3
Arquitetura para tratamento e compressão de dados de sensores

3.1
Visão geral

A arquitetura proposta neste trabalho especifica um processo para o tratamento e compressão de dados adquiridos por sensores. Este processo inclui as etapas descritas abaixo.

Figura 2: Arquitetura da solução de compressão

Em linhas gerais, para compactar uma série histórica de N medições, é necessário que estas sejam carregadas nas estruturas de dados básicas. Neste momento, não estamos preocupados com o tempo em que as medições foram realizadas. Na seqüência, a transformada Wavelet é aplicada utilizando uma função

base pré-definida obtendo como resultado uma lista de N coeficientes. Como visto anteriormente, o algoritmo que implementa a transformada organiza os coeficientes de forma a posicionar aqueles que representam sinal de alta freqüência no final do conjunto. Então, de acordo com o critério definido para o descarte, estes coeficientes que representam a componente de alta freqüência do sinal são eliminados. Neste ponto a quantidade de coeficientes eliminados pode ultrapassar noventa por cento. Os coeficientes restantes sofrem um processo de suavização cuja intensidade também é configurável. Neste processo, alguns coeficientes que satisfaçam à condição básica são aproximados para zero, ou seja, passam a não representar variação no sinal.

Com uma quantidade bastante reduzida de coeficientes, já existe um estágio inicial de compactação, visto que, se armazenarmos apenas os coeficientes, conseguiremos reconstruir o sinal com a inversa da transformada Wavelet usada. Porém, como estamos buscando otimizar ao máximo a compressão, usaremos um mecanismo de codificação baseado na representação ótima de seqüências repetidas. No entanto, como a lista de coeficientes é uma lista de números naturais, podemos ter uma infinidade de valores muito próximos, mas que não sejam exatamente iguais. Para resolver este problema, antes de realizarmos a codificação, os coeficientes devem ser quantizados. O resultado desta etapa é uma estrutura conhecida como tabela de quantização, onde cada elemento contido nela é o valor representativo de uma série de coeficientes que foram classificados em um determinado intervalo estatístico. Uma vez calculada esta tabela, a lista dos coeficientes é percorrida e cada coeficiente é substituído pelo índice da tabela onde se encontra o seu valor representativo. O resultado deste processo é uma lista de números inteiros com alta freqüência de repetição, o que nos dá o cenário ideal para que um algoritmo de codificação possua resultados ótimos. Assim sendo, podemos aplicá-lo e seu resultado será uma tabela de símbolos com suas determinadas representações binárias. Detalhes desta etapa podem ser vistos na seção anterior.

Neste momento, concluímos o processo de compressão do sinal independente do tempo, todavia se faz necessário armazenarmos as informações relativas à data e hora onde cada medição ocorreu a fim de, no futuro, restabelecer sua relação temporal. Para codificar as datas e horas das medições, o algoritmo percorre toda série temporal e, para cada dia, coleta estatísticas como: quantidade de medi-

ções no dia, quantidade de medições por hora e cinco pontos chave contendo minuto para cada hora. Desta forma podemos garantir que os pontos estarão relacionados no tempo com precisão de hora, sendo que as componentes de minutos e segundos serão inferidas com base no número de pontos medidos naquela hora e nos pontos chave que servirão como guias.

Para realizar a descompactação, o processo é semelhante, porém pelo caminho inverso. Iniciamos com o arquivo binário, o qual deve ser carregado nas estruturas internas. A primeira estrutura a ser restabelecida é a lista de dias compactados. Como o tamanho do conjunto é conhecido, o algoritmo pode estimar as medições de data e hora baseada nas estatísticas coletadas no momento da compressão. As outras estruturas do arquivo correspondem à tabela de quantização e àquelas geradas pelo algoritmo de codificação, que deve ser executado para restaurar a lista de números inteiros referentes aos índices da tabela, que por sua vez não está compactada. Com a lista de números inteiros recuperada, ela deve ser percorrida substituindo cada valor pelo contido na tabela de quantização na posição indicada pelo número. O resultado desta etapa é uma lista de números naturais que se aproximam dos coeficientes originais. A precisão apresenta perdas aceitáveis introduzidas pelo processo de quantização. Com os coeficientes restabelecidos (com alguma perda), podemos executar a transformada Wavelet inversa e restaurar, de forma aproximada, os valores medidos originalmente. Esta aproximação se dá devido ao descarte dos componentes de alta freqüência e da suavização dos coeficientes restantes, sem esquecer, evidentemente, da perda introduzida pela quantização. Como resultado, temos um sinal mais limpo e fácil de analisar.

3.2 Transformada Wavelet

Existem diversas funções base para aplicações de Wavelets, cada qual com suas características específicas. O que diferencia cada uma delas é sua capacidade de aproximação. Na seção de preliminares, para efeito didático, tomamos como exemplo a função base de Haar que foi selecionada devido à sua simplicidade. Por ser uma função descontínua, sua aproximação não é a melhor dentre as funções base, mas sua utilização é bastante popular, pois sua simplicidade permite gerar códigos muito rápidos. No entanto, para este trabalho, escolhemos uma função

mais complexa com um potencial de aproximação melhor, conhecida como D4, por usar quatro coeficientes de escala. Esta função é uma das funções ortonormais com suporte compacto desenvolvidas por Ingrid Daubechies que produz médias mais precisas do que as funções de Haar, principalmente em resoluções mais baixas. Este fato foi decisivo na escolha da função, pois quanto mais baixa a resolução, menos coeficientes de detalhe são necessários para representar o sinal e conseqüentemente maiores são as taxas de compressão adquiridas com melhor qualidade do sinal comprimido.

As funções base desenvolvidas por Daubechies são inúmeras e variam basicamente na quantidade de coeficientes de filtro usados. Selecionamos a D4, pois o custo benefício é o melhor para os propósitos deste trabalho, uma vez que oferece boa capacidade de aproximação e usa um número relativamente pequeno de coeficientes de escala, o que torna o código bem rápido no processamento. Abaixo segue a função D4.

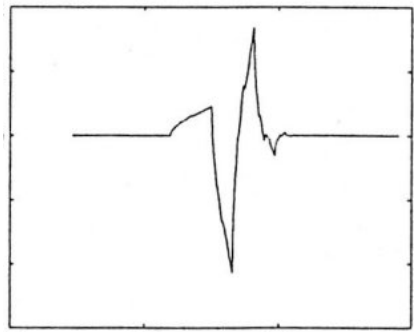


Figura 3: Função base Daubechies D4

Existe uma vasta literatura sobre transformadas Wavelet ([14], [15], [16], [17]) e a conceituação matemática desta técnica está fora do escopo deste trabalho. No entanto, explicaremos de forma resumida o funcionamento do algoritmo, que é detalhado em [21].

Assim como a transformada rápida de Fourier (FFT), a transformada wavelet discreta (DWT) é uma operação linear muito rápida que opera em um vetor de dados cujo tamanho é uma potência inteira de dois.

Para entendermos como funciona a transformada wavelet D4, vamos considerar que seus quatro coeficientes de filtro sejam: C_0, C_1, C_2 e C_3. Então a matriz de transformação seria:

$$T = \begin{bmatrix}
c_0 & c_1 & c_2 & c_3 & & & & & & & & \\
c_3 & -c_2 & c_1 & -c_0 & & & & & & & & \\
 & & c_0 & c_1 & c_2 & c_3 & & & & & & \\
 & & c_3 & -c_2 & c_1 & -c_0 & & & & & & \\
\vdots & \vdots & & & & & \ddots & & & & & \\
 & & & & & & & c_0 & c_1 & c_2 & c_3 & \\
 & & & & & & & c_3 & -c_2 & c_1 & -c_0 & \\
c_2 & c_3 & & & & & & & & c_0 & c_1 & \\
c_1 & -c_0 & & & & & & & & c_3 & -c_2 &
\end{bmatrix}$$

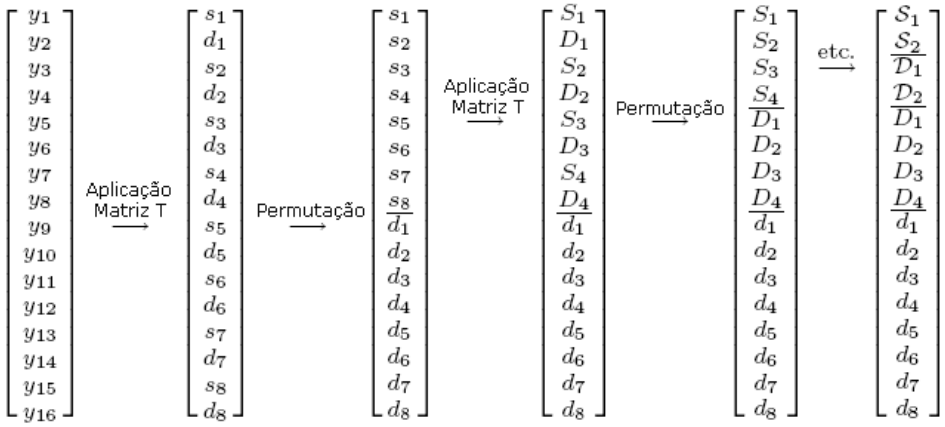
As entradas em branco significam zeros nesta matriz. Para realizarmos a transformada devemos multiplicá-la pelo vetor coluna contendo os valores numéricos. O objetivo desta operação é realizar duas convoluções relacionadas, removendo metade dos valores resultantes de cada uma delas e reorganizando as duas metades resultantes em um mesmo conjunto final.

Não nos aprofundaremos nos métodos utilizados por Ingrid Daubechies para calcular os coeficientes utilizados, pois são facilmente encontrados numericamente tabulados na literatura [21]. Os valores seguem abaixo:

$$C_0 = (1+\sqrt{3})/4\sqrt{2}$$
$$C_1 = (3+\sqrt{3})/4\sqrt{2}$$
$$C_2 = (3-\sqrt{3})/4\sqrt{2}$$
$$C_3 = (1-\sqrt{3})/4\sqrt{2}$$

A transformada Wavelet discreta (DWT) consiste na aplicação da matriz T de coeficientes Wavelet hierarquicamente. Primeiro é aplicada a todo vetor coluna com os dados originais de tamanho N. Depois aplicado ao vetor suavizado de N/2 posições, depois no suavizado do suavizado de N/4 posições e assim sucessivamente até que reste apenas um vetor com duas (2) posições. Este procedimento é conhecido como *algoritmo piramidal*. O resultado da DWT consiste em um vetor coluna contendo estes dois valores super-suavizados e todos os componentes de

detalhe acumulados ao longo das rodadas totalizando N valores, como mostrado no exemplo abaixo:



No caso do nosso vetor coluna de entrada, N é igual a 16. Em casos onde o vetor de entrada seja maior, mais passos intermediários serão necessários.

No vetor resultante, S1 e S2 são conhecidos como "coeficientes da função mãe" enquanto os outros são os coeficientes de detalhe. Como podemos notar, os coeficientes mais ao final do vetor são aqueles que representam os detalhes mais minuciosos.

Abaixo segue o código referente ao algoritmo piramidal para a transformada Wavelet de uma dimensão [21]:

```
void wt1(float a[], unsigned long n, int isign,
      void (*wtstep)(float [], unsigned long, int))
{
      unsigned long nn;

      if (n < 4) return;
      if (isign >= 0) {
            for (nn=n; nn>=4; nn>>=1) (*wtstep)(a,nn,isign);
      } else
        {
            for (nn=4; nn<=n; nn<<=1)
            (*wtstep)(a,nn,isign);
        }
}
```

Note que a função **wt1** recebe como parâmetro o ponteiro para uma outra função **wtstep**, no caso, que implemente uma função Wavelet mãe. Para este trabalho, implementamos D4:

```
#include "nrutil.h"
#define C0 0.4829629131445341
#define C1 0.8365163037378079
#define C2 0.2241438680420134
#define C3 -0.1294095225512604

void daub4(float a[], unsigned long n, int isign)
{
        float *wksp;
        unsigned long nh,nh1,i,j;

        if (n < 4) return;
        wksp=vector(1,n);
        nh1=(nh=n >> 1)+1;
        if (isign >= 0) {
                for (i=1,j=1;j<=n-3;j+=2,i++) {

        wksp[i]=C0*a[j]+C1*a[j+1]+C2*a[j+2]+C3*a[j+3];
        wksp[i+nh] = C3*a[j]-C2*a[j+1]+C1*a[j+2]-C0*a[j+3];
                }
                wksp[i]=C0*a[n-1]+C1*a[n]+C2*a[1]+C3*a[2];
                wksp[i+nh] = C3*a[n-1]-C2*a[n]+C1*a[1]-C0*a[2];
        } else {
                wksp[1]=C2*a[nh]+C1*a[n]+C0*a[1]+C3*a[nh1];
                wksp[2] = C3*a[nh]-C0*a[n]+C1*a[1]-C2*a[nh1];
                for (i=1,j=3;i<nh;i++) {

        wksp[j++]=C2*a[i]+C1*a[i+nh]+C0*a[i+1]+C3*a[i+nh1];
        wksp[j++] = C3*a[i]-C0*a[i+nh]+C1*a[i+1]-C2*a[i+nh1];
                }
        }
        for (i=1;i<=n;i++) a[i]=wksp[i];
        free_vector(wksp,1,n);
}
#undef C0
#undef C1
#undef C2
#undef C3
```

O parametro **isign** da função executa a transformada inversa se for passado como **–1**. Os outros parâmetros são o vetor **a[]** contendo os dados originais e **n** contendo o tamanho de **a**, obrigatoriamente uma potência de dois.

3.3
Suavização por descarte de coeficientes

De acordo com a natureza do negócio monitorado, uma série histórica pode conter tanta informação que talvez seja possível abrir mão de algo sem prejudicar sua análise, principalmente se estivermos falando de longos períodos de tempo. Por exemplo, se analisarmos 365 dias de dados originados por sensores de pressão no fundo de um poço de petróleo, com freqüência média de uma medição por minuto, podemos atentar para os seguintes fatos:

- Sinal composto por aproximadamente 524.288 medições
- Grande probabilidade da adição de ruídos ao sinal
- Não há necessidade de precisão de minutos e segundos
- Maior interesse no comportamento geral da curva
- Deve conter detalhes suficientes para permitir a detecção de padrões de comportamento.

Sabemos que ao aplicarmos a transformada Wavelet em uma série de 524.288 pontos, teremos uma lista de coeficientes de igual tamanho. Além disso, sabemos que esta lista os coeficientes representativos nas duas primeiras posições e coeficientes de detalhes nas posições posteriores sendo que, quanto mais no final da lista, mais detalhe ele representa. Desta forma, consideramos que a maior parte dos componentes de alta freqüência ficará no final da lista, isto inclui:

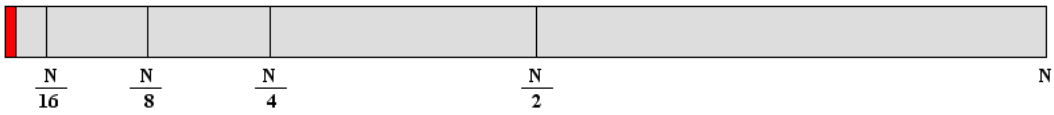
- Coeficientes causadores de mudança rápida nos detalhes
- Coeficientes de Freqüência alta (efeito tremido)
- Coeficientes representando ruídos causados por pequenas variações na medição dos sensores

Desta forma, quando existe uma boa quantidade de pontos, podemos diminuir a resolução da curva sem perda expressiva de detalhes. Para sabermos o

que descartar na lista de coeficientes basta entendermos como os coeficientes de detalhe são gerados ao longo do processo. Para isso, basta saber que, a partir dos valores médios representativos da curva situados nas primeiras duas posições da lista, podemos dar um passo em direção à diminuição de resolução de acordo com a tabela abaixo. Note que, por definição, quanto maior o nível de resolução, menos coeficientes são utilizados e conseqüentemente, menos detalhes a curva passa a possuir, ou seja, quanto maior o nível de resolução menor a resolução da curva.

Nível de Resolução	**Nº. de Coeficientes utilizados**
R	**$N/(2^R)$**
...	**...**
4	**N/16**
3	**N/8**
2	**N/4**
1	**N/2**
0	**N**

Configuração geral da lista de coeficientes:



As primeiras duas posições são os coeficientes de aproximação geral da curva (vermelho), os outros coeficientes de detalhes podem ser removidos de acordo com a resolução desejada para a análise do sinal, sempre do final para o início, respeitando o fato de que a proporção a ser descartada sempre é uma potência de dois (2).

Para baixarmos a resolução ao nível 1, podemos considerar apenas a lista de coeficientes da posição inicial até n/2. Desta forma, da posição (n/2 + 1) até a posição n, todos os coeficientes serão zerados. Para visualizarmos melhor o efeito deste descarte, recorremos ao exemplo da seção anterior, onde temos a seguinte série original:

[9 7 3 5]

Após a aplicação da transformada Wavelet usando a função base de Haar, obtivemos o seguinte resultado:

Resolução	Aproximação	Detalhes
4	[9 7 3 5]	
2	[8 4]	[1 -1]
1	[6]	[2]

[6 2 1 -1]

Com este resultado conseguimos restabelecer o sinal original sem nenhuma perda. Agora vejamos o que acontece quando descartarmos metade dos coeficientes. O resultado da transformada passa a ser:

[6 2 0 0]

Para restabelecer o sinal original o processo seria:

Resolução	Aproximação	Detalhes
1	[6]	[2]
2	[8 4]	[0 0]
4	[8 8 4 4]	

Logo, podemos comparar os resultados:

Original:

[9 7 3 5]

Restaurando depois da transformada com descarte de 50% dos coeficientes:

[8 8 4 4]

Graficamente podemos comparar as duas curvas:

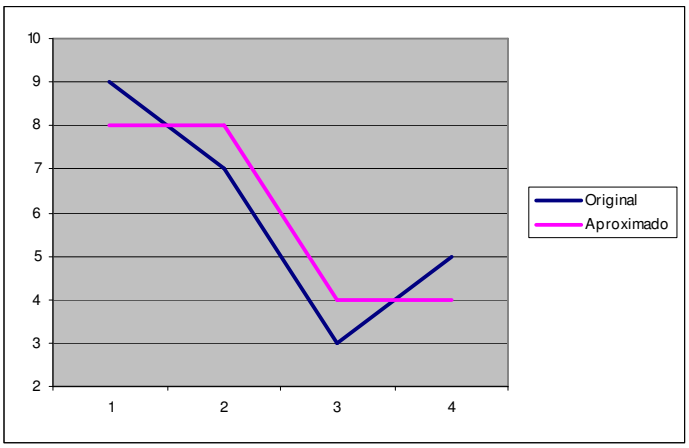


Figura 4: Exemplo de aproximação por Haar

No caso deste exemplo, a aproximação não possui uma precisão boa visto que o sinal original contém apenas 4 pontos, mas não é difícil imaginar que com uma série longa as duas curvas estarão muito mais próximas. Vejamos, como exemplo, um sinal com aproximadamente 2.500 pontos, usando função base Daubechies4.

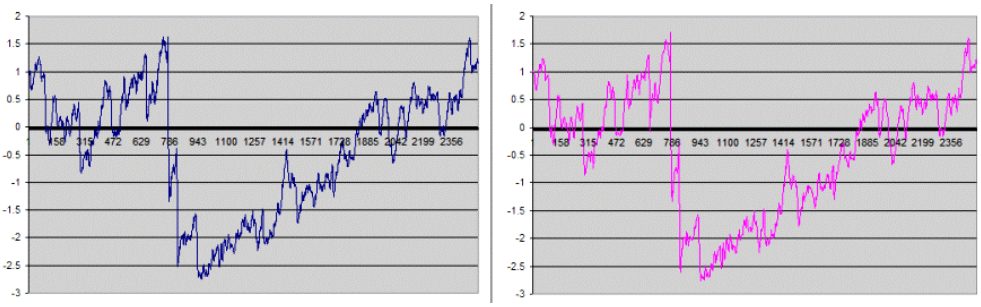


Figura 5: Original X Usando 50% dos coeficientes com D4

Usando os mesmos conceitos, podemos baixar a resolução da curva até onde for conveniente, apenas descartando coeficientes na proporção de potência de dois 2. Vejamos a tabela:

Resolução	Coeficientes usados	% coeficientes usados
N	[1...N]	100%
N/2	[1...N/2]	50%
N/4	[1...N/4]	25%
N/8	[1...N/8]	17,5%
N/16	[1...N/16]	8,25%
...	...	...

Podemos perceber então que armazenar apenas a lista dos coeficientes válidos, ao invés do sinal completo, nos trás a vantagem de poder representá-lo com menos informação, além de remover componentes de alta freqüência responsáveis por representarem ruídos ou detalhes minuciosos demais. Logo, podemos considerar o descarte de coeficientes como um método de compressão com perdas, onde tais perdas são componentes do sinal que não causam impactos profundos nas atividades de análise do mesmo.

3.4
Tratamento dos coeficientes para remoção de ruídos

A atividade de remoção de ruídos em um sinal ruidoso é conhecida como *denoising*. O objetivo deste método não é suavizar a curva, pois isto implicaria em remover os componentes de alta freqüência e reter os de baixa, exatamente como explicado na seção anterior, o objetivo é remover qualquer ruído presente e reter qualquer sinal presente, independente da freqüência. Por exemplo, quando remo-

vemos ruídos em um sinal que representa uma música, nós queremos preservar os graves e agudos da canção, e remover os chiados que não pertencem à música.

Para realizarmos esta atividade usamos um método chamado de *Wavelet Shrinkage* [32], proposto inicialmente por Donoho *et al* [5]. Este método não-paramétrico é aplicado diretamente sobre os coeficientes da transformada Wavelet, ou seja, no domínio da transformada.

Assumindo que o dado observado **X(t) = S(t) + N(t)** contém o sinal real S(t) e ruído adicionado N(t) como funções no tempo t. Consideramos W(.) e W^{-1}(.) as funções responsáveis pela transformada wavelet e sua inversa, respectivamente. Assumimos que a função D(. , φ) representa a operação de remoção de ruídos com um limite suave de φ. Pretendemos aplicar a técnica proposta em X(t) com o objetivo de obter S´(t) como uma estimativa de S(t). Então, três passos resumem o problema:

Y = W(X)	Realiza a transformada Wavelet
Z = D(Y, φ)	Aplica Wavelet Shrinkage com fator φ
S´ = W^{-1}(Z)	Realiza a transformada Wavelet Inversa

As operações W e W^{-1} já foram explicadas em seções anteriores. A operação D recebe como entrada a lista de coeficientes resultantes, Y, da transformada e um fator φ conhecido como *Shrinkage Factor* ou *Fator de Achatamento*. A idéia do método consiste em manipular diretamente a lista de coeficientes Wavelet ($C_1, C_2, \ldots, C_N$) fazendo com que aqueles menores que um determinado limite φ sejam redefinidos para 0. Os maiores que φ serão "achatados" de φ, mantendo-se o sinal. Os novos coeficientes gerados seguem a função abaixo:

$$C_i' = \begin{cases} 0 & \text{Se } |C| < \varphi \\ sinal(C_i)(|C_i| - \varphi) & \text{Se } |C| >= \varphi \end{cases}$$

Esta função define, em termos práticos, uma técnica não linear para remoção de ruídos por um limite suave. Como podemos ver, se trata de um método relativamente simples. Na verdade, o maior problema é definir o limite φ. Diga-

mos que o nosso conjunto de dados original possua N elementos. Então, aplicando uma transformada wavelet de base ortogonal, teremos N coeficientes Y. Temos duas opções para cálculo do fator:

- Dependente exclusivamente de N.
- Depende dos dados em si (adaptativo).

De fato, podemos gerar uma grande gama de procedimentos para remoção de ruídos utilizando Wavelet Shrinkage através de combinações entre a função base da transformada Wavelet e a maneira como definimos o fator φ. Uma das técnicas mais simples de cálculo do coeficiente é definida por Donoho *et al* [5] e conhecida como *VisuShrink*. Esta técnica define um limite universal dependente única e exclusivamente de N. No caso:

$$\varphi = \sqrt{2 \log N}$$

O cálculo deste fator não implica em custo de processamento devido à simplicidade. Seu nome tem origem no fato de oferecer uma importante vantagem visual: uma reconstrução quase sem ruídos. A explicação para isso é que ao efetuarmos a transformada Wavelet em sinais desprovidos de ruídos, obtemos uma lista de coeficientes muito esparsa e contendo essencialmente zeros. Após a contaminação do sinal com ruído, estes coeficientes tornam-se diferentes de zero. Ao reconstruirmos um sinal contaminado com ruídos, veremos uma curva com características visuais estranhas contendo pequenas variações rápidas em sua extensão. O limite $\varphi = \sqrt{2 \log N}$ evita este problema pois aumenta a probabilidade de cada coeficiente zero contaminado por ruído voltar a ser zero.

Certamente outras técnicas mais avançadas existem e atingem resultados melhores em algumas situações. Donoho *et al* também propõe outros métodos para cálculo do fator φ, dentre eles o "RiskShrink", "SureShrink" e "WaveJS"

Para o escopo do trabalho proposto, foi implementada técnica de "Wavelet Shrinkage", porém o coeficiente φ é configurável, e não calculado automaticamente.

3.5
Quantização dos coeficientes

A técnica de quantização é uma das mais simples formas de compressão com perdas. Ela é aplicada quando precisamos representar uma quantidade muito grande ou infinita de valores com um conjunto reduzido de códigos. Neste trabalho será aplicada nos coeficientes resultantes da transformada Wavelet, uma vez que eles são representados em ponto flutuante. Com o objetivo de criar uma lista reduzida de números inteiros com alto grau de repetição, criando um ambiente ótimo para a execução posterior de um algoritmo de codificação, usaremos um método de quantização com intervalos adaptativos.

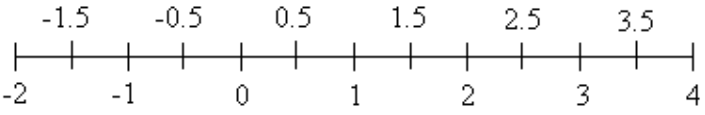
A idéia geral é fazer uma pré-classificação dos valores da lista de coeficientes em intervalos. O número de intervalos pode ser configurado e não interfere no tamanho do conjunto final. Para que a classificação possa ser feita, descobrimos o valor mínimo e o máximo na lista de coeficientes, depois dividimos o espaço de acordo com a quantidade de intervalos, por exemplo:

Mínimo: -2

Máximo: 4

Intervalos: 12

Intervalo entre as divisões: $\frac{4-(-2)}{12} = \frac{6}{12} = 0.5$



Uma vez particionado o espaço, a lista de coeficientes é percorrida para classificação de cada valor em um destes intervalos. Nesta etapa não estamos interessados em saber quais valores caíram em quais intervalos e sim a quantidade de valores em cada intervalo, como em um histograma. Então, considerando uma lista hipotética com 235 coeficientes, um exemplo de classificação segue:

Intervalo	Inicio	Fim	Freqüência	Freqüência %
1	>= -2.0	< -1.5	0	0
2	>= -1.5	< -1.0	8	3.40
3	>= -1.0	< -0.5	55	23.40
4	>= -0.5	< 0.0	38	16.17
5	>= 0.0	< 0.5	62	26.38
6	>= 0.5	< 1.0	30	12.77
7	>= 1.0	< 1.5	18	7.66
8	>= 1.5	< 2.0	12	5.11
9	>= 2.0	< 2.5	0	0.00
10	>= 2.5	< 3.0	4	1.70
11	>= 3.0	< 3.5	8	3.40
12	>= 3.5	<= 4.0	0	0.00

Tabela 3: Exemplo de freqüências por intervalo de quantização

Como podemos notar, este exemplo hipotético reflete a realidade encontrada nos testes realizados para este trabalho, onde existem grandes concentrações de valores em alguns intervalos, logo uma quantização uniforme não seria apropriada.

Este exemplo se baseia em uma lista hipotética de 235 coeficientes em ponto flutuante e a idéia é representá-los com uma quantidade sensivelmente menor de números inteiros. Esta quantidade é definida como "nível de quantização" e está diretamente relacionado à capacidade de compressão (inversamente proporcional) e a precisão (diretamente proporcional). Em suma, se tivermos nível de quantização alto, teremos boa precisão, mas a compressão não será muita, em contrapartida, quanto menor o nível de quantização, maior a capacidade de compressão e aumento de perda de precisão. Se usarmos 32 níveis de quantização neste conjunto hipotético com um método de quantização uniforme, teríamos que distribuir estes níveis uniformemente no intervalo [-2, 4]. Usando a técnica desenvolvida, os níveis são distribuídos proporcionalmente nos intervalos de acordo com a freqüência. Isso trás a vantagem de podermos usar uma maior quantidade de ní-

veis em intervalos que concentram maiores quantidades de valores. Então, neste caso teríamos:

Intervalo	Inicio	Fim	Freqüência %	Níveis
1	>= -2.0	< -1.5	0	0
2	>= -1.5	< -1.0	3.40	1
3	>= -1.0	< - 0.5	23.40	7
4	>= -0.5	< 0.0	16.17	5
5	>= 0.0	< 0.5	26.38	8
6	>= 0.5	< 1.0	12.77	4
7	>= 1.0	< 1.5	7.66	2
8	>= 1.5	< 2.0	5.11	2
9	>= 2.0	< 2.5	0.00	0
10	>= 2.5	< 3.0	1.70	1
11	>= 3.0	< 3.5	3.40	2
12	>= 3.5	<= 4.0	0.00	0

Tabela 4: Exemplo de atribuição de níveis de quantização por intervalo

Após feita a distribuição como na tabela acima, cada intervalo é subdividido em uma quantidade de níveis proporcional à freqüência de coeficientes contidos nele. Ao final do processo teremos uma tabela com 32 novos intervalos, como segue abaixo:

Intervalo	Inicio	Fim	Valor Representativo (média)
1	>= -1.500	< -1.000	-1.250
2	>= -1.000	< -0.928	-0.964
3	>= -0.928	< -0.857	-0.892
4	>= -0.857	< -0.785	-0.821
5	>= -0.785	< -0.714	-0.750
6	>= -0.714	< -0.642	-0.678
7	>= -0.642	< -0.571	-0.607
8	>= -0.571	< -0.500	-0.535

...	...	...	...
31	>=3.000	< 3.250	3.125
32	>= 3.250	<= 3.500	3.375

Tabela 5: Exemplo de definição de valores representativos por intervalo

A esta tabela, chamamos *tabela de quantização* e podemos agora percorrer a lista original de coeficientes classificando os valores nos intervalos desta tabela. Cada valor será substituído pelo número do intervalo onde se encaixa. No fim do processo, teremos uma lista com o mesmo tamanho, mas representada com números inteiros, no domínio de [1, Níveis de quantização]. Neste exemplo teríamos uma lista de 235 valores inteiros variando de 1 a 32, fazendo com que ocorra um alto índice de repetição. Vejamos, se a lista de coeficientes originais fosse:

C = [-0.93, -0.90, -0.75, -0.58, -1.2, ... , -0.98]

Teríamos uma lista de coeficientes quantizados parecida com:

Cq = [2, 3, 5, 7, 1, 2, 2, 3, 7, 9, 3, 5, 1, 4, 6, 7, 20, 12, 27, 30, ... , 2]

Como podemos notar, usando os índices dos intervalos da tabela de quantização, alcançamos um alto nível de repetição na nova lista de coeficientes quantizados, tornando esta lista ideal para a aplicação de técnicas de codificação como o código de Huffman ou algoritmo de codificação aritmética. A aplicação destas técnicas traz excelentes ganhos em termos de compressão. No entanto, o método de quantização introduz erros, pois cada índice da tabela de quantização contém um valor representativo que não é igual ao original, mas à média entre o inicial e o final do intervalo onde aquele valor se encaixa. Embora esse erro introduzido seja reduzido quando distribuímos uma maior quantidade de níveis de quantização para os intervalos de maior freqüência, este erro pode representar grande variação no sinal compactado se as configurações de intervalo e níveis não forem balanceadas em relação a quantidade de coeficien-

tes. Considerando o exemplo proposto nesta seção, a lista de coeficientes recuperada através da tabela de quantização pode ser comparada com os valores originais abaixo:

C = [-0.93, -0.90, -0.75, -0.58, -1.2, ... , -0.98] ► Original

Cr = [-0.96, -0.89, -0.75, -0.60, -1.2, ... , -0.96] ► Recuperado

3.6
Codificação dos coeficientes quantizados

Após a técnica de quantização ser aplicada aos coeficientes, obtemos uma lista de números inteiros com alto grau de repetição. Para tirar proveito desta situação, aplicamos um algoritmo de codificação ao conjunto com a finalidade de obtermos uma maneira mais eficiente de representá-lo. A aplicação do código de Huffman é feita apenas na lista de índices, ficando a tabela de quantização em seu estado natural.

Como sabemos, a codificação de Huffman [28] é um método de compressão sem perdas que usa uma tabela de códigos de tamanho variável para representar os símbolos originais. Esta tabela de códigos é criada de uma maneira particular (árvore de Huffman) baseada na probabilidade estimada da ocorrência de cada símbolo original. O método de escolha da representação de cada símbolo resulta em códigos livres de prefixo, o que significa que um conjunto de bits que o representa nunca é igual ao prefixo de nenhum conjunto representativo de qualquer outro. Desta forma, torna-se possível à decodificação, pois cada símbolo é substituído pelos conjuntos binários definidos na tabela de códigos, símbolo por símbolo. Os códigos binários são atribuídos a cada símbolo na tabela de codificação de acordo com sua probabilidade de freqüência, quanto maior a freqüência de um determinado símbolo, menor será o código para representá-lo.

No caso deste trabalho, o problema resume-se a representar de maneira eficiente uma extensa lista de números inteiros (símbolos), que servem de índice para os valores representativos da tabela de quantização. O alfabeto para o código de Huffman tem o tamanho da tabela de quantização, ou seja, é igual à quantidade de níveis de quantização definida. No caso do nosso exemplo, o alfabeto seria

composto pelos símbolos no intervalo [1, 32], como se tratam de números inteiros do tipo “short int”, que possuem domínio [0, 65535], sua representação em C/C++ ocupa 2 bytes (16 bits) por símbolo. Logo, se aplicássemos o código de Huffman na lista abaixo:

Cq = [2, 3, 5, 7, 1, 2, 2, 3, 7, 9, 3, 5, 1, 4, 6, 7, 20, 12, 27, 30, ... , 2]

Teríamos a seguinte tabela de codificação hipotética:

Entrada	Símbolo	Freqüência	Código	Total de Bits / Símbolo
1	2	50	111	5 x 3 = 15
2	7	42	010	4 x 3 = 12
3	3	41	000	4 x 3 = 12
4	9	29	1101	2 x 4 = 8
5	5	24	1010	2 x 4 = 8
6	4	22	1000	2 x 4 = 8
7	20	21	0111	2 x 4 = 8
8	1	21	0010	2 x 4 = 8
9	6	18	1011	1 x 4 = 4
10	30	14	0110	1 x 4 = 4
...	...	...	...	...
32	12	10	00110	1 x 5 = 5

Tabela 6: Exemplo de tabela de codificação de Huffman

Em posse da tabela, o próximo passo é substituir os símbolos pelos códigos binários na seqüência em que aparecem, no caso do exemplo teríamos:

Cq = [**2**, 3, **5**, 7, **1**, 2, **2**, 3, **7**, 9, **3**, 5, **1**, 4, 6, 7, 20, 12, 27, 30, ... , 2]
Ch = [**111**000**1010**010**0010**111**111**000**010**1101**000**1010**0010**...111]

Como o algoritmo de Huffman gera códigos livres de prefixo, podemos percorrer todo o conjunto binário da esquerda para a direita identificando as seqüências de forma indubitável no momento da decodificação.

Analisando a tabela, podemos perceber que o símbolo que aparece mais freqüentemente na lista de coeficientes quantizados é o número 2, com 50 ocorrências. Sem a representação de Huffman, armazenar todas as ocorrências deste símbolo ocuparia 100 bytes, visto que um inteiro do tipo “short int” precisa de 2 bytes para ser representado. Porém, como o código gerado pelo algoritmo de Huffman para representar o símbolo 2 precisa apenas de 3 bits, as cinqüenta ocorrências precisariam apenas de cento e cinqüenta 150 bits, ou aproximadamente 19 bytes, o que representa uma quantidade 5 vezes menor de bits para representar, sem perdas, o mesmo conjunto de dados. Substituindo todos os símbolos por seus códigos correspondentes teremos uma seqüência binária que representa toda a lista de coeficientes quantizados utilizando muito menos espaço em memória. Além disso, a tabela de codificação deve ser persistida para que seja possível a decodificação.

Para o desenvolvimento do protótipo, implementamos uma versão do código de Huffman adaptada para receber como entrada uma lista de números inteiros. Este algoritmo foi selecionado por apresentar baixa complexidade e ser bastante popular no meio acadêmico. Nada impede que outros algoritmos de codificação substituam o escolhido, inclusive o método de codificação aritmética produz ganhos significativos em relação ao método de Huffman, porém é sensivelmente mais complexo em termos computacionais além de envolver diversas questões de patentes, já que a IBM detém os direitos sobre vários de seus métodos.

3.7
Codificação de datas e horas

Ao fim do processo de codificação dos coeficientes, a técnica proposta neste trabalho está concluída para a problemática que inclui a compressão e tratamento do sinal. No entanto, esta compressão não inclui informações que permitam relacioná-lo com o tempo, ou seja, ao realizarmos a descompressão dos dados, teremos como resultado uma lista de números em ponto flutuante sem nenhuma relação com tempo, tornando impossível, por exemplo, descobrir dia e hora onde determinado padrão de comportamento ocorreu.

Para resolver este problema foi desenvolvida uma técnica de compressão de informações temporais. Esta técnica se baseia no fato de que, durante a análise de um sinal ao longo de uma série histórica grande, não se faz necessário uma precisão muito alta e sim que a idéia de tempo seja passada ao individuo responsável pela análise. Fica claro que, ao analisarmos uma curva de temperatura em uma série histórica de quinze (15) dias, é importante saber o momento onde aconteceram os maiores picos; no entanto, certamente não faz diferença se a informação não estiver com precisão de minuto e segundo.

A técnica proposta nesta seção introduz perda de precisão e, por esse motivo, as taxas de compressão alcançadas são mais agressivas. Não faria sentido armazenar a informação temporal de forma a ocupar muito mais espaço em memória do que os valores numéricos que representam à curva em si. Como a informação temporal é tratada em separado, o algoritmo proposto para a compressão recebe como entrada apenas a lista das datas/horas. Mantendo duas listas em paralelo (data/hora, valores medidos), podemos relacionar os valores de tempo e medição através da posição dos elementos na lista, permitindo manter a coerência mesmo com tratamento em separado.

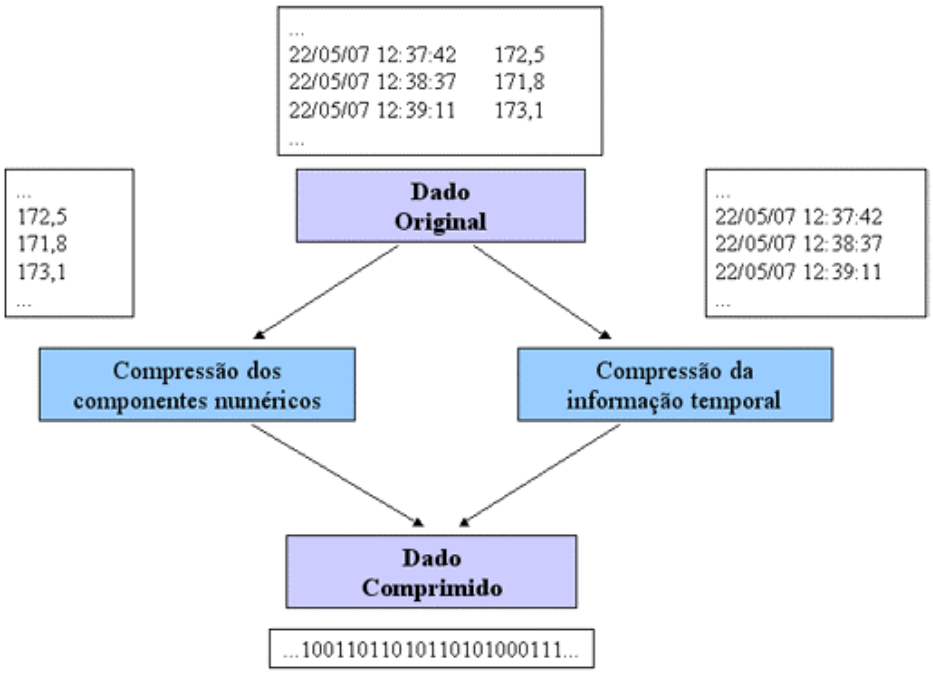


Figura 6: Esquema de compressão de informações temporais

Para o processo de compressão da informação temporal, a estratégia é montar uma estrutura de dados que represente um dia de medição com um tamanho fixo e muito menor do que um dia de medição real. Isto quer dizer que, se houver um dia com 50 medições e um outro com 300, ambos gastarão o mesmo espaço em memória.

O processo é iniciado percorrendo toda a lista em busca das seguintes informações (por dia):

- Número do dia
- Número do mês
- Número do ano
- Quantidade de medições no dia
- Freqüência de medições por hora
    - Número N_0 de medições na hora 00
    - Número N_1 de medições na hora 01
    - Número N_2 de medições na hora 02
    - Número N_3 de medições na hora 03

- Número N_4 de medições na hora 04
- ...
- Número N_{23} de medições na hora 23

- Ajuste de minutos
  - Componentes de minutos chave para Hora 0
    - Minuto da primeira medição da hora 0
    - Minuto da medição $N_0/4$
    - Minuto da medição $N_0/2$
    - Minuto da medição $3N_0/4$
    - Minuto da medição N_0
  - ...
  - Componentes de minutos chave para Hora 23
    - Minuto da primeira medição
    - Minuto da medição $N_{23}/4$
    - Minuto da medição $N_{23}/2$
    - Minuto da medição $3N_{23}/4$
    - Minuto da medição N_{23}

Por exemplo, se tivéssemos uma série iniciando em 22/05/2007 as 14h00min:

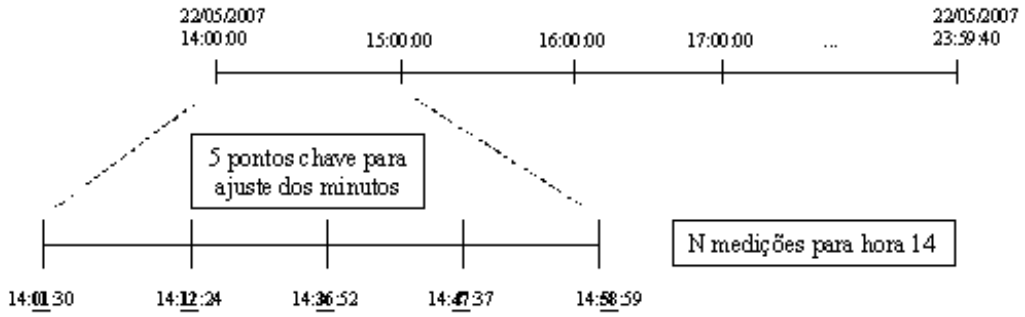


Figura 7: Exemplo de Coleta de estatísticas para compressão de tempo

O primeiro procedimento seria coletar as estatísticas do dia 22/05/2007 e para armazenar estas estatísticas propomos uma estrutura de dados simples que tem como principal característica ter tamanho fixo. Esta estrutura representa estatísticas de um dia, portanto para representar toda a série precisaremos de uma lista deste tipo.

A definição do tipo segue abaixo:

```
typedef struct TCompactDay
{
      unsigned char dia;                        // 1 byte
      unsigned char mes;                        // 1 byte
      unsigned short int ano;                   // 2 bytes
      unsigned int qtdPontosDia;                // 4 bytes
      unsigned short int frequenciaHora[24];    // 48 bytes
      unsigned char ajusteMinutos[120];         // 120 bytes
} TCpDay;
```

Como podemos perceber pela definição da estrutura de dados que representa um dia compactado, o espaço em memória ocupado é de 176 bytes / dia. Isto quer dizer que, em uma série histórica de um (1) ano, o espaço gasto para armazenar as informações temporais seria aproximadamente 62 Kbytes, independente da quantidade de medições em cada dia. Sem esta representação compactada, esta mesma série histórica com dados medidos a cada 30 s teria 1051200 medições e ao armazenarmos cada uma das componentes de tempo em uma estrutura de dados como o "time_t" do C que ocupa 8 bytes, teríamos um espaço total gasto de 8.2 Mbytes.

Devemos analisar a perda de precisão para poder avaliar a viabilidade da aplicação da técnica em determinadas situações onde a precisão é de suma importância. Veremos um exemplo extraído de um conjunto temporal real com 1 hora de duração, iniciando em 24/09/2006 00:00:11 com fim em 24/09/2006 00:59:33. A estrutura de dados seria carregada da seguinte forma após o levantamento estatístico:

```
TcompactDay.dia = 24;
TcompactDay.mes = 9;
TcompactDay.ano = 2006;
TcompactDay.qtdPontosDia = 88
TcompactDay.frequenciaHora = { 88, 0, 0, 0, 0, 0, 0, ... , 0 };
TcompactDay.ajusteMinutos = { 0, 14, 28, 44, 59, ... , 0 }
```

Para descompactar os dados a partir das informações coletadas e armazenadas na estrutura acima, o algoritmo monta a data inicial 24/09/2006 lendo os três primeiros campos (dia, mês e ano). A partir de então, analisa o vetor **frequenciaHora**, que indica quantas medições foram feitas na hora referente ao índice do valor. No caso acima, frequenciaHora[0] = 88 o que significa que na hora 0 (entre meia-noite e uma da manhã) houve 88 medições. Continuando a análise deste vetor podemos descobrir que não existem informações fora deste intervalo. O algoritmo considera cinco pontos chave por hora para ajuste de minutos. Estes pontos estão no vetor ajusteMinutos em seqüência, do índice zero ao quatro pertencem a hora zero, do cinco ao nove pertencem a hora 1 e assim por diante. Logo, na hora 0 são inferidos os seguintes pontos chave:

1) 00:00:00
2) 00:14:00
3) 00:28:00
4) 00:44:00
5) 00:59:00

Como são 88 medições para a hora zero 0, cada um dos 4 intervalos deverá possuir 22 pontos de medição, logo o algoritmo faz a inferência destes pontos. A tabela abaixo apresenta os resultados obtidos:

Quadrante	Tempo Original	Tempo Compactado	Erro	Erro Médio
	24/09/2006 00:00:11	**24/09/2006 00:00:00**	0:00:11	
	24/09/2006 00:00:52	**24/09/2006 00:00:42**	**0:00:10**	
	24/09/2006 00:01:33	**24/09/2006 00:01:24**	**0:00:09**	
	24/09/2006 00:02:14	**24/09/2006 00:02:06**	**0:00:08**	
	24/09/2006 00:02:55	**24/09/2006 00:02:48**	**0:00:07**	
	24/09/2006 00:03:36	**24/09/2006 00:03:30**	**0:00:06**	
	24/09/2006 00:04:17	**24/09/2006 00:04:12**	**0:00:05**	
	24/09/2006 00:04:58	**24/09/2006 00:04:54**	**0:00:04**	
	24/09/2006 00:05:38	**24/09/2006 00:05:36**	**0:00:02**	
	24/09/2006 00:06:19	**24/09/2006 00:06:18**	**0:00:01**	
1	**24/09/2006 00:07:00**	**24/09/2006 00:07:00**	**0:00:00**	**0:00:07**
	24/09/2006 00:07:41	**24/09/2006 00:07:42**	**0:00:01**	
	24/09/2006 00:08:22	**24/09/2006 00:08:24**	**0:00:02**	
	24/09/2006 00:09:03	**24/09/2006 00:09:06**	**0:00:03**	
	24/09/2006 00:09:44	**24/09/2006 00:09:48**	**0:00:04**	
	24/09/2006 00:10:25	**24/09/2006 00:10:30**	**0:00:05**	
	24/09/2006 00:11:06	**24/09/2006 00:11:12**	**0:00:06**	
	24/09/2006 00:11:47	**24/09/2006 00:11:54**	**0:00:07**	
	24/09/2006 00:12:28	**24/09/2006 00:12:36**	**0:00:08**	
	24/09/2006 00:13:09	**24/09/2006 00:13:18**	**0:00:09**	
	24/09/2006 00:13:50	**24/09/2006 00:14:00**	**0:00:10**	
2	**24/09/2006 00:14:31**	**24/09/2006 00:14:00**	0:00:31	**0:00:22**
	24/09/2006 00:15:12	24/09/2006 00:14:42	0:00:30	
	24/09/2006 00:15:53	24/09/2006 00:15:24	0:00:29	
	24/09/2006 00:16:34	24/09/2006 00:16:06	0:00:28	
	24/09/2006 00:17:15	24/09/2006 00:16:48	0:00:27	
	24/09/2006 00:17:55	24/09/2006 00:17:30	0:00:25	
	24/09/2006 00:18:36	24/09/2006 00:18:12	0:00:24	
	24/09/2006 00:19:17	24/09/2006 00:18:54	0:00:23	
	24/09/2006 00:19:58	24/09/2006 00:19:36	0:00:22	
	24/09/2006 00:20:39	24/09/2006 00:20:18	0:00:21	
	24/09/2006 00:21:20	24/09/2006 00:21:00	0:00:20	
	24/09/2006 00:22:01	24/09/2006 00:21:42	0:00:19	
	24/09/2006 00:22:42	24/09/2006 00:22:24	0:00:18	
	24/09/2006 00:23:23	24/09/2006 00:23:06	0:00:17	
	24/09/2006 00:24:04	**24/09/2006 00:23:48**	**0:00:16**	

	24/09/2006 00:24:45	24/09/2006 00:24:30	0:00:15	
	24/09/2006 00:25:26	24/09/2006 00:25:12	0:00:14	
	24/09/2006 00:26:07	24/09/2006 00:25:54	0:00:13	
	24/09/2006 00:26:48	24/09/2006 00:26:36	0:00:12	
	24/09/2006 00:27:29	24/09/2006 00:27:18	0:00:11	
	24/09/2006 00:28:09	24/09/2006 00:28:00	0:00:09	
3	24/09/2006 00:28:50	24/09/2006 00:28:00	0:00:50	0:00:23
	24/09/2006 00:29:31	24/09/2006 00:28:44	0:00:47	
	24/09/2006 00:30:12	24/09/2006 00:29:27	0:00:45	
	24/09/2006 00:30:53	24/09/2006 00:30:11	0:00:42	
	24/09/2006 00:31:34	24/09/2006 00:30:55	0:00:39	
	24/09/2006 00:32:15	24/09/2006 00:31:38	0:00:37	
	24/09/2006 00:32:56	24/09/2006 00:32:22	0:00:34	
	24/09/2006 00:33:37	24/09/2006 00:33:05	0:00:32	
	24/09/2006 00:34:18	24/09/2006 00:33:49	0:00:29	
	24/09/2006 00:34:59	24/09/2006 00:34:33	0:00:26	
	24/09/2006 00:35:40	24/09/2006 00:35:16	0:00:24	
	24/09/2006 00:36:21	24/09/2006 00:36:00	0:00:21	
	24/09/2006 00:37:02	24/09/2006 00:36:44	0:00:18	
	24/09/2006 00:37:43	24/09/2006 00:37:27	0:00:16	
	24/09/2006 00:38:24	24/09/2006 00:38:11	0:00:13	
	24/09/2006 00:39:05	24/09/2006 00:38:55	0:00:10	
	24/09/2006 00:39:46	24/09/2006 00:39:38	0:00:08	
	24/09/2006 00:40:26	24/09/2006 00:40:22	0:00:04	
	24/09/2006 00:41:07	24/09/2006 00:41:05	0:00:02	
	24/09/2006 00:41:48	24/09/2006 00:41:49	0:00:01	
	24/09/2006 00:42:29	24/09/2006 00:42:33	0:00:04	
	24/09/2006 00:43:10	24/09/2006 00:43:16	0:00:06	
	24/09/2006 00:43:51	24/09/2006 00:44:00	0:00:09	
4	24/09/2006 00:44:32	24/09/2006 00:44:00	0:00:32	0:00:15
	24/09/2006 00:45:13	24/09/2006 00:44:43	0:00:30	
	24/09/2006 00:45:54	24/09/2006 00:45:26	0:00:28	
	24/09/2006 00:46:35	24/09/2006 00:46:09	0:00:26	
	24/09/2006 00:47:16	24/09/2006 00:46:51	0:00:25	
	24/09/2006 00:47:57	24/09/2006 00:47:34	0:00:23	
	24/09/2006 00:48:38	24/09/2006 00:48:17	0:00:21	
	24/09/2006 00:49:18	24/09/2006 00:49:00	0:00:18	
	24/09/2006 00:49:59	24/09/2006 00:49:43	0:00:16	
	24/09/2006 00:50:40	24/09/2006 00:50:26	0:00:14	

	24/09/2006 00:51:21	**24/09/2006 00:51:09**	**0:00:12**
	24/09/2006 00:52:02	**24/09/2006 00:51:51**	**0:00:11**
	24/09/2006 00:52:43	**24/09/2006 00:52:34**	**0:00:09**
	24/09/2006 00:53:24	**24/09/2006 00:53:17**	**0:00:07**
	24/09/2006 00:54:05	**24/09/2006 00:54:00**	**0:00:05**
	24/09/2006 00:54:46	**24/09/2006 00:54:43**	**0:00:03**
	24/09/2006 00:55:27	**24/09/2006 00:55:26**	**0:00:01**
	24/09/2006 00:56:08	**24/09/2006 00:56:09**	**0:00:01**
	24/09/2006 00:56:49	**24/09/2006 00:56:51**	**0:00:02**
	24/09/2006 00:57:30	**24/09/2006 00:57:34**	**0:00:04**
	24/09/2006 00:58:11	**24/09/2006 00:58:17**	**0:00:06**
	24/09/2006 00:58:52	**24/09/2006 00:59:00**	**0:00:08**
	24/09/2006 00:59:33	**24/09/2006 00:59:00**	**0:00:33**

Tabela 7: Exemplo de decodificação das séries históricas comprimidas

Analisando a desempenho do algoritmo neste conjunto de dados, vemos que o erro médio total é de 17 segundos, considerado aceitável para o propósito deste trabalho. Este método é interessante visto que alcança uma precisão aceitável para análise de séries históricas. De fato, em casos onde a precisão é um requisito fundamental, esta técnica não deve ser utilizada e consequentemente devemos arcar com o custo do armazenamento integral destas informações. É importante atentar para o fato de que, quando a série histórica é completa, o erro introduzido é menor. Em casos onde o sinal apresenta buracos, os pontos serão distribuídos de acordo com os minutos chave, o que eventualmente pode diminuir a precisão.

3.8
Estrutura do arquivo binário

Ao final do processo de codificação de datas e horas, temos uma série de estruturas de dados que fornecem suporte ao processo de compressão, além da lista codificada de índices da tabela de quantização. Para concluir o processo de compressão, um formato de arquivo binário foi desenvolvido para armazenar todas as estruturas necessárias. As estruturas de dados primárias são as seguintes:

```
/* Lista de números em ponto flutuante */
typedef struct {
 int* value;
 int size;
} INT_DATA;

/* Lista de números em ponto flutuante */
typedef struct
{
  float* value;
  int size;
} FLOAT_DATA;

/* Lista representando uma sequência binária */
typedef struct
{
  unsigned char* value;
  int size;
} BIN_DATA;
```

A tabela de quantização (Q) não passa de uma lista de números em ponto flutuante. Logo, ela está representada em uma estrutura do tipo FLOAT_DATA. Para armazená-la no arquivo, primeiro armazenaremos o tamanho da estrutura (Tq) e em seguida a estrutura referente à tabela de quantização (Q). Além disso, teremos uma lista de dados binários contendo a Árvore de Huffman (AH) em conjunto com a seqüência de dados codificados (D), sendo precedida pelo tamanho da

lista binária (Tb). Além destes dados, armazenamos também o tamanho do conjunto (N).

Desta forma, o arquivo fica como no esquema abaixo:

[Tq + Q + Tb + [AH + D] + N]

Para a abertura do arquivo, é restabelecida a tabela de quantização (Q) após ler o intervalo de memória baseado no tamanho da estrutura (Tq), logo em seguida é restabelecida a estrutura binária (AH+D) também com base no tamanho da estrutura (Tb). Estes dados binários são passados ao algoritmo de decodificação de Huffman que recupera a árvore binária de Huffman (AH) e percorre a seqüência binária (D) recuperando os dados e armazenando-os em uma estrutura do tipo INT_DATA. Esta estrutura possui os índices da tabela de quantização, logo, a estrutura é percorrida e para cada valor inteiro lido (i), o coeficiente em ponto flutuante é recuperado da tabela de quantização (Q) acessando Q[i] e armazenado-o em uma estrutura do tipo FLOAT_DATA. Nesta estrutura estão os coeficientes relevantes (CR) da transformada (isto exclui a seqüência de zeros do final, criada no momento do descarte de coeficientes durante o processo de compressão). Logo, temos que ler o tamanho do conjunto (N), pois sabemos que, após a transformada, foram criados N coeficientes. Desta forma podemos completar a lista de coeficientes relevantes com uma quantidade de zeros igual a (N – Quantidade (CR)). De posse da lista completa de coeficientes, podemos aplicar a transformada Wavelet inversa e restabelecer os valores tratados do sinal original.