 0 permitidos, parando de adicionar novos valores quando limite é atingido.
- ▶ Termos de peso alto são considerados e termos de peso baixo são desconsiderados.

Tratamento dos Acumuladores A_d

Regra simples:

E os termos restantes?

1. *Quit*

Ignora contribuição dos termos não processados (mesmo se eles são para documentos que já tenham acumuladores processados).

2. *Continue*

Continua o processamento de entradas do arquivo invertido:

- documentos ainda sem acumuladores são ignorados.
- documentos para os quais acumuladores tenham sido criados continuam a ter contribuições para o seu cosseno acrescidas.

Tratamento dos Acumuladores A_d

Regra simples:

1. *Quit*

- Processamento rápido: entradas não processadas são as mais longas.
- Resultados pobres de *ranking*.

2. *Continue*

- *Retrieval performance* comparável ao método exato para o cosseno.

3. Experimentos TREC com conjunto de consultas padrão:

- 5000 acumuladores são suficientes para extrair os 1000 top documentos.

Ordenação

- ▶ Como evitar ordenação?

$$N = 1.000.000$$

$$N \log N = 20 \text{ milhões}$$

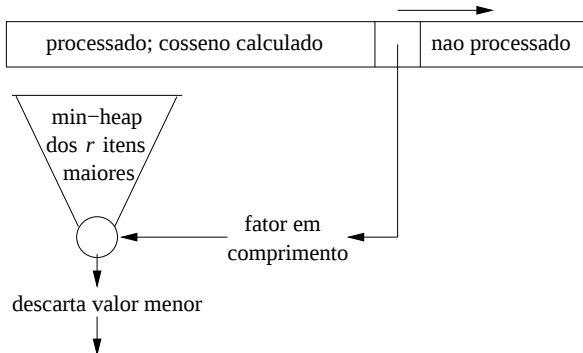
$$r \ll N$$



Uso de um *heap*

- ▶ Construção $\approx 2N$
 - r extrações $\approx 2r \log N$
- ▶ Qual o problema de usar um *heap* com N elementos?
 - espaço para armazenar N endereços de documentos.

Ordenação



- Solução: armazenar no *heap* r valores (*min-heap*).

Algoritmo:

- Inicia com os primeiros r acumuladores (com endereço dos documentos). Custo: $2r$.
- Lê novo acumulador: se maior que o menor valor do *heap* $\Rightarrow$ substituir.

Ordenação: Seleciona r Valores de Topo

1. $H \leftarrow \{\}$. H é o *heap*.
2. **Para** $1 \leq d \leq r$ **faça**
 - (a) $A_d \leftarrow A_d/W_d$.
 - (b) Armazene o endereço do documento d .
 - (c) $H \leftarrow H + \{A_d\}$.
3. Construa H no *heap*.
4. **Para** $r + 1 \leq d \leq N$ **faça**
 - (a) $A_d \leftarrow A_d/W_d$.
 - (b) **Se** $A_d > \min\{H\}$ **então**
 - i. $H \leftarrow H - \min\{H\} + \{A_d\}$.
 - ii. “Peneire” H .
 - iii. Armazene o endereço do documento d .

H agora contém os r valores do *cos seno* exatos do topo.
5. **Para** $1 \leq i \leq r$ **faça**
 - (a) Selecione d tal que $A_d = \max\{H\}$.
 - (b) Recupere o documento d e o apresente para o usuário.
 - (c) $H \leftarrow H - \{A_d\}$.

Processamento de Consultas

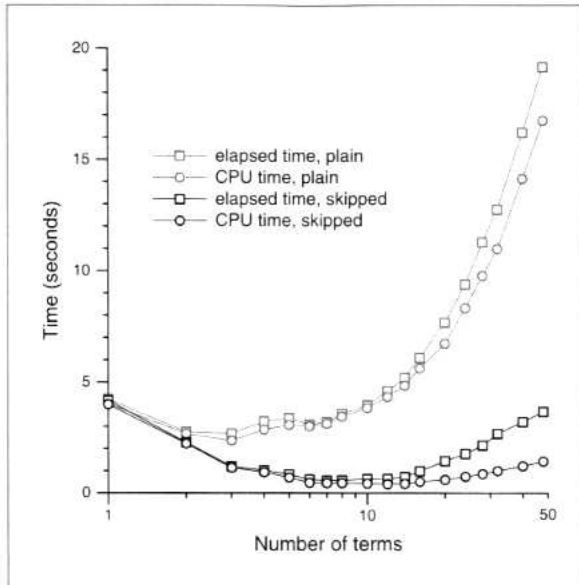


Figure 9.10 Time required by *mg* for Boolean queries.

Processamento de Consultas

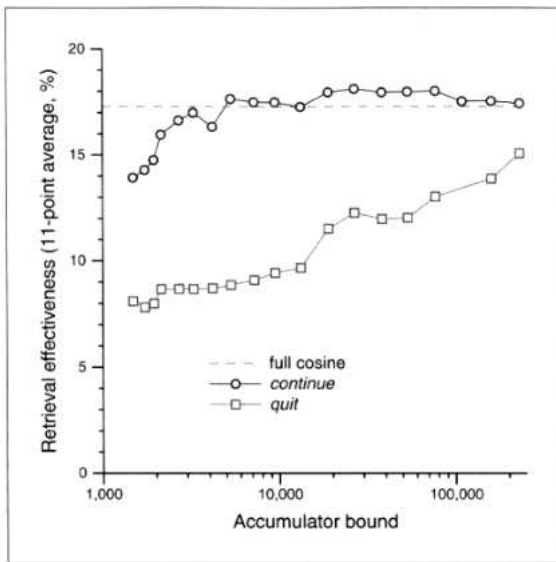
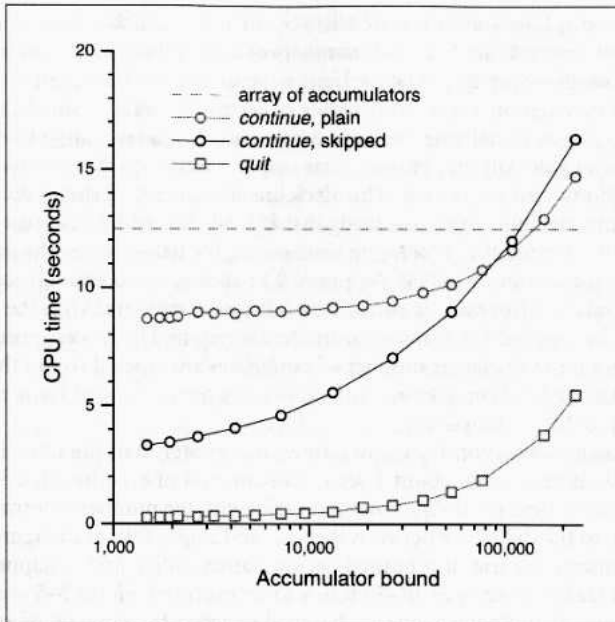


Figure 9.11 Ranking effectiveness: *quit* and *continue*.

Processamento de Consultas



Medida do Cosseno: Algoritmo Básico

1. Crie e inicialize uma estrutura de acumuladores (A).
2. **Para cada** termo t na consulta **faça**.
 - (a) Recupere lista invertida do termo t do disco.
 - (b) **Para cada** par $[d, f_{d,t}]$ na lista invertida t **faça**:
 - i. **Se** $A_d \notin A$ **então**
 $A_d = 0$,
 $A = A + \{A_d\}$
 - ii. $A_d = A_d + w_{q,t} * w_{d,t} (sim_{q,d,t})$
3. Divida cada acumulador A_d por W_d .
4. Recupere os r maiores valores para os acumuladores e retorne os documentos correspondentes.

Medida do Cosseno

$$C_{q,d} = \frac{\sum_t sim_{q,d,t}}{W_d}$$

$$sim_{q,d,t} = w_{q,t} \times w_{d,t}$$

similaridade parcial entre q e d em relação ao termo t

$w_{x,t}$: peso de t na consulta ou no documento

$$W_d = \sqrt{\sum_t w_{d,t}^2}$$

W_d : comprimento do documento d (pré-computado)

$$w_{d,t} = \log_2 f_{d,t} \times \log_2 \frac{N}{f_t}$$

Persin, Zobel & Sacks-Davis: JASIS, 1996

$$w_{d,t} = (1 + \log_e f_{d,t}) \times \log_e (1 + \frac{N}{f_t})$$

WMB, MG, 1999

$$w_{d,t} = (1 + \log_2 f_{d,t}) \times \log_2 \frac{N}{f_t}$$

BYR, MIR, 2011

Medida do Cosseno

Estruturas de Dados para A_d :

- ▶ Aloca acumulador para documento d no início
- ▶ Adiciona dinamicamente um acumulador para cada documento d quando $sim_{q,d,t} > 0$

Medida do Cosseno

Custo de avaliar a medida do cosseno pode ser muito alto:

- consultas podem conter muitos termos
- uma medida de similaridade é atribuída a cada documento contendo um dos termos da consulta

Solução:

- técnicas para filtrar documentos durante a ordenação
- acumulador é incrementado somente se a combinação $tf \times idf$ é grande o suficiente para causar alteração na ordem final

Medida do Cosseno

Duas heurísticas adicionais para limitar o número de acumuladores:

1. Estratégia de corte que processa listas invertidas na ordem w_t decrescente, permitindo criar acumuladores enquanto $w_t > \max_{t \in Q} \frac{w_t}{3}$
 - Avaliação do restante dos termos não cria mais acumuladores, mas podem contribuir quando já houver A_d criado
2. Processa uma lista invertida apenas se w_t do termo mais todos os outros termos restantes é tal que o documento ranqueado na posição $r + 1$ poderia entrar nos r documentos do topo
 - Garante os top r documentos mas não necessariamente na mesma ordem da medida exata do cosseno

Medida do Cosseno

Como avaliar:

1. Potencialmente caro para grandes coleções

- tempo CPU
- largura de banda de disco
- demanda por memória

2. Soluções

- limitar o número de documentos candidatos ao ranking
- ordenar por $f_{d,t}$
- representação eficiente dos W_d

Objetivo:

- procurar não usar mais memória e tempo de CPU necessários ao processamento da consulta booleana

Medida do Cosseno - Persin

Persin, Zobel & Sacks_Davis, "Document Retrieval with Frequency-Sorted Indexes", JASIS 47(10), 749-764, 1996

Custos Principais:

- ▶ Memória
 - ▶ armazena as medidas de similaridade
 - ▶ A_d : um acumulador por documento
- ▶ Acesso a disco
 - ▶ transferência das listas invertidas do disco para a memória
- ▶ Tempo CPU
 - ▶ para processar informação do índice

Medida do Cosseno - Persin

Limitação do número de candidatos usando Persin, Zobel & Sacks_Davis:

- ▶ Lista invertida ordenada por $f_{d,t}$

$\langle 5; (1, 2), (2, 2), (3, 5), (4, 1), (5, 2) \rangle$

$\langle 5; (3, 5), (1, 2), (2, 2), (5, 2), (4, 1) \rangle$

outra representação:

$\langle 5; (5, 1 : 3), (2, 3 : 1, 2, 5), (1, 1 : 4) \rangle$

permite usar diferenças nos números dos documentos.

Medida do Cosseno - Persin

Como usar lista invertida ordenada por $f_{d,t}$:

- ▶ Normalmente idf é previamente calculado e usado para ordenar os termos da consulta de forma que termos mais raros são processados primeiro
- ▶ Entretanto, $f_{d,t}$ também é importante. Logo, uma boa solução é processar todas as listas em paralelo, um bloco de cada vez:
 - ▶ primeiro bloco da lista invertida de cada termo da consulta é lido e o produto $(1 + \log_e f_{d,t}) \times (\log_e 1 + \frac{N}{f_t})$ é calculado para cada bloco
 - ▶ todos os documentos de cada bloco recebem a mesma contribuição no acumulador (mesmo $f_{d,t}$)
 - ▶ processar na ordem decrescente de produto (maiores contribuições primeiro)

Medida do Cosseno - Persin

Como usar lista invertida ordenada por $f_{d,t}$:

Vantagens:

1. Mais precisa

- cada bloco é processado no momento certo
- se limite no número de acumuladores é usado, processamento pode usar *quit* ou *continue* quando limite é atingido, e agora sabemos que contribuições desprezadas são ainda menores

2. Menos processamento

- um item de qualquer lista com $f_{d,t} = 1$ dificilmente será processado

Medida do Cosseno - Persin

Como usar lista invertida ordenada por $f_{d,t}$:

Vantagens:

3. Menos transferência de disco
 - listas longas raramente são lidas na sua totalidade
4. Ranking melhor do que *quit* e *continue* com lista invertida ordenada pelo número do documento

Desvantagens:

1. Difícil processar consultas booleanas

Medida do Cosseno - Persin

Como filtrar documentos:

- ▶ Transição gradual da inclusão para a omissão de documentos
 1. termos da consulta são ordenados por *idf* decrescente (como no algoritmo básico)
 2. antes de processar cada termo t calcula dois parâmetros:
 S_{ins} e S_{add} , $S_{add} \leq S_{ins}$
 - S_{add} permite ignorar itens da lista invertida
 - S_{ins} permite ignorar alguns documentos
- ▶ Propostas anteriores decidiam processar ou descartar listas invertidas inteiras
- ▶ Persin permite decidir processar ou rejeitar documentos individuais

Medida do Cosseno - Persin: Algoritmo

1. Crie e inicialize uma estrutura de acumuladores (A).
2. Ordene termos da consulta por idf decrescente.
3. $S_{max} = 0$
4. **Para cada** termo t na consulta **faça**.
 - (a) Compute os valores para os filtros f_{ins} e f_{add} .
 - (b) Recupera lista invertida do disco.
 - (c) **Para cada** par $[d, f_{d,t}]$ na lista invertida **faça**.
 - i. **Se** $f_{d,t} > f_{ins}$ **então**
Crie A_d se necessário
 $A_d = A_d + w_{q,t}.w_{d,t} (sim_{q,d,t})$
 - ii. **Se não, Se** $f_{d,t} > f_{add}$ e A_d existe **então**
 $A_d = A_d + w_{q,t}.w_{d,t} (sim_{q,d,t})$
 - iii. **Se não**, Desvie para o próximo termo da consulta
 - iv. $S_{max} = \max(S_{max}, A_d)$.
5. Divida cada acumulador A_d por W_d .
6. Recupere os r maiores valores para os acumuladores e retorne os documentos correspondentes.

Medida do Cosseno - Persin

Como obter f_{ins} e f_{add}

- ▶ São função da similaridade parcial acumulada do documento mais relevante no momento (S_{max})

$$S_{ins} = C_{ins} \times S_{max},$$

$$S_{add} = C_{add} \times S_{max}$$

onde, $0 \leq C_{add} \leq C_{ins}$, não constantes

- ▶ $sim_{q,d,t} \leq w_{q,t} \times w_{d,t} = f_{d,t} \times idf \times f_{q,t} \times idf$

ou,

$$\frac{sim_{q,d,t}}{f_{q,t} \times idf^2} \leq f_{d,t}$$

(processa elemento como condição sobre $f_{d,t}$)

logo,

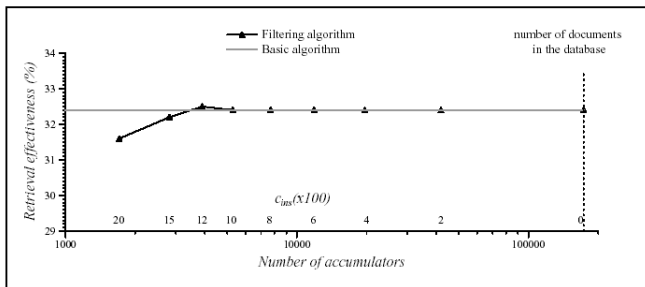
$$f_{ins} = \frac{C_{ins} \times S_{max}}{f_{q,t} \times idf^2}, \quad f_{add} = \frac{C_{add} \times S_{max}}{f_{q,t} \times idf^2}$$

(Valores de f_{ins} e f_{add} são constantes durante o processamento de uma lista invertida)

Ajuste da Constante de Inserção c_{ins}

Efetividade da recuperação: média da precisão nos 11 pontos de revocação (0%, 10%, ..., 100%).

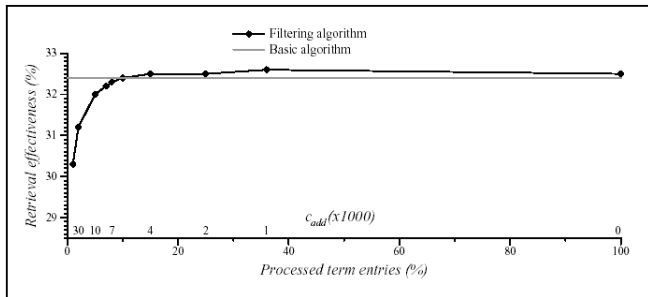
- Fixar $c_{add} = 0$ e aumentar c_{ins} até atingir o valor máximo que ainda oferece uma efetividade igual ou melhor que o algoritmo básico ($c_{ins} = c_{add} = 0$).



- O valor escolhido foi $c_{ins} = 0.12$, que usa apenas 4,000 acumuladores dentre o total de 173,000 acumuladores.

Ajuste da Constante de Adição c_{add}

- Fixar $c_{ins} = 0.12$ e medir a efetividade para valores crescentes de c_{add} .



- O valor escolhido foi $c_{add} = 0.007$, pois oferece uma efetividade igual ao do algoritmo básico processando apenas 10% das entradas das listas invertidas.

Medida do Cosseno - Persin

Observações sobre f_{ins} e f_{add}

- ▶ permitem transição suave da aceitação para rejeição de elementos nas listas invertidas
- ▶ fica progressivamente mais difícil adicionar ou atualizar os acumuladores
- ▶ para primeiros termos o valor de S_{max} é pequeno e o valor do idf é alto
- ▶ na medida que S_{max} cresce e idf diminui f_{ins} e f_{add} crescem até que, no limite, todos $f_{d,t} < f_{add}$ (e processar a lista a partir deste ponto não promove mudanças significativas nos acumuladores)

Medida do Cosseno - Persin

Nova versão:

1. Cálculo de f_{ins} e f_{add} , na versão original, ocorre uma vez para cada termo t e permanece igual durante o processamento da lista invertida do termo t
2. Na nova versão, o cálculo de f_{ins} e f_{add} ocorre toda vez que S_{max} é alterado.

Medida do Cosseno - Persin

1. Crie e inicialize uma estrutura de acumuladores (A).
2. $S_{max} = 0$
3. **Para cada** termo t na consulta **faça**.
 - (a) Compute os valores para os filtros f_{ins} e f_{add} .
 - (b) Recupere primeiro bloco de cada lista do disco.
 - (c) Calcule $sim_{q,d_s,t}$ e Insere $sim_{q,d_s,t}$ em H .
4. **Enquanto** houver blocos em H **faça**.
 - (a) Extrai bloco de H e insere próximo bloco da lista.
 - (b) Calcule $sim_{q,d_s,t}$.
 - i. **Se** $fd, t > f_{ins}$ **então**
Crie A_d se necessário
$$A_d = A_d + w_{q,t}.w_{d,t} (sim_{q,d,t})$$
 - ii. **Se não, Se** $fd, t > f_{add}$ e A_d existe **então**
$$A_d = A_d + w_{q,t}.w_{d,t} (sim_{q,d,t})$$
 - iii. **Se** $A_d > S_{max}$ **então**
$$S_{max} = \max(S_{max}, A_d).$$

Compute os valores para os filtros f_{ins} e f_{add} .
5. Divida cada acumulador A_d por W_d .
6. Recupere os r maiores valores para os acumuladores e retorne os documentos correspondentes.

Medida do Cosseno - Persin

Exemplo:

$$t_1 \rightarrow < 4; (4, 2 : 1, 3), (2, 1 : 2), (1, 1 : 5) >$$

$$t_2 \rightarrow < 1; (2 : 1, 5) >$$

$$t_3 \rightarrow < 2; (8, 1 : 7), (4, 1 : 6) >$$

$$w_{d,t} = tf \times idf = \log_2 f_{d,t} \times \log_2 \frac{N}{f_t}$$

$$N = 8$$

$$idf_{t_1} = \log_2 \frac{8}{4} = 1$$

$$idf_{t_2} = \log_2 \frac{8}{1} = 3$$

$$idf_{t_3} = \log_2 \frac{8}{2} = 2$$

$tf \times idf$:

$$Bloco_{1,t_1} = \log_2 4 \times 1 = 2$$

$$Bloco_{1,t_2} = \log_2 2 \times 3 = 3$$

$$Bloco_{1,t_3} = \log_2 8 \times 2 = 6$$

$$Bloco_{2,t_1} = \log_2 2 \times 1 = 1$$

$$Bloco_{2,t_3} = \log_2 4 \times 2 = 4$$

$$Bloco_{3,t_1} = \log_2 1 \times 2 = 0$$

Medida do Cosseno - Persin

Exemplo:

Ordem de processamento dos blocos:

1. Persin Original: (t_2 , t_3 e t_1)
(2,1:5), (8,1:7), (4,1:6), (4,2:1,3), (2,1:2), (1,1:5)
2. Nova versão:
(8,1:7), (4,1:6), (2,1:5), (4,2:1,3), (2,1:5)