3
Código de Baixa Densidade, Irregular e com Taxa Variável

3.1.
Escolha do código

Segundo [67], dentre os métodos de codificação de canal mais eficientes aplicados ao codificador WZ, destacou-se, em termos de desempenho, o método LDPC, principalmente em sua forma irregular. Otimizando e complementando as técnicas de codificação e decodificação mais eficientes e consideradas o "estado da arte" em códigos de canal para DVC, buscou-se realizar um trabalho mais profundo (e mais eficiente), a fim de alcançar um desempenho similar aos métodos convencionais de codificação híbrida de vídeo, para uma complexidade de codificação menor, embora o processo de decodificação do DVC seja bem mais complexo.

3.2.
DVC baseado em Síndromes

A codificação assimétrica da fonte pode ser descrita, de forma resumida, pela figura 3.1. Assim, uma fonte *X* (com alfabeto finito) é transmitida sem perdas usando o menor número possível de bits. A informação lateral estatisticamente dependente *Y* está disponível somente no decodificador. O codificador deve, portanto, comprimir *X* na ausência de *Y* e o decodificador utiliza *Y* para auxiliar na recuperação de *X*.

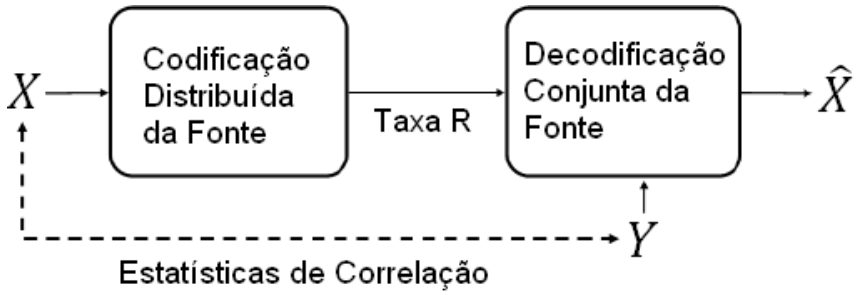


Figura 3.1: Cenário resumido de codificação assimétrica da fonte.

Seguindo o proposto por Pradhan et al. para codificação de vídeo baseada em síndromes [7], o codificador distribuído de fonte comprime X e gera uma síndrome S, segundo o código de canal C, e esta síndrome é enviada para o decodificador. Após o recebimento da Síndrome, o decodificador de canal, executando uma decodificação de forma conjunta, consegue restringir os possíveis valores de X ao conjunto de palavras-código (coset) representado por S em C. Então, o decodificador distingue $\hat{X}$ dentre os elementos deste coset como o elemento com maior probabilidade de ter sido enviado, segundo o critério de Hamming, dada a informação lateral correlata Y.

O impressionante potencial deste método tem sido demonstrado por várias implementações com o uso de códigos-turbo [10, 12, 14] e códigos de verificação de paridade de baixa densidade [22]. Nesses esquemas de codificação, uma compressão eficiente e realizável, próxima ao limite de Slepian-Wolf, depende da escolha de um código de canal apropriado, e nesse contexto, as dependências entre X e Y desempenham o papel das estatísticas do canal de correlação. Se as estatísticas deste "canal dependente" são conhecidas tanto pelo codificador quanto pelo decodificador, eles podem, em concordância, utilizar um código otimizado para este canal e uma taxa observada próxima ao limite estabelecido por Slepian-Wolf.

A codificação de vídeo de baixa complexidade através do método de codificação distribuída da fonte, por exemplo, trata um frame de vídeo como uma fonte X e sua predição no decodificador como uma informação lateral Y [54]. Visto que os dados de vídeo são altamente não-ergódicos, ou melhor, variantes, a taxa de compressão admissível varia e não pode ser prevista pelo codificador.

Nessa situação, um esquema de taxa adaptativa com realimentação ou canal de retorno seria uma solução bem atrativa. Desta forma, o codificador transmite uma síndrome curta, baseada em um código menos preciso e o decodificador tenta realizar a decodificação. No evento em que a decodificação obtém êxito, o decodificador sinaliza este fato para o codificador, o qual, então, continua com o próximo bloco de dados da fonte. Entretanto, se a decodificação falha, o codificador aumenta a síndrome, transmitindo bits adicionais e gerando, desta forma, uma síndrome mais longa, baseada em um código com valores mais suaves. Realiza iterações até que a síndrome enviada contenha informação suficiente para uma decodificação bem sucedida. Obviamente, este método só é

viável se houver um canal de retorno e se o tempo despendido no processo iterativo não for muito longo.

Para implementar um método de codificação de fonte com taxa adaptativa e com a informação lateral no decodificador, alguns autores [61–63] utilizaram o código LDPC perfurado. Embora o código LDPC perfurado possa ser aplicado a este problema, seu desempenho é baixo mesmo depois de executar uma razoável perfuração [68].

Com o objetivo de solucionar este problema, a saber, o de codificação distribuída com taxa adaptativa, propõe-se a utilização de uma melhor arquitetura para codificação distribuída de vídeo, através do desenvolvimento de um código de verificação de paridade de baixa densidade, irregular, acumulado e adaptativo, denominado LIA (LDPC Irregular, Acumulado e Adaptativo) o qual tira proveito de técnicas de alto desempenho utilizadas pelos códigos fontanais (aperfeiçoados) de taxa infinita (*rateless codes*) propostos por Luby [76] e dos códigos LDPC [22].

3.3. Escolha do código

Conforme mencionado na seção anterior, os códigos LDPC (na função de códigos decodificadores de síndrome) têm sido utilizados eficientemente na codificação distribuída de taxa fixa. Uma nova maneira de utilizar tal código como parte do esquema adaptativo seria transmitir os bits da síndrome em estágios e permitir a decodificação após o recebimento de cada incremento da síndrome. Entretanto, o desempenho dos códigos de alta compressão empregados na codificação de canal é bastante limitado, pois geralmente são códigos não adaptativos, cujas características estruturais impedem a transferência de informação através do algoritmo de decodificação iterativa [65, 66]. Por esse motivo, apresentamos um método para gerar códigos de taxa adaptativa baseados no código LDPC, criado por Gallager [89], para codificação distribuída da fonte, cujo desempenho não degrada em altas taxas de compressão. O funcionamento do código proposto fundamenta-se no LDPC acumulado, desenvolvido por Varodayan [71].

O código LIA é chamado “irregular” por não ter um número regular de conexões entre os nós de paridade e os nós das variáveis, o que diminui bastante a correlação entre as palavras-código geradas, aumentando a distância de Hamming entre elas e, consequentemente, diminui a possibilidade de erro na decodificação. O adjetivo “acumulado” atribuído ao código significa que se faz uso de um acumulador para transmitir as síndromes a fim de enviar somente uma parte delas, as quais são formadas pela soma das síndromes originais, segundo uma determinada regra, o que implica em uma diminuição na quantidade de bits transmitidos. As síndromes originais são aquelas geradas inicialmente pelo código adotado. O termo “adaptativo” empregado na denominação do código significa que ele pode variar o percentual de paridade na palavra código e aumentar o grau de verificação de erros à custa de um pequeno incremento na quantidade de bits provenientes da síndrome.

Assim, a estrutura e o funcionamento do código LIA foram projetados para adequação à codificação distribuída de fonte com taxa variável, buscando alcançar um melhor desempenho do que os códigos alternativos que carecem de flexibilidade quanto a este requisito.

3.3.1. Fundamentos da Codificação LDPC Irregular

Os códigos turbo e LDPC, os melhores códigos corretores de erros da atualidade [71], partilham de um fato em comum que é o uso de métodos de decodificação iterativa que recorrem a algoritmos semelhantes. Entretanto, segundo Aaron [71], com o código LDPC irregular alcança-se um melhor desempenho quando utilizado no codificador Wyner-Ziv. Em particular os códigos LDPC são caracterizados por grafos bipartidos de Tanner e grafos de fatores [90] e são decodificados com um algoritmo de transferência de mensagens ou propagação de probabilidades entre nós dos grafos, fazendo uso também do algoritmo da *soma-e-produto* [91] usado nos códigos-turbo. Além de utilizar-se o algoritmo da *soma-e-produto* para aumentar a eficiência da correção de erros, há a possibilidade de ser utilizada uma versão simplificada e mais rápida, ou seja, com maior velocidade na decodificação iterativa, o qual é chamado de algoritmo de *produto-e-máximo* [91]. Entretanto, o algoritmo de *produto-e-máximo* possui uma

capacidade de correção de erros um pouco menor que o de *soma-e-produto*. O objetivo deste capítulo é enfatizar a decodificação de canal proposta neste trabalho utilizando como base o código LDPC, cuja construção é descrita no Anexo A desta tese.

3.3.2. Probabilidades e LLR *a posteriori*

É habitual reconstruir a sequência $\underline{x}$ a partir da sequência recebida $\underline{y}$, a qual pode ser considerada também como uma estimativa de $\underline{x}$, usando-se um entre dois métodos de decodificação probabilística: estimação de máxima verosimilhança (ou estimação ML, de *Maximum Likelihood*) e estimação da máxima probabilidade *a posteriori* (ou estimação MAP, de *Maximum A Posteriori*) [91]. O primeiro método (ML) calcula a probabilidade condicional *a priori* $p\left(\underline{y} \mid x_i\right)$, para determinar a sequência $\underline{\hat{x}}$ que maximiza $p\left(\underline{y} \mid \underline{\hat{x}}\right)$. Já o segundo método calcula a probabilidade condicional *a posteriori* $p\left(x_i \mid \underline{y}\right)$, segundo a eq. (3.1), a qual é também designada por APP (*A Posteriori Probability*), para determinar a probabilidade associada ao símbolo x_i.

$$p\left(x_i \mid \underline{y}\right) = p\left(\underline{y} \mid x_i\right)\frac{p(x_i)}{p\left(\underline{y}\right)} \tag{3.1}$$

O método que é mais adequado à estimação de bits no decodificador de canal proposto é o método de estimação MAP já que se deseja, neste trabalho, calcular a probabilidade de ter sido enviado cada bit x_i dado que se recebeu a sequência $\underline{y}$, uma estimativa de $\underline{x}$, empregada como informação lateral na abordagem deste trabalho. Além disso, costuma-se utilizar para fins de decisão o logaritmo da razão de probabilidades, expresso por:

$$L(x_i \mid y) = \ln\frac{p(x_i = 1 \mid y)}{p(x_i = 0 \mid y)} \tag{3.2}$$

o qual é denominado LLR *a posteriori* (*Log-Likelihood Ratio* – logaritmo da razão de probabilidades a posteriori).

O critério de decisão final sobre o bit x_i é o habitual: se a probabilidade APP (eq. (3.1)) ou a razão LLR (eq. (3.2)) estiverem acima de um determinado limiar de decisão será escolhido $\hat{x}_i$ = 1, se estiverem abaixo escolher-se-á $\hat{x}_i$ = 0. O

limiar de decisão é 0,5 no caso da APP e 0 no caso da LLR. Assim, este critério pode ser expresso da seguinte forma:

$$x_i = \begin{cases} 1, & L(x_i \mid y) > 0 \\ 0, & L(x_i \mid y) < 0 \end{cases} \tag{3.3}$$

Portanto, se a LLR for positiva escolhe-se o bit 1 e se for negativa escolhe-se o bit 0. As probabilidades e as LLR *a posteriori* podem ser calculadas de diversas maneiras. Utilizando a codificação turbo, por exemplo, é normal usar o algoritmo BCJR [92]. Com códigos LDPC é costume recorrer a grafos e a algoritmos de transferência de mensagens entre seus nós. Estas mensagens contêm estimativas de probabilidades ou quantidades a elas relacionadas, como a LLR.

3.3.3. Transferência de Probabilidades entre Nós dos Grafos

Os algoritmos de decodificação baseados em grafos são chamados genericamente de *algoritmos de transferência de mensagens*. O nome que efetivamente tomam depende um pouco do contexto: enquanto que na teoria da codificação e na presença de grafos de fatores se prefere a designação *algoritmo da soma-e-produto* (*Sum-Product algorithm*, ou SP), em redes bayesianas a comunidade de inteligência artificial usa mais o termo *algoritmo de propagação de probabilidades* (*Belief Propagation algorithm*, ou BP). O que aconteceu é que, por vias e em aplicações muito distintas, ambas as comunidades chegaram a um algoritmo que a outra desconhecia, mas que o tempo veio a provar que, afinal, era o mesmo. O algoritmo BP foi formulado por J. Pearl [92] e chama-se assim porque propaga "crenças" (probabilidades) de uns nós de grafos para os outros. Tanner generalizou-o com o algoritmo SP [93]. Mais tarde, já depois do aparecimento dos códigos-turbo em 1993, o algoritmo da soma-e-produto foi aplicado com sucesso aos grafos de fatores, o que resultou em sua formulação mais genérica [91]. Na prática, estes algoritmos de transferência de mensagens são algoritmos do tipo SISO (*Soft Input-Soft Output*), pois suas entradas e saídas, medidas de probabilidades, são valores reais.

Consideremos então um grafo de Tanner com nós de variáveis x_i e nós de controle f_j a "trocar" mensagens deste gênero: a variável x_i questiona f_j sobre as

"opiniões" que os outros x_i têm dela e f_j responde-lhe com as "opiniões" que conhece. Tais perguntas (questões) e respostas refletem-se na notação que será utilizada:

$m_{ij}(x_i)$ – mensagem do nó de variável x_i para o nó de paridade f_j

$r_{ji}(x_i)$ – mensagem do nó de paridade f_j para o nó de variável x_i

Estas mensagens ou probabilidades são função de um único argumento (a variável x_i) quer circulem num sentido quer no outro. A mensagem de x_i para f_j representa a probabilidade de x_i assumir certo valor, dado o valor observado dessa variável (a quantidade $p(x_i|y_i)$) e dados os valores que recebeu dos outros nós de paridade. Como x_i é binária, a mensagem pode ser escrita como:

$$m_{ij}(x_i) = \begin{bmatrix} m_{ij}(x_i = 0) \\ m_{ij}(x_i = 1) \end{bmatrix} = \begin{bmatrix} p(x_i = 0 | \sim \{f_j\}, \underline{y}) \\ p(x_i = 1 | \sim \{f_j\}, \underline{y}) \end{bmatrix} \tag{3.4}$$

Na eq. (3.4), $\sim \{f_j\}$ representa o conjunto das funções (ou nós de paridade) ligadas a x_i exceto f_j. De igual modo, a mensagem de f_j para x_i é a probabilidade de x_i ter certo valor e a equação de paridade f_j [89, 90] ser satisfeita, dado que se recebeu $\underline{y}$. Logo, a mensagem de f_j para x_i pode ser expressa por:

$$r_{ji}(x_i) = \begin{bmatrix} r_{ji}(0) \\ r_{ji}(1) \end{bmatrix} = \begin{bmatrix} p(x_i = 0, f_j(\,) = 1 | \underline{y}) \\ p(x_i = 1, f_j(\,) = 1 | \underline{y}) \end{bmatrix} \tag{3.5}$$

Suponhamos, por exemplo, que o nó da variável x_i está ligado a três nós de paridade e recebe do decodificador, no qual está presente a informação lateral, a probabilidade fixa p_i, como está ilustrado na figura 3.2(a). No início de todo o processo de transferências a variável difunde p_i pelos três nós de paridade de forma igual, ou seja, não se precisa de algo mais que os valores de p_i como ponto de partida do algoritmo (figura 3.2(b)). Depois, cada um dos nós de paridade envia a x_i uma nova mensagem que tem em conta exclusivamente a "opinião" (as estimativas) sobre x_i que acabou de receber de outros nós de variáveis (não mostrados na figura).

O recebimento das mensagens pelos nós de variáveis encerra a primeira iteração de cálculos. Daqui em diante esse procedimento repete-se com transferências dos nós de variáveis para os nós de paridade e vice-versa, de um

lado para o outro até se atingir um determinado número pré-estabelecido de iterações ou quando já houver "certezas" suficientes sobre a confiabilidade das mensagens em circulação (por exemplo, verificando se a estimativa final $\hat{x}$ satisfaz $\hat{x}.H^T = 0$, onde H é a matriz de verificação de paridade).

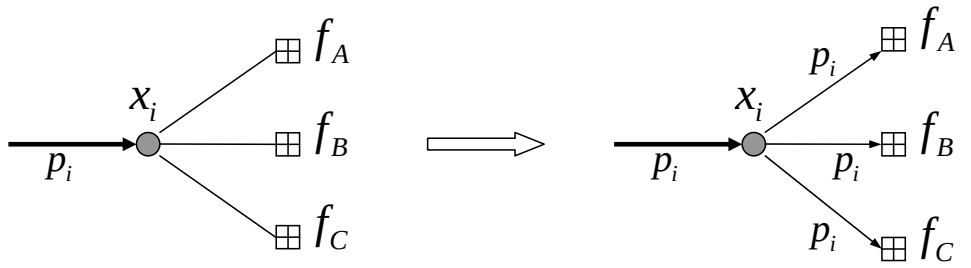
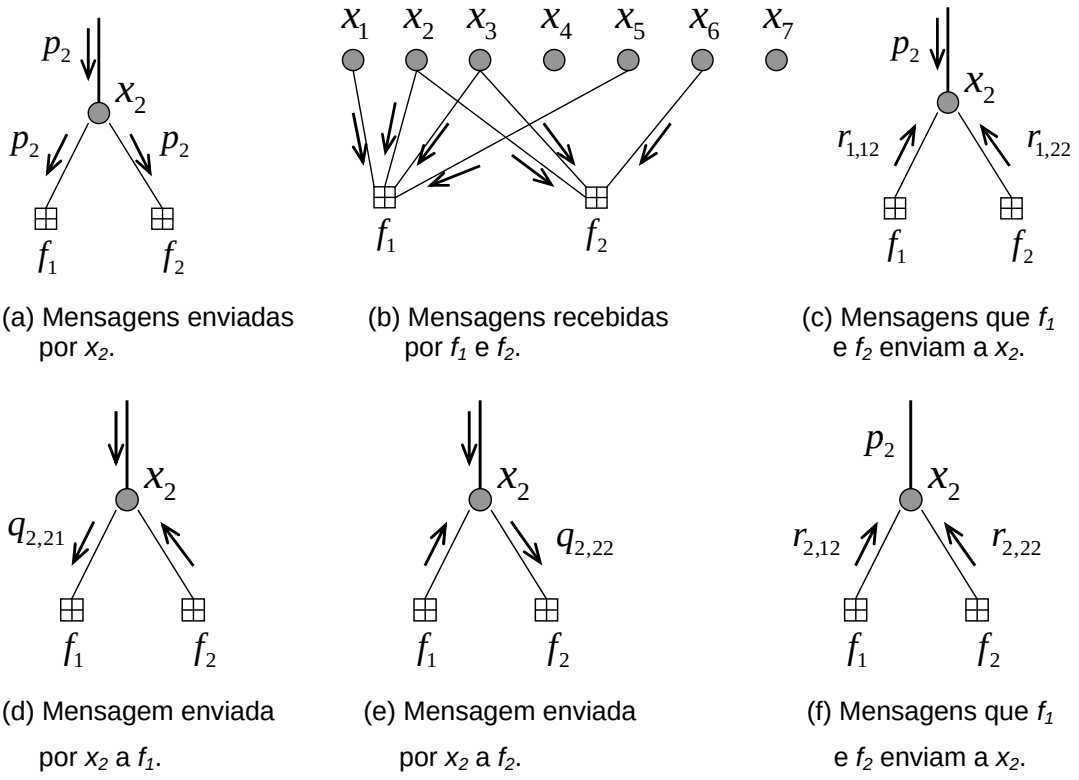


Figura 3.2: Início da difusão de mensagens dos nós de variáveis para os nós de paridade.

Para exemplificar com um caso concreto consideremos o grafo de Tanner de uma sequência $\underline{x}$ qualquer de n bits (n variáveis x_i) e λ nós de paridade f_i, sendo λ bem menor que n, conforme ilustra a figura 3.3 e seus vários passos relativos à variável x_2.



(a) Mensagens enviadas por x_2.

(b) Mensagens recebidas por f_1 e f_2.

(c) Mensagens que f_1 e f_2 enviam a x_2.

(d) Mensagem enviada por x_2 a f_1.

(e) Mensagem enviada por x_2 a f_2.

(f) Mensagens que f_1 e f_2 enviam a x_2.

Figura 3.3: Exemplo de transferência de mensagens entre os nós de grafos bipartidos.

1. No início, ver figura 3.3(a), a variável x_2 espalha pelos nós de paridade f_1 *e* f_2 a mensagem (a probabilidade *a priori*) que recebeu do canal de correlação, p_2, onde $p_2 = p(y \mid x_2)$.

2. Mas o nó f_1 também recebeu estimativas de x_1, x_3 e x_5 e o nó f_2 também recebeu estimativas de x_3 e x_6, conforme figura 3.3(b).

3. Cada um dos nós f_1 *e* f_2 processa as mensagens que não vieram de x_2 e envia a x_2 o resultado, chamados de $r_{1,12}$ e $r_{1,22}$, respectivamente (ver figura 3.3(c)). Aqui se utilizou a seguinte nomenclatura para os índices de *r*: o número antes da vírgula indica a iteração atual; o primeiro algarismo após a vírgula indica o nó de paridade; e o segundo algarismo após a vírgula indica o nó de variável.

4. O processo agora se repete com as mensagens a circular das variáveis para os nós de paridade e destes para as variáveis: de acordo com a figura 3.3(d) e voltando à variável x_2, esta envia a f_1 uma mensagem $m_{2,21}$ (da 2ª iteração) baseada no que os outros nós de paridade, p_2 e f_2, lhe enviaram, e envia a f_2 uma outra mensagem, $m_{2,22}$, esta baseada no que recebeu de p_2 e f_1 (ver figura 3.3(e)).

5. Os nós f_1 *e* f_2 enviam a x_2 novas mensagens (respostas) mais refinadas e o processo se repete (ver figura 3.3.(f)).

Note que x_i recebe sempre a mesma mensagem de probabilidade *a priori*, p_i, em todas as iterações. Convém notar que termos como “mensagens”, “probabilidades” e “estimativas de probabilidades” são usados neste capítulo praticamente como sinônimos, consoante o contexto.

3.3.4. Regras de Atualização de Mensagens Probabilísticas

No algoritmo da *soma-e-produto* as mensagens são calculadas de acordo com as eq. (3.6) a (3.9) abaixo:

- Do nó de variável x_i para o nó de paridade f_j:

$$q_{ij}(0) = K_{ij}(1 - p_i)\prod_{j' \neq j} r_{j'i}(0) \qquad (3.6)$$

$$q_{ij}(1) = K_{ij}\, p_i \prod_{j' \neq j} r_{j'i}(1) \tag{3.7}$$

Nas equações (3.6) e (3.7) acima, *j'* representa genericamente todos os nós de paridade, exceto o nó *j* em questão (para o qual a mensagem está sendo enviada).

- Do nó de paridade f_j para o nó de variável x_i:

$$r_{ji}(0) = \frac{1}{2}\left\{1 + \prod_{i' \neq i}\left[1 - 2q_{i'j}(1)\right]\right\} \tag{3.8}$$

$$r_{ji}(1) = 1 - r_{ji}(0) \tag{3.9}$$

Nas equações (3.8) e (3.9) acima, *i'* representa genericamente todos os nós de variável, exceto o nó *i* em questão (para o qual a mensagem está sendo enviada).

O fator K_{ij} é um fator de normalização com valor tal que $q_{ij}(0) + q_{ij}(1) = 1$ e a função de várias variáveis $f_j(x_1,x_2,\ldots)$ é uma função indicadora de pertença ao código que só toma dois valores, 0 e 1, consoante as equações de paridade do código serem ou não satisfeitas. Assim, pode-se escrever:

$$f_j(x_1, x_2, ..., x_n) = \delta(x_1 \oplus x_2 \oplus ... \oplus x_n) = \begin{cases} 1 & x_1 \oplus x_2 \oplus ... \oplus x_n = 0 \\ 0 & x_1 \oplus x_2 \oplus ... \oplus x_n \neq 0 \end{cases} \tag{3.10}$$

Na eq. (3.10), $\delta(x)$ representa o delta de Kronecker:

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases} \tag{3.11}$$

As eq. (3.6) e (3.7) expressam que a mensagem que uma variável transmite a um dos nós de paridade a que está ligada, q_{ij}, é igual ao produto, devidamente normalizado, das mensagens que acabou de receber dos outros nós de paridade, $r_{j'i}$, incluindo a do canal p_i. As eq. (3.8) e (3.9) expressam que a mensagem que um nó de paridade envia a um dos nós de variáveis a que está ligado, r_{ji}, é igual à soma dos produtos das mensagens que o nó acabou de receber, $q_{i'j}$, das outras variáveis e que satisfazem a equação de paridade de f_j.

A estimativa final da probabilidade *a posteriori* $p(x_i|\underline{y})$ produzida por cada nó de variável x_i é igual ao produto normalizado de todas as mensagens recebidas dos nós de paridade, incluindo a mensagem *a priori* do canal:

$$p(x_i \mid \underline{y}) = K_i \prod_j r_{ji}(x_i) \tag{3.12}$$

Alternativamente, as mensagens dos nós de paridade f_i para o nó de variável x_i podem ser escritas numa forma mais compacta como:

$$r_{ji}(x_i) = \frac{1}{2}\left\{1 + (-1)^{x_i} \prod_{i' \neq i} \left[1 - 2q_{i'j}(1)\right]\right\} \tag{3.13}$$

Como $q_{ij}(0) + q_{ij}(1) = 1$, então:

$$K_{ij}(1 - p_i) \prod_{j' \neq j} r_{j'i}(0) + \prod_{j' \neq j} r_{j'i}(1) = 1, \tag{3.14}$$

de onde se tira que:

$$K_{ij} = \frac{1}{p_i \prod_{j' \neq j} r_{j'i}(1) + (1 - p_i) \prod_{j' \neq j} (1 - r_{j'i}(1))} \tag{3.15}$$

Considerando-se que para o código proposto estamos trabalhando apenas com variáveis binárias e mensagens de um único elemento com a estimativa de $p(x_i = 1|\underline{y})$, pode-se simplificar a notação ainda mais se forem escritos r_{ji} *e* q_{ij} em vez de $r_{ji}(1)$ *e* $q_{ij}(1)$ e se não usar $r_{ji}(0)$ e $q_{ij}(0)$ por serem redundantes devido à normalização. Desta forma, serão utilizadas as seguintes equações de atualização:

- Do nó de variável x_i para o nó de paridade f_i:

$$q_{ij} = K_{ij} p_i \prod_{j' \neq j} r_{j'i} \tag{3.16}$$

onde

$$K_{ij} = \frac{1}{p_i \prod_{j' \neq j} r_{j'i} + (1 - p_i) \prod_{j' \neq j} (1 - r_{j'i})} \tag{3.17}$$

- Do nó de paridade f_i para o nó de variável binária x_i (expressão mais simples):

$$1 - 2r_{ji} = \prod_{i' \neq i} (1 - 2q_{i'j}) \tag{3.18}$$

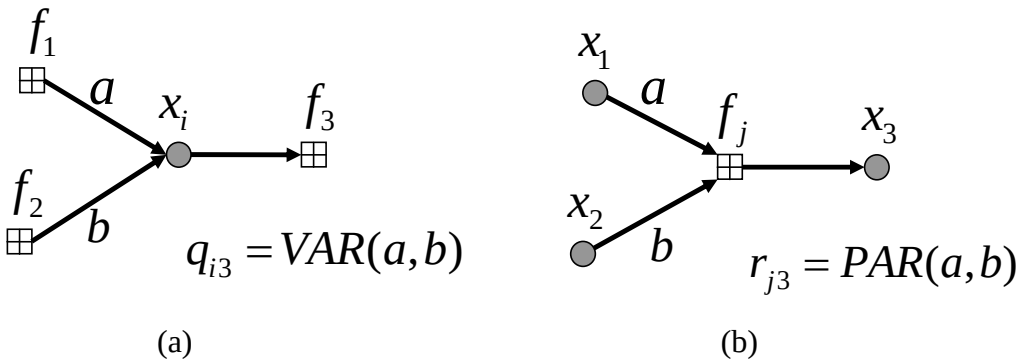
a qual resulta em:

$$r_{ji} = \frac{1}{2}\left[1 - \prod_{i' \neq i} (1 - 2q_{i'j})\right] \tag{3.19}$$

3.3.5. Método para Síntese de nós de Síndrome

As mensagens dos nós de um dado grau podem ser expressas à custa de mensagens de nós de grau inferior. Consideremos, por exemplo, uma mensagem de um nó de variável de grau 3, como é ilustrado na figura 3.4. Se tiver recebido as mensagens *a* e *b* de dois nós de paridade, o nó de variável envia para o terceiro nó de paridade uma mensagem que é função de *a* e *b* (e que será chamado aqui de *VAR*(a,*b*)):

Grau 3: $$VAR(a,b) = \frac{ab}{ab + (1-a)(1-b)} \tag{3.20}$$



(a) (b)

Figura 3.4: As funções VAR e PAR em nós de grau 3.

Consideremos agora um nó de variável de grau 4 ao qual chegam as mensagens a, b e c. A mensagem que ele transmite para o quarto nó de paridade é:

Grau 4: $$VAR(a,b,c) = \frac{abc}{abc + (1-a)(1-b)(1-c)} \tag{3.21}$$

Dividindo o numerador e o denominador da eq. (3.20) por *ab* + (1-*a*)(1-*b*) e fazendo:

$$\beta = \frac{ab}{ab + (1-a)(1-b)} \tag{3.21}$$

E, portanto,

$$1-\beta = \frac{(1-a)(1-b)}{ab+(1-a)(1-b)}, \tag{3.22}$$

obtém-se:

$$VAR(a,b,c) = \frac{\beta c}{\beta c+(1-\beta)(1-c)} = VAR(\beta,c)\,. \tag{3.23}$$

Como $\beta = VAR(a,b)$, então:

$$VAR(a,b,c) = VAR(c,VAR(a,b)) \tag{3.24}$$

logo, estamos exprimindo a mensagem de grau 4 à custa da mensagem de grau 3. Generalizando para nós de grau n, tem-se que:

Grau *n*: $$q_{in} = VAR(f_1, f_2, \ldots, f_{n-1}) = VAR(f_{n-1}, VAR(f_1, \ldots, f_{n-2})) \tag{3.25}$$

Seguindo o mesmo raciocínio, chega-se a algo semelhante relacionado aos nós de paridade:

Grau 3: $$PAR(a,b) = a(1-b)+b(1-a) \tag{3.26}$$

Grau *n*: $$r_{jn} = PAR(x_1, x_2, \ldots, x_{n-1}) = PAR(x_{n-1}, PAR(x_1, \ldots, x_{n-2})) \tag{3.27}$$

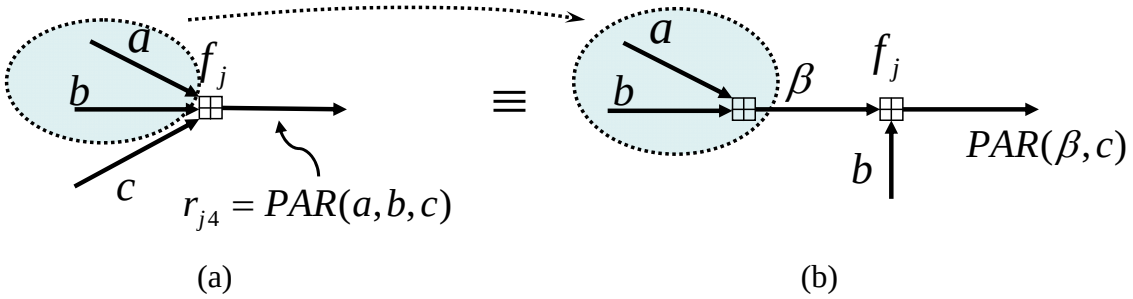
Para chegar nas eq. (3.25) acima, deve-se partir da eq. (3.17). Assim, para nós de grau 3 (ver exemplo ilustrado pela figura 3.4), tem-se:

$$r_{jn} = PAR(a,b) = \frac{1}{2}\left[1-\prod_{i'\neq i}^{n}(1-2q_{i'j})\right] = \frac{1}{2}\left[1-(1-2a)(1-2b)\right] =$$

$$r_{jn} = PAR(a,b) = a-ab+b-ab = a(1-b)+b(1-a) \tag{3.28}$$

E se o nó de paridade tiver, por exemplo, grau 4 (ver figura 3.5), a mensagem que será enviada pode ser expressa como:

$$PAR(a,b,c) = \frac{1}{2}\left[1-(1-2a)(1-2b)(1-2c)\right] \tag{3.29}$$



(a) (b)

Figura 3.5: Síntese de nó de paridade de grau 3.

No entanto, fazendo $(1-2a)(1-2b)=1-2\beta$, ou seja, $\beta = PAR(a,b)$, então chega-se a:

$$PAR(a,b,c) = \frac{1}{2}\left[1-(1-2\beta)(1-2c)\right] = PAR(\beta,c)$$

$$PAR(a,b,c) = PAR(c, PAR(a,b)) \tag{3.30}$$

Esta propriedade dos grafos de Tanner, descrita pelas equações (3.26) e (3.28), é bastante útil neste trabalho já que se pretende transmitir somente as síndromes acumuladas, onde uma síndrome acumulada representará as síndromes de grau inferiores, conforme descreve a seção 3.3.6.1.

3.3.6. Algoritmo de Produto-e-Máximo

Observe o nó de paridade de grau 3 da figura 3.3(b). a mensagem transmitida resulta de um produto (conforme eq. (3.19)) que, como se viu, vale $r_{j3} = PAR(a,b) = a(1-b)+b(1-a)$. Segundo [66], uma simplificação do algoritmo substitui este cálculo pelo valor máximo entre duas parcelas:

$$r_{j3} \simeq máx\left(a(1-b)+b(1-a)\right) \tag{3.32}$$

Para um nó de grau 4, a simplificação do cálculo seria descrita por:

$$r_{j4} \simeq máx\left(a(1-b)(1-c), b(1-a)(1-c), c(1-a)(1-b), abc\right) \tag{3.33}$$

Com nós de paridade de grau superior as multiplicações da eq. (3.19) são simplificadas de maneira idêntica. Seguindo a mesma lógica utilizada na eq. (3.33), toma-se apenas o valor máximo dos diversos produtos envolvidos:

$$r_{jn} \simeq máx\big(q_{1n}(1-q_{2n})...(1-q_{n-1\,n}), q_{2n}(1-q_{1n})...(1-q_{n-1\,n}),$$
$$..., q_{n-1\,n}(1-q_{1n})...(1-q_{n-2\,n}), q_{1n}q_{2n}...q_{n-1\,n}\big) \qquad (3.34)$$

A este algoritmo simplificado, em que as mensagens dos nós de variáveis resultam de um produto e as mensagens dos nós de paridade são encontradas através de um valor máximo, dá-se o nome de algoritmo produto-e-máximo.

3.3.6.1. Codificação com o Código LIA

De forma resumida, o codificador LDPC irregular acumulado e adaptativo (LIA) consiste em um LDPC irregular, o qual foi apresentado na seção anterior, gerador de síndromes, concatenado com um acumulador e com uma saída de comprimento variável (síndrome acumulada), cujo funcionamento é descrito nas seções abaixo. A palavra “irregular” está associada à distribuição de probabilidades associada aos graus das síndromes, e a palavra adaptativa está relacionada à transmissão de uma síndrome acumulada de comprimento variável, conforme a requisição de taxa mínima sinalizada pelo decodificador. Um exemplo de um codificador LDPC concatenado com um acumulador é mostrado na figura 3.6 seguinte.

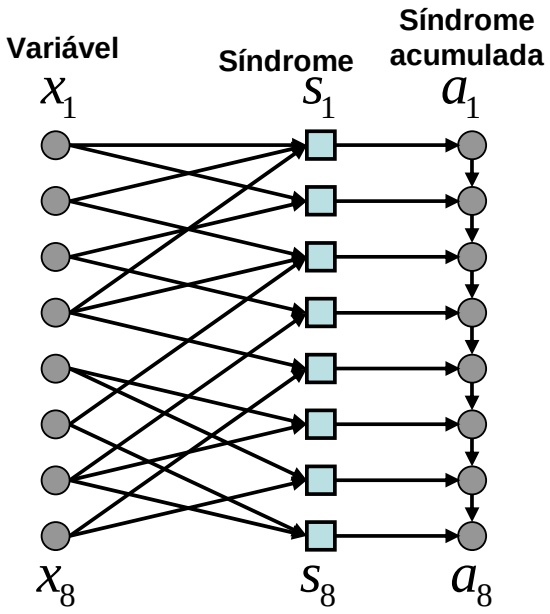


Figura 3.6: Codificador de baixa densidade, irregular e acumulado.

Conforme ilustra a figura 3.6, primeiro os bits da fonte (x_1,..., x_8) são somados em módulo 2 nos nós da síndrome conforme o processo de geração de conexões irregulares através da matriz paridade gerada pelo código, ou seja, segundo a estrutura pré-definida do grafo LDPC (a qual é conhecida também pelo decodificador), gerando os bits da síndrome (s_1,..., s_8). Estes bits da síndrome são, por sua vez, acumulados em módulo 2, produzindo a síndrome acumulada (a_1,..., a_8), conforme a equação abaixo:

$$a_i = \sum_{j=1}^{i} s_j \quad \text{ou seja,} \quad a_i = \begin{cases} s_i, & i = 1 \\ s_i \oplus a_{i-1}, & i \neq 1 \end{cases} \tag{3.35}$$

O codificador armazena a síndrome acumulada e a transmite incrementalmente para o decodificador. Esta estrutura de codificador pode ser simplesmente reformulada como um código de canal IRA estendido (eIRA – *extended Irregular Repeat Accumulate code*) [67]. A interpretação do código eIRA, porém, trata os nós das síndromes acumuladas como nós de variáveis de grau 2, induzindo a uma distribuição de graus diferente daquela utilizada aqui para o codec DVC, a qual é irregular (não é um valor constante para todos os nós) e muito maior que 2. A próxima seção demonstra que somente esta distribuição de graus (do LDPC acumulado) é invariante sob a operação de taxa adaptativa.

3.3.6.2. Decodificação com o Código LIA

O decodificador LIA trabalha de forma adaptativa em relação à taxa, modificando seu grafo de decodificação cada vez que ele recebe um incremento de síndromes acumuladas. Primeiro assuma, por questão de argumento, que a síndrome acumulada inteira (a_1,..., a_8) foi recebida. Depois, tomando-se as diferenças consecutivas em módulo 2 destes valores, obtêm-se as síndromes (s_1,..., s_8):

Codificação		Decodificação	
$a_1 = s_1$	$\Rightarrow$	$s_1 = a_1$	
$a_2 = a_1 \oplus s_2$	$\Rightarrow$	$s_2 = a_1 \oplus a_2$	
$a_3 = a_2 \oplus s_3$	$\Rightarrow$	$s_3 = a_2 \oplus a_3$	(3.36)
$\vdots$		$\vdots$	
$a_n = a_{n-1} \oplus s_n$	$\Rightarrow$	$s_n = a_{n-1} \oplus a_n$	

O método iterativo de decodificação LDPC, ajustado para trabalhar com o envio de síndromes, pode ser aplicado sobre o mesmo grafo (figura 3.7) que foi usado para codificar (s_1,..., s_8) de (x_1,..., x_8). Todas as variáveis x_i são obtidas efetuando-se a operação inversa. Nesse exemplo ilustrado pela figura 3.7, $x_2 = s_2 \oplus s_3$ e $x_6 = s_2 \oplus s_4$.

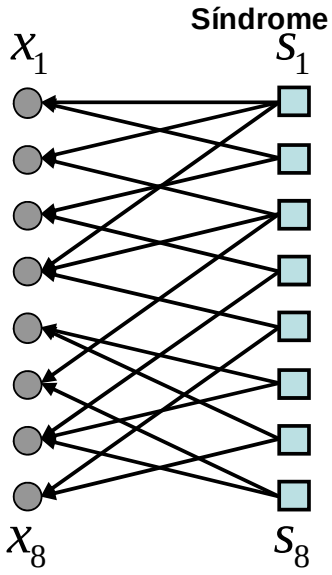


Figura 3.7: Grafo de decodificação das variáveis com o codificador transmitindo todas as síndromes acumuladas.

Para decodificação, os nós da fonte x_i são associados a distribuições de probabilidade condicionais dos bits da fonte dada a informação lateral, as quais são denominadas $p\left(x_1 \mid \Omega_1, \underline{y}\right)$, $p\left(x_2 \mid \Omega_2, \underline{y}\right)$, ... , $p\left(x_8 \mid \Omega_8, \underline{y}\right)$, onde Ω_i é o conjunto de nós de síndrome aos quais x_i está ligado. Por conseguinte, as mensagens são recuperadas, obtendo-se os nós da síndrome e por seguinte, os nós da fonte (de acordo com as equações em [23]). Este processo se repete até que as estimativas dos bits da fonte tenham convergido. O processo detalhado de decodificação dos bitplanes é descrito na seção 4.6.1. A indicação de uma recuperação correta dos valores da fonte é testada em relação aos bits da síndrome, com uma chance muito pequena de uma sinalização positiva falsa. Quando o número de bits recebidos se iguala ao número de bits da fonte, o desempenho atingido transmitindo-se as síndromes acumuladas (a_1,..., a_8) não é diferente daquele obtido quando se transmite somente as síndromes (s_1,..., s_8), visto que os grafos de decodificação resultantes são idênticos.

A modificação da estrutura do grafo de decodificação torna-se mais relevante em altas taxas de compressão, conforme o exemplo seguinte: considere a utilização do código proposto com uma razão de compressão igual a 2. No exemplo apresentado aqui, isto corresponde à transmissão de somente um subconjunto de síndromes acumuladas de índice par (a_2, a_4, a_6, a_8). A operação consecutiva de diferença em módulo 2 no decodificador produzirá ($s_1 + s_2$, $s_3 + s_4$, $s_5 + s_6$, $s_7 + s_8$). A figura 3.8 mostra o grafo no qual teriam sido codificadas as síndromes ($s_1 + s_2$, $s_3 + s_4$, $s_5 + s_6$, $s_7 + s_8$) de (x_1,..., x_8).

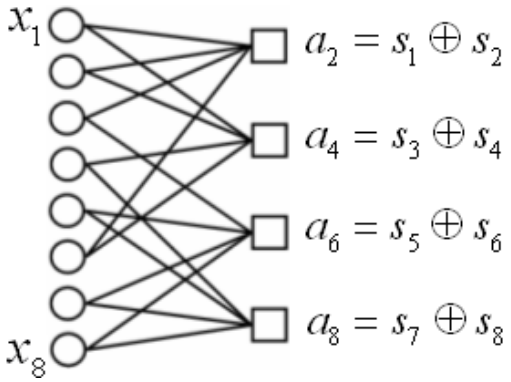


Figura 3.8: Grafo de codificação com o codificador transmitindo somente as síndromes acumuladas de índice par.

O grafo da figura 3.8 mantém o grau dos nós da fonte (neste exemplo, grau 2) quando comparado ao da figura 3.6. Portanto, ele pode ser usado para uma decodificação iterativa efetiva com os bits da fonte estimados com base nas probabilidades $p\left(x_i \mid \Omega_i, \underline{y}\right)$. Após completar a decodificação, a fonte recuperada, ou melhor, a palavra-código decodificada, pode ser testada em relação à síndrome a fim de verificar se há necessidade de realizar uma nova correção de erros. Para comparação, se o subconjunto de síndromes (s_2, s_4, s_6, s_8) fosse transmitido ao invés do subconjunto de síndromes acumuladas, o grafo de decodificação seria severamente degradado e inadequado à decodificação iterativa, conforme ilustrado na figura 3.9, onde se observa que as variáveis x_3, x_4 e x_8 não teriam disponíveis os valores das síndromes a eles ligadas.

A complexidade de codificação e decodificação do código LIA é linear em relação ao número máximo de conexões, o qual é invariante sob a construção proposta. Além disso, o número de conexões é linear em relação ao comprimento do código para uma distribuição de graus fixa. Portanto, a complexidade de codificação e decodificação é *O(n)*, onde *n* é o comprimento do bloco do código em bits.

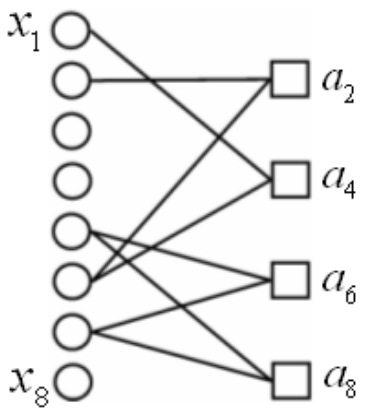


Figura 3.9: Grafo de decodificação com o codificador transmitindo os bits das síndromes de índice par.

3.4. Considerações sobre o Código Proposto

Considerar-se-á, agora, algumas propriedades do código LIA proposto e será mostrado como as técnicas do método LDPC existente foram aperfeiçoadas para uso na configuração do codec proposto.

Começamos com a observação de que o grafo de decodificação da figura 3.8 é obtido a partir do grafo da figura 3.6, juntando nós de síndrome adjacentes. As conexões ligadas a cada nó combinado a_i são aquelas que foram conectadas a um de seus nós constituintes, mas não a ambos (pois um dupla conexão se anula em soma módulo 2). Para assegurar que nenhuma conexão é perdida sobre diversos passos de junção, deve-se fazer um projeto bastante cuidadoso no que diz respeito à forma e ao grau de compressão do código.

Uma estratégia mais simples para garantir um número constante de conexões para todos os grafos de decodificação é começar o projeto com o grafo com maior relação de compressão. Conhecer a ordem de transmissão da síndrome acumulada, a taxa mínima $R_{WZ,mín}$ e o incremento de síndromes permite a dedução dos grafos para cada um dos códigos de menor razão de compressão. Cada incremento de bits recebidos da síndrome acumulada resulta na divisão de um nó de síndrome em dois nós adjacentes. A chave para manter um número constante de conexões em todos os grafos é particionar o conjunto de conexões do antigo nó da síndrome em um conjunto de conexões do novo par. Assim, a distribuição global dos graus pode ser usada para regular o desempenho dos códigos [80].

Quando o número de bits recebidos iguala ao número de bits da fonte, existe um objetivo adicional no projeto: as equações que geram os bits da síndrome acumulada devem ser independentes dos bits da fonte. Logo, para uma taxa de codificação igual a 1, isto garante a decodificação da fonte através de técnicas de álgebra linear, sem considerar a qualidade da informação lateral.

3.4.1.1. Avaliação da influência das Iterações do Decodificador LIA na Reconstrução dos Bitplanes

Neste experimento, busca-se avaliar a influência das iterações na decodificação do frame WZ. Assim, foram geradas algumas curvas com a PSNR

do frame WZ ao longo da sequência, em função do número máximo de iterações estabelecidas para o decodificador LIA. As figuras 3.10 a 3.18 mostram os resultados deste experimento para dois níveis de qualidade diferentes, ou seja, para dois valores de passo de quantização (QP) diferentes para *intraframes* e para duas TQ diferentes para os frames WZ.

Observando as figuras 3.10, 3.11, 3.12, 3.14 e 3.16, pode-se notar que o aumento quase geométrico do número de iterações não é proporcional ao aumento relativo da PSNR, ou seja, embora o número de iterações consideradas seja quase uma progressão geométrica de razão 2, a diferença de PSNR entre as curvas com i e com aproximadamente $2i$ iterações vai diminuindo à medida que i aumenta. Portanto, não é vantajoso utilizar um grande número de iterações na decodificação, pois isto demandaria um grande (e proporcional) tempo de processamento e implicaria em um ganho insignificante de qualidade. Para comprovação, percebam que as curvas que foram geradas utilizando-se 20 e 50 iterações possuem quase o mesmo PSNR, sendo que em alguns pontos este valor se iguala nas duas.

As figuras 3.11, 3.14, 3.16 e 3.18 apresentam as diferenças entre a PSNR utilizando uma e cinquenta iterações, para as configurações acima. Para as duas sequências, nota-se a grande variação da influência das iterações na decodificação do frame WZ ao longo da sequência. Comparando as curvas para as duas sequências, percebe-se que a influência das iterações para o caso da sequência *Foreman* é bem maior que para a sequência *News*, pois o uso de 50 iterações na primeira fornece um aumento de 0,4 a 1,21 dB para TQ=18 e de 0,39 a 1,28 dB para TQ=7, enquanto que na sequência *News* este aumento é de 0,27 a 0,63 dB para TQ=18 e de 0,07 dB a 0,64 dB para TQ=7, mostrando certa variabilidade do número de iterações adotadas (e sua influência) em relação à sequência a ser codificada.

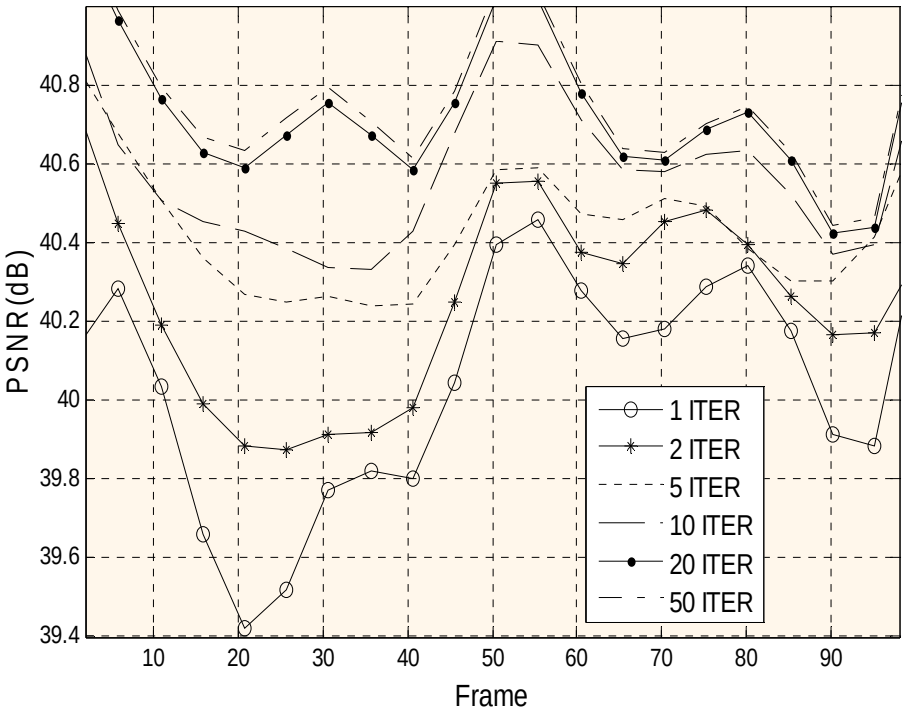


Figura 3.10: PSNR do frame Wyner-Ziv para diferentes números de iterações do decodificador LIA, para sequência *Foreman*, com QP = 25 (intra) e TQ = 18 (WZ).

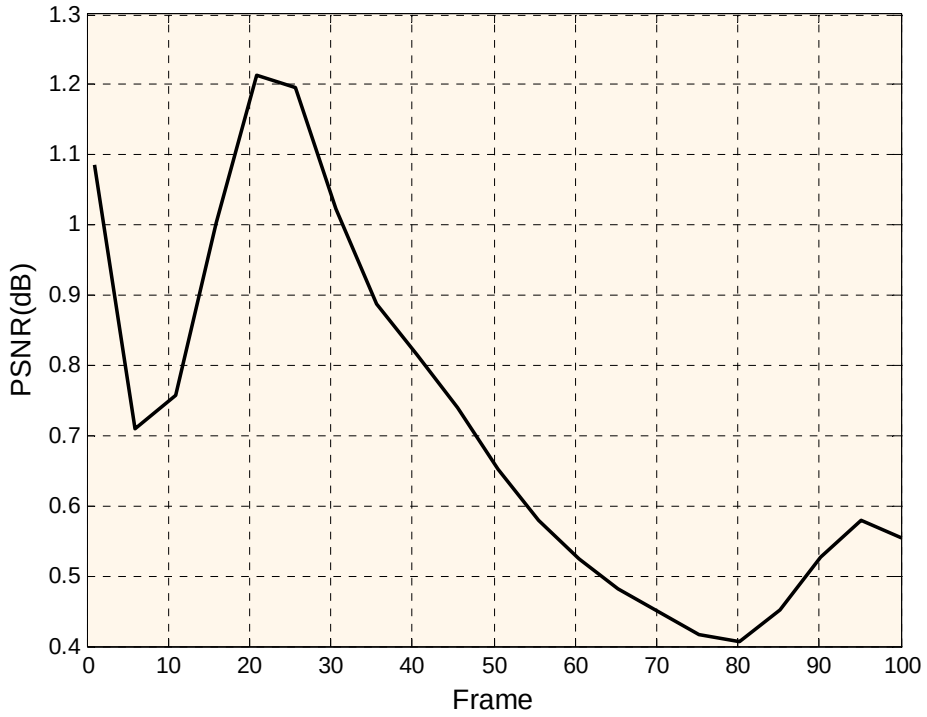


Figura 3.11: Diferença entre a PSNR do frame Wyner-Ziv utilizando uma e cinquenta iterações, para a sequência *Foreman*, com QP = 25 (intra) e TQ = 18 (WZ).

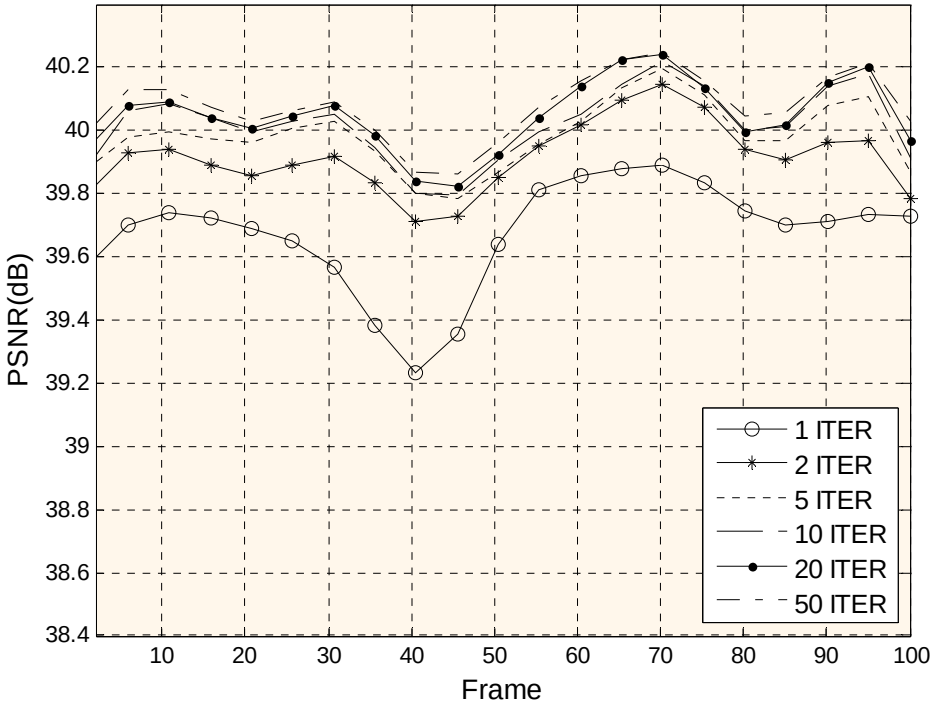


Figura 3.12: PSNR do frame Wyner-Ziv para números diferentes de iterações do decodificador LIA, para sequência *News*, com QP = 25 (intra) e TQ = 18 (WZ).

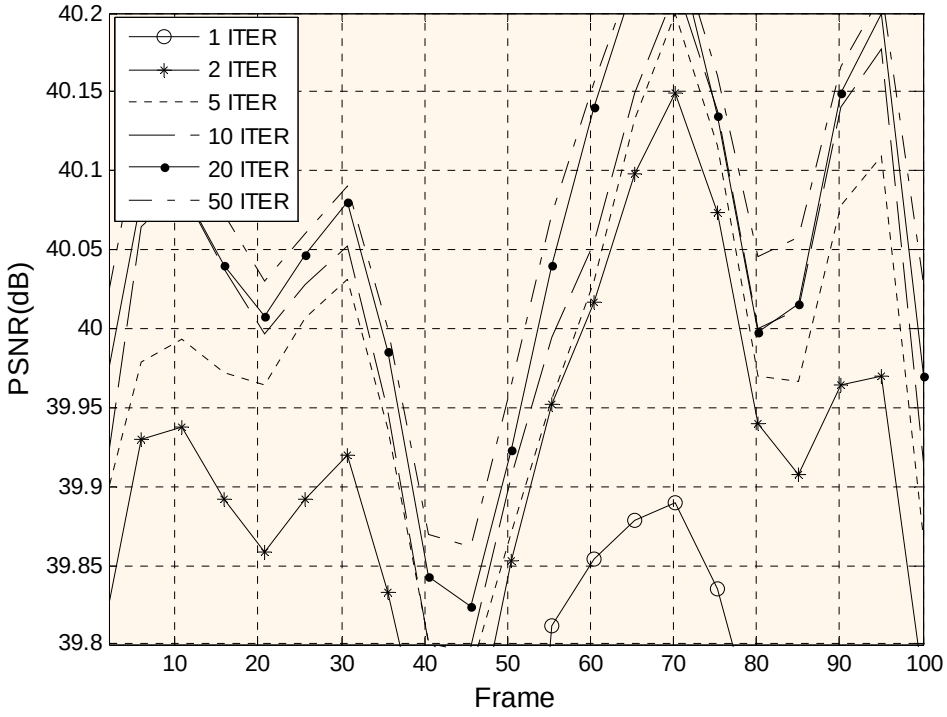


Figura 3.13: PSNR do frame Wyner-Ziv para números diferentes de iterações do decodificador LIA, para sequência *News*, com QP = 25 (intra) e TQ = 18 (WZ), em uma escala maior (melhor visualização).

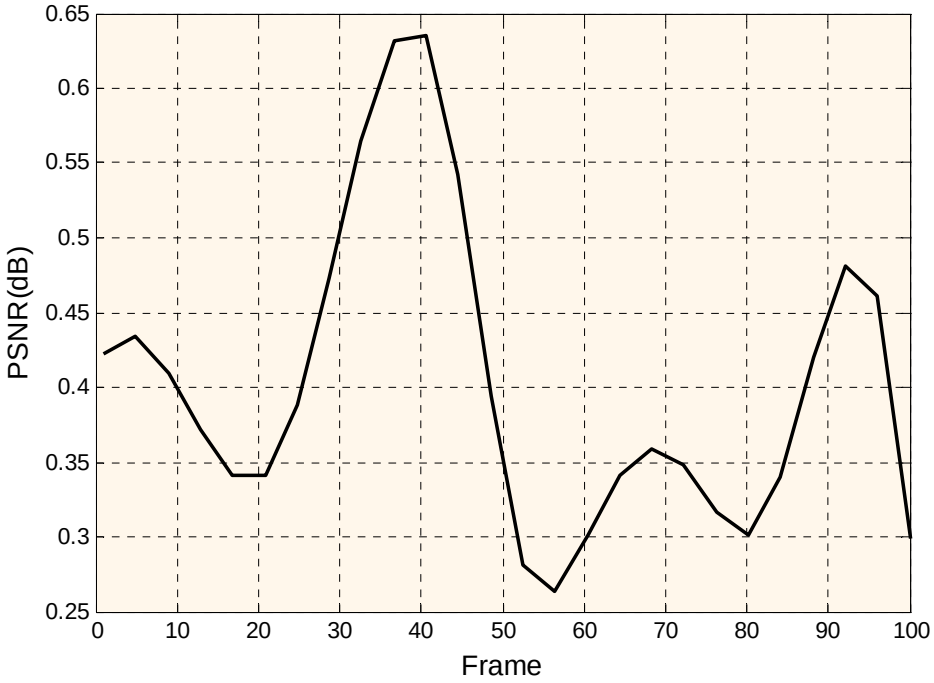


Figura 3.14: Diferença entre a PSNR do frame Wyner-Ziv utilizando uma e cinquenta iterações, para a sequência *News*, com QP = 25 (intra) e TQ = 18 (WZ).

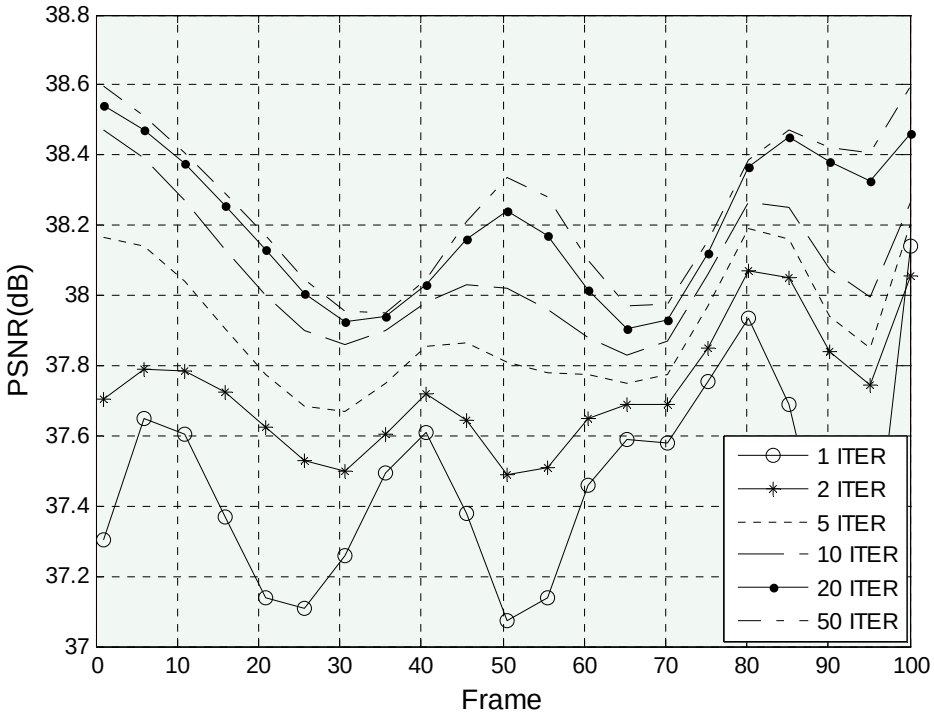


Figura 3.15: PSNR do frame Wyner-Ziv para números diferentes de iterações do decodificador LIA, para sequência *Foreman*, com QP = 26 (intra) e TQ = 7 (WZ).

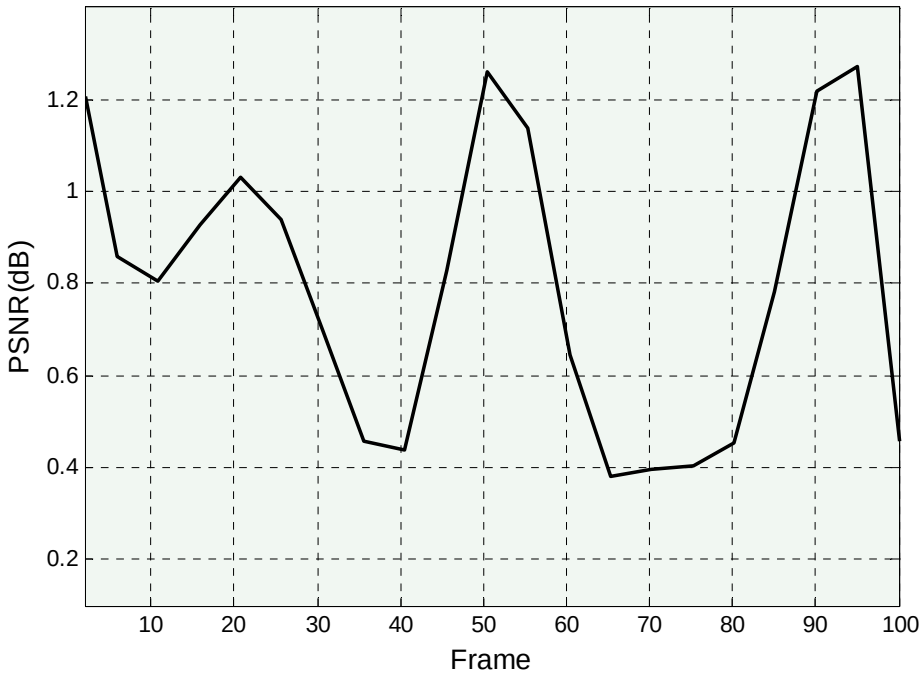


Figura 3.16: Diferença entre a PSNR do frame Wyner-Ziv utilizando uma e cinquenta iterações, para a sequência *Foreman*, com QP = 26 (intra) e TQ = 7 (WZ).

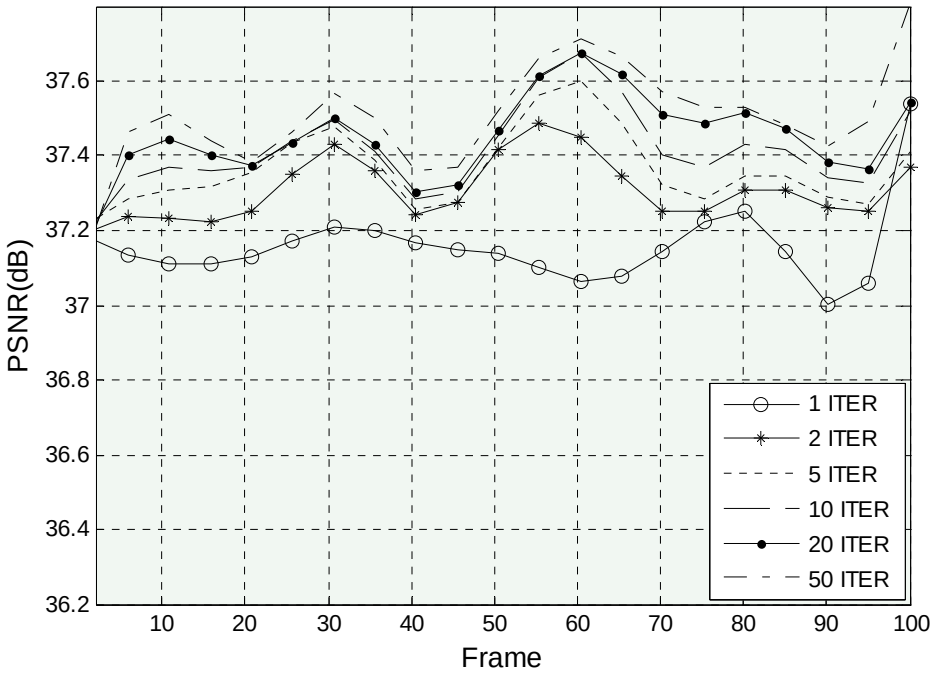


Figura 3.17: PSNR do frame Wyner-Ziv para números diferentes de iterações do decodificador LIA, para sequência *News*, com QP = 26 (intra) e TQ = 7 (WZ).

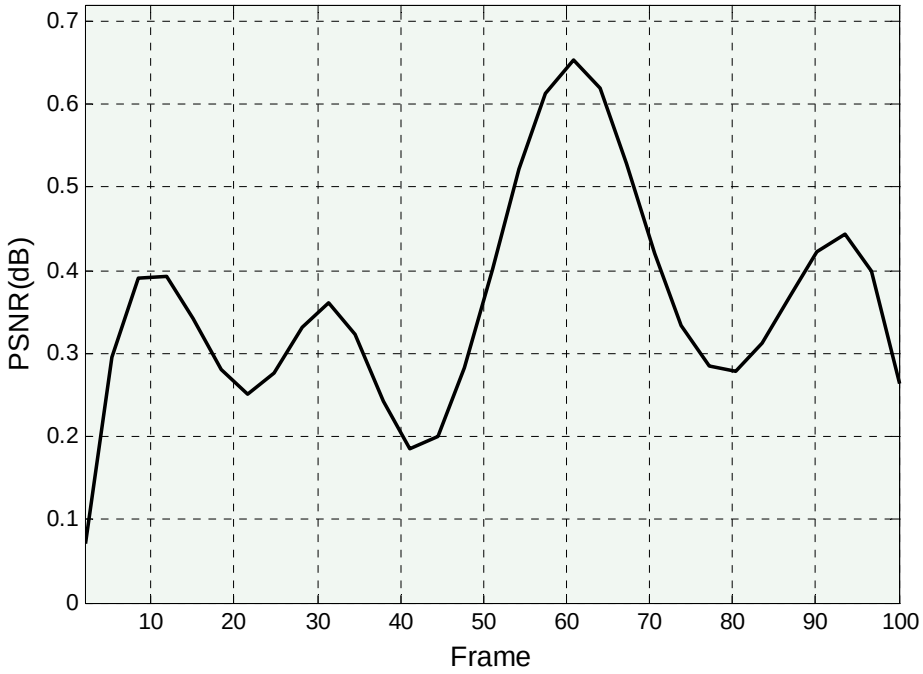


Figura 3.18: Diferença entre a PSNR do frame Wyner-Ziv utilizando uma e cinquenta iterações, para a sequência *News*, com QP = 26 (intra) e TQ = 7 (WZ).