Igrep - Um Sistema para Busca Aproximada em Textos Indexados*

Márcio Drumond Araújo Nivio Ziviani

Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Caixa Postal 702, Campus da UFMG, Pampulha
Belo Horizonte, MG, Brasil - CEP: 30123-970
{drumond,nivio}@dcc.ufmg.br

Resumo

O presente trabalho apresenta o sistema IGREP para busca aproximada em textos de grande porte. O IGREP faz uso de uma lista invertida, composta por uma tabela contendo o vocabulário de palavras do texto e uma lista de endereços no texto correspondendo às ocorrências no mesmo de cada palavra do vocabulário. O tamanho do vocabulário é menos do que 1% do tamanho do texto como um todo, fazendo com que seja possível a sua manutenção em memória principal durante todo o processo de busca. Para consultas contendo uma palavra, a busca fica restrita apenas ao vocabulário. Para consultas contendo mais de uma palavra, a busca fica restrita ao vocabulário e respectivas listas de endereços, não havendo pois nenhum acesso ao texto na fase de pesquisa. O sistema permite desde a busca com erros ou não de palavras e frases até buscas de seqüências complexas contendo conjuntos de caracteres, caracteres coringa e expressões regulares arbitrárias. A construção do índice é $O(n)$ na prática, fazendo uso de uma única leitura seqüencial do texto. O tamanho do índice é aproximadamente $\frac{2}{3}$ do tamanho total do texto. O tempo de busca é $O(\sqrt{n})$ para casos típicos. Os resultados experimentais mostram que o sistema funciona bem na prática: para um texto de 1 *gigabyte*, casamentos compostos por 3 palavras com até 1 erro são recuperados em aproximadamente 6 segundos. No caso de busca sem erro, as ocorrências são obtidas em menos de meio segundo.

Abstract

This work presents a system called IGREP for exact and approximate string matching in large indexed texts. The index is composed by a table containing the vocabulary of words of the text and a list of addresses in the text correponding to the addresses of the occurrences of each word. The size of the vocabulary is less than 1% of the size of the text and hence can be kept in main menory during the whole querying process. The

*Este trabalho foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e Projeto RITOS/CYTED.

most significant time in the querying process is spent in this table in almost all kinds of searches. At the same time, the text is not accessed at all. The system still allows a large number of variations, since the simple exact and approximate word and phrase searches until complex string searches containing sets of characters, wild cards and arbitrary regular expressions. The whole index can be built in $O(n)$ time in practice through a single sequential pass over the text. Its size is about $\frac{2}{3}$ of the text size. On the other hand, the retrieval times are about $O(\sqrt{n})$ for typical situations. Experimental results show that the system works well in practical situations: for a one-gigabyte text, the matches of a phrase composed by 3 words allowing up to 1 error can be retrieved in approximately 6 seconds. In exact searches, the matches are found in under half a second.

1 Introdução

O problema da busca em texto dentro da área de recuperação de informação tem ganhado muita popularidade ultimamente. Nesses problemas, o usuário expressa sua necessidade de informação através de seqüências ou padrões fornecidos para serem pesquisadas e o sistema de informação recupera as posições onde tais seqüências ocorrem dentro do texto. Quando o texto é muito grande, normalmente utiliza-se alguma técnica de indexação para uma maior eficiência. Uma técnica simples e popular de indexação é a lista invertida. Ela é adequada quando o padrão a ser pesquisado é formado por palavras comuns. Como esse é um tipo de busca extremamente comum, por exemplo, em sistemas de busca utilizados na Internet, a lista invertida tem se tornado bastante popular em tal contexto.

Uma fraqueza que os sistemas para busca em textos de grande porte comercialmente disponíveis é a necessidade de se fazer a busca de padrões considerando somente a grafia correta dos mesmos devido à utilização de estruturas de indexação. Entretanto, em muitas situações o padrão procurado e/ou o texto não são exatos, devido a reconhecimento óptico de caractares (OCR), erros de digitação ou de grafia incorreta ou porque está se procurando padrões de forma aproximada. Por exemplo, um nome procurado pode estar digitado de forma incorreta no texto ou então o usuário não se lembra da sua grafia correta.

Mais formalmente o problema da busca aproximada em textos consiste em encontrar todas as seqüências no texto que estão numa dada distância menor ou igual a k do padrão p. A distância entre os dois é dada pelo conceito de distância de edição [Lev66] que considera o número mínimo de inserções, remoções e substituições de caracteres necessárias para transformar uma seqüência no padrão.

A solução clássica para o problema da busca aproximada tem custo $O(mn)$, onde m é o tamanho do padrão e n é o tamanho do texto [Sel80]. Uma das primeiras tentativas de se obter um algoritmo linear para solucionar esse problema foi feita por Ukkonen [Ukk85]. Desde então, há uma grande coleção de trabalhos sobre o assunto, onde [BYG92, WM92, ST95, BYN96] representam uma lista parcial do que foi feito mais recentemente.

Do ponto de vista prático, um paradigma recém surgido de grande importância é o denominado *bit*-paralelismo, desenvolvido por Baeza-Yates e Gonnet [BYG92]. Nesse paradigma, o estado da busca é representado como um número inteiro, sendo que as transições são simuladas através de operações *bitwise*. Wu e Manber [WM92] estenderam o método para a

busca aproximada genérica considerando o conceito de distância de edição. O algoritmo de Wu e Manber tem custo $O(kn)$, onde k é o número de erros. Recentemente, Baeza-Yates e Navarro [BYN96] conseguiram um algoritmo $O(n)$ para a busca aproximada.

Por outro lado o problema da busca aproximada em textos de grande porte, onde é necessária a utilização de índice, é considerado um problema não resolvido em [WM92]. Os primeiros trabalhos considerando o uso de estruturas de indexação nesse problema surgiram somente após essa data [MW93, Ukk93, Cob95, ST96].

No presente trabalho é apresentado um sistema eficiente para a recuperação de padrões de forma exata e aproximada em textos de grande porte. É mostrado que o sistema chamado IGREP é eficiente tanto na parte de construção do índice quanto na parte de busca. O índice pode ser construído em um tempo proporcional $O(n)$ na prática, com complexidade de espaço também $O(n)$. O tempo de busca é proporcional a $O(\sqrt{n})$ e $O(\sqrt{n}\log_2 n)$ nos casos mais típicos.

O sistema IGREP tem sido testado de forma bem sucedida em textos da ordem de 1 *gigabyte*. Ele ainda suporta diversas variações para o problema da busca aproximada. Além da busca de palavras e frases comuns, o sistema permite a busca de padrões contendo conjuntos de caracteres (intervalos, conjuntos arbitrários, complementos e caracteres coringa), buscas aproximada e exata sendo feitas num mesmo padrão e expressões regulares arbitrárias. O sistema está na versão 1.0, disponível através de FTP anônimo no endereço `ftp://ftp.dcc.ufmg.br/pub/research/nivio/igrep`.

O texto está organizado como se segue. A seção 2 descreve o índice e o processo de sua construção. A seção 3 descreve o processo de busca. A seção 4 apresenta os resultados analíticos. A seção 5 mostra os principais resultados experimentais. Finalmente, na seção 7 são apresentadas as conclusões do trabalho.

2 A Lista Invertida

O índice utilizado no presente trabalho é baseado em uma lista invertida que possui dois componentes: o primeiro contém todas as palavras distintas do texto (ou seja, o vocabulário) e o segundo contém os endereços das ocorrências de todas as palavras do vocabulário do texto, na ordem em que cada ocorrência aparece.

A figura 1 ilustra a estrutura da lista invertida para um texto exemplo contendo seis palavras. Cada posição do vocabulário contém uma palavra distinta do texto e uma referência para a última posição de sua respectiva lista de ocorrências.

Para responder a uma busca do usuário, apenas o vocabulário e as listas de ocorrências são necessárias, não havendo necessidade de acessos a texto em nenhum momento. Para padrões compostos por uma única palavra, a busca se restringe ao vocablário. Se algum casamento for detectado no vocabulário, automaticamente o sistema já obtém todas as ocorrências daquela palavra no texto através das listas de ocorrências. Quando o padrão contém mais de uma palavra, o sistema procura no vocabulário por cada palavra separadamente recuperando as respectivas listas de ocorrências. Com as listas em mãos, o IGREP realiza o que é chamado interseção das listas, procurando por referências que têm o mesmo posicionamento relativo encontrado dentro do padrão. A cada ocorrência em que a distância relativa entre as

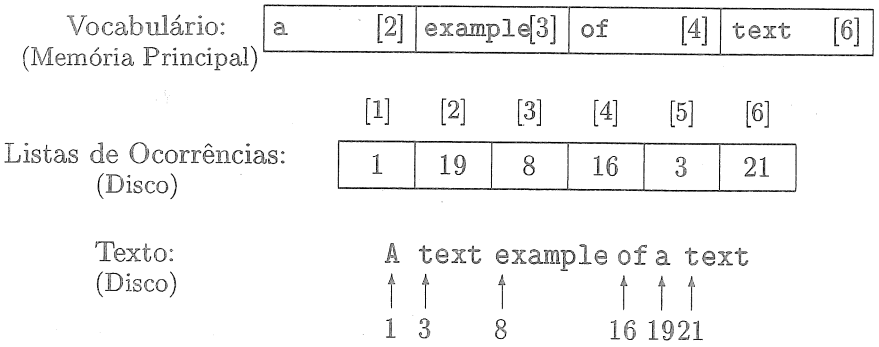

Figura 1: Estrutura da lista invertida

diferentes palavras casadas com as palavras do padrão satisfazem o mesmo distanciamento encontrado entre as palavras no padrão, tem-se um casamento completo.

Para ilustrar o processo de busca no índice são apresentados dois exemplos. A busca exata do padrão "`text`" na figura 1 envolve uma busca binária no vocabulário, obtendo a lista de ocorrências equivalente ao intervalo $[5, 6]$ no arranjo de listas. O outro exemplo considera a busca com 2 erros do padrão "`text sample`". Primeiro o sistema busca a palavra "`text`" obtendo o intervalo $[5, 6]$ no arranjo de listas. Depois, ele busca a palavra "`sample`" retornando como casamento a palavra "`example`" com dois erros correspondente ao intervalo $[3, 3]$. As duas listas contém os endereços $\{3, 21\}$ e $\{8\}$ respectivamente, correspondendo aos endereços contidos nos intervalos $[5, 6]$ e $[3, 3]$ do arranjo de listas. Através da interseção entre as duas listas de ocorrências, o sistema indica que na posição 3 há um casamento do padrão como um todo, levando em conta que a seqüência "`text example`" apresenta a mesma distância relativa entre suas palavras em relação ao padrão "`text sample`".

2.1 A Construção da Lista Invertida

O algoritmo utilizado durante o processo de construção da lista invertida é o de Moffat e Bel [MB95]. Ao final da construção, obtém-se dois arquivos em disco magnético: um contendo o vocabulário e o outro as listas de ocorrências. No primeiro passo da construção se faz a leitura seqüencial do texto, palavra por palavra, pesquisando cada palavra em uma tabela *hash*. Se a palavra não existe na tabela, o sistema insere a palavra na mesma. Ao mesmo tempo que ocorre a identificação das palavras no texto, o sistema armazena também o endereço da ocorrência da palavra no final de uma lista em memória principal.

O sistema, entretanto, trabalha com a possibilidade da lista invertida não caber em memória principal. Nesse caso, ele considera um espaço limitado para o armazenamento das ocorrências das palavras. A cada momento que esse espaço é preenchido, ordena-se o mesmo e salva-o em disco concatenando-o num arquivo responsável pelo armazenamento de todas as ocorrências. Cada bloco salvo em disco é chamado de *dump* (descarga). Esse processo é repetido até que todas as ocorrências tenham sido identificadas e gravadas no arquivo.

O próximo passo consiste na intercalação dos blocos de ocorrências ou *dumps* obtidos

na etapa anterior. Basicamente, o sistema faz uso do algoritmo *mergesort in-place* sobre o arquivo. Esse enfoque considera que a cada passo da intercalação, ao invés de se gerar um novo arquivo contendo blocos r vezes maiores, onde r corresponde ao número de caminhos utilizados durante a intercalação, os blocos resultantes são salvos sobre o mesmo arquivo que contém os blocos lidos. A intercalação *in-place* proporciona uma economia significativa de espaço em disco durante a construção da lista invertida. Porém, o seu uso geralmente faz com que os blocos sejam salvos fora de ordem. Nesse caso, o sistema utiliza uma estrutura de referência indireta que contém o verdadeiro posicionamento físico dos blocos dentro do arquivo. A figura 2 mostra como os blocos aumentam de tamanho a cada passo considerando uma intercalação de 3 caminhos.

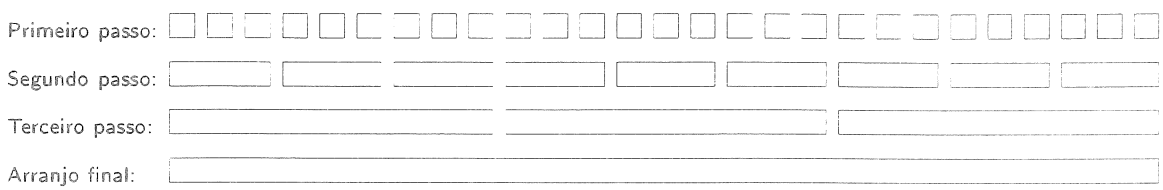

Figura 2: Intercalação considerando 3 caminhos

Finalizada a intercalação, a última etapa consiste em fazer a permutação dos blocos de tal forma que eles fiquem armazenados de forma contígua no arquivo. Ao final da permutação, tem-se em mãos os dois arquivos desejados.

3 O Processo de Busca

O sistema IGREP permite basicamente dois tipos de busca: padrões compostos por uma palavra e padrões correspondendo a frases. Nos dois casos o sistema consegue procurar por ocorrências exatas ou aproximadas no texto. Cada uma dessas possibilidades envolve algoritmos e tarefas diferentes utilizados pelo sistema. A seguir, descreve-se os principais tipos de busca realizadas.

3.1 Padrões Compostos por Uma Palavra

A característica mais importante do comportamento do sistema em relação a padrões compostos por uma palavra é que somente o vocabulário é consultado. As respectivas listas de ocorrências são automaticamente recuperadas através dos casamentos encontrados no vocabulário.

A busca no vocabulário pode ser binária ou seqüencial. A busca exata de uma palavra comum, ou seja, uma palavra sem nenhum caractere com significado sintático especial para o sistema envolve uma busca binária no vocabulário. Caso a palavra seja encontrada, o sistema tem em mãos a sua respectiva lista de ocorrências automaticamente.

A busca aproximada por sua vez envolve uma busca seqüencial no vocabulário. Para palavras comuns, o sistema utiliza o algoritmo de Baeza-Yates e Navarro [BYN96]. O tempo gasto com esse algoritmo é independente do número de erros com o qual se realiza a busca ($O(n)$) para pequenos padrões, sendo extremamente rápido na prática. O algoritmo é baseado

na simulação de um autômato não determinístico, sendo que a cada caractere inspecionado no vocabulário, as transições no autômato são calculadas em tempo $O(1)$. Por exemplo, em uma arquitetura de 32 *bits*, o algoritmo permite que um padrão com até 9 caracteres seja procurado em um tempo $O(n)$ com qualquer número de erros. Isso é razoável para os propósitos do presente trabalho visto que a maioria das palavras têm menos que 9 caracteres na prática. Experimentos da seção 5 mostram que buscas no vocabulário com esse algoritmo demoram cerca de 1 segundo.

Para padrões mais complexos envolvendo símbolos com significado sintático especial faz-se uso do algoritmo de Wu e Manber [WM92]. Esse algoritmo é extremamente flexível, permitindo a busca de padrões complexos, conforme descrição abaixo. Entretanto, tal flexibilidade torna o algoritmo menos eficiente, dependente do número de erros durante a busca ($O(kn)$). A seguir são mostrados alguns padrões complexos juntamente com as possíveis palavras que casariam com os mesmos no texto:

- "`t[a-z]xt`": palavras iniciadas por "`t`", seguido de qualquer caractere entre "`a`" e "`z`", seguido de "`xt`".
- "`t[aeiou]xt`": palavras iniciadas por "`t`", seguido de qualquer vogal, seguido de "`xt`".
- "`t[^aeiou]xt`": palavras iniciadas por "`t`", seguido de qualquer caractere diferente de vogal, seguido de "`xt`".
- "`t.xt`": palavras iniciadas por "`t`", seguido de qualquer caractere, seguido de "`xt`".
- "`t#xt`": palavras iniciadas por "`t`", seguido de qualquer caractere do alfabeto zero ou mais vezes em qualquer ordem, seguido de "`xt`".
- "`t(e|ai)xt`": palavras iniciadas por "`t`", seguido de "`e`" ou de "`ai`", seguido de "`xt`". Nesse caso, o padrão é visto como uma expressão regular.
- "`t(e|ai)*xt`": palavras iniciadas por "`t`", seguido de "`e`" ou de "`ai`" zero ou mais vezes em qualquer ordem, seguido de "`xt`". Esse padrão também é uma expressão regular.
- "`<te>xt`": o sistema busca as ocorrências da palavra "`text`" considerando que o prefixo "`te`" tem de ocorrer de forma exata e o sufixo "`xt`" pode ocorrer de forma aproximada.

O sistema ainda permite a busca insensível ao caso, sem diferenciar letras maiúsculas de minúsculas (o normal é diferenciá-las). O usuário também tem a possibilidade de atribuir pesos diferentes a cada um dos erros considerados pelo conceito de distância de edição. Normalmente, o sistema considera cada um dos erros com peso unitário.

3.2 Padrões Compostos por Mais de Uma Palavra

Para padrões contendo mais de uma palavra, o sistema procura por cada palavra separadamente no vocabulário e depois realiza a interseção entre as listas de ocorrências obtidas com o intuito de identificar os casamentos do padrão como um todo.

Na busca aproximada, após a busca no vocabulário das palavras, as listas de ocorrências obtidas durante a busca no vocabulário são agrupadas primeiro por palavra do padrão e depois pelo número de erros com o qual são encontradas. O processo de interseção é repetido satisfazendo todas as combinações de erros entre as palavras do padrão tais que o somatório do número de erros de cada palavra seja menor ou igual a k. No caso de um padrão contendo j palavras numa busca permitindo k erros, o sistema faz $\binom{j+k}{k}$ interseções.

Para cada interseção, o sistema seleciona o conjunto de listas com o menor número de ocorrências. Esse conjunto é percorrido seqüencialmente, fazendo buscas binárias ou seqüenciais nas listas das palavras vizinhas, dependendo da relação entre os tamanhos dos dois conjuntos de listas. Caso o sistema encontre alguma ocorrência nas listas da palavra vizinha que satisfaça a distância entre as duas palavras no padrão, repete-se o processo para o conjunto de listas da palavra seguinte. Isso é feito até que todos os conjuntos de todas as palavras do padrão tenham sido verificados. Se para uma dada ocorrência do menor conjunto o sistema encontra ocorrências em todos os conjuntos de listas das demais palavras que satisfazem a distância relativa entre as palavras do padrão, então ele notifica um casamento completo. O custo de uma interseção é proporcional ao número de ocorrências do menor conjunto de listas multiplicado pelo número de buscas binárias realizadas durante a mesma.

4 Resultados Analíticos

4.1 A Construção da Lista Invertida

A construção da lista invertida consiste em três etapas. Na primeira etapa o texto é lido, sendo que as ocorrências são identificadas e trazidas para uma área da memória principal de tamanho M. A cada momento que essa área é preenchida, seu conteúdo é ordenado e concatenado no arquivo contendo as ocorrências das palavras do texto. Considerando que o texto possui n ocorrências, são obtidos $\frac{n}{M}$ blocos. Como cada ordenação custa $O(M \log_2 M)$, tal etapa tem um custo de $O(n \log_2 M)$.

A intercalação dos blocos de ocorrências tem o mesmo custo do algoritmo *mergesort* considerando n elementos. Com isso, tal etapa custa $O(n \log_2 n)$ para o sistema.

Por fim, a reorganização dos blocos tem o custo de uma leitura seqüencial no arquivo que contém as ocorrências, ou seja, $O(n)$. Portanto, levando em conta que essas etapas são executadas em seqüência, o custo da construção da lista é o custo da etapa mais cara, ou seja,

$$O(n \log_2 n)$$

4.2 O Processo de Busca

O processo de busca apresenta resultados diferentes dependendo do tipo do padrão e do número de erros permitidos durante a busca. Entretanto, para se avaliar tais custos, é necessário que se tenha em mãos uma estimativa do tamanho do vocabulário em função do tamanho do texto. A equação de Heaps [Hea78] é uma estimativa bem próxima da realidade, sendo dada pela expressão

$$v = cn^{\beta} \qquad (1)$$

onde $c > 0$ e $0 < \beta < 1$. Essa estimativa mostra claramente que a taxa de crescimento do vocabulário é decrescente em função do crescimento do texto.

Na seção 5 é mostrado que esse resultado faz com que seja possível a manutenção de todo o vocabulário em memória principal durante o processo de busca. A seguir são mostrados os custos para cada situação do processo de busca.

4.2.1 Padrões Compostos por Uma Palavra

Como a busca de padrões contendo uma palavra se resume a busca no vocabulário, o custo total da busca de tais padrões corresponde ao custo da busca dos mesmos no vocabulário. A seguir descreve-se as três situações de busca de padrões contendo uma palavra.

A Busca Exata de Padrões Contendo uma Palavra Comum Em tal situação, o sistema faz busca binária no vocabulário. Lembrando que, de acordo com a equação 1, o tamanho do vocabulário é $O(n^{\beta})$, o custo da busca binária no mesmo é, portanto,

$$O(\log_2 n).$$

A Busca Aproximada de Padrões Contendo uma Palavra Comum Na busca aproximada, a ordenação do vocabulário não é mais vantajosa, obrigando o sistema a fazer a busca da palavra no vocabulário seqüenciamente. Para palavras comuns, utilitiza-se o algoritmo de Baeza-Yates e Navarro. Considerando que esse algoritmo é independente do número de erros k permitidos durante a busca, o custo de sua utilização é o custo de uma leitura seqüencial do vocabulário, ou seja,

$$O(n^{\beta}).$$

A Busca de Padrões Contendo uma Palavra Complexa A busca por padrões contendo uma única palavra complexa, independente do número de erros, é feita utilizando o algoritmo de Wu e Manber. Ao contrário do caso anterior, o custo do algoritmo de Wu e Manber depende do número de erros com o qual se busca o padrão. Para cada caractere da base de dados, faz-se $k + 1$ operações. Logo, considerando a pesquisa seqüencial no vocabulário, o custo da busca nesse caso é

$$O((k + 1)n^{\beta}).$$

4.2.2 Padrões Compostos por Mais de Uma Palavra

Nesta seção faz-se a análise da busca exata e aproximada de frases. Quando o padrão contém mais de uma palavra, comum ou não, o sistema busca cada palavra separadamente no vocabulário, utilizando o algoritmo mais adequado em cada caso. Nesse caso, o processo de interseção das listas de ocorrências torna-se necessário.

A Busca Exata de Frases O sistema faz uma busca binária no vocabulário para cada palavra do padrão. Caso todas as palavras sejam encontradas, realiza-se a interseção entre suas listas de ocorrências. Considerando que nesse caso há j palavras no padrão, onde $j > 1$, o custo da busca no vocabulário de todas as palavras do padrão é $O(j \log_2 n)$.

Para que seja possível a realização da interseção entre as listas é necessário que se leia as mesmas para a memória principal. O custo disso é determinado identificando o tamanho

esperado das listas de ocorrências. Lembrando que o vocabulário tem tamanho $O(n^{\beta})$, o tamanho médio das listas é, portanto, $O(n^{1-\beta}) = O(n^{\alpha})$. Logo, havendo j palavras no padrão, a leitura das listas tem custo $O(jn^{\alpha})$.

O processo de interseção em si tem custo $O(C(j-1)\log_2 n)$, onde C representa o custo de se percorrer a menor lista seqüencialmente. O valor de C é determinado considerando a distribuição de Zipf [Zip49], dada pela equação

$$f(i) = \frac{n}{i^{\theta} H_v^{(\theta)}} \tag{2}$$

onde $H_v^{(\theta)} = \sum_{i=1}^{v} \frac{1}{i^{\theta}}$ corresponde à série harmônica de ordem θ e v representa o número de palavras contidas no vocabulário. Essa equação mostra o número de ocorrências esperado para a i-ésima palavra mais freqüente do texto.

Considerando que a escolha das palavras do padrão feita pelo usuário não depende da sua distribuição dentro do texto, há uma chance igual de que uma palavra escolhida pelo usuário seja qualquer palavra do vocabulário. Dessa forma, as palavras escolhidas pelo usuário tendem a se distribuir uniformemente ao longo do domínio da equação de Zipf. Usando isso como base, o tamanho da menor lista é $O(1)$ mesmo para o pior caso em que o padrão é composto por duas palavras. Dessa forma, a busca exata de frases contendo j palavras comuns tem um custo de

$$O(j \log_2 n) + O(jn^{\alpha}) + O((j-1)\log_2 n) = O(jn^{\alpha})$$

A Busca Aproximada de Frases Nesse caso, o processo de busca no vocabulário faz uso do algoritmo de Baeza-Yates e Navarro para cada palavra do padrão. Levando em conta que o tamanho do vocabulário é $O(n^{\beta})$, o custo total da busca no vocabulário considerando que haja j palavras no padrão é $O(jn^{\beta})$.

Na busca aproximada e na busca de palavras complexas, entretanto, há a possibilidade de que a palavra procurada se case com mais de uma palavra do vocabulário. Dessa forma, para cada palavra do padrão há um conjunto de listas de ocorrências cujo tamanho médio é $O(n^{\alpha})$. O total de ocorrências para essas palavras é dado pela soma dos tamanhos de tais listas de ocorrências. Uma estimativa para isso é a utilização de uma aproximação $O(n^{\gamma})$ para esse valor, onde $\alpha \leq \gamma \leq 1$. Nesse caso, o custo de se ler as listas de ocorrências de todas as palavras do padrão é $O(kjn^{\gamma})$, onde k representa o número de erros.

O custo esperado de se percorrer o menor conjunto de listas durante uma interseção é, portanto, $O(n^{\gamma})$. Como no pior caso se faz $j-1$ buscas binárias nos conjuntos de listas das palavras vizinhas para cada ocorrência da palavra menos freqüente numa dada interseção, o custo da interseção é $O(n^{\gamma}(j-1)\log_2 n)$. Lembrando que para k erros são necessárias $\binom{j+k}{k}$ interseções, o custo total das interseções é

$$O\left(\binom{j+k}{k} n^{\gamma}(j-1)\log_2 n\right)$$

sendo que esse também é o custo total da busca aproximada de frases, visto que as interseções representam a etapa mais cara do processo.

A Busca de Frases com Palavras Complexas Como já foi visto anteriormente, por palavras complexas entende-se expressões regulares, caracteres coringa e conjuntos de caracteres. Nesse caso, tanto na busca exata quanto na aproximada, é possível que a palavra se case com mais de uma palavra do vocabulário. A busca no vocabulário usando o algoritmo de Wu e Manber, considerando j palavras, tem custo $O((k+1)jn^{\beta})$, onde k é o número de erros.

Utilizando a mesma aproximação do caso anterior, pode-se considerar que cada conjunto de listas de cada palavra do padrão tem $O(n^{\gamma})$ ocorrências. Logo, a leitura das listas tem custo $O((k+1)jn^{\gamma})$.

Da mesma forma também são realizadas $\binom{j+k}{k}$ interseções. Com isso, o custo dessa etapa é

$$O((j-1)n^{\gamma}\log_2 n)$$

Logo, as interseções representam a etapa mais cara nesse caso, sendo, portanto, responsáveis pelo custo total da busca.

5 Resultados Experimentais

A seguir são mostrados alguns resultados experimentais referentes a construção da lista invertida e ao processo de busca. Os textos utilizados são os da coleção *TREC* [Har95], sendo que a tabela 1 mostra algumas estatíticas sobre os mesmos. Os testes foram feitos utilizando uma Sun SparcStation 4 com 128 *megabytes* de memória principal executando o sistema operacional Sun Solaris 2.5.1.

Arquivos	Texto		Vocabulário		Vocab./Texto	
	Tamanho (bytes)	#Palavras	Tamanho (bytes)	#Palavras	Tamanho	#Palavras
ap	237.766.005	37.740.089	1.530.192	201.115	0,64	0,53
doe	180.515.212	27.124.239	1.795.783	211.196	0,99	0,78
fr	219.987.476	32.000.223	1.043.869	132.129	0,47	0,41
wsj	262.757.554	40.741.508	1.511.951	198.818	0,58	0,49
ziff	242.660.178	38.047.824	1.639.677	216.482	0,68	0,57
az	480.426.183	75.787.913	2.574.518	336.716	0,54	0,44
adfwz	1.143.686.425	175.653.883	4.629.371	573.661	0,40	0,32

Tabela 1: Arquivos texto da coleção TREC collection

Como se pode ver na tabela 1, o tamanho do vocabulário dos textos sempre é menos de 1% do tamanho dos mesmos, viabilizando a utilização do vocabulário em memória principal durante o processo de busca.

5.1 A Construção do Índice

A tabela 2 mostra os valores da constante de proporcionalidade c e de β para equação 1 e o tamanho médio das listas de ocorrências considerando os textos utilizados. Os valores de

c e de β são obtidos medindo o tamanho do vocabulário a cada *megabyte* lido do texto e aplicando mínimos quadrados sobre o conjunto de medidas.

Texto	c	β	n^{α}
ap	26.8	0.46	187,7
doe	10.8	0.52	128,4
fr	13.2	0.48	242,2
wsj	43.5	0.43	204,9
ziff	11.3	0.51	175,8
az	9.2	0.52	225,1
adfwz	4.8	0.56	306,2

Tabela 2: Resultados experimentais dos coeficientes das equações de Heaps e de Zipf

Como se pode observar, os valores de β para todos os textos é próximo de 0,5, fazendo com que para os textos utilizados o tamanho do vocabulário seja de aproximadamente

$$O(\sqrt{n}).$$

Outro resultado trata-se do valor reduzido de n^{α}. Esse resultado mostra que o custo dos processos de leitura das listas de ocorrências para a memória principal e de interseção têm um valor razoável na prática. Um valor baixo para n^{α} também indica que n^{γ} também é razoavelmente reduzido considerando que $\gamma \rightarrow \alpha$ quando o número de erros permitidos durante a busca é pequeno.

Os tempos de construção da lista invertida são mostrados na tabela 3. Nela, além dos tempos totais de construção para os textos WSJ, AZ e ADFWZ, são mostrados os tempos gastos na etapa de intercalação. Tais medidas foram tomadas utilizando 96 *megabytes* de memória principal.

Tempos de construção do índice			
Arquivo	Tamanho (*megabytes*)	Tempo total (min)	Tempo de *merge* (min)
wsj	250.6	58.5	13.8
az	458.2	122.7	33.9
adfwz	1090.7	248.9	79.8

Tabela 3: Resultados experimentais da construção do índice

Apesar do algoritmo de construção do índice ser $O(n \log_2 n)$, vê-se claramente na prática que ele tem comportamento $O(n)$, pois a componente $O(n \log_2 n)$ ocorre somente na fase de intercalação, correspondendo a cerca de 20% a 30% do tempo total.

5.2 Resultados do Processo de Busca

Aqui são mostrados os tempos para a busca exata e busca aproximada para diferentes taxas de erros ($k = 0, 1, 2, 3$), para frases compostas por 1, 2, 3, 4 e 5 palavras. As tabelas 4 e 6 mostram os tempos de busca obtidos, respectivamente, para os textos WSJ e ADFWZ.

	igrep				
	Número de palavras na busca				
k	1	2	3	4	5
0	0.077 ± 0.002	0.233 ± 0.024	0.242 ± 0.013	0.281 ± 0.010	0.343 ± 0.009
1	0.584 ± 0.007	1.99 ± 0.22	2.15 ± 0.12	2.59 ± 0.07	3.16 ± 0.05
2	0.848 ± 0.008	8.27 ± 1.77	4.26 ± 0.38	4.65 ± 0.29	5.06 ± 0.21
3	1.30 ± 0.04	34.1 ± 8.9	14.6 ± 3.5	11.2 ± 1.9	8.97 ± 0.72

Tabela 4: Tempo de busca em segundos para o texto WSJ usando o IGREP

Em relação ao texto WSJ, foram feitas medidas utilizando a mesma amostra de padrões com a versão 3.0 do GLIMPSE [MW93]. O GLIMPSE é uma das únicas ferramentas disponíveis para a busca aproximada em textos indexados. Ele faz uso de uma lista invertida associada com um processo de busca seqüencial. Nesse caso, os resultados mostram um resultado bastante satisfatório do IGREP em relação ao GLIMPSE quando utilizado em textos de maior porte.

	glimpse versão 3.0				
	Número de palavras na busca				
k	1	2	3	4	5
0	28.7 ± 0.6	25.9 ± 0.5	24.1 ± 0.6	23.4 ± 0.4	22.8 ± 0.3
1	132.6 ± 0.5	133.4 ± 0.8	133.2 ± 0.9	135 ± 1	*
2	161 ± 1	161 ± 1	163 ± 1	162 ± 1	*
3	190 ± 1	191 ± 2	194 ± 3	*	*

* O `glimpse` não permite a busca aproximada de frases contendo mais de 32 caracteres

Tabela 5: Tempo de busca em segundos para o texto WSJ usando o `glimpse`

	igrep				
	Número de palavras na busca				
k	1	2	3	4	5
0	0.095 ± 0.006	0.456 ± 0.054	0.411 ± 0.034	0.438 ± 0.026	0.475 ± 0.027
1	1.55 ± 0.02	6.18 ± 0.61	6.10 ± 0.33	7.09 ± 0.24	8.38 ± 0.22
2	2.29 ± 0.03	37.8 ± 9.7	17.8 ± 2.5	15.4 ± 1.8	15.3 ± 1.3
3	3.45 ± 0.10	108 ± 30	51.8 ± 11.5	34 ± 8	37 ± 11

Tabela 6: Tempo de busca em segundos para o texto ADFWZ usando o IGREP

Na tabela 7 são mostrados os tempos para alguns padrões complexos. O último padrão, em especial, corresponde a uma expressão regular.

igrep				
Padrão	k			
	0	1	2	3
`<exe>cutive`	0.0313	3.18	6.92	11.4
`earl# retir[aeiou]#<ent> program`	4.60	14.5	38.0	191
`acc[aeiou]*unt compri[ms](es|ent)`	10.2	22.2	*	*

* A versão atual do igrep não permite busca aproximada de expressões regulares quando o número de erros é maior ou igual à menor seqüência entre parênteses

Tabela 7: Tempos de busca em segundos para frases contendo palavras complexas no texto WSJ usando o igrep

6 Conclusões

A busca aproximada em textos indexados é uma área com poucos resultados práticos que tem sido bastante explorada recentemente. Os métodos desenvolvidos até então não representam soluções definitivas para o problema e isso também inclui o presente trabalho. Ainda há inúmeras dificuldades em se utilizar a busca aproximada em estruturas de indexação, principalmente por estas últimas serem desenvolvidas visando agilizar processos de busca exata. Além disso, há poucos estudos referentes ao desenvolvimento de estruturas de indexação objetivando a busca aproximada.

A principal contribuição do trabalho é justamente mostrar a possibilidade de se fazer buscas aproximadas de padrões contendo palavras complexas utilizando uma estrutura de indexação bem conhecida: a lista invertida. É mostrado claramente que a redução do espaço de pesquisa proporcionado pela utilização do vocabulário do texto normalmente é suficiente para a viabilização de tais buscas, mesmo considerando textos da ordem de 1 *gigabyte* ou mais. O sistema aqui implementado apresenta custos iguais a $O(\log_2 n)$, $O(\sqrt{n})$ e $O(\sqrt{n}\log_2 n)$ nos casos mais típicos. Tais resultados tornam o sistema atraente quando comparado com outros sistemas disponíveis destinados a buscas do mesmo tipo. Os tempos obtidos nos testes também confirmam isso, reforçando a idéia de que a lista invertida é uma boa alternativa em tais situações.

Outra contribuição diz respeito à implementação e disponibilização do sistema no endereço `ftp://ftp.dcc.ufmg.br/pub/research/nivio/igrep`. O IGREP foi implementado na linguagem *ANSI C*, contando hoje com cerca de 390 *kbytes* de código fonte (cerca de 33 mil linhas de código). Atualmente ele já foi testado em três plataformas de forma satisfatória: Sun Solaris 2.5.1, SunOS 4.1.4 e Linux para i386.

Entre as possíveis extensões para o trabalho está a adaptação de novos algoritmos para a busca no vocabulário. Outra alteração sem grandes custos seria fazer com que o sistema como um todo trabalhasse com mais de um arquivo simultaneamente. Isso permitiria a utilização do IGREP como um sistema de busca em árvores de diretórios como o GLIMPSE. Outra possibilidade seria fazer o sistema lidar com formatos específicos de arquivos texto. Isso permitiria a sua utilização em diferentes aplicações, como por exemplo, em sistemas de busca na Internet caso ele conseguisse entender o formato *HTML* (*Hiper Text Markup Language*).

Referências

[BYG92] R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, 1992.

[BYN96] R. Baeza-Yates and G. Navarro. A faster algorithm for approximate string searching. In *Proc. Combinatorial Pattern Matching*, volume 1075, pages 1–13. Springer-Verlag Lecture Notes in Computer Science, 1996.

[Cob95] A. L. Cobbs. Fast approximate matching using suffix trees. In *Proc. Combinatorial Pattern Matching*, pages 41–54. Springer-Verlag Lecture Notes in Computer Science, 1995.

[Har95] D. K. Harman. Overview of the third text retrieval conference. In *Proc. Third Text REtrieval Conference (TREC-3)*, pages 1–19, Gaithersburg, Maryland, USA, 1995. National Institute of Standards and Technology Special Publication.

[Hea78] J. Heaps. *Information Retrieval - Computational and Theoretical Aspects.* Academic Press, NY, 1978.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Sov. Phys. Dokl.*, 10:707–710, 1966.

[MB95] A. Moffat and T. Bell. In situ generations of compressed inverted files. *Journal of the American Society for Information Science*, 46(7):537–550, 1995.

[MW93] W. Manber and S. Wu. Glimpse: A tool to search through entire file systems. Technical Report 93-34, Dept. of Computer Science, The University of Arizona, Oct. 1993.

[Sel80] P. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1:359–373, 1980.

[ST95] E. Sutinen and J. Tarhio. On using q-gram locations in approximate string matching. In P. Spirakis, editor, *Proceedings Third Annual European Symposium on Algorithms (ESA'95)*, pages 327–340, Corfu, Greece, 1995. Springer-Verlag Lecture Notes in Computer Science.

[ST96] E. Sutinen and J. Tarhio. Filtration with q-samples in approximate string matching. In *Proc. Combinatorial Pattern Matching*, pages 50–61. Springer-Verlag Lecture Notes in Computer Science, 1996.

[Ukk85] E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137, 1985.

[Ukk93] E. Ukkonen. Approximate string-matching over suffix trees. In *Proc. Combinatorial Pattern Matching*, pages 228–242. Springer-Verlag Lecture Notes in Computer Science, 1993.

[WM92] S. Wu and U. Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, 1992.

[Zip49] G. K. Zipf. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, Cambridge, Mass., 1949.