4 Compactação de dados

Apesar da alta disponibilidade e custo em constante decréscimo de dispositivos de armazenamento de dados como discos rígidos e memória de acesso aleatório, o volume de dados sempre será um fator importante para aplicações intensivas sobre os dados, como os próprios sistemas de banco de dados, e para resolver os desafios de áreas como geografia e biologia cujas descobertas vêm gerando grandes volumes de dados.

Por esta razão, a compressão ou compactação de dados pode ser considerada como uma estratégia para reduzir o espaço de armazenamento físico, melhorar as taxas na transferência de dados, especialmente em aplicações distribuídas, e ainda, para obter melhor desempenho em aplicações intensivas sobre os dados.

Nesta seção serão introduzidos conceitos, aplicabilidade e principais soluções sobre compactação de dados, com ênfase no caso de dados armazenados em SGBDs. Por fim será analisado o caso especifico dos bancos de seqüências de DNA ou biosseqüências.

4.1 Compactação

A ***compressão de dados*** é a transformação ou codificação de um conjunto de símbolos em outro com um tamanho reduzido [LH87]. A compressão de dados é estudada dentro da teoria da codificação, uma das aplicações diretas mais relevante da teoria da informação [WIT06]. Alguns textos em inglês fazem a diferenciação entre os termos compactação e compressão, usando o primeiro para expressar as técnicas de codificação reduzida que não preserva toda a informação, neste trabalho estes termos serão usados como sinônimos.

Por ***codificação*** entende-se como um processo para levar de uma representação a outra. O dado transformado é chamado de ***código***. O primeiro componente de uma codificação é a ***entrada***, como por exemplo, um arquivo texto. O segundo componente é chamado de ***codificador***, e é onde está implementada toda a lógica da codificação, e por último, como saída, está o código gerado, a nova representação dos dados da entrada.

No entanto, a compressão de dados não é como outras codificações, como por exemplo, códigos para transmissão de dados, um protocolo, ou ainda codificações com o fim de resguardar a segurança da informação. A compressão de dados utiliza uma codificação como uma estratégia de armazenamento de dados e otimização de aplicações.

Definimos ***esquema de codificação*** como um algoritmo de codificação. Os dados de *entrada* são tratados como palavras, ou seja, a leitura vai acontecendo até ser encontrada uma seqüência de símbolos para gerar a codificação. Com relação ao tamanho dos dados de entrada, pode ser considerada como uma palavra cada caractere ou símbolo, ou ainda, seqüências de tamanho maiores. É possível ter entradas formadas por palavras de tamanho fixo ou tamanho variável. As palavras de tamanho fixo têm sempre o mesmo tamanho ou número de símbolos, já em tamanho variável, o conjunto é formado por palavras de tamanhos diferentes. Por exemplo, o alfabeto de entrada AE_1 = {a,e,i,o,u} contém palavras de tamanho fixo, e AE_2 = {"estava", "estávamos", "estavas", "estavam"}, palavras de tamanho variável.

Uma *codificação* é considerada ***distinta*** se existe um mapeamento de um para um entre as palavras do alfabeto de entrada para o código gerado. Ou seja, para cada palavra lida é gerado um único código, e vice-versa.

Para uma codificação distinta, se diz que a *saída* é ***independente do prefixo*** quando um código pode ser decodificado sem a necessidade de ler os caracteres que o seguem. Por exemplo, seja o conjunto C = {"1", "100000", "00"} e a palavra "1000000001" para ser decodificada. Ao ler o primeiro caractere não é possível definir a palavra correspondente no conjunto pré-definido dos códigos já que poderia ser a primeira opção, "1", ou a segunda "100000". E assim, ao ler o próximo caractere, "0", tampouco é esclarecido, pois ainda poderia ser a primeira palavra do conjunto e o início da terceira. Somente com a leitura do último caractere é possível saber, já que, se o número de zeros fosse ímpar, corresponderia à segunda palavra, mas neste caso, "1000000001" seria decodificada como "1", "00", "00", "00", "00", e "1" [LH87].

Por definição, a compressão de dados busca detectar redundância nos dados e tenta removê-la através de uma representação reduzida. Comumente isto é conseguido usando representações compactas para dados que mais se

repitam e outras mais longas para aqueles que apresentem menor freqüência. As redundâncias podem ser dos seguintes tipos [RV93]:

- Distribuição dos caracteres: dentro do domínio ou alfabeto de entrada é comum a ocorrência mais freqüente de alguns símbolos frente a outros. É possível tirar proveito dessa distribuição de freqüências para gerar códigos menores para os caracteres mais freqüentes e outros mais longos para os menos comuns.

- Repetição dos caracteres: é comum aparecerem varias repetições de um caractere. Essa seqüência pode ser substituída por uma única aparição do caractere e um valor indicando o número de repetições. Obviamente, esta estratégia só resultará em códigos menores quando o número de repetições for maior que dois. Um exemplo deste tipo de redundância é um número composto de vários zeros seguidos.

- Padrões: a freqüência de padrões é similar à distribuição de caracteres, porém para uma seqüência de caracteres ou símbolos. Por exemplo, em um texto em português é comum aparecer um ponto seguido de um espaço; esta seqüência pode ser substituída por um único caractere, gerando assim, um código mais enxuto.

- Redundância posicional: freqüência de um dado referente à posição dentro do arquivo de entrada. Um exemplo deste tipo de redundância ocorre em arquivos de imagens, onde uma linha pode repetir-se na mesma posição.

A seguir serão apresentadas algumas classificações de compactação.

4.1.1 Classificações

As estratégias de compactação podem ser classificadas de um modo geral em reversíveis e não-reversíveis. As ***não-reversíveis*** são geralmente utilizadas com arquivos de imagem, som e vídeo. Com esta técnica a redução do tamanho da representação é obtida descartando alguns dados e mantendo somente os mais relevantes. Sendo assim, no processo de descompressão não é possível obter o arquivo original. Como exemplo, podemos ter imagens com nitidez reduzida, sons com menor qualidade.

Por outro lado, as ***reversíveis*** são aquelas que ao aplicar o processo contrário à compressão é possível obter os dados originais sem perda de informação. Esta técnica é geralmente usada para arquivos de texto.

Os algoritmos de compressão também podem ser classificados em estáticos e dinâmicos. Os algoritmos ***estáticos*** são aqueles cuja relação entre uma palavra e sua representação compactada é única. Já para os ***dinâmicos ou adaptativos***, a codificação varia com o tempo. Por exemplo, ao codificar os primeiros dados, uma palavra pode aparecer com alta freqüência e, em conseqüência, ser representada por um código menor, enquanto para outras partes da leitura da entrada que está sendo codificada, a mesma palavra pode ser substituída por um código de maior tamanho.

Outro critério comumente adotado para agrupar estratégias de compressão de dados é a dependência em relação aos dados de entrada. Assim, existem soluções que exploram características próprias dos dados para gerar uma codificação. Estas soluções são chamadas de **semanticamente dependentes** ou **baseadas no contexto** *(context-based)***.** Por outro lado, as soluções que podem ser aplicadas em qualquer contexto são classificadas como **semanticamente independentes**.

A **eficácia** de uma compactação é medida pela taxa de compressão, dada pela fórmula apresentada no Figura 12. Nesta fórmula quanto maior o valor da taxa de compressão maior será a redução no arquivo de dado original.

$$\textbf{Taxa de compressão} = \frac{\textbf{Tamanho do arquivo original}}{\textbf{Tamanho do arquivo compactado}}$$

Figura 12 - Fórmula para Cálculo de Taxa de Compressão

A seguir são apresentados os principais algoritmos de compressão de dados aplicáveis a qualquer contexto. Serão expostas as características de cada um dos algoritmos de acordo as definições apresentadas acima, uma avaliação da taxa de compressão e complexidade.

4.1.2 Algoritmos de compressão reversível

Os algoritmos mais populares de compressão reversível podem ser agrupados em duas categorias:

1. Baseados em técnicas estatísticas.
2. Baseados em técnicas de substituição de caracteres.

A primeira categoria engloba os algoritmos que utilizam a freqüência de cada palavra da entrada para definir a nova representação, sendo que caracteres mais freqüentes são representados por um código de menor tamanho e os menos freqüentes, códigos maiores. Já nas técnicas baseadas na substituição de caracteres, ou modelagem baseada em dicionário, é construída uma tabela com as palavras que vão sendo lidas da entrada e associando códigos às mesmas. Em ocorrências posteriores ao registro da palavra na tabela, faze-se referência ao código associado na primeira leitura da palavra.

Um dos principais representantes da primeira categoria é o algoritmo de Huffman. Este esquema, que associa códigos às palavras segundo as probabilidades de ocorrência das mesmas na entrada, foi criado em 1952 por David Huffman [Huf52]. Esta solução gera códigos de tamanho variável a partir de palavras de tamanho fixo. Em arquivos texto, cada caractere é considerado como uma palavra a ser codificada, e os códigos gerados são independentes do prefixo, isto é, a cadeia de *bits* que representa uma palavra dada nunca é o prefixo da representação de outra palavra.

Existem variações estáticas e dinâmicas do algoritmo. Nas últimas, a tabela de freqüências dos caracteres iniciais, parte da lógica do codificador, vai sendo adaptada durante o processo de compressão e, posteriormente, na descompressão.

Como o código de Huffman pode ser aplicado a qualquer área de conhecimento, como arquivos textos, arquivos de imagens e sons, comprimindo *byte* a *byte*, a taxa de compressão pode variar dependendo das características da redundância dos dados de entrada.

Existem muitas adaptações e variações baseadas no algoritmo de Huffman, como as que realizam algum pré-processamento sobre os dados de entrada para melhorar a taxa de compressão ou ainda exploram características do contexto [Cor85]; assim como implementações que exploram estruturas de

dados distintas para armazenar a tabela de freqüências dos caracteres de maneira a obter melhor desempenho [Huf05].

Outra técnica baseada em estatísticas bastante explorada com bons resultados de compressão é a codificação aritmética [WI+87]. Esta técnica também é baseada em estatísticas sobre freqüências dos caracteres, porém usando números no intervalo [0,1] para gerar os códigos. Estudos comparando a taxa de compressão do algoritmo de Huffman e o da codificação aritmética divergem sobre a superioridade de um sobre o outro. Em geral o grau de compressão da codificação aritmética é tão bom quanto o valor obtido por Huffman para a mesma entrada [RV93].

A lista de algoritmos de compactação é extensa, alguns com pequenas variações sobre os mais tradicionais, outros combinando várias técnicas, mas ainda vale ressaltar aqui outro exemplar do grupo de algoritmos baseados em estatísticas. Trata-se do PPM, *Prediction by Partial Match* [CT97], que apresenta uma alta taxa de compressão prevendo a probabilidade do próximo símbolo, usando para isto um contexto formado por alguns símbolos precedentes. O número de símbolos precedentes a ser considerado é um parâmetro de entrada para o algoritmo e é designado com a ordem do algoritmo.

O algoritmo conhecido como LZ, (Lempel-Ziv) [ZL77], está baseado na substituição de caracteres. Gera códigos de tamanho fixo a partir de palavras de tamanho variável. As palavras vão sendo registradas em uma tabela, associadas a um código. Quando uma palavra lida já se encontra na tabela, é gerado como código o valor associado a ela na primeira aparição. Sendo assim, o LZ é uma solução dinâmica.

Entre os esquemas de compressão que exploram as características dos dados, aqueles semanticamente dependentes, pode-se citar o *Run-Length*, *Difference Mapping*, até os mais populares hoje em dia como o MPEG (*Moving Pictures Experts Group*), que tratam do armazenamento e transmissão de arquivos de imagem, explorando as redundâncias espaciais e temporais deste tipo de dado; também vale citar o esquema MP3, *MPEG-1/2 Audio Layer 3*, que comprime áudio com eficiência sem perda substancial de qualidade, cuja técnica de compressão se baseia em estudos sobre psico-acústica, retirando da música o que o ouvido humano não conseguiria perceber, devido às limitações da audição humana, pelos fenômenos de *mascaramento* de sons.

Existem outros esquemas para tratar a compressão de diferentes tipos de dados, como aqueles presentes em sistema de banco de dados relacionais, explorando-se a arquitetura destes sistemas e as características dos diferentes tipos de dados contidos nessas aplicações, assim também como outras áreas de pesquisa como a biologia. Na seção seguinte são detalhados os requisitos dos sistemas de banco de dados e os esquemas de compressão propostos para esta área. Em seguida, são apresentadas as soluções para as biosseqüências, objeto de estudo deste trabalho.

4.2 Compactação em SGBDs

A estruturação, acesso e manipulação de grandes volumes de dados sempre foi uma questão importante na construção de aplicações denominadas intensivas sobre os dados. De um modo em geral, foi observado que subconjuntos desses grandes volumes de dados compartilham propriedades e estruturas de dados similares[RV93], como por exemplo o formato de um campo para armazenar números de telefone.

Os SGBDs foram criados com o objetivo de permitir que aplicações intensivas sobre dados possam ser construídas mais facilmente fornecendo um grau de transparência quanto às estruturas dos dados utilizadas e a forma de acessá-los e manipulá-los. Além disso, SGBDs visam oferecer mecanismos eficientes para operar sobre esses volumes de dados. Diante deste desafio, surgiu um modelo específico de banco de dados denominado de SGBD relacional (SGBDR) o qual introduziu a abstração de tabela (relação, da matemática) permitindo assim o armazenamento e manipulação de dados relacionados de forma mais homogênea, ou seja, fazendo uso da propriedade de similaridade e relacionamento apresentado nos grandes volumes de dados.

Apesar do sucesso obtido pelo SGBDs relacionais, é grande o desafio de se garantir um bom desempenho diante de grandes volumes de dados que continuam a crescer constantemente. Em domínios específicos como da biologia, se observa uma duplicação a cada 14 meses das fontes de dados biológicas[3] , além do surgimento de dados com semânticas específicas como é o

[3] Fonte: http://www.ncbi.nlm.nih.gov/Database/index.html.

caso das biossequências, cujas propriedades foram apresentadas no capítulo 2. Desta forma, existe um grande desafio para SGBDs tradicionais em lidar com esses novos domínios que apresentam um crescimento constante no volume de dados.

De modo geral, os SGBDs se valem do acesso rápido à memória principal para reduzir o tempo de latência da aplicação no acesso aos dados que serão úteis para seu processamento. Manter páginas de dados em memória principal para provável uso futuro garante também maior eficiência no acesso aos dados. Enxergando o SGBD como um produtor de dados para as aplicações que os consomem, identificam-se três fatores que possuem influência direta na eficiência do SGBD em responder às demandas das aplicações: o volume de dados a ser manipulado, a natureza do dado a ser manipulado e o padrão de consumo dos dados das aplicações.

A natureza do dado está relacionada com as propriedades intrínsecas dos dados, como no caso de texto, imagem, vídeo, e com sua forma de representação, numérica, alfanumérica ou binária. Um SGBD faz uso dessas propriedades para definir estruturas de dados específicas aliadas a métodos de acesso que as manipulam. Este é o caso, por exemplo, de dados multidimensionais que necessitam de estruturas específicas para sua manipulação e acesso como, por exemplo, as conhecidas árvores R [Gut84].

O padrão de consumo de dados é definido a partir do modo como a aplicação faz uso deste dado. Por exemplo, uma determinada aplicação pode consumir os dados seqüencialmente, usando uma ordem pré-definida ou de forma aleatória. Os SGBDs utilizam técnicas para interpretar as demandas das aplicações trabalhando de forma pró-ativa no sentido de antever as suas futuras necessidades. O exemplo mais conhecido desta estratégia são as leituras antecipadas de dados (*prefetch reads*) visando trazer páginas de dados para a memória principal antes mesmo que estes sejam solicitados pela aplicação.

Como comentado no início do capítulo, a compressão tem como objetivo principal detectar as redundâncias nos dados e removê-las através de uma representação reduzida. Aplicando-se esse objetivo aos SGBDs é possível concluir que a compressão aplica-se ao fator referente ao volume de dados como forma de reduzir o número de operações de entrada e saída. Os dados compactados ocupariam menos espaço em disco e, com isso, um menor número

de blocos (ou páginas em disco) será necessário, resultando na redução do tempo gasto em operações de entrada e saída(E/S). Desta forma, o SGBD lê os dados compactados para a memória principal e depois descompacta em memória principal para seu uso posterior. Este tipo de abordagem, no entanto, traz algumas limitações com relação ao desempenho. Afinal o ganho obtido na redução de operações de entrada e saída precisa ser maior que o custo de descompactação em memória principal [WKH+00].

Os sistemas de bancos de dados apresentam algumas características que poderiam ser exploradas com sucesso ao considerar a compressão de dados.

- Grande quantidade de valores nulo, brancos ou zeros;
- Geralmente os dados não cobrem o domínio inteiro, apresentando uma distribuição de valores restrita ao um subconjunto do mesmo;
- Alguns valores acontecem com maior freqüência que outros;
- Repetições de atributos, como os atributos de junção;
- Necessidade de compressão, para economia de espaço, quando o volume de informação é grande;
- Consultas geralmente são processadas como um *pipeline*, ou seja, poucos dados são processados por vez.

Porém, o uso de dados comprimidos pelo SGBD levanta uma questão com relação às demandas das aplicações. Por exemplo, as consultas serão tratadas de forma compactada ou descompactada? Se forem tratadas de forma descompactada será necessária a descompactação prévia dos dados antes que estes possam ser usados para responder às consultas. Caso contrário, a consulta também precisará ser compactada para que seja gerado um plano de consulta consistente com os dados compactados [GHS95]. Desta forma, os métodos de acesso que se privilegiam dessas estruturas de dados precisam ser modificados para que possam operar sobre estruturas compactadas.

Em [GHS95] são apresentadas restrições quanto ao uso dessa abordagem de consultar diretamente os dados compactados, já que deve-se levar em conta os tipos das operações que manipulam tais dados. Caso as operações não sejam de igualdade, como por exemplo, maior ou igual, será necessário um algoritmo de compressão que mantenha a ordem dos dados.

Vale ressaltar como vantagens de compressão no contexto de banco de dados:

- Redução do tempo de busca em disco, já que os dados ocuparão uma porção menor do disco;

- Melhora na taxa de transferência de disco devido ao aumento da densidade da informação dos dados transferidos;

- Aumento da taxa de *hit* no *buffer* durante o processamento de uma consulta já que uma porção maior do banco de dados caberá no *buffer pool*. Isto só ocorrerá quando a implementação considerar a consulta diretamente sobre os dados compactados;

Entre as desvantagens advindas do uso de compactação em banco de dados podemos ressaltar:

- A sobrecarga de processamento devido à compressão e descompressão dos dados;

- Consultas com predicados envolvendo desigualdades não serão possíveis para esquemas de compressão que não preservem a ordem dos dados, no caso da implementação de consultas sobre dados compactados;

A compactação de dados é utilizada na maioria dos SGBDs comerciais. Como por exemplo, a versão mais atual do Oracle[OB05], o Oracle 10G, comprime os dados de um bloco ou página de disco criando uma tabela no início de cada bloco com os valores repetidos dentro das linhas de dados contidas no bloco. Desta maneira, cada bloco é auto-contido, ou seja, para descompactar um bloco é necessário somente o bloco. Outra vantagem da utilização desta técnica, do ponto de vista de implementação, deve-se a alteração do sistema apenas nas funções que manipulam blocos de dados, sendo transparente para os demais níveis de manipulação dos dados.

Também o DB2[RAV06] conta com uma forma de persistência compactada que explorar a freqüência de repetição dos valores em cada fila de uma tabela que pode ser utilizada pelos administradores do sistema. Este recurso é chamado *Venom*.

4.3 Compactação de biosseqüências

Como já mencionado, a idéia por trás desta técnica é eliminar as redundâncias de certas porções da seqüência usando uma codificação reduzida. Entretato, a simples aplicação de esquema para compressão baseado em estatísticas, por exemplo, não garante bons resultados de compactação em todos os casos, já que para as seqüências biológicas, mesmo com a presença de regiões com repetição de padrões e redundância, a regularidade da aparição de um caractere de acordo ao contexto não é tão simples como com textos comuns em língua portuguesa, por exemplo.

Ao mesmo tempo a preponderância de padrões de repetição é um fenômeno importante nas seqüências biológicas. Estas repetições podem ser randômicas muito provavelmente devido ao tamanho dos arquivos de seqüências em relação ao alfabeto bastante pequeno, ou ainda podem representar alguma informação biológica como a mutação no caso de uma duplicação de um gene[AZM+02].

Por outro lado, considerando que as seqüências de DNA são formadas por quatro letras (a, c, t, g), excluindo as ambigüidades, e as de proteínas por possíveis vinte letras, uma idéia simples de compressão usaria somente 2 *bits* para representar cada base no primeiro caso e 5 *bits* no segundo. Desta maneira, os algoritmos de compressão a serem testados com este tipo de arquivo, para garantir uma taxa de compressão ainda maior, devem usar menos *bits* que os citados acima.

Tais fatos e o estudo das propriedades deste tipo especial de texto, as biosseqüências, como a existência de palíndromos e repetições exatas ou aproximadas, e outras apresentadas no capítulo 3, acabam gerando um novo desafio para a compressão de dados, já que é possível verificar que estes dados apresentam características tais que, uma vez exploradas, poderiam gerar esquemas com taxas boas de compactação, talvez melhores que com a utilização de esquemas tradicionais como Huffman, codificação aritmética entre outros.

Deste modo, as biosseqüências são candidatas à compressão por apresentar redundância de dados. A chave do problema se torna como identificar dentro de um tempo razoável as estruturas de repetição, e como fazer

uso deste conhecimento biológico para comprimir a seqüência completa. Diferentes algoritmos específicos, com diferentes graus de eficiência, foram propostos para a compressão de biosseqüências. As soluções variam principalmente no tipo de repetição que exploram e como o fazem [AZM+02].

Em geral, os esquemas específicos para compactação de biosseqüências estão baseados nas técnicas de substituição de caracteres, como o algoritmo LZ [LZ77]. Uma alternativa para os algoritmos baseados no método de LempelZiv foi proposta [TKR03], determinando uma nova linha baseada no modelo de analogia máxima, ou NML (*Normalized maximum Likehood model*).

O primeiro algoritmo específico para compactação de seqüências de nucleotídeos, chamado de *Biocompress,* foi publicado por Grumbach e Tahi [GT93]. Este algoritmo segue o estilo LempelZiv[ZL77] baseado em substituições e explora as repetições de padrões nas biosseqüências, como a presença de palíndromos. Os resultados obtidos pelo *Biocompress* representaram melhoras consideráveis na taxa de compressão em relação aos algoritmos clássicos aplicados às biosseqüências, porém o tempo de execução é alto.

O *DNACompress*[CLM+02] é considerado uma das melhores soluções baseada em esquema de compressão por substituição para seqüências biológicas. É executado em dois passos. No primeiro, um programa chamado *PatternHunter* é usado para encontrar padrões significativos, e no segundo, os padrões encontrados no primeiro passo são codificados, sendo representados por um ponteiro para a posição na qual o padrão aparece previamente. Grande parte do sucesso do *DNACompress* se deve a disponibilização de funções para realizar buscas aproximadas, uma tarefa executada com freqüência na Biologia. Sendo assim, é possível concluir que a integração deste tipo de busca aos algoritmos de compressão pode gerar bons resultados nos acessos futuros aos dados.

Outra solução interessante nesta linha é a apresentada em [AZM+02]. A entrada é dividida em blocos e os dados de cada bloco são ordenados lexicograficamente, criando um agrupamento de dados semelhantes conveniente para a compressão. Esta abordagem de ordenação em bloco é chamada de BWT, *Burrows-Wheeler Transform*. É usualmente realizada em três ou quatro etapas, como um *pipeline*. Mais informação sobre o funcionamento deste algoritmo pode ser encontrada no Apêndice D.

Na segunda linha de algoritmos próprios para biosseqüências está a proposta de [TKR03]. O método é conhecido como *NMLComp*, um modelo normalizado para aproximação (*Normalized maximum Likehood model*). Esta implementação gerou resultados que comprovaram a redução nos requisitos computacionais deste método sem perder a precisão, já que é comprovadamente adequado para codificação a ser utilizada em buscas aproximadas.

Em [KT05] é discutida uma otimização mais flexível deste método que conseguiu representar cada base de uma biosseqüências com 1,490 *bits* ao ser testado contra um arquivo contendo seqüências do genoma humano.

4.4 Considerações Finais

Como explicada na seção 4.1.1, a eficácia de um algoritmo de compressão é medida pela taxa de compressão. Isto é, para dizer que um algoritmo é melhor que outro se deve comparar as taxas de compressão dos mesmos, obviamente dentro de um esquema preciso e correto de testes.

Porém, do ponto de vista do desempenho de um programa, tantos aspectos teóricos, da concepção em si do algoritmo, quanto aspectos práticos, diretamente da implementação, influenciam o resultado final.

Geralmente, na publicação de um novo algoritmo, ou utilização de um algoritmo existente em um novo domínio de dados, é suficiente a analise do primeiro item, ou seja, a taxa de compressão. Tem-se como primeiro problema, provar que os dados serão realmente compactados. Como segundo objetivo, é importante conhecer o desempenho do programa, principalmente em alguns domínios como o caso de banco de dados, nos quais o desempenho da descompressão afeta diretamente o desempenho do sistema como um todo.

Desta maneira é comum encontrar tabelas de comparação de programas de compressão de dados em publicações da área. A seguir, é apresentada a **Tabela 8**, encontrada em [BW94], com alguns valores obtidos sobre os 14 arquivos da Calgary Compression Corpus[WB06]. Estes arquivos possuem formatos variados e são comumente utilizados para comparações entre algoritmos de compressão. O tamanho total dos 14 arquivos originais é de 3.141.622 bytes. Os programas *compress* e *gzip* são baseados na família LZ, já

o *comp-2* é baseado no modelo estatístico. Como é possível observar na tabela, os programas de compressão podem apresentar desempenhos diferentes para a compressão e descompressão. Por outro lado, comprova-se que nem sempre o algoritmo que obtém uma boa taxa de compressão em comparação a outras implementações, é o mais adequado para ser usado em uma descompressão *online*, por exemplo, como o caso do programa comp-2.

Programa	**Tempo total CPU**		**Tamanho do arquivo final**	**Taxa de compressão**
	compressão	*descompressão*	*(bytes)*	*bits/char*
compress	9,6	5,2	1.246.286	3,63
gzip	42,6	4,9	1.024.887	2,71
BWT	51,1	9,4	856.233	2,43
comp-2	603,2	614,1	848.885	2,47

Tabela 8 - Comparação de programas de compressão de dados

Assim também, com esta tabela é possível observar que o algoritmo BWT além de apresentar uma taxa de compressão comparável ao modelo estatístico, tem um tempo de processamento próximo aos algoritmos baseados na família LZ, considerado os melhores[AZM+02]. Do ponto de vista da implementação, sua construção se vê facilitada por utilizar algoritmos que facilmente podem conseguir-se na literatura[Huf05] em versões otimizadas. Outra característica importante desta solução, que poderá ser útil na tentativa de compactação de biosseqüências, é a possibilidade de compactação e descompactação em blocos.

4.5 Conclusão

Neste capítulo foram apresentados os conceitos da compressão de dados em geral. Também foram introduzidos os problemas e as vantagens da utilização de compressão de dados em sistemas de banco de dados.

Por último, dado o foco do trabalho, foram resumidos os desafios do uso das técnicas de compactação sobre os tipos de dados das biosseqüencias, sendo apresentadas as principais soluções aplicadas a tais dados.

Desta maneira, foi possível constatar que a base para os algoritmos de compactação de dados é bastante antiga, e que geralmente novas soluções são

criadas mediante combinações ou pequenas alterações das principais estratégias como Huffman, Codificação Aritmética e os algoritmos da família Lempel-Ziv. Quando uma nova área apresenta um tipo de dado que tenha característica que o torna um candidato à compactação, é adequado começar pelas soluções mais tradicionais.

Em resumo, o uso de técnicas de compactação em banco de dados abre uma série de oportunidades para melhorar o desempenho de consultas intensivas sobre os dados ao diminuir o número de operações de E/S requeridas. Para sistemas de banco de dados, aperfeiçoar o número de operações de E/S é muito importante e a compressão poderia ser adotada em tais sistemas na busca de uma melhora nos tempos de execução de consultas intensivas sobre os dados. Porém uma vez em memória, os dados devem ser descompactados para finalmente resolver as consultas. Sendo assim, o uso de técnicas de compressão não pode visar somente a economia de espaço, senão também conseguir que a descompressão seja feita em memória e com uma sobrecarga de execução razoável que não pese no tempo total da execução de uma consulta.

Baseado no estudo apresentado neste capítulo foi escolhido utilizar o algoritmo BWT como solução de compactação de dados de biosseqüencias no intuito de melhorar o desempenho do programa BlastP em relação ao número de operações de E/S, por apresentar uma taxa de compressão comparável aos melhores e um desempenho próximo às implementações baseadas na família LZ [AZM+02].

A proposta é realizar a compactação uma vez formatado o banco de biosseqüências com o aplicativo FORMATDB previamente à execução do BlastP, e descomprimir em tempo de execução utilizando diferentes estratégias inspiradas nas soluções para compactação em SGBDs. No capítulo seguinte será apresentada a proposta em detalhe.