CHARLES ORNELAS ALMEIDA

PODA ESTÁTICA PARA ÍNDICES INVERTIDOS BASEADA EM *LOGS*

Belo Horizonte
07 de julho de 2005

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

PODA ESTÁTICA PARA ÍNDICES INVERTIDOS BASEADA EM *LOGS*

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

CHARLES ORNELAS ALMEIDA

Belo Horizonte
07 de julho de 2005

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Poda Estática para Índices Invertidos Baseada em *Logs*

CHARLES ORNELAS ALMEIDA

Dissertação defendida e aprovada pela banca examinadora constituída por:

Ph. D. Nivio Ziviani – Orientador
Universidade Federal de Minas Gerais

Ph. D. Edleno Silva de Moura – Co-orientador
Universidade Federal do Amazonas

Ph. D. Berhier Ribeiro de Araújo Neto
Universidade Federal de Minas Gerais

Ph. D. Renato Antônio Celso Ferreira
Universidade Federal de Minas Gerais

Belo Horizonte, 07 de julho de 2005

Resumo

O crescimento inexorável do volume de documentos na World Wide Web coloca um grande desafio para as máquinas de busca, não apenas com relação a eficácia mas também com relação a eficiência de espaço e de tempo. Esta dissertação apresenta um novo método de compressão com perda (poda) para arquivos invertidos que considera o aspecto eficiência sem desconsiderar a eficácia. O método proposto é baseado na análise de *logs* de consultas passadas para obter uma grande redução no espaço ocupado pelo índice. O método pode ser utilizado em qualquer máquina de busca para melhorar sua eficiência em termos de tempo de processamento e espaço ocupado pelo índice, praticamente sem perdas na qualidade dos resultados da consulta. Experimentos utilizando uma máquina de busca real mostram que a técnica apresentada reduz os custos de armazenamento do índice em até 50% com relação ao índice sem compressão. Uma consequência dessa redução no tamanho do índice é que o tempo de processamento de uma consulta pode ser reduzido a aproximadamente 45% do tempo original, sem perda na precisão média. Considerando a qualidade do *ranking* produzido, o espaço ocupado pelo índice e o tempo de resposta a consultas, estudos comparativos com os dois melhores algoritmos de compressão de índices conhecidos na literatura mostram que o algoritmo proposto é bastante competitivo. Por exemplo, tanto a curva de similaridade entre os *rankings* quanto a precisão média das respostas do algoritmo proposto e o melhor algoritmo dentre os dois considerados na comparação se mantêm aproximadamente iguais para os diferentes níveis de poda. Quanto ao tempo de resposta o algoritmo proposto é mais rápido do que o melhor algoritmo dentre os dois considerados na comparação.

Palavras-chave: poda, compressão de índices, máquinas de busca, busca na Web, recuperação de informação.

Abstract

The relentless growth of the volume of documents in the World Wide Web poses a great challenge for search engines, not only regarding search effectiveness but also time and space efficiency. This thesis presents a new lossy compression (pruning) method for inverted files that addresses the efficiency issue without disconsidering the effectiveness. The proposed method is based on log analysis of past queries to obtain a great reduction in the size of the indices. The method can be used in any search engine to improve its efficiency in terms of query processing time and space occupied by the index, with nearly no cost in terms of the quality of query results. Experiments using a real search engine show that the technique presented reduces the storage costs of the index by up to 50% over traditional lossless compression methods. One consequence of this reduction in the size of the index is that the query processing time can be reduced to nearly 45% of the original time, with no loss in average precision. Considering the quality of the ranking, the space occupied by the index and the query answer time, comparative studies with the two best known static pruning methods show that the proposed algorithm is competitive. For example, both the similarity plot between the rankings and the average precision of the answers of the proposed algorithm and the best algorithm among the two considered are very close for all different levels of pruning. For the query answer time the proposed algorithm is faster than the best algorithm among the two considered.

Keywords: pruning, indexing, search engines, web search, information retrieval.

Agradecimentos

A Deus, acima de tudo, que me deu a salvação por meio de Jesus Cristo, meu Senhor. A Ele eu agradeço pelo amor, misericórdia, bondade e longanimidade imensuráveis, dispensados a mim a cada dia. A Ele eu louvo e dou graças pela presença consoladora e animadora, que me fez não desistir, mesmo em situações além de minhas próprias forças, me mostrando assim que ele é a minha força e que por mim mesmo, nada posso.

A meus pais Jorgelino e Maria Olinta, meus irmãos Sávio e Igor e minha namorada Talita, meus amores. A eles dedico todo o meu afeto por terem me acompanhado durante os momentos mais difíceis de minha vida. Agradeço pelo carinho, por compartilharem as alegrias e dores e principalmente por suas constantes orações.

A meu orientador, professor Nivio, pelo suporte sempre eficiente. A ele agradeço o incentivo constante e as muitas horas de dedicação, que foram determinantes para a execução desse trabalho.

A meu co-orientador, professor Edleno, que apesar da distância física sempre participou de perto deste trabalho por meio de inúmeras videoconferências.

Aos demais membros da banca examinadora, professores Berthier e Renato, por suas críticas e contribuições valorosas.

Aos amigos do Laboratório para Tratamento da Informação—os de agora, Álvaro, Anísio, Bruno, Claudine, Marco Cristo, Marco Modesto e Thierson e os de outras épocas, Cláudio, Cristiane, Dilú, Pável, Tânia e Wesley—por compartilhar dias, noites e finais de semana de trabalho.

Sumário

Capítulo 1

Introdução

1.1 Motivação

Máquinas de busca precisas e eficientes são um recurso valioso atualmente dadas as grandes proporções da Web. Em novembro de 2004, a máquina de busca Google[1] anunciou que sua base de dados possuía mais de 8 bilhões de documentos[2]. Outras máquinas de busca de escala global também têm bases de dados com bilhões de documentos. Apesar desses altos números, essas bases são apenas uma pequena amostra de um universo ainda maior (Raghavan e Garcia-Molina, 2001).

Para recuperar informação relevante para seus usuários de forma eficiente as máquinas de busca utilizam um tipo de índice chamado de arquivo invertido (Baeza-Yates e Ribeiro-Neto, 1999). O arquivo invertido associa cada termo que aparece na coleção com os documentos em que ele ocorre. Para tratar de buscas por frases ou buscas em proximidade, o arquivo invertido guarda também informação de onde cada termo aparece dentro dos documentos. O tamanho destes índices é proporcional ao tamanho da base de dados de texto armazenada pelo sistema de busca, portanto o índice toma dimensões muito grandes quando usados para busca em toda a Web.

Por outro lado, a proporção de acessos a sites provindos de máquinas de busca quase dobrou de 2002 para 2003[3]. Essa tendência é mundial, sendo que no Brasil a proporção aumentou ainda mais, indo de 8% para 19%. Embora o acesso direto ainda seja responsável pela maior parte do tráfego a um site, essa modalidade cresceu menos, cerca de 30%. É até mesmo comum ao público da Internet utilizar sua máquina de busca preferida como *bookmark*, ou seja, em vez de entrar, por exemplo, com o endereço *www.ufmg.br* no navegador, o internauta entra com a palavra-chave *ufmg* na máquina

[1] http://www.google.com

[2] http://www.google.com/googleblog/2004/11/googles-index-nearly-doubles.html

[3] http://www.websidestory.com/pressroom/pressreleases.html?id=181

de busca, que muitas vezes é sua página inicial, e acessa o primeiro link apresentado como resposta[4].

A combinação entre quantidade crescente de informação e de usuários faz com que haja uma demanda por estruturas de índice mais compactas e eficientes para máquinas de busca. Máquinas de busca na Web requerem um uso extremo de recursos computacionais, devendo ser capazes de responder a milhares de consultas por segundo. Uma única consulta pode ler centenas de megabytes de dados e consumir dezenas de bilhões de ciclos de CPU (Barroso et al., 2003). Além disso, a resposta às consultas deve apresentar os documentos em ordem de relevância para o usuário. Logo, uma utilização mais racional dos recursos computacionais envolvidos no mecanismo de busca é importante para que se possa atender rapidamente a todas as requisições.

Outra tendência que motiva a criação de índices mais compactos é o advento dos computadores de bolso. Atualmente eles já são capazes de armazenar coleções de documentos de tamanho moderado, de tal forma que a capacidade de se fazer busca indexada é desejável. No entanto, sua capacidade de armazenamento limitada impõe severas restrições de tamanho aos índices.

A compactação de índices (Ziviani et al., 2000; Carmel et al., 2001; de Moura et al., 2005) é um artifício que reduz não somente os custos de armazenamento como também melhora o tempo de resposta às consultas. O maior custo envolvido no processamento de uma consulta é o tempo gasto para ler do disco as porções de índice necessárias para o cômputo da resposta. Dessa forma, índices menores implicam menos leitura de disco, que por sua vez implicam tempos de resposta menores.

Esta dissertação propõe, implementa e avalia uma nova técnica de compactação de arquivos invertidos baseada em *logs*. Os *logs*[5] de uma máquina de busca podem ser uma boa fonte de informação para guiar a redução do tamanho dos índices. Uma forma de fazer a compactação do índice é eliminar entradas que não são de interesse para o público que utiliza a máquina de busca. Os *logs* podem ser usados para estimar quais são os interesses de pesquisa dos usuários, ajudando a determinar que informação pode ser descartada dos índices sem que os usuários percebam mudanças nos resultados obtidos. Experimentos apresentados com o método proposto indicam que é possível melhorar o desempenho de um sistema de busca reduzindo tanto os custos de armazenamento do índice como o os custos de processamento da consultas, sem perda significativa na qualidade da reposta.

[4]http://www.google.com/help/features.html#lucky

[5]Registros das consultas submetidas pelos usuários

1.2 Trabalhos Relacionados

Uma abordagem para a compactação de arquivos invertidos é a compressão sem perdas (de Moura et al., 2000; Ziviani et al., 2000). Essa técnica é chamada sem perda porque nenhuma informação é descartada do arquivo invertido. As técnicas de compressão utilizadas em arquivos invertidos partem do princípio de que números pequenos podem ser representados de forma mais compacta por meio de codificações específicas. Como a lista de documentos que contém um determinado termo da coleção é armazenada em ordem ascendente, ela pode ser considerada como uma seqüência de intervalos. Assim, o primeiro documento da lista, digamos 5, é representado normalmente, o segundo, digamos 11, é representado como 6, ou seja, 11 menos o anterior 5, e assim sucessivamente. Uma vez que o processamento da lista de ocorrências de uma palavra é normalmente feito de forma seqüencial, o número original do documento pode sempre ser recomputado pela soma dos intervalos. A lista de ocorrências composta agora de inteiros menores pode então ser representada em menos bits usando uma codificação adequada.

Outra abordagem, conhecida como poda (Persin et al., 1996; Carmel et al., 2001; de Moura et al., 2005), descarta determinadas entradas do arquivo invertido tornando-o menor, porém perdendo informação. Essas duas técnicas são complementares, sendo possível aplicar uma poda num arquivo invertido para selecionar as entradas mais importantes e em seguida comprimir o arquivo invertido podado. Possivelmente não se obtém a mesma taxa de compressão que seria obtida comprimindo um arquivo invertido não podado, porém o resultado da combinação das duas abordagens acaba sendo melhor que a aplicação de apenas uma delas.

Exemplos de métodos de poda são a remoção de *stopwords*[6] e Indexação Semântica Latente (LSI)[7] (Deerwester et al., 1990). Ambas as técnicas foram originalmente propostas com o objetivo de remover informação ruidosa do índice, eliminando termos que deterioram a precisão. No entanto, outro efeito causado por essas técnicas é a redução do tamanho do índice. O problema destas técnicas é que elas descartam listas de ocorrências inteiras de determinados termos. Assim, consultas contendo tais termos terão de ignorá-los ou retornar uma resposta vazia. Em ambos os casos, o usuário do sistema de busca não ficará satisfeito.

Um tipo de poda menos drástica e mais eficaz é a que elimina apenas as entradas menos importantes das listas de ocorrências. Um exemplo é a poda de *Persin* (Persin et al., 1996), que é aplicada no momento em que a consulta é processada. Normal-

[6]Remoção do arquivo invertido das entradas de termos muito freqüentes e de pouco valor semântico, como por exemplo, preposições, artigos, alguns pronomes, etc.

[7]Do inglês, Latent Semantic Indexing

mente os termos da pesquisa são processados seqüencialmente, começando do termo mais raro para o termo mais freqüente na coleção. Porém, o processamento de cada termo é terminado assim que determinada condição é atingida, mesmo que sua lista de ocorrências não tenha sido inteiramente lida e processada. Isso reduz o tempo de processamento das consultas, entretanto, as entradas das listas de ocorrências que não foram lidas ainda fazem do parte do índice, que não diminui de tamanho.

Enquanto o método de *Persin* propõe que a poda seja feita durante o processamento da consulta, o método de *Carmel* (Carmel et al., 2001) propõe que essa poda seja feita estaticamente, logo depois que o índice é construído. Assim como o método de *Persin*, somente as entradas menos importantes de cada lista de ocorrências são podadas, mas essa poda implica remover literalmente essas entradas do índice em vez de simplesmente não processá-las. Podar o índice estaticamente, além de reduzir o tempo de processamento das consultas reduz também o tamanho do índice.

Um problema que é comum tanto à poda de *Persin* quanto à poda de *Carmel* é que a poda é feita sem levar em conta o relacionamento entre os termos. Essa estratégia acaba deteriorando a qualidade das respostas, principalmente para consultas conjuntivas. Em consultas desse tipo é necessário que todos os termos procurados estejam presentes nos documentos da resposta, mas como a poda é feita tratando de cada termo individualmente, muitos documentos que seriam bons candidatos acabam sendo descartados. Por exemplo, suponhamos que o documento d seja um bom candidato para responder à consulta pelos termos t_a e t_b, mas ao podar a lista de ocorrências do termo t_a, o documento d foi descartado. Assim, mesmo que o documento d tenha sido preservado na lista de ocorrências do termo t_b, esse documento não será dado como resposta à consulta.

Um trabalho recente que leva em conta a relação entre os termos é o método de poda estática baseada em localidade, o *Lbpm* (de Moura et al., 2005). Esse método determina quais são as entradas individualmente mais importantes na lista de ocorrências dos termos, mas preserva também as entradas de termos que ocorrem próximos a estes dentro dos documentos. A principal diferença entre o método *Lbpm* e o método proposto nesta dissertação é que o *Lbpm* procura predizer quais termos serão pesquisados em conjunto utilizando o conteúdo dos documentos, enquanto o nosso método olha para as consultas já submetidas para fazer essa predição.

1.3 Contribuições

A principal contribuição desta dissertação é a apresentação de um novo método de compressão com perda (poda) para arquivos invertidos, baseado na análise de *logs*

de consultas passadas. O método proposto pode ser utilizado em qualquer máquina de busca para melhorar sua eficiência em termos de tempo de processamento e espaço ocupado pelo índice, praticamente sem perdas na qualidade dos resultados da consulta. Outras contribuições específicas do trabalho são:

- Proposição, descrição e implementação de um novo algoritmo de compressão com perda baseado na análise de *logs* (capítulo 3).
- Estudo comparativo do comportamento do algoritmo proposto com dois algoritmos apresentados na literatura, o algoritmo de *Carmel* (Carmel et al., 2001) e o algoritmo *Lbpm* (de Moura et al., 2005) (capítulo 4), considerando os seguintes aspectos:
  - qualidade do *ranking* produzido;
  - espaço ocupado pelos índices gerados pelos algoritmos;
  - tempo de resposta de cada algoritmo.
- Estudo dos principais algoritmos de poda conhecidos na literatura (capítulo 2), com modificação do algoritmo para poda dinâmica de *Persin* (Persin et al., 1996), para tratar consultas conjuntivas (seção 2.3.1), estudo do método de *Carmel* (Carmel et al., 2001) (seção 2.3.2), estudo do método *Lbpm* (de Moura et al., 2005) (seção 2.3.2) e estudo de medidas de similaridade de *rankings* para utilização nos experimentos (seção 2.4.2).

1.4 Organização da Dissertação

Esta dissertação está dividida em cinco capítulos, dos quais este é o primeiro. O capítulo 2 apresenta os conceitos básicos necessários para o entendimento do texto, bem como as principais medidas de relevância utilizadas nos experimentos realizados. O capítulo 3 apresenta o novo algoritmo de poda proposto. O capítulo 4 apresenta os resultados experimentais. O capítulo 5 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Conceitos Básicos

2.1 Arquivos Invertidos

O esquema de indexação comumente utilizado pelos sistemas de recuperação de informação é o *arquivo invertido* (Baeza-Yates e Ribeiro-Neto, 1999). Um arquivo invertido é composto por um *vocabulário* e um conjunto de *listas invertidas*. O vocabulário contém cada termo t presente na coleção de documentos e o número f_t de documentos contendo t. Existe uma lista invertida para cada t, constituída dos identificadores d dos documentos contendo o termo e da freqüência $f_{d,t}$ de t em d. Logo, cada entrada de uma lista invertida é um par $\langle d, f_{d,t} \rangle$ de valores.

2.2 Modelo Espaço Vetorial

O método de ordenação das respostas mais conhecido e utilizado nos sistemas de recuperação de informação é o *modelo espaço vetorial* (Salton e McGill, 1986). Nesse modelo, cada termo do vocabulário representa uma dimensão. Documentos e consultas são então mapeados em vetores multidimensionais neste espaço vetorial. Quanto maior é o valor do cosseno entre um vetor que representa uma consulta e um vetor que representa um documento, maior é a similaridade entre esta consulta e este documento. De forma prática, a similaridade entre o documento d e a consulta q é definida por

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

sendo W_d o tamanho do documento d e $sim_{q,d,t}$, a similaridade parcial entre q e d com relação ao termo t, definida por

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

em que $w_{x,t}$ é o peso de t no documento d ou na consulta q. Acumuladores A_d são usados para manter os somatórios parciais $\sum_t sim_{q,d,t}$ de cada documento. Os valores de W_d são previamente computados com a expressão

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

e armazenados em disco. Vários métodos para calcular $w_{x,t}$ são propostos (Witten et al., 1999). Dentre eles, adotamos

$$w_{q,t} = 1$$

e

$$w_{d,t} = \ln f_{d,t} \cdot \ln \frac{N}{f_t}$$

sendo N o número de documentos na coleção.

Uma forma de garantir que nenhum documento que contenha todos os termos da consulta seja descartado prematuramente é não utilizar poda. O algoritmo 2.1 é o método básico para processar consultas conjuntivas sem poda e com ordenação da resposta.

No algoritmo básico, $A(d).sim$ acumula as similaridades parciais entre d e q com relação a t, ao passo que $A(d).cont$ conta quantos dos termos da consulta ocorreram no documento d. Ao final do algoritmo, os documentos procurados são aqueles em que $A(d).cont = |Q|$, sendo $|Q|$ igual à quantidade de termos presente na consulta[1]. Como estamos interessados apenas nos k melhores documentos, recuperamos aqueles que possuem os maiores valores de similaridade $A(d).sim$ com a consulta.

```
para cada documento d da coleção faça
    A(d).sim ← 0;
    A(d).cont ← 0;
para cada termo t da consulta faça
    Recupere a lista invertida de t no disco;
    para cada entrada ⟨d, f_{d,t}⟩ na lista invertida faça
        A(d).cont ← A(d).cont + 1;
        A(d).sim ← A(d).sim + sim_{q,d,t};
Divida cada acumulador A(d).sim por W_d;
Identifique os k acumuladores com maior valor A(d).sim e A(d).cont = |Q|;
Recupere os k documentos correspondentes;
```

Algoritmo 2.1: Algoritmo básico para consultas conjuntivas

[1]Assumindo que todos os termos de uma consulta são distintos.

O algoritmo básico é uma boa alternativa no que diz respeito à qualidade da resposta. Se partirmos do pressuposto de que a ordenação da resposta dada pela função de similaridade utilizada é boa, então a ordenação dada pelo algoritmo básico também será boa. Apesar de ser eficaz quanto à qualidade da resposta, o algoritmo básico possui um grande problema de eficiência. Nesse algoritmo, as listas invertidas de todos os termos da consulta têm que ser inteiramente processadas. No caso de máquinas de busca, em que a coleção de documentos é composta de bilhões de páginas Web, as listas invertidas podem ser muito grandes. Isso faz com que o desempenho do algoritmo básico, em termos de eficiência, seja muito ruim.

2.3 Métodos de Poda

Técnicas de poda conhecidas na literatura (Persin et al., 1996; Carmel et al., 2001) podem ser muito bem aplicadas quando não há a necessidade de que os documentos da resposta contenham todos os termos da consulta. Normalmente esses mecanismos funcionam descartando alguns pares $\langle d, f_{d,t} \rangle$ das listas invertidas durante o processamento de uma consulta (Persin et al., 1996) ou previamente, como é o caso da poda estática (Carmel et al., 2001). No caso de consultas disjuntivas, a eliminação dessas entradas não significa que o documento em questão está descartado da resposta pois outro par $\langle d, f_{d,t} \rangle$ referente a outro termo nesse mesmo documento pode acabar incluindo-o nas primeiras posições do conjunto resposta. Com isso, a poda acaba não modificando muito o resultado das consultas, mas garante um ganho significativo de desempenho.

O mesmo princípio poderia ser usado para processar consultas conjuntivas, porém a decisão por descartar uma entrada de uma lista invertida pode ser extremamente arriscada. Nesse caso, descartar um par $\langle d, f_{d,t} \rangle$ equivale a dizer que o termo não ocorreu no documento em questão. Essa pode ser uma escolha prematura mesmo que a similaridade parcial relativa à entrada descartada seja baixa, pois ao processar as outras listas invertidas, podemos descobrir que todos os outros termos da consulta ocorreram naquele documento com altas similaridades parciais.

2.3.1 Poda Dinâmica

No método de poda dinâmica, as listas invertidas são ordenadas pelo valor de $f_{d,t}$ e a similaridade parcial relativa a um par $\langle d, f_{d,t} \rangle$ é tratada de forma diferenciada, dependendo da relação entre a freqüência $f_{d,t}$ e duas freqüências limiares, uma de inserção f_{ins} e outra de adição f_{add}. Para o caso $f_{d,t} \geq f_{ins}$, cria-se uma entrada na estrutura de acumuladores, caso ela ainda não exista, e a similaridade parcial é

computada, atualizando o acumulador $A(d)$. Quando $f_{ins} > f_{d,t} \geq f_{add}$, a similaridade parcial só atualiza o acumulador $A(d)$, caso ele já exista na estrutura de acumuladores. A partir do momento em que temos $f_{add} > f_{q,t}$ todo o restante da lista invertida pode ser abandonado, uma vez que ela está ordenada por $f_{d,t}$. O algoritmo 2.2 apresenta a estratégia de poda de *Persin*.

```
Ordene o termos da consulta por f_t decrescente;
S_max ← 0;
para cada termo t da consulta faça
    Compute os valores de f_ins e f_add;
    Recupere a lista invertida de t no disco;
    para cada entrada ⟨d, f_d,t⟩ da lista invertida faça
        se f_d,t ≥ f_ins então
            Crie um acumulador A(d) se necessário;
            A(d).sim ← A(d).sim + sim_q,d,t;
        senão se f_d,t ≥ f_add E A(d) já foi criado então
            A(d).sim ← A(d).sim + sim_q,d,t;
        termine o processamento da lista invertida atual;
        S_max ← max(S_max, A(d).sim);
Divida cada acumulador A(d).sim por W_d;
Identifique os k acumuladores com maior valor A(d).sim;
Recupere os k documentos correspondentes;
```

Algoritmo 2.2: Estratégia de poda de *Persin*

A poda de *Persin* define os valores das freqüências limiares de inserção por

$$f_{ins} = \frac{c_{ins} \cdot S_{max}}{f_{q,t} \cdot w_t^2}$$

e adição por

$$f_{add} = \frac{c_{add} \cdot S_{max}}{f_{q,t} \cdot w_t^2}.$$

Nessa expressão, S_{max} corresponde à maior similaridade computada durante o processamento das listas invertidas. As constantes c_{ins} e c_{add} são utilizadas para controlar o nível da poda. Elas devem ser ajustadas de acordo com a coleção de documentos (Persin et al., 1996), de forma a se obter o melhor balanceamento entre economia de recursos computacionais e qualidade da resposta.

A adaptação mais ingênua da poda de *Persin* para responder a consultas conjuntivas é direta e pode ser vista no algoritmo 2.3. Nesse algoritmo, $A(d).cont$ é utilizado como no algoritmo básico para contar quantos dos termos da consulta ocorreram em cada documento. As similaridades parciais entre o documento e a consulta são somadas no campo $A(d).sim$ do acumulador. Observe ainda que quando algum par $\langle d, f_{d,t} \rangle$

do i-ésimo termo da consulta referencia um acumulador $A(d)$, sua similaridade só é computada e somada se todos os termos anteriores também ocorreram no documento d, nesse caso $A(d).cont = i$.

```
Ordene os termos da consulta por f_t decrescente;
S_max ← 0;
para cada termo t_i da consulta (0 ≤ i ≤ |Q|) faça
    Compute os valores de f_ins e f_add;
    Recupere a lista invertida de t_i do disco;
    para cada entrada ⟨d, f_d,t⟩ da lista invertida faça
        se i ≠ 0 E (A(d) não está no conjunto de acumuladores ou A(d).cont < i)
        então
            Continue;
        se f_d,t ≥ f_ins então
            Crie um acumulador A(d) se necessário;
            A(d).cont ← A(d).cont + 1;
            A(d).sim ← A(d).sim + sim_q,d,t;
        senão se f_d,t ≥ f_add E A(d) já foi criado então
            A(d).cont ← A(d).cont + 1;
            A(d).sim ← A(d).sim + sim_q,d,t;
        termine o processamento da lista invertida atual;
        S_max ← max(S_max, A(d));
Divida cada acumulador A(d).sim por W_d; Identifique os k acumuladores com
maior valor A(d).sim e A(d).cont = |Q|; Recupere os k documentos
correspondentes;
```

Algoritmo 2.3: Adaptação da estratégia de poda de *Persin* para consultas conjuntivas

Apesar do algoritmo de *Persin* ser facilmente adaptado para tratar consultas conjuntivas, existem muitos pontos que devem ser investigados. Como não existe nenhum trabalho que estudou esta possibilidade, é necessário verificar se é possível encontrar as constantes c_{ins} e c_{add} que balanceiam economia no processamento e qualidade de resposta para o caso de consultas conjuntivas. Esse estudo foi feito neste trabalho, demonstrando que a adaptação desse algoritmo gera resultados muito ruins, mesmo para podas muito pequenas, conforme pode ser visto no capítulo 4.

2.3.2 Poda Estática

Existem dois métodos de poda estática propostos na literatura, o método de *Carmel* (Carmel et al., 2001) e o método *Lbpm*[2] (de Moura et al., 2005). O ponto de

[2]Do inglês, *locality based pruning method*

partida para esta dissertação foi o método de *Carmel*, mas ao longo do desenvolvimento deste trabalho surgiu o *Lbpm*, que apresentou resultados melhores que o método de *Carmel* em termos de precisão de resultados. O método de poda estática baseado em *log*, que estamos propondo, é comparado no capítulo 4 com estes dois métodos, que são descritos a seguir.

2.3.2.1 Poda Estática de *Carmel*

A idéia do método de *Carmel* é usar o *ranking* da máquina de busca para computar a importância de cada entrada nas listas invertidas e determinar quais entradas resultam em uma contribuição baixa o suficiente para serem removidas sem prejuízo da resposta. Para isso, submetemos individualmente cada termo do vocabulário da coleção como uma consulta à maquina de busca. O resultado disso é que para cada termo t temos a lista resultante $R(t)$ de documentos relacionados a t em ordem decrescente de importância, segundo os critérios de *ranking* da máquina de busca.

A lista resultante $R(t)$ pode ser encarada como um *ranking* para os documentos, uma vez que quanto maior for o *ranking* do documento d em $R(t)$, maior é a chance de que d apareça nos resultados de uma consulta contendo o termo t. A poda de *Carmel* consiste em tomar apenas os documentos que aparecem na parte do topo de $R(t)$ como sendo a nova lista de documentos relevantes para o termo t. Cada entrada da lista invertida que representa a ocorrência de um termo t em um documento d é removida do índice se d não está presente no topo de $R(t)$.

A quantidade de entradas que compõem o topo de $R(t)$ de cada termo é determinada por um parâmetro δ. Este parâmetro apenas preserva entradas que correspondem a documentos em $R(t)$ cujo *ranking*, dado pela máquina de busca, é pelo menos δ vezes o maior *ranking* de todos os documentos em $R(t)$. O resultado é uma nova lista chamada $R_\delta(t)$. Por exemplo, se $\delta = 0,7$, cada documento com *ranking* maior ou igual a 70% do maior *ranking* estará em $R_\delta(t)$. O valor de δ, entre 0 e 1, é determinado experimentalmente. Um bom valor para δ reduzirá o tamanho do índice e conseqüentemente o tempo de resposta das consultas, mas deverá preservar um número de elementos que seja suficiente para dar uma boa aproximação dos *rankings* das respostas de topo de grande parte das consultas submetidas ao sistema.

O algoritmo 2.4 descreve como podar um arquivo invertido usando a poda estática de *Carmel*. O algoritmo toma como entrada um arquivo invertido e o parâmetro δ, gerando como saída um arquivo invertido podado. Assim como nos algoritmos anteriores, $A(d)$ é o valor acumulado das similaridades parciais entre o documento d e a consulta, mas para facilitar o entendimento podemos vê-lo como o *ranking* de d em relação à consulta.

```
para cada termo t no arquivo invertido faça
    Recupere a lista invertida de t do disco;
    para cada entrada ⟨d, f_{d,t}⟩ na lista invertida faça
        A(d) ← ranking de t em relação a d;
    τ_t ← δ max(A(d));
    para cada acumulador A(d) faça
        se A(d) > τ_t então
            Insira ⟨d, f_{d,t}⟩ em R_δ(t);
Escreva R_δ(t) no disco;
```

Algoritmo 2.4: Algoritmo de *Carmel* para poda estática

Nossos experimentos indicam que este método é de fato útil para consultas disjuntivas, mas seus ganhos não são tão significativos em um cenário onde a maioria das consultas são conjuntivas, como de fato acontece nas máquinas de busca da Web.

2.3.2.2 Poda Estática Baseada em Localidade (*Lbpm*)

Um ponto fraco do método de *Carmel* é seu desempenho em termos de qualidade das respostas para consultas conjuntivas. Consultas desse tipo, bem como consultas do tipo frase, requerem que entradas de determinados documentos sejam preservadas em listas invertidas de diferentes termos. Essa condição não é satisfeita pelo método de *Carmel*, que poda as lista invertidas de cada termo individualmente. Esse problema é descrito de maneira mais detalhada no capítulo 3.

O *Lbpm* é uma variação do método de *Carmel* que tenta predizer quais conjuntos de termos ocorrem juntos nas consultas dos usuários e usa essa informação para preservar documentos em comum nas listas invertidas destes termos.

O primeiro passo do *Lbpm* consiste em determinar quais ocorrências de termos são individualmente importantes para cada documento. Esse passo pode ser feito por meio da poda de *Carmel*. Em seguida, essa informação é utilizada para determinar quais são as sentenças significativas de cada documento, que são aquelas em que ocorrem as entradas selecionadas pela poda de *Carmel*. Essas sentenças são então ordenadas e selecionadas para compor uma coleção de documentos menor que a original, composta apenas das sentenças significativas selecionadas como mais importantes. Por fim, a nova coleção é indexada.

A seleção das sentenças significativas mais importantes de cada documento é feita por meio do algoritmo 2.5. Seja $T(d)$ a lista dos termos significativos para o documento d. Esta lista é obtida "desinvertendo" as listas invertidas obtidas pela poda de *Carmel*. Seja $S(d)$ a lista das sentenças significativas do documento d. Considerando que uma função $comum(S(d), T(d))$ retorna as sentenças em $S(d)$ que têm o maior número de

termos em comum com $T(d)$, o algoritmo utiliza essa função para ordenar as sentenças significativas.

O algoritmo 2.5 seleciona sentenças até que não existam mais sentenças significativas ou até que o tamanho do novo documento, em número de termos, atinja a porcentagem p do tamanho do documento original. O algoritmo faz uma cópia de $T(d)$ em $T'(d)$ e executa um laço, que a cada passo, adiciona em $S'(d)$ a sentença S de $S(d)$ que possui o maior número de termos em comum com $T'(d)$. O tamanho de $S'(d)$ é incrementado do tamanho da sentença S selecionada. Em seguida, S é removida de $S(d)$ e os termos em S são removidos de $T'(d)$, para que a seleção de cada sentença de $S'(d)$ seja baseada em diferentes conjuntos de termos. Caso as sentenças já selecionadas cubram todos os termos em $T(d)$, $T'(d)$ fica vazio. Quando isso ocorre, $T(d)$ é novamente copiado em $T'(d)$ iniciando assim uma nova rodada de escolhas baseada em todos os termos significativos para o documento d.

$p \leftarrow$ porcentagem final desejada;
$T(d) \leftarrow$ conjunto dos termo significativos para d;
$S(d) \leftarrow$ conjunto das sentenças significativas para d;
$T'(d) \leftarrow T(d)$;
repita
  $S_{comum} \leftarrow comum(S(d), T'(d))$;
  $S'(d) \leftarrow S'(d) \cup S_{comum}$;
  $tamanho \leftarrow tamanho + |S_{comum}|$;
  $T'(d) \leftarrow T'(d) - \{x | x \in S_{comum} \wedge x \in T'(d)\}$;
  $S(d) \leftarrow S(d) - S_{comum}$;
  se $T'(d) = \emptyset$ **então**
    $T'(d) \leftarrow T(d)$;
até ($tamanho \geq p|d|$) Ou $S(d) = \emptyset$;
Retorne $S'(d)$;

Algoritmo 2.5: Algoritmo *Lbpm* para seleção das sentenças significativas

Ao preservar as sentenças com o maior número de termos significativos, o *Lbpm* consegue obter uma qualidade de respostas para consultas conjuntivas melhor que a obtida pelo método de *Carmel*, conforme pode ser visto no capítulo 4.

2.4 Avaliação de Resultados

Para qualquer sistema de recuperação de informação o objetivo a ser alcançado é retornar documentos relevantes às consultas submetidas e no menor tempo possível. Costumamos chamar estes dois aspectos respectivamente de eficácia e eficiência.

O uso de técnicas de poda tem por objetivo melhorar a eficiência do processo sem, contudo, prejudicar a eficácia. Isto é, um bom mecanismo de poda é aquele que diminui o tempo de resposta às consultas de uma forma geral, mas que continue a retornar documentos tão relevantes quanto os que seriam retornados sem o uso da poda.

Enquanto uma melhora na eficiência pode ser facilmente obtida, o problema de manter a eficácia é mais difícil. No entanto, não é incomum que uma poda bem feita consiga melhorar ambos os aspectos. Isto pode parecer pouco natural, mas é entendido quando lembramos que bases de dados tão populares e acessíveis como a Web estão sujeitas a armazenar uma enorme quantidade de informação pobre ou mesmo inútil. Ao podar esse tipo de documento estamos diminuindo o tamanho da base, e com isso ganhando em eficiência, ao mesmo tempo em que melhoramos a qualidade da coleção, e com isso ganhamos em eficácia.

A avaliação da eficiência do sistema é feita sem maiores dificuldades usando métricas como tempo médio de resposta das consultas, avaliando tempo de processamento, quantidade de memória e disco utilizada, dentre outros. Já a avaliação da eficácia é mais difícil, pois dizer se um documento é relevante para uma dada consulta é uma tarefa subjetiva. A seguir apresentamos dois métodos que utilizamos para medir a qualidade dos resultados obtidos depois de feita a poda.

2.4.1 Precisão e Revocação

Uma maneira de verificar a relevância dos documentos recuperados é o cálculo de precisão e revocação (Baeza-Yates e Ribeiro-Neto, 1999; Witten et al., 1999). Essa é a medida padrão utilizada para avaliar a qualidade da respostas de um sistema de recuperação de informação, sendo largamente utilizada.

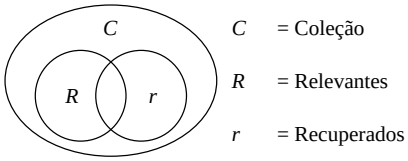

Figura 2.1: Precisão–Revocação

Na figura 2.1, considere que C seja a coleção de todos os documentos da base e que para uma dada consulta, R seja o conjunto dos documentos relevantes e r seja o conjunto de documentos recuperados. A precisão se define por

$$P = \frac{|R \cap r|}{|r|}$$

e a revocação por

$$R = \frac{|R \cap r|}{|R|}.$$

Intuitivamente, a precisão nos dá uma medida de quão apurado foi o resultado, pois nos diz quantos dos documentos recuperados são realmente relevantes, enquanto a revocação nos dá uma medida da cobertura do resultado, pois nos diz quantos dos documentos relevantes foram recuperados. Normalmente essas duas medidas são apresentadas como um gráfico de precisão em função da revocação ou como a precisão média dos dez pontos de revocação (Witten et al., 1999). Nesta dissertação utilizamos a precisão média.

Para utilizar essa métrica é necessário que se tenha avaliado de antemão quais são os documentos relevantes para um conjunto de consultas. Esse é um processo trabalhoso e demorado uma vez que o julgamento da relevância ou não de um determinado documento deve ser feito por uma pessoa. Por isso, muitas vezes as medidas de precisão e revocação não são viáveis. Assim, outro método de avaliação de qualidade que tem sido utilizado é a medida de similaridade de *rankings*.

2.4.2 Similaridade de *Rankings*

A medida de similaridade consiste em comparar dois *rankings* para verificar quão parecidos eles são um do outro. No nosso caso, o objetivo da medida de similaridade é verificar quão próximos são os *rankings* dados pelo índice podado em relação aos *rankings* dados pelo índice original.

A medida de similaridade que utilizaremos é baseada no método tau de Kendall (Kendall e Gibbons, 1990), que é utilizado para comparar a similaridade entre permutações de conjuntos. O método original associa uma penalidade $S(i, j) = 1$ para cada par i, j de itens distintos do conjunto em que i aparece antes de j em uma permutação e j aparece antes de i na outra permutação. A soma dessas penalidades para todos os pares i, j dá uma penalidade final entre 0 (quando as permutações são idênticas) e $n(n-2)/2$ (quando uma permutação é o inverso da outra).

Quando comparamos os *rankings* gerados pelo índice podado com o índice original, pode acontecer de um elemento estar presente em um *ranking* e não estar presente no outro. Dessa forma, utilizamos uma modificação do método tau de Kendall (Fagin et al., 2003). Nessa versão modificada, comparamos o topo das duas listas, que é onde devem aparecer os documentos mais relevantes. Para cada par i, j, em que i e/ou j aparece no topo, isto é, entre os k primeiros elementos de pelo menos uma das listas, associamos as seguintes penalidades $S(i, j)$:

- Caso 1: se i e j aparecem no topo de ambas as listas, então $S(i, j)$ é definido como

no método original. A saber, se i e j estão na mesma ordem, então $S(i,j) = 0$, do contrário $S(i,j) = 1$;

- Caso 2: se i e j aparecem no topo de uma lista (digamos, na lista 1) e apenas j ou i aparece no topo da outra lista (lista 2), e se i está à frente de j na lista 1, então $S(i,j) = 0$, do contrário $S(i,j) = 1$. Intuitivamente, sabemos que i está à frente de j na lista 2, uma vez que i aparece no topo da lista 2 enquanto j não aparece.

- Caso 3: se apenas i aparece no topo de uma lista e apenas j aparece no topo da outra lista, então $S(i,j) = 1$. Intuitivamente, sabemos que i está à frente de j na primeira lista e que j está à frente de i na outra lista.

- Caso 4: se i e j aparecem no topo de uma das listas mas nem i nem j aparecem no topo da outra lista, então $S(i,j) = 1/2$. Intuitivamente, não temos informação suficiente para determinar se i e j aparecem na mesma ordem nas duas listas, então associamos uma penalidade neutra de $1/2$.

A soma final das penalidades será então um valor entre 0 (quando os dois *rankings* são idênticos) e $k(3k-1)/2$ (quando os dois *rankings* são disjuntos). Obtemos o valor normalizado x', entre 0 e 1, fazendo $x' = 1 - 2x/k(3k-1)$. Assim, o valor 1 ocorre quando as duas listas são idênticas e o valor 0 ocorre quando as listas são disjuntas.

Capítulo 3

Algoritmo de Poda Proposto

As técnicas de poda de *Persin* e de *Carmel* eliminam entradas das listas invertidas tratando cada termo de forma isolada. Essa abordagem funciona bem quando se trabalha com consultas disjuntivas, no entanto, não é boa para responder a consultas conjuntivas. Por exemplo, será que o conjunto de documentos que melhor respondem à consulta "trabalho" bem como os que melhor respondem à consulta "infantil" inclui os documentos que melhor respondem à consulta conjuntiva contendo os termos "trabalho" e "infantil"?

A figura 3.1 ilustra bem a situação decrita acima. Ela mostra o mesmo exemplo apresentado no artigo que propõe o *Lbpm* (de Moura et al., 2005). A lista invertida do termo a era originalmente composta pelos documentos 3, 15, 1, 8, 14, 2, 31, 4, e 13, e a lista do termo b, pelos documentos 2, 1, 7, 9, 43, 25, 16, 3 e 21. As áreas sombreadas correspondem aos documentos que foram eliminados pela poda. Nesse cenário, uma consulta conjuntiva contendo os termo a e b conterá apenas o documento 1 como resposta. Entretanto, os documentos 2 e 3 fariam parte da resposta se a consulta tivesse sido feita sobre o índice original. Além disso, estes dois documentos são possivelmente importantes, já que eles aparecem no topo das lista de a e de b individualmente. Esse exemplo mostra que uma poda que não leve em conta o relacionamento entre os termos pode descartar entradas importantes.

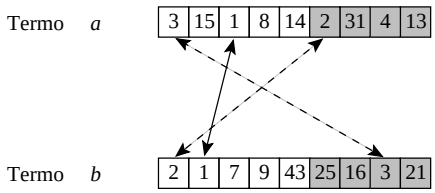

Figura 3.1: Informação perdida ao podar as listas invertidas dos termos individualmente

Dada essa deficiência, nossa proposta é uma variação no método de *Carmel* que procura predizer quais termos apareceram em conjunto nas consultas feitas pelos usuários. Para descobrir os relacionamentos entre os termos nosso método faz uso de *logs* de consultas passadas da máquina de busca.

3.1 Extração de Informação dos *Logs*

Quando as pessoas usam máquinas de busca para recuperar algum tipo de informação, suas consultas e os sites clicados são armazenados em *logs*. Tais dados trazem informações valiosas: as consultas podem indicar relações entre termos e os sites clicados indicam que, em uma primeira análise, o usuário da máquina de busca considerou tal site possivelmente relevante em relação a sua consulta. Nesse sentido, os *logs* podem ser vistos como um registro do conhecimento e da inteligência de milhões de pessoas.

No caso específico das consultas, quando um usuário de uma máquina de busca utiliza um conjunto de termos para fazer uma pesquisa, podemos inferir que os termos utilizados possuem algum grau de relação entre si. Ou melhor, sabemos que o usuário espera que existam documentos na Web que contenham os termos pesquisados. Logo, ao fazer uma poda é importante que tais documentos sejam preservados nas listas invertidas dos termos envolvidos naquela consulta. É justamente nessa observação que se baseia o nosso método.

Analisando o *log* estabelecemos que há um relacionamento entre dois termos caso eles ocorram em uma mesma consulta. O processamento do *log* resulta em um arquivo em que, para cada termo t, temos uma lista dos termos que possuem relacionamento com t. Definimos esse conjunto de termos relacionados a t como $R(t)$. Por exemplo, digamos que o termo a co-ocorra com os termos b, c e d em diferentes consultas. Assim, depois de processar o *log*, a lista de relacionamentos de a seria $R(a) = \{b, c, d\}$ e, além disso, ambas as listas de relacionamento dos termos b, c e d conteriam uma entrada para o termo a.

Antes de gerar as listas de relacionamento de cada termo, fazemos um processamento prévio no *log* a fim de diminuir a quantidade de relacionamentos que vamos levar em consideração. Conforme mostramos na próxima seção, estas listas de relacionamentos servem para expandir a poda de *Carmel* a fim de contemplar as consultas conjuntivas. O problema de considerar todos os relacionementos do *log* é que a expansão acaba gerando listas invertidas tão grandes quanto as originais, logo, selecionamos apenas os relacionamentos mais freqüentes.

A seleção dos relacionamentos mais freqüentes do *log* é feita utilizando conceitos da área de mineração de dados (Agrawal et al., 1993). Mais especificamente, selecionamos

os conjuntos freqüentes maximais contendo dois ou mais termos usando um suporte de 1%. É importante salientar que a ferramenta utilizada em nossos experimentos[1] utiliza um conceito de suporte um pouco diferente (Borgelt e Kruse, 2002) do conceito original (Agrawal et al., 1993). Também é importante dizer que o valor de suporte foi escolhido de maneira que a quantidade de pares de termos presentes nos conjuntos fosse aproximadamente igual a 1/5 dos pares presentes no *log* do mês de janeiro, que foi o mês utilizado para o processo de poda. Esse valor foi escolhido empiricamente.

3.2 O Processo de Poda

Uma vez obtidas as relações entre os termos o próximo passo é fazer a poda das listas invertidas preservando as relações encontradas no *log*. Para cada termo t, nosso objetivo é determinar quais são os documentos mais importantes que contêm t, quer seja individualmente, quer seja acompanhado dos termos aos quais t se relaciona.

Esse processo é composto de duas etapas executadas para cada termo t:

1. Encontrar as melhores entradas da lista invertida de t individualmente;
2. Expandir a lista invertida resultante de t, acrescentando as entradas que são importantes para os termos aos quais t se relaciona.

No nosso método, a etapa 1 do processo de poda é obtida por meio da proposta de *Carmel*. Chamemos de $Carmel(t)$ a função que retorna a lista invertida podada do termo t por meio da poda de *Carmel* e chamemos de $Original(t)$ a função que retorna a lista invertida original de t.

Seja $R(t) = \{r_1, ..., r_n\}$ o conjunto dos termos que se relacionam com t, obtido na fase de processamento do *log*. O resultado da etapa 2 é obtido pela função $Expandido(t)$ definida por:

$$Expandido(t) = [\bigcup_{i=1}^{n} Carmel(r_i) \cup Carmel(t)] \cap Original(t).$$

Tomando o exemplo anterior, em que o termo a se relaciona com os termos b, c e d, teríamos:

$$Expandido(a) = [Carmel(b) \cup Carmel(c) \cup Carmel(d) \cup Carmel(a)] \cap Original(a).$$

Dessa forma, ainda que determinada entrada do termo a não esteja entre as melhores individualmente ela pode ser preservada caso apareça entre as melhores entradas de algum dos termos que possuem relação com a.

[1]Disponível em: http://fuzzy.cs.uni-magdeburg.de/ borgelt/doc/apriori/apriori.html#download

O problema da perda de informação observado no exemplo da figura 3.1 é solucionado usando essa técnica. Como os termos a e b se relacionam, o documento 2 que não estava entre os melhores do termo a passa a ser preservado, uma vez que ele está entre os melhores documentos do termo b. Similarmente, o documento 3 passa a ser preservado na lista invertida do termo b. A figura 3.2 mostra como ficariam as listas invertidas desses termos após as duas etapas do processo de poda.

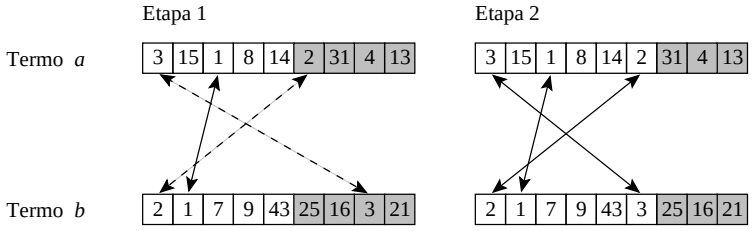

Figura 3.2: Informação preservada ao levar em conta o relacionamento entre os termo

3.3 Descrição do Algoritmo

O algoritmo 3.1 mostra como encontramos os relacionamentos entre os termos. Lembramos que o primeiro passo da poda consiste em selecionar os conjuntos de termos freqüentes maximais obedecendo um determinado suporte, conforme já foi dito anteriormente. Então criamos uma uma lista para cada termo do vocabulário da coleção. Em seguida, para cada conjunto freqüente maximal, tomamos seus termos aos pares $\langle t_i, t_j \rangle$, inserindo t_i na lista de t_j, caso t_i ainda não esteja presente e vice-versa. Note que são descartados os termos que não fazem parte do vocabulário da coleção, logo, somente processamos conjuntos que possuem pelo menos dois termos existentes no vocabulário.

Por fim, gravamos em disco a lista dos termos r aos quais cada termo t se relaciona. O resultado é um arquivo parecido a um índice invertido, porém, em vez haver, para cada termo t, a lista dos documentos em que t ocorre, temos a lista de termos r que ocorreram nas mesmas consultas que t.

O algoritmo 3.2 mostra como é feita a expansão das listas invertidas podadas na etapa 1. O processo consiste em excluir dentre os documentos da lista invertida original do termo t aqueles documentos que não ocorrem nas listas invertidas, geradas pelo método de *Carmel*, do termo t e de seus termos relacionados r.

```
para cada termo t do vocabulário faça
    Crie uma lista de relacionamentos R(t) vazia;
para cada conjunto freqüente maximal m faça
    para cada par de termos t_i, t_j de m faça
        se t_i e t_j pertencem ao vocabulário então
            Insira t_i em R(t_j);
            Insira t_j em R(t_i);
para cada termo t do vocabulário faça
    Escreva R(t) em disco;
```

Algoritmo 3.1: Geração das listas de relacionamento dos termos

```
Execute a poda de Carmel; // algoritmo 2.4
Gere as listas de relacionamentos; // algoritmo 3.1
para cada documento d da coleção faça
    Crie um acumulador A(d) = 0;
para cada termo t do vocabulário faça
    Recupere a lista invertida sem poda de t;
    para cada entrada ⟨d, f_{d,t}⟩ da lista invertida sem poda faça
        A(d) ← A(d) + 1;
    Recupere a lista invertida podada de t; // poda de Carmel
    para cada entrada ⟨d, f_{d,t}⟩ da lista invertida podada faça
        A(d) ← A(d) + 1;
    Recupere a lista de relacionamentos de t;
    para cada termo relacionado r faça
        Recupere a lista invertida podada de r; // poda de Carmel
        para cada entrada ⟨d, f_{d,r}⟩ da lista invertida podada faça
            A(d) ← A(d) + 1;
    Crie uma nova lista invertida L(t);
    para cada documento d faça
        se A(d) > 1 então
            Insira ⟨d, f_{d,t}⟩ em L(t);
para cada termo t do vocabulário faça
    Escreva L(t) em disco;
```

Algoritmo 3.2: Expansão da poda estática gerada pelo método de *Carmel*

Capítulo 4

Resultados Experimentais

Neste capítulo são descritos os experimentos realizados com o novo método de poda proposto e com os demais métodos existentes na literatura, visando a avaliação da idéia de se utilizar *logs* de consultas passadas como fonte de informação para a poda. Comparamos os métodos em termos de tempo de processamento, precisão dos resultados e diferença entre a resposta fornecida pelo sistema antes da poda e depois da poda. A seção 4.2 apresenta a comparação da diferença das respostas. A seção 4.3 apresenta a comparação da precisão das respostas. A seção 4.4 apresenta a comparaçãao dos tempos de resposta dos três métodos de poda estática avaliados.

4.1 Ambiente de Execução e Coleções de Teste

Todos os experimentos de medida de desempenho foram realizados em uma máquina com 1 gigabytes de memória principal, disco Serial ATA, processador Intel Pentium 4 de 2.4 GHz e sistema operacional Linux com kernel versão 2.6.

Os experimentos foram realizados sobre a coleção de documentos WBR-99. A coleção WBR-99[1] é composta de um conjunto de documentos coletados da Web brasileira no ano de 1999. Os documentos foram tomados da base de dados da máquina de busca TodoBR[2]. A WBR-99 contém aproximadamente 6 milhões de documentos tanto em formato de texto como em formato indexado. Esta coleção conta ainda com um conjunto de todas as consultas submetidas ao TodoBR no mês de novembro de 1999, sendo que para 50 destas consultas existe um conjunto de documentos relevantes. A tabela 4.1 apresenta os principais dados sobre a coleção WBR.

O *log* de novembro de 1999 foi utilizado apenas nos experimentos de medida de precisão. Para os experimentos de similaridade de *ranking* e de tempo de execução,

[1] http://www.linguateca.pt/Repositorio/WBR-99/

[2] http://www.todobr.com.br

utilizamos o conjunto de consultas submetidas à máquina de busca TodoBR no período de janeiro a outubro de 2003. outros dados sobre este *log* são mostrados na tabela 4.2.

Coleção WBR-99	
Tamanho total	20 gigabytes
Tamanho do texto	16 gigabytes
Número de documentos	5.939.061
Número de termos	2.669.965
Consultas avaliadas	50
Documentos Avaliados	4.117

Tabela 4.1: Dados sobre a coleção WBR-99

Log de Consultas TodoBR-2003	
Meses	janeiro a outubro
Total de consultas	2.158.756
Média de palavras por consulta	3.4
Consultas com mais de um termo	65%
Conjuntivas	87%
Frases	12%
Disjuntivas	1%

Tabela 4.2: Dados sobre a coleção WBR-99

Para verificar a precisão dos resultados foram utilizados as 50 consultas avaliadas do *log* de novembro de 1999, enquanto que para verificar a similaridade dos rankings foram utilizados dois conjuntos de 1000 consultas escolhidas aleatoriamente do *log* de fevereiro de 2003, sendo um conjunto composto por consultas disjuntivas e outro por consultas conjuntivas.

4.2 Impacto na Ordenação da Resposta

Os gráfico das figuras 4.1 e 4.2 mostram a similaridade entre os 20 primeiros resultados retornados pelo índice original e pelos índices podados em diferentes taxas de poda. Utilizamos a versão modificada da medida de similaridade de Kendall tau (vide seção 2.4.2) para comparar os resultados. Conforme pode ser visto, a eficácia do método de *Carmel* para consultas disjuntivas é muito boa, sofrendo uma perda muito pequena, mesmo para uma poda de 70% do índice. Por outro lado, quando tratamos as consultas como conjuntivas a qualidade se degrada rapidamente.

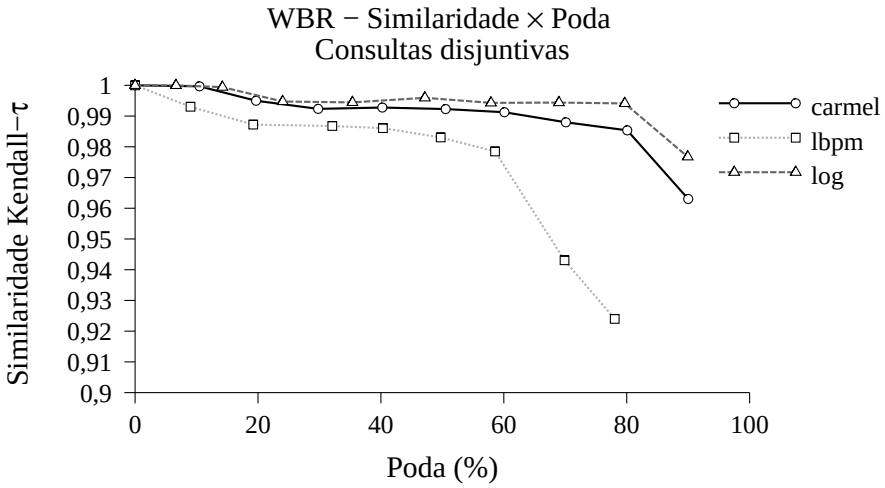

Figura 4.1: Comparação da similaridade das respostas disjuntivas para os três métodos de poda estática avaliados sobre a coleção WBR-99

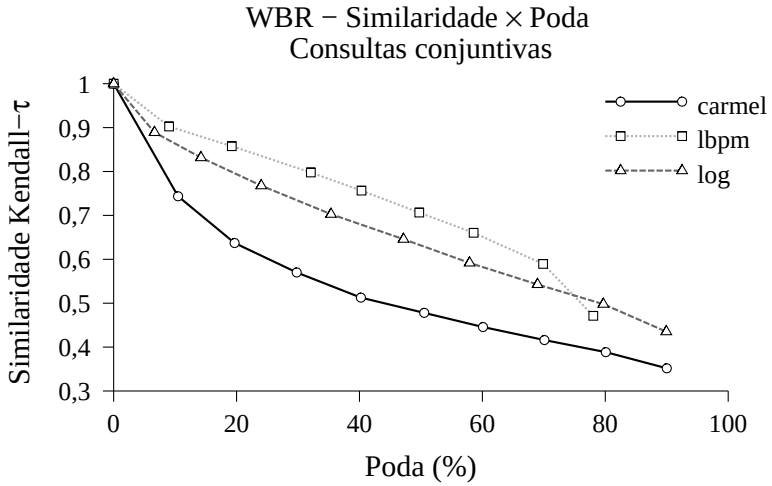

Figura 4.2: Comparação da similaridade das respostas conjuntivas para os três métodos de poda estática avaliados sobre a coleção WBR-99

Estes gráficos também mostram que nosso método dá resultados melhores que os do método de *Carmel* tanto para as consultas disjuntivas quanto para as consultas conjuntivas. É importante salientar que as consultas utilizadas nesses experimentos foram retiradas do *log* do mês de fevereiro, enquanto as consultas que utilizamos para gerar os índices utilizando nosso método foram extraídas do *log* do mês de janeiro. Em comparação com o *Lbpm*, nosso método obtém resultados bem próximos, superando-o por pouco no caso das consultas disjuntivas e perdendo por pouco no caso das conjuntivas.

Também realizamos experimentos para verificar o que acontece com a similaridade da resposta à medida em que aumentamos a distância entre o momento em que foi realizada a poda e o momento em que foram realizadas as consultas. Esta avaliação é importante porque na medida em que esta diferença de tempo aumenta, espera-se que haja uma relação cada vez menor entre as consultas utilizadas para guiar o processo de poda e as consultas correntes. Como conseqüência pode-se ter uma degradação da qualidade da resposta com o passar do tempo.

Os resultados desse experimento podem ser observados no gráfico da figura 4.3. Pode-se perceber que a similaridade das respostas varia pouco em um período de 9 meses. Isso significa que o índice não precisa ser atualizado com periodicidade muito freqüente para que o processo de poda baseado em *logs* dê bons resultados.

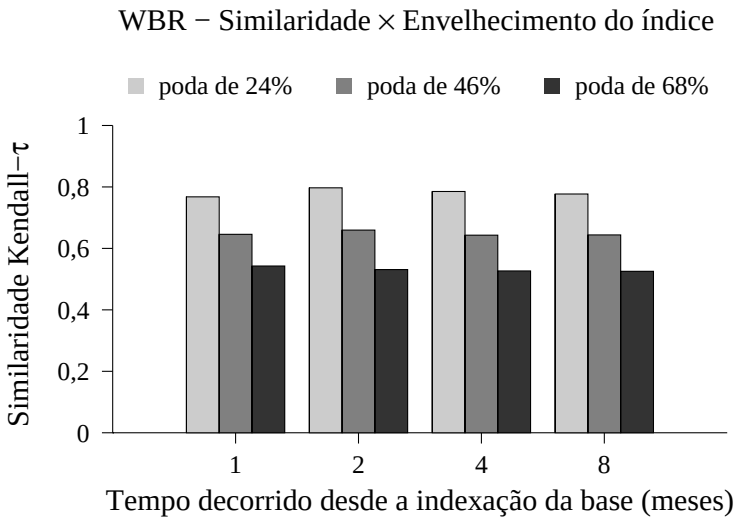

Figura 4.3: Qualidade da resposta à medida em que o índice podado envelhece

4.3 Qualidade da Resposta

Além das diferenças na ordenação das respostas, outro fator importante a ser avaliado é a qualidade das mesmas. Os resultados deste experimento podem ser vistos no gráfico da figura 4.4.

Esta métrica é importante por indicar o impacto do sistema de poda na qualidade dos serviços prestados pelo sistema de busca aos seus usuários. O gráfico mostra a precisão média obtida com os diferentes métodos de poda estática ao variarmos o percentual de poda. Nota-se que os método *Lbpm* e *log* fornecem resultados similares em termos de precisão para todos os níveis de poda experimentados. Este resultado

comprova que os logs de consultas passadas são fontes de informação tão úteis para o processo de poda quanto o próprio texto dos documentos.

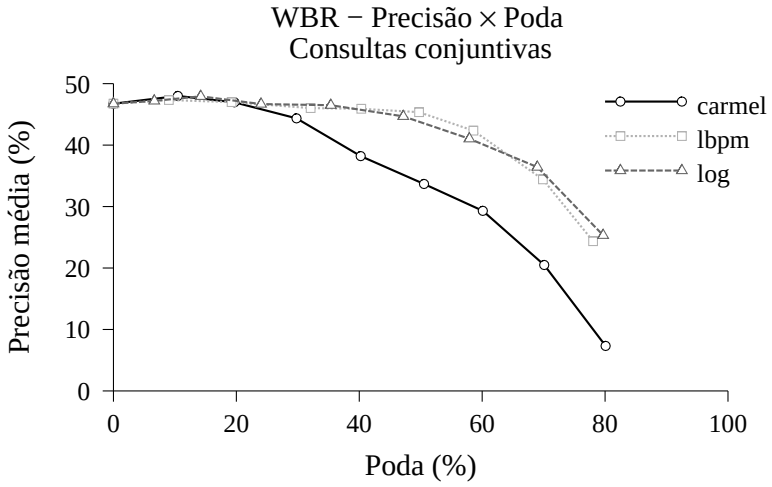

Figura 4.4: Comparação da precisão para os três métodos de poda estática avaliados

4.4 Tempo de Execução

Esta seção mostra as diferenças no tempo de execução de consultas em índices gerados pelos distintos processos de poda. A primeira vista, não haveria razão para uma variação no tempo de execução ao utilizarmos níveis de poda similares em diferentes métodos. Contudo, as estratégias de eliminação de entradas influenciam o tempo de execução assim como influenciam a qualidade das respostas.

A figura 4.5 mostra o tempo médio de execução das consultas utilizando cada um dos três métodos experimentados quando variamos o percentual de poda. A porcentagem do tempo é em relação ao tempo gasto utilizando o índice sem poda. No experimento foram utilizadas apenas consultas conjuntivas, uma vez que as mesmas são mais populares e apresentam custo de processamento bem maior que o de consultas disjuntivas. Como pode ser observado, o método de *Carmel* apresenta ganhos significativos no tempo de execução mesmo com percentuais pequenos de poda. Contudo, a vantagem de desempenho obtida pelo método de *Carmel* é anulada pelas perdas na qualidade das respostas obtidas pelo método. Dentre os três métodos, o que apresenta melhor relação entre custo e benefício é o método baseado em *logs*, que apresenta tempos de resposta melhores que o *Lbpm* com qualidade de resultados similares.

Uma explicação para as diferenças de tempo de execução entre os métodos está na forma como os mesmos eliminam entradas de termos mais freqüentes da coleção.

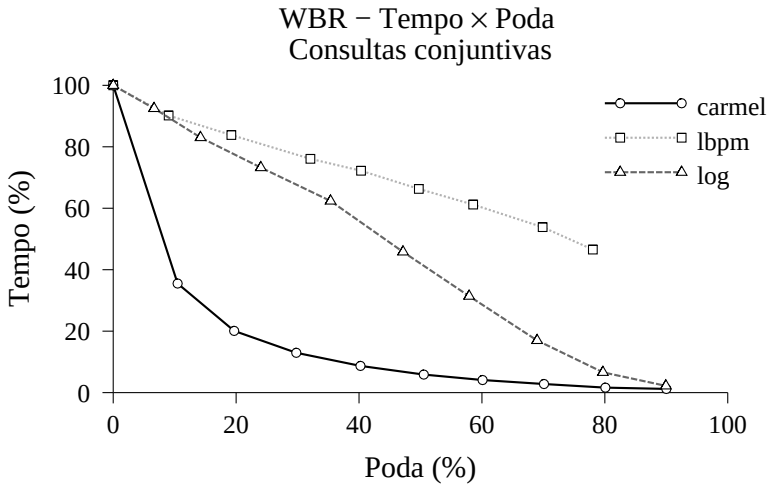

Figura 4.5: Comparação dos tempos de processamento para os três métodos de poda estática avaliados

O método de *Carmel* normalmente elimina mais entradas de termos freqüentes que os outros dois métodos. O *Lbpm* elimina entradas dos termos baseando-se em co-ocorrências dos termos dentro dos textos. Por esta razão o método *Lbpm* tende a preservar mais entradas de palavras comuns, as quais são conhecidas como *stopwords*. O método baseado em logs também é suscetível ao problema de preservar em demasia entradas de palavras comuns, contudo, como o mesmo é baseado em logs e não no texto, a quantidade de ruídos inseridos no método é menor.

Para ilustrar o problema, verificando o tamanho das listas das 250 palavras mais freqüentes da coleção WBR e um nível de poda de 50%, percebemos que o método de *Carmel* preservou em média cerca de 119 mil entradas do índice, o método baseado em logs cerca de 200 mil e o método *Lbpm* cerca de 440 mil. Estes valores comprovam que o *Lbpm* tende a preservar mais entradas de palavras comuns. O problema com a preservação destas palavras é que as mesmas também costumam aparecer em consultas, causando um aumento no tempo médio de processamento.

Capítulo 5

Conclusões e Trabalhos Futuros

Esta dissertação apresentou um novo método de compressão com perda (poda) para arquivos invertidos que considera o aspecto eficiência sem desconsiderar a eficácia. O método proposto é baseado na análise de *logs* de consultas passadas para obter uma grande redução no espaço ocupado pelo índice. O método pode ser utilizado em qualquer máquina de busca para melhorar sua eficiência em termos de tempo de processamento e espaço ocupado pelo índice, praticamente sem perdas na qualidade dos resultados da consulta.

Os experimentos mostram que a técnica apresentada reduz os custos de armazenamento do índice em até 50% com relação ao índice sem compressão. Para essa taxa de compressão, a perda de precisão é de apenas 2%. Já para uma taxa de compressão de 70% a perda de precisão é de 10%, que é uma perda relativamente baixa dada a enorme redução do tamanho do índice. A relação custo-benefício obtida entre taxa de poda e precisão dos resultados é extremamente vantajosa. Com a aplicação de nossa técnica é possível prover busca indexada eficaz até mesmo para dispositivos com capacidade de armazenamento bastante limitadas, como é o caso de computadores de bolso.

Uma consequência dessa redução no tamanho do índice é que o tempo de processamento de uma consulta pode ser reduzido a aproximadamente 45% do tempo original, sem perda na precisão média. A uma taxa de poda de 70% o tempo de processamento cai para quase 20% do tempo original com uma perda de 10% da precisão dos resultados. Novamente, o custo-benefício entre a taxa de poda e o tempo de processamento é bastante vantajoso.

Considerando a similaridade do *ranking* produzido com o *ranking* original, o espaço ocupado pelo índice e o tempo de resposta a consultas, os estudos comparativos com os métodos de *Carmel* e *Lbpm* mostram que nosso método agrega as maiores vantagens individuais daqueles. Para uma mesma taxa de poda, o método de *Carmel* consegue tempos de resposta muito melhores, especialmente para consultas conjuntivas. Por

outro lado, sua poda reduz sensivelmente a similaridade com o *ranking* original. Já o *Lbpm* consegue manter a similaridade com o *ranking* original em níveis muito melhores que *Carmel*. Por outro lado, apesar de o ganho em tempo de processamento ser significativo em relação ao índice original, ele é bem mais alto do que o tempo obtido pelo método de *Carmel*. Nosso método une o melhor dos dois mundos, mantendo a similaridade do *ranking* bem próxima aos níveis do *Lbpm* e mantendo o tempo de resposta não tão baixo quanto o método de *Carmel*, mas muito menor que o *Lbpm*.

Embora nosso método perca para o *Lbpm* em termos de similaridade com o *ranking* original, não se nota diferenças na qualidade das respostas em termos de precisão. É interessante notar também que para taxas de poda mais baixas, entre 10% e 20%, todos os métodos de poda estática apresentaram um pequeno aumento na precisão dos resultados. Esse fato também é observável com na poda dinâmica (Persin et al., 1996).

Conforme observado na seção 4.4, o tamanho das listas invertidas das *stopwords* é determinante no que diz respeito ao tempo de processamento das consultas. Semelhantemente, esse tipo de termo tem papel importante na precisão dos resultados quando se aplica um método de poda estática. Embora a contribuição das *stopwords* para a ordenação das respostas seja muito baixa, descartar documentos das listas invertidas destes termos é uma tarefa arriscada, pois nas consultas as *stopwords* ocorrem em conjunto com outros termos para os quais os documentos descartados podem ser importantes. Esse fenômeno pode ser claramente observado em nossos experimentos. Para todas as taxas de poda, o método de *Carmel* tende a podar mais as listas invertidas das *stopwords* enquanto nosso método e o *Lbpm* tendem a preservar tais listas. Como conseqüência a qualidade das respostas obtidas por estes mostrou-se superior à qualidade das respostas daquele.

Baseado-se nas observações acerca das *stopwords*, uma proposta de tabalho futuro é que ao podar arquivos invertidos estaticamente, seja utilizada uma estratégia diferente para as *stopwords*. Em vez de podar tais listas, poderíamos utilizar uma codificação que seja mais econômica que a lista invertida para o caso específico de termos muito freqüentes. Uma idéia inicial seria o uso de um *bitmap* em que assinalaríamos apenas a presença ou ausências do termo em cada documento por meio de um único bit.

Algumas verificações e experimentos já foram realizados no sentido do uso do *bitmap*.

- Os 40 termos mais freqüentes da coleção WBR-99 são *stopwords*;

- Aproximadamente 87% do tempo de processamento das consultas são gastos no processamento desses 40 termos;

- Na coleção WBR-99, a soma do tamanho dos bitmap desses 40 termos ocuparia aproximadamente 7% do tamanho de suas listas invertidas somadas.
- Para todos os níveis de poda, quando não somamos as contribuições parciais relativas a esses 40 termos:
  - A similaridade do *ranking* com o original sem poda cai muito pouco;
  - A precisão dos resultados aumenta levemente;

Todas as observações acima indicam que o uso do *bitmap* para as *stopwords* em conjunto com a poda estática dos demais termos leve a excelentes resultados tanto em termos de economia de espaço de armazenamento como em precisão dos resultados.

Referências Bibliográficas

Agrawal, R.; Imielinski, T. e Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 207–216. ACM Press.

Baeza-Yates, R. e Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison Wesley, 1st edition edição.

Barroso, L. A.; Dean, J. e Holzle, U. (2003). Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28.

Borgelt, C. e Kruse, R. (2002). Induction of association rules: apriori implementation. In *Proceedings of the 15th Conference on Computational Statistics*, pp. 395–400. Springer.

Carmel, D.; Cohen, D.; Fagin, R.; Farchi, E.; Herscovici, M.; Maarek, Y. S. e Soffer, A. (2001). Static index pruning for information retrieval systems. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 43–50. ACM Press.

de Moura, E. S.; dos Santos, C. F.; Fernandes, D. R.; Silva, A. S.; Calado, P. e Nascimento, M. A. (2005). Improving web search performance via a locality based static pruning method. In *Proceedings of the 14th International Worl Wide Web Conference*, pp. 235–244.

de Moura, E. S.; Navarro, G.; Ziviani, N. e Baeza-Yates, R. (2000). Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems*, 18(2):113–139.

Deerwester, S. C.; Dumais, S. T.; Furnas, G. W.; Landauer, T. K. e Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Fagin, R.; Kumar, R. e Sivakumar, D. (2003). Comparing top k lists. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 28–36. Society for Industrial and Applied Mathematics.

Kendall, M. G. e Gibbons, J. D. (1990). *Rank correlation methods*. Edward Arnold, 5th edition edição.

Persin, M.; Zobel, J. e Sacks-Davis, R. (1996). Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47(10):749–764.

Raghavan, S. e Garcia-Molina, H. (2001). Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 129–138. Morgan Kaufmann.

Salton, G. e McGill, M. J. (1986). *Introduction to Modern Information Retrieval*. McGraw-Hill Companies.

Witten, I. H.; Moffat, A. e Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 2nd edition edição.

Ziviani, N.; de Moura, E. S.; Navarro, G. e Baeza-Yates, R. (2000). Compression: A key for next-generation text retrieval systems. *IEEE Computer*, 33(11):37–44.