Determinação de Códigos Convolucionais de Memória Unitária Ótimos: um Problema de Otimização via Seleção e Alocação de Recursos

Amir Said e Reginaldo Palazzo Jr.

Neste trabalho é apresentado um procedimento via programação matemática, em direção a uma sistematização do problema de determinação de códigos convolucionais ótimos, especificamente aqueles pertencentes à classe de memória unitária. Este procedimento utiliza técnicas típicas de equivalência e redução de problemas, comumente empregadas na análise de complexidade e intratabilidade, e culmina com o estabelecimento de dois problemas de otimização, um de seleção e outro de alocação de recursos. A solução destes problemas consiste na determinação dos tipos de colunas das submatrizes geradoras bem como na sua ordenção correta. Por fim, são apresentados novos códigos convolucionais de memória unitária, obtidos através deste novo procedimento.

1. Introdução

O problema de determinação de códigos convolucionais ótimos a partir de propriedades algébricas ou modelos matemáticos advém desde a proposta, feita por Elias [**1**] em 1955, desta classe de códigos. Um passo importante nesta direção foi dado por Viterbi [**2**] em 1967, quando do estabelecimento do algoritmo que hoje leva o seu nome que, posteriormente, provou-se ser equivalente à aplicação da programação dinâmica ao problema de decodificação por máxima verossimilhança. No ano seguinte, Odenwalder [**3**] foi o primeiro que efetivamente apresentou códigos convolucionais segundo o algoritmo apresentado em [**2**]. Desde então, muitos pesquisadores têm apresentado novos códigos convolucionais através de algoritmos que são aprimoramentos da adaptação do algoritmo de Viterbi à seleção do código ótimo, via busca exaustiva das matrizes geradoras. Dentro desta categoria encontramos os algoritmos propostos por Larsen [**4**], Paaske [**5**], Johannesson e Paaske [**6**], Massey e Costello [**7**] e Daut et alli [**8**].

Os autores são professores da Faculdade de Engenharia Elétrica da UNICAMP, Caixa Postal 6101, 13081, Campinas, SP.

Um novo avanço foi dado por Lee [**9**] em 1976, quando da proposta de códigos convolucionais de memória unitária. Embora não apresentando uma estrutura algébrica equivalente aos códigos de bloco, a conexão foi feita considerando-se a truncagem do código convolucional com relação ao primeiro comprimento de restrição, dando origem a um código de bloco equivalente. Mostrou-se que todos os códigos convolucionais podem ser enquadrados na classe dos códigos convolucionais de memória unitária, o que garante que os códigos ótimos encontrados nesta categoria, em relação à capacidade de correção de erros, não são inferiores aos pertencentes a outras categorias. Além disto, Lee mostrou que o uso de memória unitária (que, de certo modo, é equivalente a se operar em bytes ao invés de bits) permite melhorar sensivelmente a capacidade de correção dos códigos convolucionais, sem aumentar a complexidade na decodificação pelo algoritmo de Viterbi.

Entretanto, em 1983, uma proposta de equivalência do problema de determinação de códigos convolucionais com o problema de máximo fluxo em redes, apresentada em [**10**], possibilitou a introdução de novas técnicas com estrutura matemática objetivando a sistematização da resolução do problema em questão. Como resultado, uma quantidade considerável de novos códigos convolucionais foi apresentada. No entanto, tais coçdigos estão restritos a um comprimento de restrição igual a 10 para taxas 1/n e até 5 para taxa b/n ($2 \leq b \leq 5$).

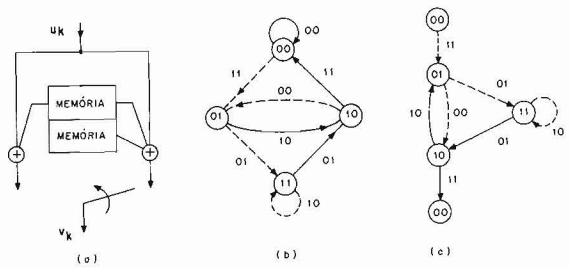
A principal contribuição deste trabalho é a apresentação de uma formulação matemática do problema de determinação de códigos convolucionais ótimos, até então inexistente. Desta forma procuramos estender de forma consistente as idéias e resultados apresentados em [**9**] e [**10**], adaptados para códigos convolucionais de memória unitária. É definido um novo procedimento, idealizado a partir de processos de equivalência e redução do problema, que conduz à sua decomposição em dois subproblemas de otimização, a saber, um de seleção e um de alocação de recursos. Como resultado, temos a capacidade de obter novos códigos convolucionais binários de memória unitária ótimos.

Na Seção 2, os processos de equivalência e redução são apresentados. Na Seção 3, demonstra-se como o problema original pode ser decomposto e é descrita a correspondente formulação matemática. Na Seção 4, mostra-se a resolução do primeiro subproblema (seleção) e, na Seção 5, a resolução do segundo subproblema (alocação). Os resultados computacionais correspondentes são apresentados na Seção 6.

2. Equivalência e Redução

Em geral, a solução de problemas complexos passa por uma caracterização de equivalência, seguida por um processo de redução. Este procedimento tem por objetivo buscar uma solução do problema original via equivalência, uma vez que o problema reduzido dispõe de uma estrutura matemática bem definida. Este procedimento foi utilizado em [**10**] com o objetivo de sistematizar a determinação de códigos convolucionais específicos, bem como classes de códigos convolucionais invariantes e variantes no tempo, pois a falta de uma estrutura algébrica acarreta dificuldades substanciais na resolução deste problema. Vários algoritmos foram propostos ao longo dos últimos 18 anos, objetivando fornecer códigos convolucionais ótimos segundo alguns critérios convenientes, principalmente a maximização da distância livre, que é bastante próxima da maximização da capacidade de correção. Porém, a complexidade inerente ao problema não permitiu ainda uma solução fechada.

O codificador convolucional pode ser visto como uma máquina de estados finitos. Sua representação matemática é dada pela quintupla (S, I, O, p, q), onde S é o conjunto de estados; I é o conjunto de entradas; O é o conjunto de saídas; p: IxS → S e q: IxS → O são funções que determinam a transição de estado e saída, respectivamente. Nesta seção estaremos interessados em focalizar a atenção nos processos de equivalência e redução mencionados. A equivalência do problema de determinação de um código convolucional específico com o problema de determinação do fluxo máximo em redes depende fortemente da conjetura a ser apresentada. Antes, porém, de estabelecê-la, iremos, através de um exemplo, explicitar esta equivalência. Caracterizaremos um código convolucional através da tripla (n,b,m), onde n é o comprimento da palavra código ramo, b é o comprimento dos dados de entrada e m é o número de memórias. Tal código tem taxa r = b/n. A **Fig. 1 (a)** mostra



u_k – dados de entrada ab – estado ab
v_k – saída codificada xy – bits da saída codificada

Figura 1. Representações de um codificador convolucional.

a representação esquemática de um codificador binário com parâmetros m = 2 e r = 1/2. Na mesma figura temos os correspondentes diagrama de estados (**b**) e diagrama de estados particionado (**c**).

Note que, a partir da **Fig. 1(c)**, podemos iniciar o processo de equivalência fazendo as associações entre as variáveis dos problemas equivalentes apresentadas na **Tabela 1.**

(a)	diagrama de estados particionado	⟷	rede de fluxos capacitada
(b)	peso de Hamming da palavra gerada na transição i, j (ω_{ij})	⟷	fluxo entre os nós i e j (x_{ij})
(c)	soma dos pesos das palavras entrando e saindo de cada estado	⟷	fluxo total entrando e saindo de cada nó
(d)	distância livre do código d_{free}	⟷	máximo fluxo na rede φ

Tabela 1. Equivalência de problemas.

A associação (c) nada mais é do que a constatação da propriedade de conservação de fluxo, enquanto que a associação (d) é o critério de otimalidade a ser utilizado. Como a distância livre do código é o fator considerado mais importante para minimizar a probabilidade de erro na decodificação [2], a maximização deste parâmetro é usada para definir a otimalidade do codificador.

Como o codificador convolucional é um sistema linear discreto no tempo (para simplificar, supomos também invariante no tempo), o diagrama de estados pode ser utilizado para a obtenção das equações de estado e de saída do sistema linear associado. Representando por $\mathbf{S}$ (k) o vetor de estados (nós intermediários); por $\mathbf{i}$ (k) o vetor de entrada; por $\mathbf{o}$ (k) o vetor de saída; por $\bar{\mathbf{P}}$ e $\bar{\mathbf{Q}}$ as matrizes de transição de estado; e finalmente por $\bar{\mathbf{R}}$ e $\bar{\mathbf{S}}$ as matrizes de formação da saída, resultam as equações de estado e de saída

$$\mathbf{s}(k+1) = \bar{\mathbf{P}}\,\mathbf{s}(k) + \bar{\mathbf{Q}}\,\mathbf{i}(k) \qquad (1)$$

$$\mathbf{o}(k) = \bar{\mathbf{R}}\,\mathbf{s}(k) + \bar{\mathbf{S}}\,\mathbf{i}(k) \qquad (2)$$

Para um codificador binário de memória unitária, por exemplo, temos a matriz $\bar{P}$ nula, $\bar{Q}$ igual à identidade, $\bar{S} = G_0$ e $R = G_1$, onde G_0 e G_1 são matrizes binárias e as operações em (**1**) e (**2**) são definidas em GF(2). Veremos melhor esta classe de códigos posteriormente.

A matriz $\bar{P}$ é a matriz de transição incorporando apenas os estados intermediários do diagrama de estados particionado. Chamaremos de $\tilde{A}$ a matriz de transição aumentada, que difere de $\bar{P}$ pela inclusão dos estados inicial e final e chamaremos de $\tilde{a}_{ij}$ aos elementos de $\tilde{A}$. Sejam ainda $L_i(\tilde{A})$ e $C_j(\tilde{A})$ os valores resultantes do produto dos elementos da i-ésima linha e j-ésima coluna de $\tilde{A}$, isto é

$$L_i(\tilde{A}) = \prod_{j=1}^{g} \tilde{a}_{ij}$$

e

$$C_j(\tilde{A}) = \prod_{i=1}^{g} \tilde{a}_{ij}$$

onde g é o número de estados.

A descrição da rede de fluxo pode assim ser realizada. Seja **G** = (**N**, **R**) um grafo direcionado onde **N** é um conjunto finito de elementos denominados nós ou estados, isto é, **N** = {1, 2, ..., g} e **R** um conjunto também finito de pares ordenados (i, j) denominados ramos. isto é, **R** = { (i, j) : i, j ∈ **N** }. Seja a capacidade c_{ij} do conjunto **N**, os elementos 1 e g serão designados respectivamente nós **origem** e **destino** e os demais elementos serão designados por nós **intermediários.**

Seja x_{ij} ($0 \leq x_{ij} \leq c_{ij}$) o fluxo do ramo capacitado (i, j) do grafo **G**. A descrição matemática da conservação dos fluxos numa rede é dada por

$$\sum_i x_{ij} - \sum_k x_{jk} = \begin{cases} -\varphi, \text{ se } j = 1 \\ 0, \text{ se } j \neq 1 \text{ e } j \neq g \\ \varphi, \text{ se } j = g \end{cases} \qquad (3)$$

A interpretação de (**3**) pode ser resumida conforme descrito a seguir. Se j = 1, então o primeiro somatório é zero pois não existem fluxos de ramo chegando ao nó 1 e o segundo somatório fornece a soma de todos os fluxos saindo do nó 1. Se j = g, só teremos o primeiro somatório visto que o segundo fornece a soma de todos os fluxos de ramo saindo do destino e esta soma é igual a zero.

Por outro lado, se $j \neq 1$ e $j \neq g$, então ambos os somatórios resultam no mesmo valor de fluxo total saindo ou entrando em cada nó da rede. Quando isto acontece, diz-se que a rede é limitada e balanceada.

A afirmação feita acima para os casos $j = 1$ e $j = g$, de que o somatório dos fluxos de ramo iguala ao máximo fluxo da rede, está baseada no Teorema de Máximo Fluxo – Mínimo Corte [**13**] que, por sua vez, estabelece que o fluxo máximo em uma rede é igual à soma dos fluxos nos ramos pertencentes ao mínimo conjunto de corte.

Com esses elementos em mãos podemos estabelecer a seguinte conjetura [**10**].

Conjetura

Se existir um codificador convolucional invariante no tempo não sistemático com parâmetros m e $r = b/n$ tal que

$$L_i(\tilde{\mathbf{A}}) = C_j(\tilde{\mathbf{A}}) = D^{\varphi} \quad \text{para todo } i, j \in \mathbf{N}$$

(onde D é a função de Bhattacharyya) então o código é dito ótimo.

Dessa forma, a determinação dos códigos convolucionais ótimos não sistemáticos e específicos pode ser caracterizada como um problema de fluxo máximo em redes. Uma vez que a conjetura é assumida verdadeira, o primeiro passo na solução do problema será a determinação do fluxo máximo como função dos parâmetros do código. Assim, temos o teorema que se segue [**11**].

Teorema 1

O fluxo máximo para códigos convolucionais não sistemáticos, específicos e invariantes no tempo sobre GF(2), com memória m e taxa $r = b/n$, é dado por

$$\varphi = n\,2^{b-1}$$

A redução do problema original passa a ser descrita matematicamente como uma rede onde todos os fluxos satisfazem às condições de otimalidade do problema $\mathbf{P}_1$, sendo ω_{ij} o peso de Hamming do ramo (i, j):

P_1 Maximizar $\sum_{j \epsilon \Omega} \omega_{ij}$

sujeito a

$$\sum_i \omega_{ij} - \sum_k \omega_{jk} = \begin{cases} -\varphi, \text{ se } j = 1 \\ 0, \text{ se } j \neq 1 \text{ e } j \neq g \\ \varphi, \text{ se } j = g \end{cases}$$

$$0 \leq w_{ij} \leq c_{ij}$$

onde Ω é o conjunto de nós intermediários.

Esta redução é característica de problemas combinatoriais que, sob certas condições, podem ser facilmente solucionados. No caso mais geral, este problema combinatorial ("shortest weight-constrained path") é do tipo NP-completo, sendo do tipo P (polinomial) se, ao truncarmos a treliça em k intervalos de tempo, todas as comparações entre pares de palavras código resultantes apresentarem o mesmo peso ou todos os comprimentos forem iguais. Entretanto, podemos aplicar o processo de redução através do estabelecimento de alguns parâmetros que excluirão, numa primeira aproximação, um grande número de soluções factíveis. Um desses parâmetros é a distância mínima do código. Dessa forma, temos o seguinte teorema [**11**].

Teorema 2

Para qualquer código convolucional sobre o GF(q) com memória m, taxa r = b/n e mdc (b,n) = 1, a distância mínima é limitada superiormente por:

Caso 1: b = 1 e q = 2 (conjunto de corte unitário)

$$d_{min} \leq \left[\frac{2^{p^*-1}}{2^{p^*}-1} \right] (p^* + m) \varphi$$

Caso 2: a) b > 2 e q um primo ou potência de um primo;
b) b = 1 e q um primo ímpar ou potência de um primo (conjunto de corte arbitrário)

$$d_{min} \leq \left[\frac{q^{p^*} - 1}{q^{p^*} - 1} \right] m \varphi$$

onde p^* é o inteiro positivo que minimiza o termo à direita em cada uma das inequações acima.

É interessante comentar que o Teorema 2 estabelece um conjunto de soluções bastante reduzido de $\mathbf{P}_1$. Notamos também que, em geral, a distância mínima, d_{min}, iguala-se à distância livre, d_{free}, dos códigos convolucionais ótimos. Isto é, a menor distância mínima dos códigos convolucionais ótimos é alcançada já no primeiro comprimento de restrição (m+1 janelas de tempo na treliça).

Uma vez que a equivalência e redução ficaram estabelecidas, nosso próximo objetivo é explorar o emprego das técnicas de programação matemática no encaminhamento do algoritmo a ser proposto nas seções que se seguem. Estaremos voltados para a classe dos códigos convolucionais de memória unitária (que abreviaremos para CCMU), pois a mesma apresenta a propriedade surpreendente de que, para uma determinada multiplicidade de taxa, a distância livre deste código pode ser superior à distância livre de qualquer outra classe de códigos convolucionais conhecidos [**9**] e [**12**].

Antes de passarmos efetivamente para a formulação do problema, definiremos código de memória unitária como aquele que satisfaz à seguinte relação:

$$\mathbf{y}_k = \mathbf{G}_0 \mathbf{u}_k \oplus \mathbf{G}_1 \mathbf{u}_{k-1}$$

onde $\mathbf{u}_k = (u_{k1}, u_{k2}, \ldots, u_{kb})$, $\mathbf{y}_k = (y_{k1} y_{k2}, \ldots, y_{kn})$, $k \leq 0$; com $\mathbf{u}_k = 0$ para $k < 0$, são vetores binários; $\mathbf{G}_0$ e $\mathbf{G}_1$ são as submatrizes da matriz geradora; $\mathbf{u}_k$ representa um bloco de b dígitos de informação a serem codificados no instante k; $\mathbf{y}_k$ representa a correspondente palavra código-ramo, de comprimento n, gerada no instante k e $\oplus$ representa a adição em GF(2) (binário). Nota-se que o código é chamado de memória unitária porque apenas a informação relativa a um período anterior (que pode conter vários dígitos) é armazenda. Pode-se mostrar facilmente [**9**] que um codificador com várias memórias em série pode ser transformado, através do uso das memórias em paralelo, em um codificador de memória unitária. Para isto basta caracterizar as matrizes $\mathbf{G}_0$ e $\mathbf{G}_1$, do CCMU, como sendo as matrizes com dimensão (bm) x (nm) formadas a partir das matrizes bxn do código convolucional original. A transformação inversa, isto é, do CCMU para um código convolucional com memória $m > 1$ nem sempre é possível. Em conseqüência, pode-se afirmar que os códigos de memória múltipla formam um subconjunto da classe dos códigos de memória unitária.

3. Resolução por Programação Matemática

A análise das características de um CCMU através do diagrama de transição de estado é bastante facilitada porque este diagrama contém toda a informação necessária para se avaliar o desempenho do código. Assim, o uso desta representação é interessante quando são dadas as matrizes geradoras e se deseja avaliar a capacidade de correção do código correspondente, o que ocorre quando se procura determinar códigos ótimos através de algoritmos de enumeração extensiva (implícita ou explícita) das matrizes geradoras. Contudo, estes algoritmos de enumeração extensiva têm sua aplicação limitada pelo crescimento exponencial das dimensões do problema. Uma alternativa é usar algoritmos heurísticos que procuram obter uma solução ótima, ou próxima da ótima, melhorando gradativamente uma solução disponível. Veremos agora como uma análise de outros tipos de representação dos CCMUs pode levar à idealização de métodos eficientes de busca de códigos ótimos. Inicialmente, devemos considerar como os códigos de bloco estão relacionados com os CCMUs. Como um código convolucional é linear, sua capacidade de correção depende das distâncias de Hamming entre as palavras código, que são iguais aos pesos de Hamming das palavras não nulas. Por isto, nos parágrafos seguintes usaremos igualmente os termos peso e distância de Hamming, implicitamente excluindo a palavra código nula.

Seja d^*_{free} a máxima distância livre dos códigos (n, b, 1). Para que um CCMU específico tenha distância livre igual a d^*_{free} é necessário que todas as palavras código truncadas, com comprimento 2n, 3n, ..., tenham peso de Hamming maior que ou igual a d^*_{free}. Isto significa que as matrizes geradoras $\mathbf{G}_0$ e $\mathbf{G}_1$ devem ser tais que o peso mínimo das palavras dos códigos de bloco gerados pelas matrizes apresentadas em (**4**) sejam maiores que ou iguais a d^*_{free}.

$$\left[\mathbf{G}_0 \quad \mathbf{G}_1\right], \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & 0 \\ 0 & \mathbf{G}_0 & \mathbf{G}_1 \end{bmatrix}, \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & 0 & 0 \\ 0 & \mathbf{G}_0 & \mathbf{G}_1 & 0 \\ 0 & 0 & \mathbf{G}_0 & \mathbf{G}_1 \end{bmatrix}, \ldots \qquad (4)$$

Esta condição pode ser usada para calcular limitantes superiores de d^*_{free}. Seja $\delta(n, b)$ o valor máximo do peso mínimo das palavras de um código de bloco (n,b). Podemos então limitar d^*_{free} por [**9**] e [**12**]

$$d^*_{free} \leq \delta([p+1]n, pb), \qquad p = 1, 2, \ldots \qquad (5)$$

Esta representação por uma seqüência de códigos de blocos, que é relacionada com a representação das transições de estado por uma árvore, não é muito utilizada pela dificuldade de se considerar explicitamente todas as palavras

código de comprimentos múltiplos de n. Contudo, analisando (5) é fácil concluir que, se quisermos que o CCMU gerado por G_0 e G_1 tenha distância livre igual a d^*_{free}, uma condição necessária, mas não suficiente, é que o peso mínimo do código de bloco gerado por $[G_0\ G_1]$ seja maior ou igual a d^*_{free}. Além disto, para códigos com taxas não muito altas ($1/2 \leq r \leq 3/4$), a experiência prática demonstra que, nos códigos ótimos, a distância livre é determinada pela distância entre as palavras geradas pela matriz $[G_0\ G_1]$, que têm comprimento 2n, já que as palavras geradas pelas outras matrizes têm comprimento 3n, 4n, ..., e portanto devem ter um peso maior, dependendo do número de bits redundantes (n-b). No diagrama de transição de estado, esta suposição é equivalente a se considerar que os "caminhos" de menor peso de Hamming são os que saem do estado **0**, passam por apenas um estado intermediário e voltam imediatamente ao estado **0**.

A distância de Hamming entre as palavras código geradas pela matriz $[G_0G_1]$ depende apenas da configuração binária (tipos) de suas colunas e não da ordem destas colunas na matriz, já que esta corresponde a uma matriz geradora de um código de bloco linear e, para estes códigos, se trocarmos de posição as colunas, alteraremos as palavras código mas não o seu peso de Hamming. No caso das matrizes geradoras das palavras de comprimento maior que 2n que aparecem em (4), esta ordem é importante para determinar a distância entre as palavras código devido ao interrelacionamento entre as submatrizes G_0 e G_1 na formação de suas colunas.

Estas propriedades, que foram mais facilmente visualizadas por esta representação do CCMU por códigos de bloco, são suficientes para se propor a decomposição do problema de determinação do código ótimo em dois subproblemas menores e de resolução mais simples. A estratégia de decomposição a ser empregada objetiva eliminar a necessidade de testar um número muito grande de soluções. As duas etapas foram definidas conforme descrito a seguir.

Processo de Seleção dos Tipos de Coluna da Matriz Geradora

Determinar um conjunto de 2n colunas tais que, se usadas para formar $[G_0G_1]$, independentemente da sua ordenção, maximizam o peso mínimo do código de bloco gerado por $[G_0G_1]$. Como vimos anteriormente, esta etapa define uma condição necessária e portanto o peso destas palavras deve ser maior ou igual a d^*_{free}. Nesta etapa as matrizes G_0 e G_1 são analisadas como em um código de bloco.

Processo de Alocação dos Tipos nas Colunas da Matriz Geradora

A partir do conjunto de tipos de colunas obtido pelo processo anterior, determi-

nar que alocação destes tipos nas colunas de $\mathbf{G}_0$ e $\mathbf{G}_1$ formará uma matriz geradora cujo código convolucional tem distância livre igual a d^*_{free}. Pode-se notar que apenas nesta etapa irá se considerar $\mathbf{G}_0$ e $\mathbf{G}_1$ como matrizes geradoras de um código convolucional.

O ponto fundamental na aplicação eficiente desta estratégia de decomposição é a conjetura de que, ao se avançar da primeira para a segunda etapa, cada conjunto de soluções que não é eliminado conterá, com grande probabilidade, soluções ótimas. Na prática, isto significa que podemos analisar um pequeno número de soluções do problema de seleção de tipos de colunas (ao invés de todas) e, neste conjunto, encontrar uma solução ótima do problema original através do algoritmo de alocação. Com esta condição, é possível a aplicação de algoritmos heurísticos de busca bastante eficientes na resolução dos subproblemas, conforme veremos nas seções 4 e 5. Inicialmente pode-se considerar que esta conjetura aplica-se apenas aos códigos de taxa mais baixa ($r \leq 1/2$), onde o maior número de bits redundantes conduz a problemas com mais graus de liberdade. Contudo, resultados práticos mostram que esta conjetura pode ser válida mesmo para taxas altas, como $r = 7/9$.

4. Determinação das Colunas da Matriz Geradora

Como vimos anteriormente, pode-se formular um algoritmo de busca de códigos convolucionais ótimos a partir de códigos de bloco, mas os códigos de bloco normalmente usados, que são sistemáticos, não são os mais indicados para formar códigos convolucionais ótimos. É necessária, então, uma definição matemática do problema, a partir da qual possa ser criado um algoritmo capaz de gerar um conjunto relativamente grande de códigos não sistemáticos, satisfazendo às condições de distância apresentadas na Seção 3.

A determinação de um código de bloco linear (2n,b) gerado por uma matriz binária $\mathbf{G} = [\mathbf{G}_0\ \mathbf{G}_1]$ e com máxima distância mínima de Hamming (capacidade de correção) pode ser formulada como um problema de otimização. Neste caso, é interessante que esta formulação tenha certas propriedades como linearidade e convexidade, o que é dificultado pelo fato das operações de geração do código, com a matriz $\mathbf{G}$, serem definidas em GF(2) (operações binárias). Para contornar esta dificuldade, pode-se usar o artifício de construir uma tabela com os resulados das operações binárias, como veremos a seguir.

Inicialmente define-se um vetor de variáveis de problema $\mathbf{c}$, tal que $c_i = r$ se r tipos de colunas de $\mathbf{G}$ tiverem representação binária igual a i. As colunas com todos os bits iguais a 0 não são representadas e $\mathbf{c}$ tem dimensão $N = 2^b-1$. A seguir, definimos uma matriz $\mathbf{M}_b$, com dimensão NxN, que contém to-

dos os possíveis valores não nulos dos vetores binários de dimensão b. Por exemplo, se b=3, temos

$$\mathbf{M}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Podemos notar que a matriz $\mathbf{M}_b$ contém todos os possíveis tipos de colunas de **G** e $\mathbf{M}^t_b$ (sua transposta) contém todos os possíveis vetores binários não nulos de entrada do codificador. Desta forma, se definirmos $\mathbf{H}_b = \mathbf{M}_b{}^t \otimes \mathbf{M}_b$, onde a operação entre as matrizes é feita em GF(2), $\mathbf{H}_b$ irá conter todos os resultados binários dos produtos de todos os possíveis tipos de colunas da matriz **G** por todos os possíveis vetores binários de entrada. O peso de Hamming das palavras código correspondentes pode ser calculado somando os termos das linhas de $\mathbf{H}_b$ correspondentes às colunas pertencentes a **G**. Portanto, podemos calcular o vetor de pesos das palavras código, **d**, através de

$$\mathbf{d} = \mathbf{H}_b \mathbf{c} \qquad (6)$$

Com estas definições, todas as operações binárias são feitas no produto $\mathbf{M}_b{}^t \otimes \mathbf{M}_b$ e as propriedades de distância do código, gerado pela matriz **G**, podem ficar implícitas em (**6**). A representação da matriz **G** pelo vetor de variáveis **c** já inclui a propriedade de que a ordem das colunas não irá alterar os pesos das palavras código. Note que (**6**) justifica a escolha da represetação das matrizes geradoras pelas suas colunas.

Chegamos então a uma forma de calcular os pesos de Hamming das palavras código que não envolve operações binárias diretamente, que é linear, mas com a restrição de que os elementos do vetor **c** sejam inteiros. Como, normalmente, temos um grande número de códigos cuja distância mínima é igual a d^*_{free}, para minimizar a probabilidade de erro, é interessante minimizar também o número de palavras código que têm peso igual a d^*_{free}. Com este intento, definimos uma função de penalização da forma

$$\pi\ (x, y) = \begin{cases} (y - x)^2 \text{ se } y \geq x \\ 0 \text{ caso contrário} \end{cases}$$

Podemos então formular o problema P_2 como:

$$P_2 \qquad \text{Minimizar} \quad z(\mathbf{c}) = \sum_{i=1}^{N} \pi(d_i, d^*_{free}+1) \tag{7}$$

$$\text{sujeito a} \quad \mathbf{H}_b\,\mathbf{c} = \mathbf{d} \tag{8}$$

$$\sum_{i=1}^{N} c_i = 2n \tag{9}$$

$$c_i \in \{0, 1, 2, \ldots, 2n\} \tag{10}$$

Nesta formulação, a função objetivo em (7) é convexa e não linear, penalizando a ocorrência de palavras com peso menor que d^*_{free}. Caso todas as palavras tenham peso maior ou igual a d^*_{free}, a função objetivo minimiza o número de palavras com peso igual a d^*_{free}. A restrição em (8) representa apenas uma transformação de variáveis para simplificar a notação e **d** é o vetor com os pesos de Hamming das palavras código. A restrição em (9) faz com que a quantidade dos tipos de colunas seja igual ao número de colunas da matriz geradora, que, pela restrição em (10), deve ser um número inteiro. Como (8) é apenas uma mudança de variável, podemos notar que as restrições em (9) e (10) definem o chamado **problema de seleção,** ou seja, selecionar um conjunto de 2n tipos de colunas que irá maximizar uma função avaliadora da capacidade de correção do código.

O problema P_2 é um problema de otimização com variáveis inteiras, que pode ser resolvido por várias ferramentas de programação matemática [**13**]. As considerações sobre a escolha do método mais apropriado e sua descrição seriam muito extensas para serem incluídas neste trabalho. Em [**14**] é apresentado um algoritmo heurístico desenvolvido especialmente para resolução deste problema.

5. Ordenação das Colunas da Matriz Geradora

A partir das soluções do problema P_2, temos uma determinada quantidade de conjuntos de tipos de colunas que, se usadas para formar G_0 e G_1, satisfazem a condição necessária de que nenhuma palavra do código convolucional de tamanho 2n terá peso menor que d^*_{free}. Além disso, pelo que foi concluído na Seção 3, existe uma grande probabilidade de que algumas das ordenações

destas colunas irão gerar códigos convolucionais não catastróficos com distância livre igual a d^*_{free} e, dentre estas, devemos ter aquela ordenação cujo código têm a menor probabilidade de erro.

Para representação das variáveis do problema, devemos considerar que temos 2n colunas de G_0 e G_1 que podem ser colocadas em 2n posições. Trata-se, portanto, de um clássico **problema de alocação**, o que sugere representar as variáveis pela matriz **P**, de dimensão 2nx2n, tal que $p_{ij} = 1$ se o tipo i está na posição j e $p_{ij}=0$ se não está. A matriz **P** também pode ser interpretada como uma matriz de permutação aplicada a uma matriz **G** que tem uma ordenação arbitrária dos tipos de colunas determinadas pelo problema P_2.

Este problema de alocação também pode ser formulado como um problema de otimização, mas o interrelacionamento entre G_0 e G_1 na formação das palavras do código convolucional dificulta a eliminação das operações binárias na formulação do problema de otimização correspondente. O artifício utilizado na formulação do problema P_2 pode se usado, se considerarmos as posições dos tipos de G_0 ou de G_1 fixas. Neste caso, é necessario ponderar a eficiência na resolução versus a fidelidade na formulação. Neste trabalho, escolhemos a primeira opção para poder determinar códigos de maior dimensão.

A segunda etapa para formulação matemática do problema é a escolha da representação mais apropriada do código. Usando, por exemplo, a representação por códigos de blocos, considerando as palavras código de comprimento 3n, temos o seguinte problema:

P_3

Minimizar $$w(\mathbf{P}) = \sum_{l=1}^{2^{2b}-1} \pi(d_l, d_g+1)$$

sujeito a

$$\sum_{i=1}^{n} p_{ij} = 1, \qquad j \in \{1, 2, \ldots, n\} \tag{11}$$

$$\sum_{j=1}^{n} p_{ij} = 1, \qquad i \in \{1, 2, \ldots, n\} \tag{12}$$

$$p_{ij} \in \{0, 1\} \tag{13}$$

$$d_l = \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ijl}\ p_{ij}, \qquad l \in \{1, 2, \ldots 2^{2b}-1\} \tag{14}$$

A função objetivo é semelhante à do problema P_2, mas usando uma distância mínima d_g que pode ser maior que d_{free}. As restrições em (**11**) e (**12**) garantem que nenhum tipo de coluna ocupa mais de uma posição e todas as posições são ocupadas. As restrições em (**14**) representam apenas uma mudança de variável, definindo os pesos das palavras código. Os parâmetros h_{ijl} dão o incremento no peso da palavra código l, caso o tipo i esteja na posição j.

A grande dificuldade na resolução deste problema é a sua grande dimensão, que é proporcional a 2^{2b}. Por exemplo, para um código (12,7) temos aproximadamente dezesseis mil variáveis d_l e mais de dois milhões de valores de h_{ijl} que devem ser calculados e armazenados. Além disto, como as soluções de P_3 satisfazem apenas mais uma condição necessária, é preciso considerar também as palavras código com comprimento 3n, 4n,..., que produzem problemas com dimensão muito maior.

Para ser possível trabalhar com palavras código de comprimento arbitrário é preciso voltar a uma representação mais compacta e um algoritmo de análise que permita selecionar apenas as palavras de mínimo peso, que são as mais importantes na avaliação da capacidade de correção. Podemos usar as propriedades definidas na Seção 2, e incluir (**14**) na formulação de P_1 (o problema de máximo fluxo). Neste caso, temos um interrelacionamento entre os problemas de alocação de recursos e máximo fluxo, podendo-se usar um algoritmo de programação matemática que, a partir da resolução sequencial dos dois problemas, chegue ao código ótimo.

Podemos ainda usar um método heurístico para solucionar o problema de alocação, considerando empiricamente duas propriedades dos CCMUs. Primeiramente, devemos levar em conta que se alterarmos apenas dois valores de **P**, mantendo a factibilidade em relação às restrições (**11**) até (**13**), a máxima variação dos pesos das palavras código de comprimento xn, $x>2$, é igual a $\pm 2(x-2)$. Esta "pequena alteração" é equivalente a trocar duas colunas de posição. A segunda propriedade é que, normalmente, para códigos com taxa $r \geq 1/2$, o perfil das distâncias ("distance profile") apresenta um crescimento rápido dos pesos de Hamming das respectivas palavras código.

A partir destas propriedades podemos esperar que pequenas alterações nas variáveis p_{ij} produzem pequenas variações nas propriedades do código, ou seja, as variações na função avaliadora de um CCMU devem ser "suaves" em relação a pequenas variações em **P**. Assim, podemos também esperar que um algoritmo que vá cumulativamente melhorando as características do código seja viável. Contudo, trata-se de um problema bem mais complexo do que P_2, o que deve dificultar este processo de busca localizada.

A grande liberdade que se obtém através do uso de um algoritmo heurístico permite tentar minimizar também o número de palavras com peso mínimo, o que pode ser conseguido pela aplicação do algoritmo de Viterbi. Mais uma vez, é interessante mudar a representação do código para analisar as variações nas palavras de peso mínimo, usando a representação por treliça. Na **Fig. 2** temos a representação de uma treliça em 5 janelas de tempo. Podemos definir uma função de avaliação do código a partir dos resultados da aplicação do algoritmo de Viterbi. A idéia é fixar um conjunto de parâmetros $d^{(r)}_m$, iguais ou maiores que os valores mínimos esperados dos pesos das palavras de comprimento r. Se, durante a aplicação do algoritmo de Viterbi, for encontrada uma palavra com peso δ, menor que $d^{(r)}_m$, a função de penalização $\pi(\delta, d^{(r)}_m)$ anteriormente definida é somada à função de avaliação do código. O algoritmo heurístico consiste em trocar as colunas da matriz geradora, duas a duas, cada vez que isto acarreta uma diminuição na função de penalização. Em (14) é apresentada de uma descrição deste algoritmo.

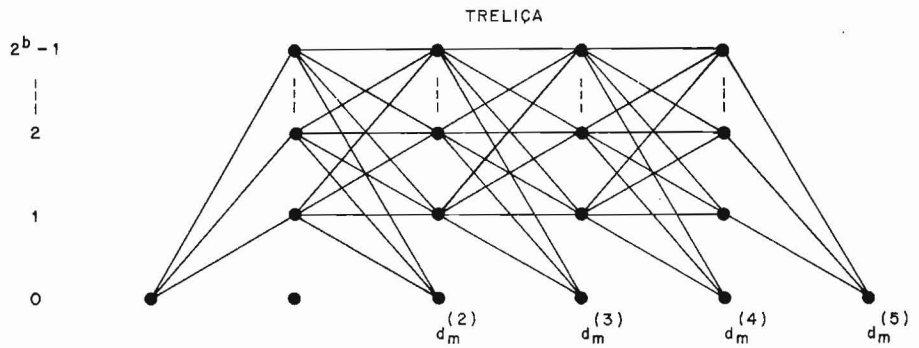


Figura 2. Diagrama de treliça.

6. Resultados Computacionais

Conforme explicado na Seção 3, onde se justifica o processo de decomposição, uma condição esperada para o sucesso da aplicação dos algoritmos de resolução dos problemas P_2 e P_3 é que os códigos procurados não tenham taxas muito altas. Os melhores resultados são esperados para taxas menores que 1/2, quando os critérios de otimalidade podem ser mais facilmente alcançados pela existência de um número suficiente de bits redundantes, de modo que o perfil de distâncias cresce rapidamente. Contudo, foi possível obter bons resultados mesmo para taxas maiores que 3/4, conforme a dimensão do problema. Neste último caso, o preço a pagar foi um menor controle da distribuição dos pesos das palavras, embora conseguindo-se códigos com distância li-

vre igual ao máximo possível (o que, para alguns autores, já representa a otimalidade do código).

Na **Tabela 2** são apresentados alguns códigos convolucionais de memória unitária ótimos, até então não conhecidos na literatura técnica. Na obtenção destes códigos, os seguintes critérios de otimalidade foram empregados (em prioridades crescente):

(i) eliminação dos códigos catastróficos;
(ii) maximização de d_{free},
(iii) minimização do número de palavras código com peso d_{free};
(iv) crescimento dos pesos das palavras código na treliça.

É interessante observar que ao se incluir o critério de crescimento do peso das palavras na treliça para selecionar os códigos com máxima distância livre, é possível eliminar os códigos catastróficos, já que estes têm um perfil de distância inferior. Não chegamos a encontrar uma classe de códigos (n,b), com $r \geq 2/3$ e $b > 4$, em que todos os códigos com máximo d_{free} são catastróficos.

Todos os códigos na **Tabela 2** têm b e n primos entre si e portanto não podem ser transformados em códigos de memória não unitária. Para se avaliar seu desempenho, são apresentados os ganhos assintóticos (γ) de cada código para o canal AWGN ("Additive White Gaussian Noise"), sendo γ para um código (n, b, m), com distância livre d_{free} e 2^m estados, calculado por

$$\gamma = 10 \log_{10} \frac{b\, d_{free}}{2n} \quad \text{dB}$$

Os valores de γ obtidos podem ser comparados com ganhos assintóticos de outros códigos ótimos, de memória múltipla, apresentados em [**15**]. Por exemplo, o novo código de memória unitária (7, 5, 1) apresenta a mesma distância livre que o código (3, 2, 3) ótimo de [**5**], porém, quando comparados seus desempenhos vemos que o código (7, 5, 1) apresenta um valor de α igual a 3,31 enquanto que o código (3, 2, 3) apresenta um valor de γ igual a 3,10. Portanto, pode-se concluir que o novo código tem um melhor desempenho com uma complexidade equivalente.

b	n	G_0	G_1	d_{free}	γ (dB)
5	6	000011 100010 011000 011110 010010	011011 110111 100100 001010 001101	4	2,22
5	7	0110001 0100100 0011011 1100000 0000011	0010110 1010101 1011100 0101011 0111010	6	3,31
5	8	00110111 01100010 11100101 00100100 01011000	11010010 01011111 00010110 11101110 10110110	7	3,40
6	11	11101011101 11001001000 10011100110 10110010110 10000001110 00111000010	10101110100 11100010001 10100010100 00000111010 11101100110 11010100100	9	3,90
6	13	0001110100000 1100110001100 0010101110011 0100010110001 0000111001001 0000101100110	1111001110001 0100011111000 0000100111010 1001011101010 0011100010111 1100011001011	12	4,42
7	9	110100010 001011100 000100010 100100100 101010000 100111100 000101111	110111010 101101110 110000011 011001110 100010010 100010100 000101000	6	3,68

Tabela 2. Exemplos de códigos ótimos encontrados.

7. Conclusões

Neste trabalho foi apresentado um procedimento direcionado à sistematização do problema de seleção de códigos convolucionais de memória unitária via técnicas de otimização, em contraposição às técnicas de busca exaustivas até então utilizadas. A determinação da matriz geradora do código foi possível graças à resolução de dois problemas de otimização inteira, cuja função objetivo penaliza as palavras código com peso de Hamming abaixo do desejado. O primeiro problema seleciona os tipos de colunas que formarão a matriz geradora. A determinação da ordem das colunas nas submatrizes geradoras foi conseguida após a resolução de um problema de alocação de recursos. Como resultado desta nova metodologia, novos códigos convolucionais de memória unitária foram apresentados.

Referências

[1] P. Elias, "Coding for Noisy Channels", IRE Convention Record, Part 4, 1955, pp.37-47.

[2] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", IEEE Transactions on Information Theory, vol. IT-13, no. 2, Abril 1967, pp. 260-269.

[3] J.P. Odenwalder, "Optimal Decoding of Convolutional Codes", PhD Dissertation, University of California, Los Angeles, 1970.

[4] K.J. Larsen, "Short Convolutional Codes with Maximum Free Distance for Rates 1/2, 1/3 and 1/4", IEEE Transactions on Information Theory, vol. IT-19, no. 3, Maio 1973, pp. 371-372.

[5] E. Paaske, "Short Binary Convolutional Codes with Maximal Free Distance for Rates 2/3 and 3/4", IEEE Transactions on Information Theory, vol. IT-20, no. 5, Setembro 1974, pp. 683-698.

[6] R. Johannesson e E. Paaske, "Further Results on Binary Convolutional Codes with an Optimum Distance Profile", IEEE Transactions on information Theory, vol. IT-24, no. 2, Março 1978, pp. 264-268.

[7] J.L. Massey e D.J. Costello, Jr., "Nonsystematic Convolutional Codes for Sequential Decoding in Space Applications", IEEE Transactions on Communication Technology, vol. COM-19, no. 5, Outubro 1971, pp. 806-813.

[8] D.G. Daut, J.W. Modestino e L.D. Wismer, "New Short Constraint Lenght Convolutional Code Constructions for Selected Rational Rates", IEEE Transactions on Information Theory, vol. IT-28, no. 5, Setembro 1982, pp. 794-800.

[9] L.N. Lee, "Short Unit-Memory Byte Oriented Binary Convolutional Codes Having Maximal Free Distance", IEEE Transactions on Information Theory, vol. IT-22, no. 3, Maio 1976, pp. 349-352.

[10] R. Palazzo Jr., "Analysis of Periodic Linear and Nonlinear Trellis Codes", PhD Dissertation, University of California, Los Angeles, 1983.

[11] R. Palazzo Jr., "A General Relationship Between the Maximum Flow in Networks and the Minimum Distance of Convolutional Codes Over GF(q)", 3rd Joint USSR-Swedish International Workshop on Information Theory, Sochi, USSR, Maio 1987.

[12] G. Thommesen e J. Justesen, "Bounds on Distance and Error Exponents of Unit Memory Codes", IEEE Transactions on Information Theory, vol. IT-29, no. 5, Setembro 1983, pp. 637-649.

[13] G.L. Nemhauser e L.A. Wolsey, "Integer and Combinatorial Optimization", John Wiley & Sons, 1988.

[14] A. Said e R. Palazzo Jr., "Heuristic Search: A New Method to Find Good Unit-Memory Convolutional Codes", Fourth Joint Swedish-Soviet International Workshop on Information Theory, Gotland, Suécia, Agosto 1989.

[15] S. Lin e D. J. Costello Jr., "Error Control Coding: Fundamentals and Applications", Prentice-Hall, Englewood Cliffs, N.J., 1983.

AMIR SAID formou-se em Engenharia Elétrica pela UNICAMP em 1985, obteve o título de Mestre em Engenharia Elétrica pela UNICAMP em 1988. Desde 1987 atua como Professor da Faculdade de Engenharia Elétrica da UNICAMP. Ocupou o cargo de Engenheiro associado junto à IBM do Brasil onde de maio a novembro de 1986 participou do grupo de desenvolvimento de um equipamento periférico nos Laboratórios da IBM em Tucson, Arizona e Boca Raton, Flórida. Atualmente encontra-se realizando estudos no Rensselaer Polytechnic Institute, Troy, New York para obtenção do título de Ph.D.

REGINALDO PALAZZO JUNIOR formou-se em Engenharia Elétrica pela UNICAMP em 1975, obteve o título de Mestre em Engenharia Elétrica pela UNICAMP em 1977 e os títulos de Degree of Engineering e Ph.D. ambos em Engenharia Elétrica pela University of California, Los Angeles, em 1981 e 1983, respectivamente. Durante o ano de 1984 foi consultor da TTI e Quotron System em Los Angeles. Desde 1985 pertence à Faculdade de Engenharia Elétrica da UNICAMP onde em 1987 obteve o título de Professor Livre Docente (concurso público). As áreas de pesquisa em que atua são Teoria da Informação e Codificação, Criptografia, Teoria de Comunicações e recentemente na área de Gravação Magnética/Óptica de Alta Densidade.