4
Códigos de Permutação Vetorial

Este capítulo apresenta nossas contribuições à teoria de códigos de permutação para compressão de fontes. Códigos de permutação vetorial (VPC), definidos pela primeira vez em [14], são uma extensão interessante dos códigos de permutação escalar: o alfabeto é constituído não por símbolos escalares, mas por vetores L-dimensionais. Sabe-se que o código de permutação escalar tem desempenho semelhante ao do quantizador escalar com restrição de entropia. A intuição sugere que, da mesma forma, o código de permutação vetorial deve apresentar desempenho semelhante ao do quantizador vetorial com restrição de entropia (ECVQ).

Neste capítulo, códigos de permutação vetorial são desenvolvidos e comparados com outras técnicas de compressão. Como uma das principais contribuições, a equivalência assintótica entre VPC e ECVQ é demonstrada matematicamente. A partir deste resultado, é proposto um algoritmo eficiente para o projeto de VPC, o qual até então permanecia como um problema em aberto. Este algoritmo foi implementado em computador e aplicado ao projeto de VPC's para fontes gaussiana e uniforme. São exibidos resultados experimentais comprovando que o VPC de fato atinge desempenhos comparáveis ao do ECVQ, e portanto superiores aos do SPC e do ECSQ. Como contribuições adicionais, expressões explícitas para o projeto de alfabeto de VPC e para a codificação ótima através do algoritmo CSA são também incluídas neste trabalho.

A organização deste capítulo é descrita a seguir. A Seção 4.1 apresenta a definição de códigos de permutação vetorial. Na Seção 4.2, é desenvolvido o procedimento de codificação ótima de VPC. Na Seção 4.3, a equivalência assintótica entre VPC e ECVQ é estabelecida. O projeto de códigos de permutação vetorial é abordado na Seção 4.4, onde é proposto um algoritmo eficiente para este problema. A Seção 4.5 apresenta resultados experimentais que confirmam o bom desempenho do VPC em comparação com as outras técnicas analisadas. A Seção 4.6 apresenta comentários gerais sobre o trabalho descrito neste capítulo.

4.1 Definições

Inicialmente apresentamos uma definição útil para o desenvolvimento deste capítulo.

Definição 4.1 (Função de Permutação) Uma *função de permutação* é uma função bijetora $\pi : \mathcal{A} \rightarrow \mathcal{A}$, isto é, um mapeamento biunívoco de um conjunto $\mathcal{A}$ em si mesmo.[1] Estaremos particularmente interessados no conjunto $\mathcal{A} = \{1, 2, \ldots, n\}$. Nesse caso, a função de permutação π pode ser representada através da notação

$$\pi = \big[\pi_1, \pi_2, \ldots, \pi_n\big] \tag{4-1}$$

onde $\pi_j = \pi(j)$, $j = 1, \ldots, n$, isto é, um elemento na posição j representa o valor da função para um argumento de valor j. A função inversa de π é dada por

$$\pi^{-1} = \big[\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n)\big] \tag{4-2}$$

de forma que $\pi^{-1}(\pi(j)) = \pi(\pi^{-1}(j)) = j$, $j = 1, \ldots, n$. O conjunto de todas as funções de permutação de $\{1, \ldots, n\}$ é denotado por Π_n.

Uma função de permutação pode ser utilizada para representar o rearranjo dos componentes de um vetor. A partir de um vetor $\boldsymbol{a} = (a_1, \ldots, a_n)$, podemos construir um vetor $\boldsymbol{b} = (a_{\pi(1)}, \ldots, a_{\pi(n)})$ formado pelos componentes do vetor $\boldsymbol{a}$ rearranjados de acordo com π.

Exemplo 4.2 Sejam as funções de permutação $\pi, \sigma \in \Pi_n$ tais que $\sigma = \pi^{-1}$. Sejam $\boldsymbol{a}$ e $\boldsymbol{b}$ vetores de comprimento n formados por símbolos de um alfabeto qualquer, e seja $n = 5$. Se

$$\boldsymbol{a} = (a_1, a_2, a_3, a_4, a_5) \tag{4-3}$$

$$\pi = \big[4, 1, 5, 3, 2\big] \tag{4-4}$$

e fizermos $b_j = a_{\pi(j)}$, $j = 1, \ldots, n$, teremos

$$\boldsymbol{b} = (a_4, a_1, a_5, a_3, a_2) \tag{4-5}$$

[1] Note que, na álgebra abstrata, a função de permutação π é denominada simplesmente por *permutação*. Optamos por usar uma denominação diferente de forma a não causar confusão com os dois significados tradicionais do termo "permutação": (1) uma lista ordenada obtida pelo rearranjo dos símbolos de outra lista; e (2) o ato ou efeito de *permutar* (rearranjar).

isto é, $\boldsymbol{b}$ é formado pelos componentes do vetor $\boldsymbol{a}$ ordenados de acordo com π. Analogamente, a relação inversa é dada por $a_j = b_{\sigma(j)}$, $j = 1, \ldots, n$, onde

$$\sigma = \pi^{-1} = \big[2, 5, 4, 1, 3\big]. \tag{4-6}$$

Assim,

$$\boldsymbol{a} = (b_2, b_5, b_4, b_1, b_3). \tag{4-7}$$

■

Definição 4.3 (Código de Permutação Vetorial) Seja $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$ um conjunto de vetores L-dimensionais reais distintos e seja $\{n_1, \ldots, n_K\}$ um conjunto de inteiros positivos tais que

$$n = \sum_{k=1}^{K} n_k. \tag{4-8}$$

Considere a seguinte *palavra de referência*, de comprimento n, formada por n_k repetições de cada símbolo $\boldsymbol{\mu}_k$ em ordem crescente de índice:

$$\underline{\boldsymbol{y}}_1 = (\underbrace{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_1}_{n_1}, \underbrace{\boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_2}_{n_2}, \ldots, \underbrace{\boldsymbol{\mu}_K, \ldots, \boldsymbol{\mu}_K}_{n_K}). \tag{4-9}$$

Um *código de permutação vetorial* (VPC)[2] é um quantizador vetorial estruturado, de taxa fixa e de dimensão nL, cujo dicionário $\mathcal{C} = \{\underline{\boldsymbol{y}}_i\}_{i=1}^{M}$ é formado por todas as palavras que podem ser obtidas permutando-se os componentes de $\underline{\boldsymbol{y}}_1 = (\boldsymbol{y}_{11}, \ldots, \boldsymbol{y}_{1n})$. Isto é, uma palavra $\underline{\boldsymbol{y}}$ pertence a $\mathcal{C}$ se e somente se existe uma função de permutação $\sigma \in \Pi_n$ tal que $\underline{\boldsymbol{y}} = (\boldsymbol{y}_{1,\sigma(1)}, \ldots, \boldsymbol{y}_{1,\sigma(n)})$. Os conjuntos $\{\boldsymbol{\mu}_k\}_{k=1}^{K}$ e $\{n_k\}_{k=1}^{K}$ são chamados, respectivamente, de *alfabeto* e *vetor de composição*.

A taxa do código, em bits por amostra escalar, é dada por

$$R = \frac{1}{nL} \log M \tag{4-10}$$

onde o tamanho do dicionário M é dado por

$$M = \frac{n!}{\prod_{k=1}^{K} n_k!}. \tag{4-11}$$

[2]De *vector permutation code*.

4.2
Codificação Ótima

Utilizando o esquema ótimo de codificação, uma palavra da fonte $\underline{\boldsymbol{X}} = (\boldsymbol{X}_1 \ \ldots \ \boldsymbol{X}_n)$, onde cada vetor aleatório $\boldsymbol{X}_j$ é um vetor L-dimensional real, será reproduzida pela palavra de $\mathcal{C}$ que produz a menor distorção em relação a $\underline{\boldsymbol{X}}$, isto é, pela palavra

$$\hat{\underline{\boldsymbol{X}}} = \arg\min_{\underline{\boldsymbol{y}}' \in \mathcal{C}} d(\underline{\boldsymbol{X}}, \underline{\boldsymbol{y}}'). \tag{4-12}$$

Como descrito na seção anterior, todas as palavras do dicionário $\mathcal{C}$ de um código de permutação vetorial estão relacionadas à palavra de referência $\underline{\boldsymbol{y}}_1$ através de uma permutação de símbolos. Reescrevendo (4-12) para explicitar essa relação, obtemos que o codificador ótimo reproduz $\underline{\boldsymbol{X}}$ pela palavra

$$\hat{\underline{\boldsymbol{X}}} = (\boldsymbol{y}_{1,\sigma(1)}, \ldots, \boldsymbol{y}_{1,\sigma(n)}) \tag{4-13}$$

onde σ é uma função de permutação em Π_n dada por[3]

$$\sigma = \arg\min_{\sigma' \in \Pi_n} \frac{1}{n} \sum_{j=1}^{n} d(\boldsymbol{X}_j, \boldsymbol{y}_{1,\sigma'(j)}). \tag{4-14}$$

Proposição 4.4 Se $\pi, \sigma \in \Pi_n$ e $\pi = \sigma^{-1}$, então

$$\sum_{j=1}^{n} d(\boldsymbol{x}_j, \boldsymbol{y}_{1,\sigma(j)}) = \sum_{\ell=1}^{n} d(\boldsymbol{x}_{\pi(\ell)}, \boldsymbol{y}_{1\ell}). \tag{4-15}$$

Prova. Fazendo $\ell = \sigma(j)$, temos que $j = \pi(\ell)$. Como σ é uma função de permutação, quando j varia de 1 até n a variável ℓ também varia no mesmo conjunto. Assim, substituindo $\ell = \sigma(j)$ e $j = \pi(\ell)$ no lado esquerdo da equação, obtemos o resultado desejado. □

Definição 4.5 (Ordenação de Vetores) A função que *ordena* o vetor de vetores $\underline{\boldsymbol{x}}$ *em relação* ao vetor de vetores $\underline{\boldsymbol{y}}_1$ é dada por

$$\mathrm{ord}(\underline{\boldsymbol{x}}|\underline{\boldsymbol{y}}_1) = \underline{\boldsymbol{x}}^{\pi} = (\boldsymbol{x}_{\pi(1)}, \ldots, \boldsymbol{x}_{\pi(n)}) \tag{4-16}$$

[3] A função que minimiza (4-14) não é unica. Como cada $\boldsymbol{u}_k$ se repete n_k vezes em $\underline{\boldsymbol{y}}_1$, há exatamente $\prod_{k=1}^{K} n_k$ funções de permutação que produzem a mesma palavra de reprodução $\underline{\boldsymbol{y}}_i$ a partir de $\underline{\boldsymbol{y}}_1$. Isto, no entanto, não impõe nenhuma dificuldade no método, uma vez que o índice i de $\underline{\boldsymbol{y}}_i$ pode ser facilmente encontrado através da técnica de enumeração de permutações descrita em [2].

onde

$$\pi = \arg\min_{\pi' \in \Pi_n} \frac{1}{n} \sum_{j=1}^{n} d(\boldsymbol{x}_{\pi'(j)}, \boldsymbol{y}_{1j}). \tag{4-17}$$

Note que, ao contrário do caso escalar, a ordenação de vetores não é "universal", isto é, tal ordenação é sempre relativa a um vetor de referência.

Usando (4-15) e a função de ordenação de vetores, obtemos que a distorção média associada ao código de permutação vetorial pode ser escrita como

$$D = E[d(\underline{\boldsymbol{X}}, \underline{\hat{\boldsymbol{X}}})] = E[\frac{1}{n} \sum_{j=1}^{n} d(\boldsymbol{X}_j, \hat{\boldsymbol{X}}_j)] \tag{4-18}$$

$$= E[d(\mathrm{ord}(\underline{\boldsymbol{X}}|\underline{\boldsymbol{y}}_1), \underline{\boldsymbol{y}}_1)] \tag{4-19}$$

e se a medida de distorção for de erro quadrático, teremos

$$D = \frac{1}{nL} E[\, \| \,\mathrm{ord}(\underline{\boldsymbol{X}}|\underline{\boldsymbol{y}}_1) - \underline{\boldsymbol{y}}_1 \|^2 \,]. \tag{4-20}$$

Algoritmo CSA

Se considerarmos que $\delta_{j,\ell} = d(\boldsymbol{x}_j, \boldsymbol{y}_{1\ell})$ corresponde ao "custo" de se atribuir o vetor $\boldsymbol{x}_j$ ao vetor $\boldsymbol{y}_{1\ell}$, então (4-14) se torna

$$\sigma = \arg\min_{\sigma' \in \Pi_n} \frac{1}{n} \sum_{j=1}^{n} \delta_{j,\sigma'(j)} \tag{4-21}$$

isto é, a tarefa do codificador ótimo é encontrar a função de permutação σ tal que o custo total $\sum_{j=1}^{n} \delta_{j,\sigma(j)}$ seja minimizado. Colocado nesse formato, o problema da codificação ótima de VPC se torna uma questão conhecida na literatura matemática como um *problema de atribuição* (*assignment problem*). Vários algoritmos existem para resolver esse problema, sendo que o mais eficiente até o momento é o conhecido como algoritmo CSA [16]. Este algoritmo tem complexidade $O(n^2\sqrt{n}\log n)$ e foi o que utilizamos para obter os resultados apresentados na Seção 4.5.

4.3 Equivalência entre VPC e ECVQ

Nesta seção, estabelecemos a equivalência assintótica entre o código de permutação vetorial e o quantizador vetorial com restrição de entropia.

Considere um ECVQ L-dimensional com um dicionário de tamanho K dado por $\{\boldsymbol{\mu}_k\}_{k=1}^K$ e regra de codificação tal que cada vetor de reprodução $\boldsymbol{\mu}_k$ seja produzido com probabilidade p_k. O teorema seguinte mostra que é possível construir um VPC de comprimento n tal que, quando $n \to \infty$, o desempenho do VPC equivale ao do ECVQ.

Teorema 4.1 Seja $\{\boldsymbol{X}_j\}_{j=1}^{\infty}$ uma seqüência de vetores aleatórios L-dimensionais i.i.d. e seja $\boldsymbol{X}$ um vetor genérico desse conjunto. Assuma que a medida de distorção é uma métrica. Dado qualquer ECVQ $(\{\boldsymbol{\mu}_k\}_{k=1}^K, \{p_k\}_{k=1}^K)$ tal que ambos K e $\max_{k,\ell} d(\boldsymbol{\mu}_k, \boldsymbol{\mu}_\ell)$ são finitos, se o ECVQ codifica $\boldsymbol{X}$ com taxa R e distorção D, então existe uma seqüência $\{\mathcal{C}_n\}$ de VPC's L-dimensionais de comprimento n, $n = 1, 2, \ldots$, que codificam $\boldsymbol{X}^n = (\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n)$ com correspondentes taxas R_n e distorções D_n que satisfazem ambos $\lim_{n\to\infty} R_n = R$ e $\lim_{n\to\infty} D_n = D$.

Prova. Para todo n, faz-se o alfabeto do VPC igual ao dicionário $\{\boldsymbol{\mu}_k\}_{k=1}^K$ do ECVQ, e escolhe-se cada elemento n_k do vetor de composição $\{n_k\}_{k=1}^K$ arredondando np_k de forma que

$$|np_k - n_k| \leq 1 \tag{4-22}$$

e $\sum_{k=1}^K n_k = n$. Conseqüentemente, $\lim_{n\to\infty} \frac{n_k}{n} = p_k$. Usando a fórmula de Stirling, é fácil mostrar que

$$\lim_{n\to\infty} R_n = -\frac{1}{L} \sum_{i=1}^{K} p_i \log p_i = R. \tag{4-23}$$

Seja $\boldsymbol{Y}^n$ a seqüência de palavras do ECVQ produzida ao se codificar cada componente de $\boldsymbol{X}^n$ com o ECVQ. Note que podemos considerar que a saída do ECVQ é uma nova fonte discreta, sem memória, com alfabeto $\mathcal{Y} = \{\boldsymbol{\mu}_k\}_{k=1}^K$ e distribuição $\{p_k\}_{k=1}^K$. Assim, $\boldsymbol{Y}^n$ é uma seqüência de n símbolos i.i.d obtidos a partir dessa fonte. Seja $N(\boldsymbol{\mu}_k|\boldsymbol{Y}^n)$ o número de ocorrências de $\boldsymbol{\mu}_k$ em $\boldsymbol{Y}^n$, e defina o conjunto de *seqüências fortemente típicas* [11, p.288]

$$\mathcal{T}_\delta^{(n)} = \{\boldsymbol{Y}^n \in \mathcal{Y}^n : |N(\boldsymbol{\mu}_k|\boldsymbol{Y}^n) - p_k n| < n\delta, \;\; k = 1, \ldots, K\}. \tag{4-24}$$

Combinando (4-22) e (4-24), temos

$$\mathcal{T}_\delta^{(n)} = \{\boldsymbol{Y}^n \in \mathcal{Y}^n : |N(\boldsymbol{\mu}_k|\boldsymbol{Y}^n) - n_k| < n\delta + 1, \;\; k = 1, \ldots, K\}. \tag{4-25}$$

Note que se $\boldsymbol{Y}^n \in \mathcal{T}_\delta^{(n)}$, então $\boldsymbol{Y}^n$ deve diferir de alguma palavra do VPC em no máximo $r \leq K(n\delta + 1)$ símbolos.

Agora considere o seguinte método de codificação para o VPC: se, para algum k, ocorrer $N(\boldsymbol{\mu}_k|\boldsymbol{Y}^n) > n_k$, então uma nova seqüência é gerada a partir de $\boldsymbol{Y}^n$ substituindo-se arbitrariamente um componente $\boldsymbol{\mu}_k$ por algum $\boldsymbol{\mu}_\ell$ para o qual $N(\boldsymbol{\mu}_\ell|\boldsymbol{Y}^n) < n_\ell$. Aplicando este procedimento repetidamente à nova seqüência até que nenhuma substituição adicional seja requerida, a seqüência resultante $\boldsymbol{Z}^n$, por definição, será uma palavra do VPC.

Em seguida, obtemos a distorção associada a esse esquema de compressão, $D_n = E\left[d(\boldsymbol{X}^n, \boldsymbol{Z}^n)\right]$. Como $d(\cdot,\cdot)$ satisfaz a desigualdade triangular, temos

$$\begin{aligned} D_n &= E\left[d(\boldsymbol{X}^n, \boldsymbol{Z}^n)\right] \\ &\leq E\left[d(\boldsymbol{X}^n, \boldsymbol{Y}^n)\right] + E\left[d(\boldsymbol{Y}^n, \boldsymbol{Z}^n)\right] \\ &= D + D_n^* \end{aligned} \tag{4-26}$$

onde D_n^* é a parcela adicional de distorção acarretada por se codificar com o VPC ao invés do ECVQ. Para avaliar D_n^*, obtemos inicialmente uma cota em $d(\boldsymbol{Y}^n, \boldsymbol{Z}^n)$:

$$\begin{aligned} d(\boldsymbol{Y}^n, \boldsymbol{Z}^n) &= \frac{1}{n}\sum_{j=1}^{n} d(\boldsymbol{Y}_j, \boldsymbol{Z}_j) \\ &\leq \begin{cases} \frac{r}{n} d_{max} & \text{se } \boldsymbol{Y}^n \in \mathcal{T}_\delta^{(n)} \\ \frac{n}{n} d_{max} & \text{se } \boldsymbol{Y}^n \notin \mathcal{T}_\delta^{(n)} \end{cases} \end{aligned} \tag{4-27}$$

onde $d_{max} = \max_{k,\ell} d(\boldsymbol{\mu}_k, \boldsymbol{\mu}_\ell)$ é uma constante determinística. Denotando por $\mathcal{A}_n$ o evento $\{\boldsymbol{Y}^n \in \mathcal{T}_\delta^{(n)}\}$, então D_n^* pode ser escrita como

$$D_n^* = E\left[d(\boldsymbol{Y}^n, \boldsymbol{Z}^n)\middle|\mathcal{A}_n\right] P(\mathcal{A}_n) + E\left[d(\boldsymbol{Y}^n, \boldsymbol{Z}^n)\middle|\bar{\mathcal{A}}_n\right] P(\bar{\mathcal{A}}_n). \tag{4-28}$$

Substituindo (4-27) e a cota em r, obtemos

$$D_n^* \leq \frac{K(n\delta + 1)}{n} d_{max} P(\mathcal{A}_n) + d_{max} P(\bar{\mathcal{A}}_n). \tag{4-29}$$

Pela lei dos grandes números, $P(\mathcal{A}_n) \to 1$ e $P(\bar{\mathcal{A}}_n) \to 0$ quando $n \to \infty$. Assim,

$$\lim_{n\to\infty} D_n^* \leq K\delta d_{max} = \epsilon \tag{4-30}$$

e portanto

$$\lim_{n\to\infty} D_n - D \leq \lim_{n\to\infty} D_n^* \leq \epsilon. \tag{4-31}$$

Como δ e conseqüentemente ϵ podem ser escolhidos tão pequenos quanto se queira, temos, pela definição de limite de seqüência,

$$\lim_{n\to\infty} D_n = D \tag{4-32}$$

o que completa a prova. □

Este teorema mostra que para qualquer ECVQ existe um VPC de comprimento infinito com desempenho idêntico. Note que o teorema é provado apenas para VPC's com codificação sub-ótima, e portanto não assegura o resultado recíproco de que a todo VPC de comprimento infinito corresponda um ECVQ com o mesmo desempenho. Acreditamos, porém, que esta relação seja válida, isto é, que quando o comprimento do VPC é infinitamente grande, os dois quantizadores VPC e ECVQ se tornam de fato equivalentes.

Vale a pena ressaltar que, apesar da aparente semelhança, a prova do Teorema 4.1 usa técnicas diferentes daquelas usadas na prova do Teorema 3.1. Nesta última, em particular, são usadas propriedades específicas do caso escalar que não se aplicam ao caso vetorial (como a ordenação "universal" de escalares, por exemplo), impossibilitando uma extensão direta.

4.4 Projeto de VPC

Nesta seção, procuramos projetar códigos de permutação vetorial com o objetivo de obter um bom desempenho no sentido taxa-distorção. Dados a dimensão dos símbolos do alfabeto L, o comprimento n e o tamanho do alfabeto K, um VPC é completamente caracterizado por seu alfabeto $\{\boldsymbol{\mu}_k\}_{k=1}^K$ e seu respectivo vetor de composição $\{n_k\}_{i=k}^K$. Assim como no caso escalar, somente a distorção é influenciada pelo alfabeto, enquanto ambas taxa e distorção são influenciadas pelo vetor de composição. Para o projeto de alfabeto, procedemos de maneira análoga ao caso escalar, encontrando o alfabeto que minimiza a distorção para um vetor de composição fixo. Tal analogia, no entanto, não se mostra útil no projeto de vetor de composição; nesse caso, uma diferente abordagem é utilizada, a qual é baseada no teorema descrito na seção anterior.

4.4.1 Projeto de alfabeto

Seja $\underline{\boldsymbol{X}}$ o vetor aleatório nL-dimensional produzido pela fonte, e defina

$$(\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_n) = \text{ord}(\underline{\boldsymbol{X}}|\underline{\boldsymbol{y}}_1) \tag{4-33}$$

isto é, $\underline{\boldsymbol{\xi}}$ corresponde ao vetor $\underline{\boldsymbol{X}}$ ordenado em relação a $\underline{\boldsymbol{y}}_1$. Usando a notação

$$S_0 = 0, \quad S_k = n_1 + n_2 + \cdots + n_k, \qquad k = 1, \cdots, K \tag{4-34}$$

e considerando a medida de distorção de erro quadrático, podemos dizer que a distorção associada a um código de permutação vetorial com codificação ótima é dada por[4]

$$D = \frac{1}{nL} E[\sum_{j=1}^{n} \|\boldsymbol{\xi}_j - \boldsymbol{y}_{1j}\|^2] \tag{4-35}$$

$$= \frac{1}{nL} E[\sum_{k=1}^{K} \sum_{j=S_{k-1}+1}^{S_k} \|\boldsymbol{\xi}_j - \boldsymbol{\mu}_k\|^2] \tag{4-36}$$

$$= \frac{1}{nL} E[\sum_{k=1}^{K} \sum_{j=S_{k-1}+1}^{S_k} (\boldsymbol{\xi}_j - \boldsymbol{\mu}_k)^T (\boldsymbol{\xi}_j - \boldsymbol{\mu}_k)]. \tag{4-37}$$

Desenvolvendo a expressão quadrática e notando que

$$\sum_{j=1}^{n} \|\boldsymbol{\xi}_j\|^2 = \sum_{j=1}^{n} \|\boldsymbol{X}_j\|^2 = \|\underline{\boldsymbol{X}}\|^2 \tag{4-38}$$

podemos reescrever (4-37) na forma

$$nD = E[\|\underline{\boldsymbol{X}}\|^2] - 2\sum_{k=1}^{K} \boldsymbol{\mu}_k^T \sum_{j=S_{k-1}+1}^{S_k} E[\boldsymbol{\xi}_j] + \sum_{k=1}^{K} n_k \boldsymbol{\mu}_k^T \boldsymbol{\mu}_k. \tag{4-39}$$

Para obter o vetor $\boldsymbol{\mu}_k$ que minimiza D, fazemos uso do cálculo diferencial matricial (Apêndice C). Fazendo $\partial D/\partial \boldsymbol{\mu}_k = \mathbf{0}$, obtemos

$$-2\frac{\partial}{\partial \boldsymbol{\mu}_k} \left(\boldsymbol{\mu}_k^T \sum_{j=S_{k-1}+1}^{S_k} E[\boldsymbol{\xi}_j] \right) + n_k \frac{\partial}{\partial \boldsymbol{\mu}_k} \left(\boldsymbol{\mu}_k^T \boldsymbol{\mu}_k \right) = 0 \tag{4-40}$$

$$-2 \sum_{j=S_{k-1}+1}^{S_k} E[\boldsymbol{\xi}_j] + 2n_k \boldsymbol{\mu}_k = 0 \tag{4-41}$$

[4]Observe que nesse desenvolvimento consideramos que os vetores L-dimensionais $\boldsymbol{\xi}_j$, $\boldsymbol{y}_{1j}$ e $\boldsymbol{\mu}_k$ são vetores-coluna.

e portanto

$$\boldsymbol{\mu}_k = \frac{1}{n_k} \sum_{j=S_{k-1}+1}^{S_k} E[\boldsymbol{\xi}_j], \qquad k = 1, \ldots, K. \tag{4-42}$$

Quando se utiliza os valores ótimos para $\boldsymbol{\mu}_k$ dados acima, a distorção resultante é

$$D = \frac{1}{nL} \left(E[\|\underline{\boldsymbol{X}}\|^2] - \sum_{k=1}^{K} n_k \|\boldsymbol{\mu}_k\|^2 \right). \tag{4-43}$$

É importante notar que os vetores $\boldsymbol{\xi}_j$ em (4-42) dependem de $\underline{\boldsymbol{y}}_1$ através de (4-33), que por sua vez depende de $\{\boldsymbol{\mu}_k\}$ através de (4-9). Logo, ao contrário do caso escalar, (4-42) não é uma condição suficiente para otimalidade do alfabeto, mas apenas necessária. Em todo caso, podemos proceder como no projeto de quantizadores e aplicar (4-33), (4-42) e (4-9) iterativamente para atualizar o alfabeto. A cada iteração, $\{\boldsymbol{\mu}_k\}$ é a melhor escolha possível do alfabeto de *reprodução* dado o alfabeto de *codificação* usado na iteração anterior. Uma vez que $\{\boldsymbol{\mu}_k\}$ só têm impacto na distorção, esta vai sendo reduzida a cada iteração, e portanto em algum ponto deverá atingir um ótimo, que pode ser local ou global. Este processo de atualização do alfabeto está formalizado no algoritmo descrito a seguir [15].

Algoritmo 4.1 (Projeto de Alfabeto de VPC)

1) Especifique o vetor de composição $\{n_i\}_{i=1}^K$ e o alfabeto inicial $\{\boldsymbol{\mu}_i\}_{i=1}^K$.
2) Faça $S_i = \sum_{j=1}^{i} n_j$.
3) Calcule:
   a) $\underline{\boldsymbol{y}}_1 = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \ldots, \boldsymbol{\mu}_K)$
   b) $\underline{\boldsymbol{z}} = E[\,\mathrm{ord}(\underline{\boldsymbol{X}}|\underline{\boldsymbol{y}}_1)\,]$
   c) $\boldsymbol{\mu}_k = \frac{1}{n_k} \sum_{j=1+S_{k-1}}^{S_k} \boldsymbol{z}_j, \quad k = 1 \ldots K.$
4) Repita o passo 3 até que $\{\boldsymbol{\mu}_k\}$ convirja.
5) Fim. ∎

Note que, usando o desenvolvimento do Apêndice B para o cálculo de $\underline{\boldsymbol{z}}$, a cada iteração deste algoritmo a operação de codificação é realizada T vezes, onde T é o tamanho da seqüência de treinamento. Se, além disso, utilizarmos a operação de codificação ótima (como implementada pelo algoritmo CSA) então cada iteração do projeto de alfabeto se torna uma tarefa bastante complexa. Portanto, torna-se proibitivo realizar, a cada projeto, um número não especificado de iterações até a parada do algoritmo (passo 4). Optamos por utilizar um número fixo (e pequeno) de iterações,

por exemplo, $J = 3$. Este pequeno número de iterações mostra-se geralmente suficiente para conduzir a distorção a valores próximos do ótimo — o que ocorre provavelmente devido à natureza altamente estruturada dos códigos de permutação.

4.4.2 Projeto de vetor de composição

Uma vez que a distorção de um VPC com alfabeto ótimo é dada por (4-43), vemos que o desenvolvimento apresentado na Seção 3.3.2 para SPC pode ser imediatamente estendido para o caso vetorial, bastando apenas que μ_k^2 seja substituído por $\|\boldsymbol{\mu}_k\|^2$ em (3-22). Isto é, se $p_k = n_k/n$, $k = 1, \ldots, K$, então uma condição necessária para otimalidade de vetor de composição é

$$n_k/n = p_k = \frac{2^{-\frac{1}{\lambda}\|\boldsymbol{\mu}_k\|^2}}{\sum_k 2^{-\frac{1}{\lambda}\|\boldsymbol{\mu}_k\|^2}} \qquad k = 1, \ldots, K. \tag{4-44}$$

No entanto, algumas dificuldades surgem quando se tenta utilizar esta condição em conjunto com o projeto de alfabeto para formar um algoritmo de projeto completo semelhante ao Algoritmo 3.2 para SPC. Em primeiro lugar, o Algoritmo 4.1 tem de ser aplicado diversas vezes alternadamente com (4-44) até a convergência de $\{n_k\}$, o que pode tornar a complexidade proibitiva devido aos mesmos motivos discutidos na seção anterior. Além disso, o fato de que o projeto de alfabeto encontra um $\{\boldsymbol{\mu}_k\}$ que não necessariamente é um ótimo global, aliado à intrínseca não-otimalidade de (4-44), faz com que um algoritmo combinado tenha dificuldades em encontrar códigos com um desempenho realmente bom. De fato, esta foi a situação observada em todos os experimentos realizados com este algoritmo tentativo.

As dificuldades descritas acima exigem que outra abordagem seja utilizada para o projeto de VPC's. Uma tal possibilidade é implicitamente sugerida pelo Teorema 4.1: podemos explorar a eqüivalência assintótica entre VPC e ECVQ para construir VPC's com desempenhos arbitrariamente próximos ao desempenho do ECVQ.

Procede-se da seguinte forma. Inicialmente, projeta-se um ECVQ para algum parâmetro λ de interesse. Em seguida, faz-se o alfabeto do VPC igual ao dicionário do ECVQ, e calcula-se o vetor de composição do primeiro a partir das probabilidades das palavras de reprodução do segundo, exatamente como descrito na prova do Teorema 4.1. Se o ECVQ original tem bom desempenho, então o VPC de comprimento n resultante desse proce-

dimento também terá bom desempenho, desde que n seja suficientemente grande. Note que, caso o desempenho obtido seja insatisfatório, não é necessário realizar um novo projeto: basta aumentar o valor de n e calcular correspondentemente o novo vetor de composição. O desempenho pode ser melhorado ainda mais aplicando-se o Algoritmo 4.1 para o refinamento do alfabeto obtido.

O procedimento descrito acima para o projeto de VPC baseado em um ECVQ é sumarizado no algoritmo a seguir [33].

Algoritmo 4.2 (Projeto de VPC)

1) Especifique n, L, K, e o parâmetro λ.
2) Projete um ECVQ L-dimensional com K vetores usando λ como parâmetro lagrangiano, obtendo assim um dicionário $\{\boldsymbol{\mu}_k\}_{k=1}^{K}$ e uma correspondente distribuição $\{p_k\}_{k=1}^{K}$.
3) Faça $n_k \approx p_k n$, $k = 1 \ldots K$.
4) Faça o alfabeto inicial do VPC igual a $\{\boldsymbol{\mu}_k\}_{k=1}^{K}$.
5) Atualize este alfabeto com o Algoritmo 4.1.
6) Fim. ■

A complexidade deste algoritmo é dominada pela complexidade do projeto de alfabeto. Note, porém, que o passo 5 é opcional, e pode ser omitido caso esta complexidade não seja tolerável. Isto resulta em um algoritmo modificado de complexidade idêntica à do ECVQ, e portanto independente de n.

4.5 Desempenho

Esta seção apresenta nossos resultados experimentais obtidos com as técnicas descritas neste capítulo [15, 33]. Examinamos o desempenho de VPC's para as fontes uniforme e gaussiana, e em ambos os casos foi considerada a medida de distorção de erro quadrático. Utilizamos o Algoritmo 4.2 para o projeto de VPC's, e o Algoritmo 4.1 para atualização do alfabeto. Para a codificação requerida no projeto de alfabeto e para a avaliação do desempenho, utilizamos a operação de codificação ótima, a qual foi implementada através do algoritmo CSA. De acordo com o desenvolvimento do Apêndice B, seqüências de treinamento e de teste foram

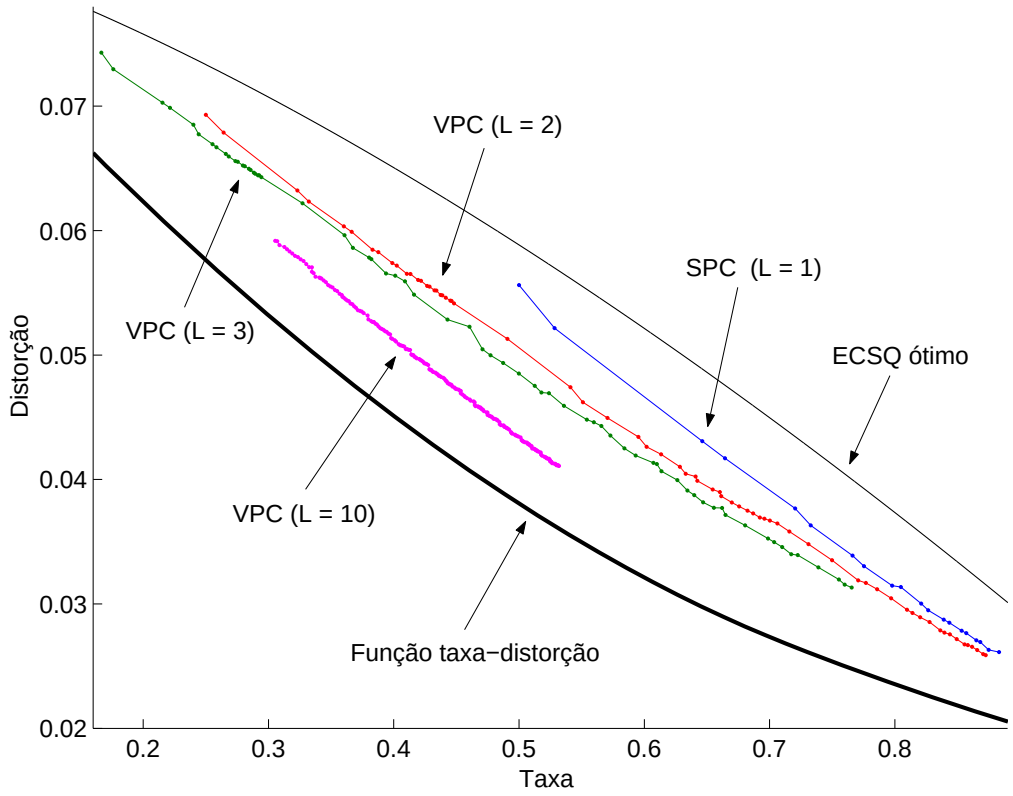

Figura 4.1: Comparação entre VPC's com $L = 1$, 2, 3 e 10 e o ECSQ ótimo para uma fonte uniforme.

L	K	n
1	2	$2, \ldots, 22$
2	$2, 3, 4$	$K, \ldots, K + 25$
3	$2, \ldots, 8$	$K, \ldots, K + 20$
10	$20, 25, \ldots, 100$	$K, \ldots, K + 17$

Tabela 4.1: Parâmetros dos VPC's projetados para uma fonte uniforme.

utilizadas, respectivamente, para a estimativa de valores esperados requerida pelas etapas de projeto e de cálculo da distorção.

4.5.1 Fonte uniforme

A Fig. 4.1 mostra uma comparação entre o desempenho de VPC's com várias dimensões de alfabeto L, o ECVQ ótimo e a função taxa-distorção, para uma fonte uniforme. Para cada $L = 1$, 2, 3 e 10, diferentes parâmetros foram usados no projeto de VPC's, como mostra a Tab. 4.1. Para cada par (L, K) foi projetado um ECVQ com $\lambda = 0$ (VQ de mínima distorção), a partir do qual VPC's com diferentes valores de n foram obtidos; estes VPC's foram em seguida submetidos à atualização do alfabeto, a qual se

mostrou relevante para o desempenho. Para cada curva correspondendo a um determinado L, foram escolhidos os VPC's com melhor desempenho dentre todos códigos com parâmetros K e n avaliados, isto é, VPC's com desempenho superável por outro VPC de mesmo L foram descartados. Utilizamos seqüências de treinamento e de teste ambas com 50000 amostras L-dimensionais em todas as simulações. Como pode ser visto na Fig. 4.1, o desempenho do VPC melhora com o aumento da dimensão de alfabeto L. Vale a pena mencionar que, nos exemplos avaliados para fonte uniforme, um aumento de n para L e K fixos tende a produzir um código com maior taxa e menor distorção, mas não necessariamente implica uma melhoria de desempenho. Este comportamento difere do que se observa para fonte gaussiana, como será visto mais adiante.

4.5.2 Fonte gaussiana

As figuras 4.2, 4.3 e 4.4 mostram comparações análogas para uma fonte gaussiana. Aqui nós projetamos ECVQ's para diversos valores de λ de forma a obter pontos para todas as taxas de interesse. À medida que λ era acrescido, o parâmetro K era eventualmente diminuído a partir de um valor inicial $K_{\max}$. Todos os ECVQ's obtidos originaram VPC's através do arredondamento do vetor de composição. Percebemos que, nesse caso, o valor exato de n não era significativo para o desempenho; por esse motivo, os valores de n_k foram obtidos a partir do arredondamento de $p_k\tilde{n}$ para o inteiro mais próximo, onde $\tilde{n}$ é o valor aproximado que se deseja ter para n. Percebemos também que os VPC's obtidos a partir do ECVQ não apresentavam melhorias de desempenho significativas quando a atualização de alfabeto era realizada, e portanto esta etapa foi omitida do projeto. Para treinamento e teste de desempenho, utilizamos seqüências formadas por 100000 amostras L-dimensionais cada uma.

A Fig. 4.2 mostra uma comparação entre os desempenhos de VPC's e ECVQ's de dimensão $L = 2$, com parâmetros $K_{\max} = 35$ e $n = 500$. O desempenho teórico do ECSQ ótimo e a curva da função taxa-distorção também são apresentados na figura. A curva de desempenho do SPC não é mostrada, porém, como visto no Capítulo 3 e relatado em [2], esta curva situa-se acima daquela correspondente ao ECSQ. Na Fig. 4.4, é mostrada uma comparação análoga para $L = 3$, $K_{\max} = 87$ e $n = 500$. Como pode ser visto na Fig. 4.3, o desempenho do VPC para fonte gaussiana melhora à medida em que n é aumentado.

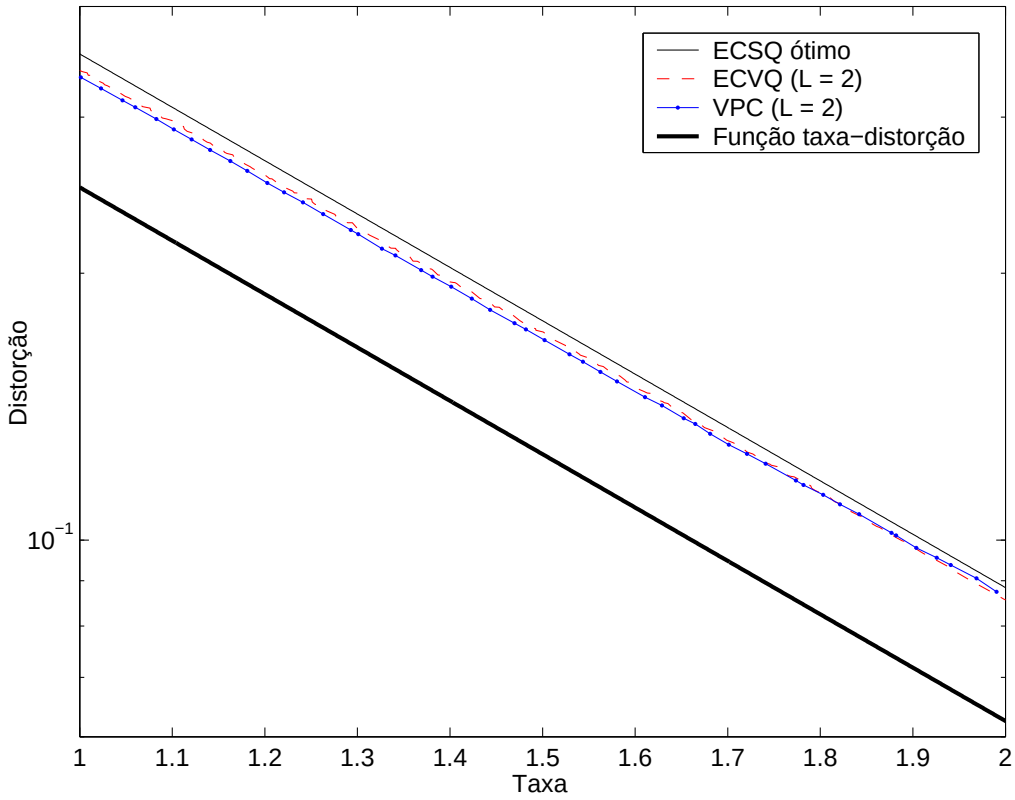

Figura 4.2: Comparação entre VPC's com $L = 2$ e o ECVQ para uma fonte gaussiana. O desempenho do ECSQ ótimo é também mostrado.

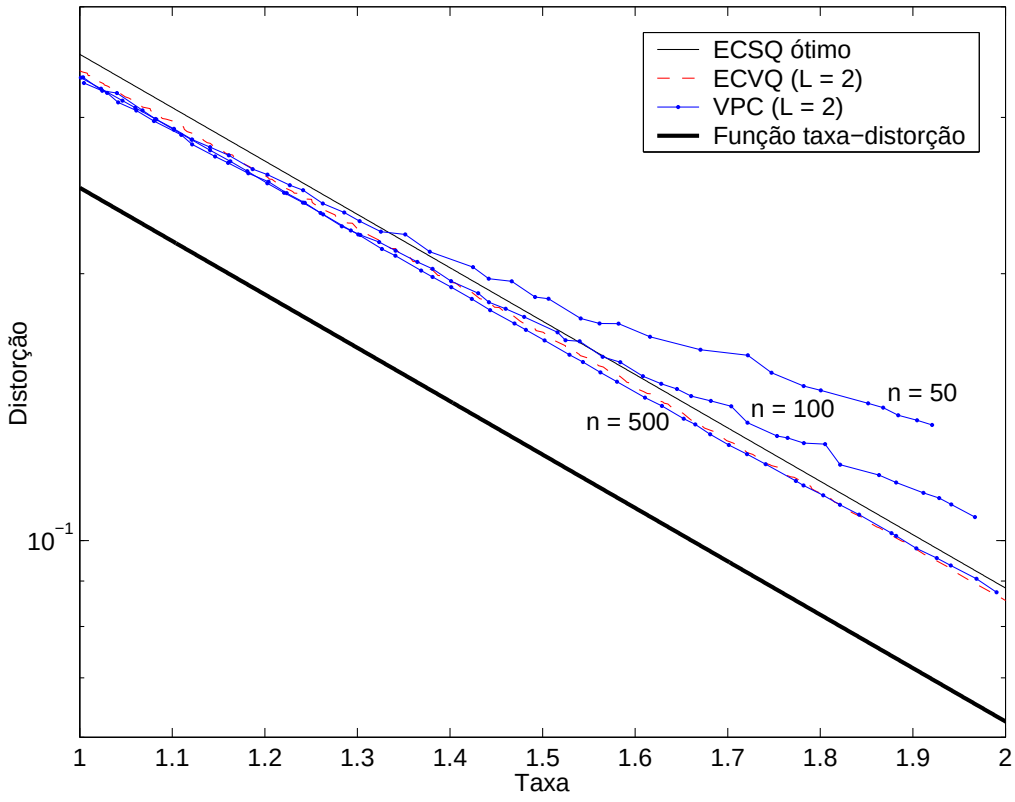

Figura 4.3: Desempenho de VPC's com $L = 2$ em função do comprimento n.

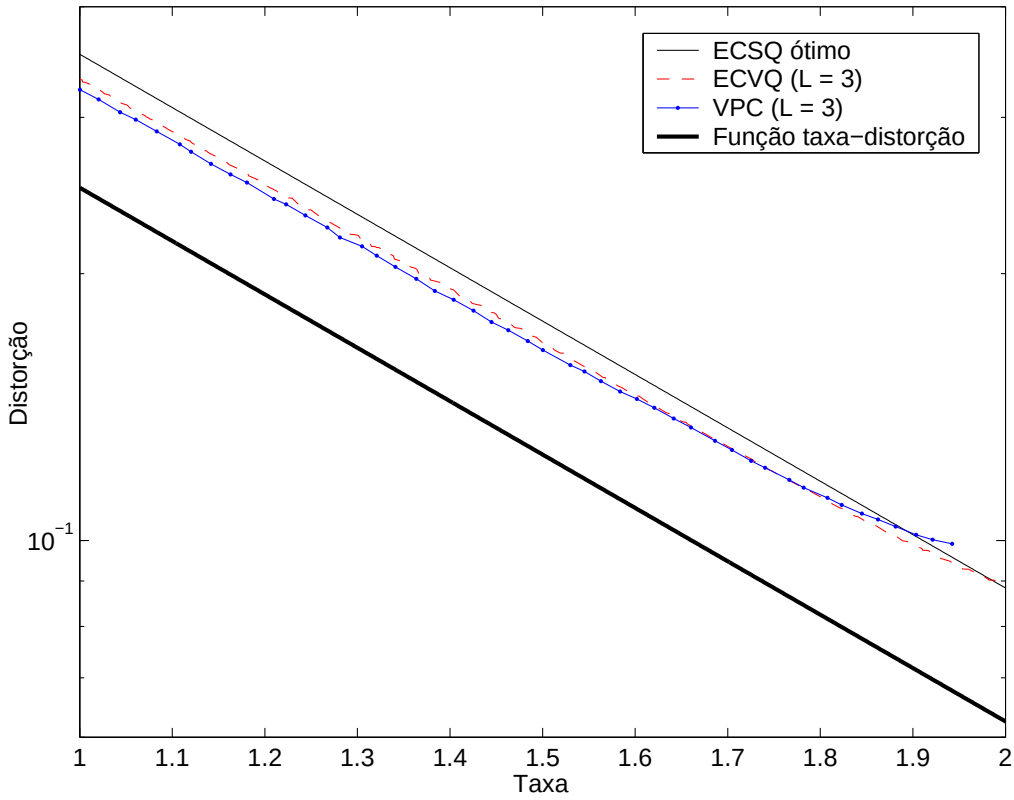

Figura 4.4: Comparação entre VPC's com $L = 3$ e o ECVQ para uma fonte gaussiana. O desempenho do ECSQ ótimo é também mostrado.

Observa-se das figuras citadas que foram obtidos VPC's com desempenho superior ao de ECVQ's de mesma dimensão[5]. Neste ponto, vale a pena ressaltar que o algoritmo de Chou para projeto de ECVQ's (Algoritmo 2.2) não fornece necessariamente os quantizadores ótimos [22]. É possível que para qualquer VPC de qualquer comprimento exista um ECVQ correspondente com desempenho superior. Assim, permanece ainda um problema em aberto se o desempenho do VPC ótimo pode superar aquele do ECVQ ótimo ou não.

4.6 Comentários

Como apontado em [2], é conhecido que o desempenho do código de permutação escalar é tipicamente pobre quando as amostras da fonte são fortemente dependentes entre si. Carregando essa analogia para o caso vetorial, podemos afirmar que o VPC terá melhor desempenho quando aplicado a uma fonte cujos símbolos vetoriais são independentes e identicamente dis-

[5]É importante ressaltar que a comparação considera ECVQ's de dimensão igual à dimensão dos símbolos dos VPC's.

tribuídos (*cf.* Teorema 4.1). No entanto, nenhuma etapa do desenvolvimento requer ou sugere que os componentes do vetor L-dimensional da fonte $\boldsymbol{X}$ sejam independentes ou identicamente distribuídos. Isto significa que, assim como o ECVQ, o VPC pode ter bom desempenho quando aplicado a uma fonte com componentes escalares correlatados, desde que a extensão dessa correlação esteja aproximadamente limitada aos L componentes de cada vetor $\boldsymbol{X}$. Além disso, é ainda permitido que tais componentes tenham distribuições distintas — note que no projeto de VPC ocorre a permutação dos vetores da fonte, mas nunca dos componentes escalares de um vetor. O alfabeto projetado para esta fonte terá vetores $\boldsymbol{u}_k$ com diferentes distribuições marginais em cada dimensão. Esta propriedade do VPC — isto é, sua forte semelhança com o ECVQ — pode permitir sua utilização também em situações onde o desempenho do SPC é insatisfatório.

No que diz respeito à implementação em computador, as rotinas que subsidiam os resultados encontrados neste trabalho foram escritas parcialmente nas linguagens MATLAB e C/C++. O ambiente de trabalho MATLAB foi escolhido por sua simplicidade e flexibilidade, enquanto as rotinas de carga computacional mais elevada foram transcritas para linguagem C e adaptadas para a interface com o MATLAB, com o intuito de garantir uma maior eficiência. A rotina que implementa o algoritmo CSA é uma versão da rotina vencedora do torneio DIMACS de algoritmos na categoria *matching*. Esta rotina, fornecida pelo primeiro autor de [16], havia sido adaptada para o problema da codificação ótima de VPC no trabalho [8], e foi posteriormente otimizada neste trabalho. O sistema completo conta com rotinas que implementam o projeto e teste de desempenho de VPC's e ECVQ's, tanto para parâmetros individuais quanto para famílias de parâmetros como as listadas na Tab. 4.1, além da rotina para o cálculo da função $R(D)$ através do algoritmo de Blahut, e outras.