UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE TELEMÁTICA

Este exemplar corresponde a redação final da Tese de Mestrado de Maria Conceição Peres Young defendida em 14/02/89

Reginaldo Palazzo Júnior

"OBTENÇÃO DE CÓDIGOS CONVOLUCIONAIS ÓTIMOS DE MEMÓRIA UNITÁRIA POR PROGRAMAÇÃO MATEMÁTICA"

AUTORA : Maria Conceição Peres Young.
ORIENTADOR : Prof. Dr. Reginaldo Palazzo Jr.

Tese apresentada a Faculdade de Engenharia Elétrica como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

14 de FEVEREIRO - 1989.

UNICAMP
BIBLIOTECA CENTRAL

ABSTRACT

This dissertation has as objective, to present a new algorithm to the systematic attaintion of optimum convolutional codes of unit memory, time invariable, as in the existing literature, a mathematical structure well defined was unknown for this purpose until then.

This structure is very important because convolutional codes are of great use, providing a high codification gain. The performance of these codes is directly related to its distance properties. It is good to reinforce that this is not a trivial task, as it belongs to the UP-hard class in its worst case. Henceforth, it is presented in this dissertation an algorithm which deals with combinatorial optimization techniques related to the problem of optimum convolutional codes determination, when this problem is characterized as the maximum flow determination in a network.

Within a few methods and techniques used, the knapsack combinatorial problem and the dynamic programming problem are emphatized. The latter, applied to code-word enumeration via Viler Algorithm as a security way to obtain reliable new results and the confirmation of some already existents.

RESUMO

Este trabalho tem por objetivo apresentar um algoritmo novo para a obtenção sistemática de códigos convolucionais ótimos de memória unitária, invariantes no tempo, uma vez que até então, na literatura existente, não era conhecida uma estrutura matemática bem definida para tal implementação. A importância de tal estrutura, reside no fato de que os códigos convolucionais são de grande aplicação, uma vez que proporcionam um alto ganho de codificação. O desempenho destes códigos está diretamente relacionado com suas propriedades de distância. Convém ressaltar que este problema é não trivial, uma vez que pertence à a classe dos NP-completos no seu pior caso. Sendo assim, apresentamos neste trabalho um algoritmo que conta com técnicas de otimização combinatorial relativas ao problema de determinação de códigos convolucionais ótimos , quando caracterizamos este problema como o de determinar o fluxo máximo numa rede [1]. Dentre alguns dos métodos e técnicas utilizadas ressaltamos o Problema Combinatorial daMochila e o Problema de Programação Dinâmica. Este último, aplicado a enumeração das palavras-código via algoritmo de Viterbi, como uma medida de segurança da veracidade dos resultados novos obtidos eà confirmação de alguns já determinados anteriormente.

" Uma pessoa é feliz como o resultado de seus próprios esforços , uma vez que reconheça os ingredientes necessários para a felicidade - gostos simples , certo grau de coragem, espírito de sacrifício e, acima de tudo, uma consciência limpa."

À minha mãe, Maria Lucia.

AGRADECIMENTOS

1- À minha mãe, Maria Lucia, meu irmão ,"Dico", meus avós, Dalila e Justo, pelo apoio e incentivos, constantes,em todos os sentidos e,principalmente, pela maior prova de amor ao compreender minha ausência física na busca do meu ideal;

2- A meu orientador, Prof. Dr. Reginaldo Palazzo Jr. , cuja dedicação, interesse e participação, puderam deixar traçado na memória, o exemplo do que é ser um profissional competente, justo e inovador;

3- Aos professores da FEE, cuja participação direta ou indireta em sugestões e apoio, colaboraram para este trabalho e ao Prof. Dr. Clóvis Perin Filho (IMECC- UNICAMP), pela boa vontade e disposição no atendimento às sugestões bibliográficas na parte de Pesquisa Operacional;

4- Aos funcionários da FEE, pelo carinho no atendimento as necessidades de um bom ambiente de estudo, em especial à "Marlene" e à secretária do Convênio Telebrás, Neli;

5- Aos amigos e colegas : Claudia, Eliane, Salimar, Luli, Ingeborg, Maria, Hamilton, Rosane, Dinho, Diniz, Roseli , Gorgônio, Miguel, Furuya, "Taka", Sonia ,Coelho, Chico,... e ,na fase de homologação, à Paulo Maurício pela força e carinho sempre presentes.

6- À CAPeS e ao convênio TELEBRÁS, pelo apoio financeiro.

ÍNDICE

PÁG.

Capítulo 1 - Introdução 01

Capítulo 2 - Revisão dos Conceitos de Codificação..... 06

2.1- Introdução.................................. 06
2.2- Revisão de Algumas Classes de Códigos....... 06
2.3- Definições e Conceitos 08
2.3.1- Parâmetros Notacionais............... 09
2.3.2- Algumas Definições Importantes....... 10
2.4- Códigos de Memória Unitária................. 23
2.5- Algoritmo de Decodificação de Viterbi....... 27
2.5.1- Introdução........................... 27
2.5.2- Diagrama de Treliça.................. 28
2.5.3- O Algoritmo de Viterbi............... 29
2.5.4- Exemplo de Aplicação de Viterbi ao Canal Binário Simétrico (BSC).......... 31

Captítulo 3 - Programação Dinâmica e Knapsack Combinatorial.................................. 33

3.1- Introdução.................................. 33
3.2- Programação Dinâmica........................ 34
3.2.1- Estrutura Básica do Problema de Programação Dinâmica........................35
3.2.2- Exemplo Protótipo de Programação Dinâmica......................................39
3.2.3- Tipos de Programação Dinâmica..........42
3.3- Knapsack Combinatorial.......................44
3.3.1- O Problema Knapsack....................47
3.3.2- Técnica do Backtrack....................49
3.3.2.1- Branch-and-Bound...............52

3.3.2.1.1- Branch-and-Bound aplicado

ao Knapsack 55

Capítulo 4 - Problema Knapsack Combinatorial aplicado aos Códigos de Memória Unitária...........60

4.1- Introdução..............................60
4.2- Equivalência dos Problemas : Determinação de Códigos Ótimos e Fluxos em Redes.............60
4.3- Problema Knapsack Combinatorial..............68
4.4- Enumeração das palavras-código...............82

Capítulo 5 - Descrição do Algoritmo Proposto e Resultados..............................88

5.1- Introdução..............................88
5.2- Descrição Geral da Implementação Computacional do Algoritmo..............................89
5.3- Especificações Computacionais do Algoritmo....90
5.3.1- Programa Knap_5.pas..................91
5.4- Resultados..............................97

Capítulo 6 - Conclusões 102

Apêndice A - Implementação do Programa DADOS.PAS 104

Apêndice B - Implementação do Programa MATVIT.PAS.......107

Apêndice C - Fluxogramas 124

CAPITULO 01

CAPÍTULO 1

INTRODUÇÃO

Desde a proposta por Elias [1] de uma nova classe de códigos corretores de erros, que utilizasse da dependência entre os símbolos a serem codificados, para se alcançar um melhor desempenho é que inúmeras pesquisas têm sido realizadas.

Deste esforço, resultou uma quantidade considerável de sub-classes de códigos, da classe originalmente proposta, cada uma apresentando características relevantes quanto a aplicações.

A Fig. 1.1 , apresenta uma retrospectiva das Subclasses dos códigos e, para melhor entende-la, faremos um breve resumo dos que serão importantes para a tese.

Os <u>códigos de árvore</u> devem ser entendidos como aqueles que processam uma sequência de informação sem particioná-la em blocos independentes, isto é, o código de árvore faz o processamento de forma contínua e associa cada sequência de informação (mesmo que semi-infinita) a uma sequência codificada. O nome código de árvore origina-se do fato das regras de codificação, para este tipo de código, serem mais convenientemente descritas através do diagrama de uma árvore.

Já os <u>códigos de bloco</u>, como o próprio nome já diz, particiona uma sequência contínua de dígitos de informação em blocos de k-símbolos, cada um com comprimento n, e opera nestes blocos de forma independente. Como os códigos de bloco possuem uma estrutura matemática melhor definida são, portanto, os que apresentam um maior desenvolvimento teórico.

Devemos entender por <u>código linear</u> aquele que consiste simplesmente de um conjunto de combinações lineares dos símbolos dados. Pelo fato dos códigos lineares obedecerem as propriedades de "fechamento", identidade e apresentarem inversas sobre as operações associativa e comutativa (que , portanto formam um **grupo abeliano**), são conhecidos, também, por **códigos de grupo**. Uma consequência da propriedade de fechamento do código linear, e

de fundamental importância para este trabalho, é que o conjunto das distâncias de Hamming de um dado vetor código a outro vetor código é o mesmo para todos os vetores códigos. A distância de Hamming entre dois vetores código é justamente o peso de Hamming da soma módulo-2 dos dois vetores.

Quanto aos <u>códigos convolucionais</u>, devemos entende-los como aqules que formam um subconjunto da classe dos códigos de árvore e que proporcionam um alto ganho de codificação. Os códigos convolucionais podem ser vistos, também, como uma classe especial dos códigos de bloco lineares. O termo convolucional aplica-se a esta classe de códigos porque a sequência de símbolos de saída, $\underline{v}$, pode ser expressa como a convolução da sequência de entrada, $\underline{u}$, com a sequência geradora do código. Assim, desde que o código seja linear, temos $\underline{v} = \underline{u}.G$, onde G é a matriz geradora do código.

Assim como os códigos de bloco lineares são uma subclasse dos códigos de bloco, os códigos convolucionais são uma subclasse de uma classe mais ampla denominada <u>códigos de treliça</u>.

Os códigos de treliça de taxa $r = k/n$, também emitem n-símbolos cada vez que k-símbolos da fonte entram no registrador, entretanto, eles podem produzir símbolos,representados por uma função arbitrária (não linear) dos bits da fonte no registrador do codificador, de qualquer alfabeto de entrada do canal. Uma vez que os estágios do registrador são os mesmos tanto para a classe dos códigos de treliça, de forma geral, quanto para os códigos convolucionais, a árvore, a treliça e o diagrama de estado serão os mesmos. Dessa forma, fica claro que os códigos de treliça estão relacionados aos códigos de bloco, da mesma forma que os códigos convolucionais estão relacionados aos códigos de bloco lineares. Tal comparação entre os códigos de bloco lineares e os códigos convolucionais é possível devido ao fato dos códigos convolucionais serem um caso especial dos códigos de bloco, isto é, se $k.K$ é o parâmetro de comprimento de memória do codificador e $r=k/n$ é a taxa do código, quando $K=1$ e $n=N$, obtemos um código de bloco com taxa $r=k/N$ e, paradoxalmente, o código de bloco linear

pode ser considerado como um caso especial dos códigos convolucionais e, como uma classe mais ampla dos códigos de bloco, pode ser considerado como um caso especial dos códigos de treliça.

Embora estas sub-classes revelem códigos com boa capacidade de correções de erros, porém , um procedimento sistemático de geração dos mesmos ainda está para ser proposto.

A dificuldade em se sistematizar tal procedimento para a construção ou a geração de códigos,advém da falta de uma estrutura matemática bem definida.

O objetivo desta tese passa a ser, então, o de apresentar um algoritmo novo para a obtenção sistemática da matriz geradora de códigos convolucionais ótimos, invariantes no tempo, através dos códigos de memória unitária [2] via knapsack combinatorial [3]. Dessa forma, procuramos proporcionar um menor grau de dificuldade na obtenção de códigos convolucionais ótimos.

Sendo assim, o presente algoritmo faz uso de técnicas de otimização combinatorial relativas ao problema de determinação de códigos convolucionais ótimos, quando caracterizamos este problema como o de determinar o fluxo máximo em uma rede [4].

Tais conceitos estão distribuídos através dos seis capítulos que constituem esta tese, da seguinte forma :

Capítulo 2 - Fornece maiores detalhes quanto aos conceitos de codificação envolvidos no contexto;

Capítulo 3 - Apresenta informações sobre o problema Knapsack combinatorial, cuja resolução fará uso da técnica do Backtrack , e conceitos de Programação Dinâmica;

Capítulo 4 - Apresenta o problema proposto sendo resolvido através das informações contidas nos capítulos precedentes;

Capítulo 5 - Apresenta a implementação sistemática do algoritmo, fazendo sua descrição geral e comentando o Programa Knap-5.pas. Neste capítulo são apresentados também os

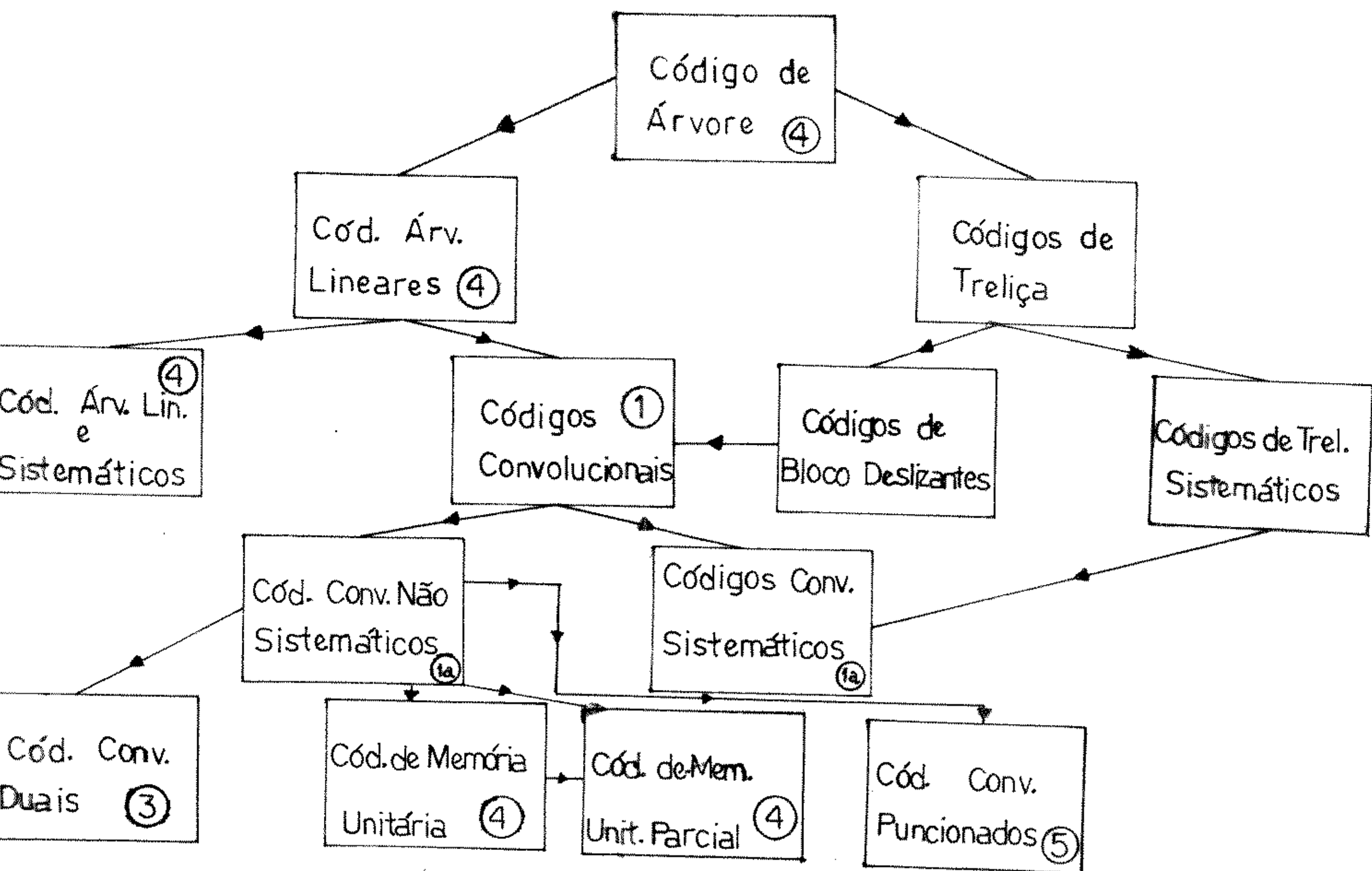
resultados obtidos através da referida implementação.

Capítulo 6 - Neste capítulo, apresentaremos as conclusões obtidas, bem como referenciamos algumas sugestões futuras visando um melhor desempenho na implementação computacional do algoritmo apresentado.

APÊNDICE A - Apresenta a implementação feita para o programa DADOS.PAS.

APÊNDICE B - Apresenta a implementação computacional feita para o programa MATVIT.PAS.

APÊNDICE C - Apresenta o fluxograma do sistema completo.



① – Elias;
(1.a) – Viterbi;
② – Forney;
③ – Odenwalder;
④ – Massey/ Lee;
⑤ –
⑥ – Robert Gray.

Fig. 1.1 – Retrospectiva das Subclasses dos Códigos.

CAPITULO 02

CAPITULO 2

REVISÃO DOS CONCEITOS DE CODIFICAÇÃO

2.1 - INTRODUÇÃO

Neste Capítulo, procuraremos estabelecer os conceitos e definições de alguns parâmetros importantes que são normalmente utilizados em Teoria de Codificação . Tais parâmetros são identidades específicas dos códigos e tais que viabilizam uma análise comparativa entre os mesmos quanto a capacidade de deteção e/ou correção de erros, complexidade, distribuição de pesos de Hamming, etc ...

Sem perda de generalidade, a teoria de codificação pode ser subdividida em duas categorias a saber uma fundamentalmente algébrica e outra probabilística. A primeira possui característica essencialmente estruturada , isto é, utiliza de estruturas de grupo, anel, corpo, etc...enquanto que a segunda é essencialmente "heurística", isto é, não possuiuma estrutura matemática adequada para sua caracterização .Nas próximas seções estaremos voltados na apresentação dos conceitos citados acima dando um maior enfoque aos códigos de memória unitária, uma vez que podemos restringir a busca de um código linearótimo (tanto de bloco como convolucional) para uma dada taxa e estado de complexidade a este tipo de código.

2.2 - REVISÃO SOBRE ALGUMAS CLASSES DE CÓDIGOS

A título de uma abordagem inicial, no que se refere a códigos, apresentaremos nesta seção algumas considerações quanto aos tipos de códigos que serão apresentados nas próximas seções. Sendo assim, devemos começar apresentando as classes principais que subdividem a classificação dos códigos em Códigos de Bloco e Códigos de Árvore. Por sua vez

tanto os códigos de bloco como os de árvore são ainda subdivididos em lineares e não lineares.

Os códigos de bloco lineares compreendem todos aqueles que possuem uma estrutura fundamentalmente algébrica, bem definida, adequada a sua caracterização; enquanto que aos códigos de árvore em geral, bem como os códigos de bloco não lineares, fica reservada a classe dos códigos cuja estrutura é essencialmente probabilística e heurística respectivamente, isto é, até então o uso de técnicas analíticas de geração para essa classe não foram caracterizadas.

Devido a este fato, uma grande quantidade de pesquisa tem sido realizada para códigos de bloco lineares, devido a facilidade estrutural matemática que apresentam.

Por um código de bloco devemos entender como sendo uma classe de códigos pertencentes a categoria essencialmente algébrica, e tal que a cada k dígitos de informação (n-k) dígitos de paridade são concatenados aos k dígitos de informação, produzindo, então uma palavra-código de comprimento n .

O objetivo da paridade consiste em aumentar o grau de confiabilidade da palavra-código quando comparada com aquela sem redundância.

Desse modo os códigos de bloco lineares podem ser caracterizados através de operações lineares sendo realizadas com os k dígitos de informação que resultarão nos (n-k) dígitos de paridade.

Uma forma sistematizada de geração dos códigos de bloco lineares vem através de suas matrizes geradoras G, cujas linhas são formadas por vetores linearmente independentes que geram um subespaço do espaço vetorial sobre um corpo de q elementos.

A decodificação desses códigos, em geral, utiliza uma matriz de paridade H tal que as linhas são vetores ortogonais aos vetores de G (subespaço gerado por G).

Por códigos de árvore entendemos a classe dos códigos pertencentes a categoria probabilística, em virtude do processo de decodificação ser probabilístico. Dentre os códigos de árvore, a classe dos códigos convolucionais é a que tem recebido maior atenção pelos pesquisadores devido a uma

série de fatores que ficarão evidenciados ao longo deste trabalho.

Básicamente, os códigos convolucionais são caracterizados como a convolução dos dígitos de informação com as submatrizes geradoras.

A matriz geradora desta classe de códigos é uma matriz semi-infinita cujos elementos são as sub-matrizes geradoras. Por sua vez, estas submatrizes são formadas de elementos zeros e uns, respectivos a existência ou não de conexões entre os registros e saídas do codificador.

Como ocorre com os códigos de bloco, os códigos de árvore podem ser classificados como lineares, caso apresentem estruturas algébricas de grupo (ou de anéis, ou de corpos), ou não-lineares.

Os processos de decodificação normalmente utilizados são : o da máxima verossimilhança, ou o algoritmo de Viterbi; o seqüêncial, que é assintóticamente ótimo; e o de lógica majoritária. Os dois primeiros processos são probabilísticos enquanto que o último é algébrico. Também nesta ordem segue a otimalidade dos processos de decodificação, isto é, o mais eficiente em termos de decodificação correta (máxima verossimilhança) até o sub-ótimo (lógica majoritária).

2.3 - DEFINIÇÕES E CONCEITOS

Dentro desta seção, procuraremos deixar claro alguns termos de fundamental importância para o entendimento dos conceitos contidos neste Capítulo. No item 2.3.1 apresentaremos a notação convencional mais utilizada de alguns parâmetros que definem um código. No item 2.3.2, apresentaremos as definições e significados de termos como distância de Hamming , peso de Hamming, distância mínima, etc ... No item 2.3.3 os conceitos de matriz geradora, equação de transição, representação modular de códigos de bloco, etc ... serão apresentados na forma vetorial e matricial, para facilidade de entendimento do algoritmo a ser apresentado em detalhes no Capítulo 4.

2.3.1 - PARAMETROS NOTACIONAIS

Passaremos a seguir a descrever alguns parâmetros através da notação usualmente empregada na especificação de um código.

Podemos representar um código de árvore através da notação

(n , k , m) onde

- n = número de saídas ou comprimento da palavra-código ramo;
- k = número de entradas ou comprimento dos dígitos de informação;
- m = número de memórias.

A Fig. 2.3.1.1 apresenta um diagrama de geração de um código de árvore com k entradas, n saídas e m memórias.

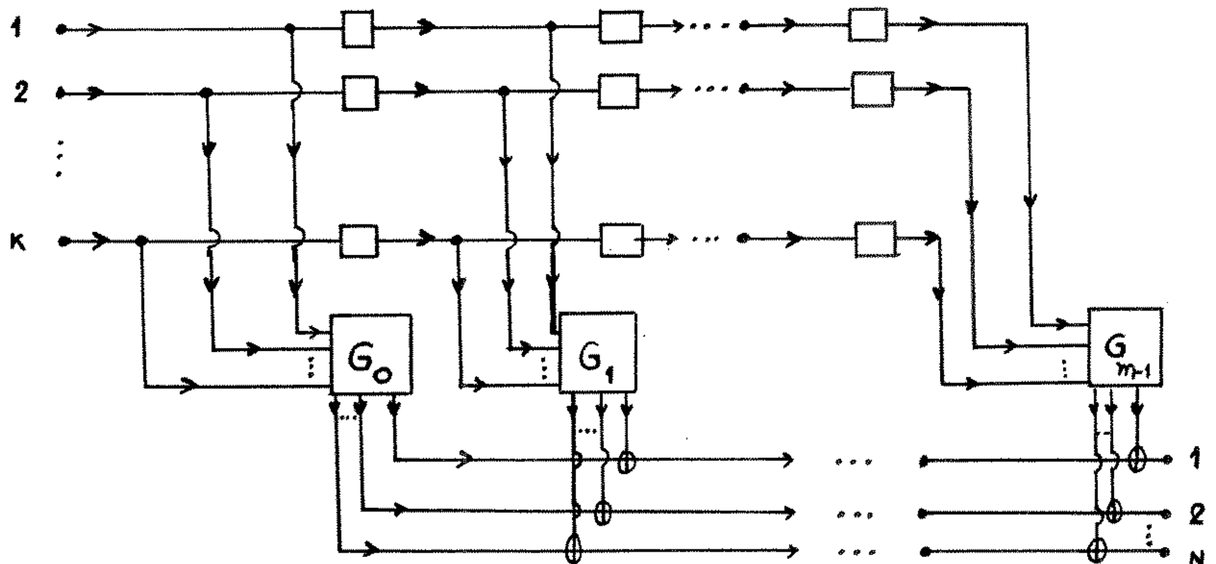


Fig. 2.3.1.1 : Esquema de um circuito apresentando n-saídas, k-entradas e (m-1)-memórias.

Da Fig. 2.3.1.1 fica evidente que a taxa deste código é dada por

$$r = \frac{k}{n}$$

A ordem do corpo de Galois, q , é outro parâmetro significativo, uma vez que o mesmo define os valores que cada componente das palavras-código possuirá .

Dessa forma, se o corpo de Galois possui q = 2 elementos, implica que os dígitos da palavra-código poderão assumir os valores "0" e "1".

Através dos parâmetros citados acima, poderemos passar as definições dos parâmetros que usaremos neste trabalho.

2.3.2 - ALGUMAS DEFINIÇÕES IMPORTANTES

A capacidade de deteção e/ou correção de erros, bem como a otimalidade de um código está diretamente relacionada com a distância mínima. Sendo assim :

PESO DE HAMMING : Devemos entender por peso de Hamming, o número de dígitos não nulos da palavra-código;

DISTÂNCIA DE HAMMING : A distância de Hamming entre as palavras-código de comprimento n, corresponde ao número de posições onde estas duas palavras-código diferem.

DISTÂNCIA MÍNIMA : A distância mínima de um código é a menor distância de Hamming encontrada entre todas as palavras-código.

EXEMPLO : Considere duas palavras-código v1 e v2 dadas por v1=(0,1,1,0,2) e v2=(1,1,0,1,2). Os pesos de v1 e v2 são respectivamente 3 e 4, e a distância de Hamming entre v1 e v2 é 3.

Como a saída da fonte, em geral, consiste de uma seqüência de dígitos de informação, codificá-los consiste em particionar esta seqüência em blocos de comprimento k e transforma-los em blocos de comprimento n dígitos, onde $n > k$. Este bloco de n dígitos recebe o nome de **PALAVRA-CÓDIGO** . Como os digitos de informaçao possuem comprimento k , então o código para ser unicamente decodificável deverá ter no mínimo q^k palavras -código, distintas, sobre GF(q).

EXEMPLO : Considere $k = 3$ e GF(2). Assim, um código que segmenta a seqüência de saída da fonte em blocos com 3 dígitos de informação , transforma cada bloco de mensagem , dentre as 2^3 blocos em uma palavra-código de 6 dígitos, é apresentado a seguir:

MENSAGEM	CODIFICADOR	PALAVRA-CÓDIGO
000	———	000000
001	———	001101
010	———	010011
011	———	011110
100	———	100110
101	———	101011
110	———	110101
111	———	111000

Através da palavra-código, podemos saber que mensagem foi transmitida, pois existe uma correspondência biunívoca entre cada palavra-código e a mensagem.

Uma vez definidos os conceitos de distância mínima, distância de Hamming, peso de Hamming , etc poderemos avançar mais um pouco, definindo conceitos que realmente foram fundamentais na sistematização proposta pelo algoritmo a ser apresentado no Capítulo 4.

MATRIZ GERADORA : Por matriz geradora devemos entender como uma maneira sistemática de gerar as palavras-código . Seja C um código de bloco designado por (n,k) sobre o corpo de q-elementos, isto é, GF(q), onde q é um primo ou potência de um primo. Então, a matriz geradora G do código C possui dimensão k X n. Representaremos G por

$$G = \begin{bmatrix} \underline{g}_0 \\ \underline{g}_1 \\ . \\ . \\ . \\ \underline{g}_{k-1} \end{bmatrix}$$

onde $\underline{g}_i$ são vetores de comprimento n.

Se o código for linear, então as k-linhas de G serão constituídas por vetores linearmente independentes que formarão a base de um subespaço k-dimensional, isto é, $GF(q^k)$. Assim, combinações lineares das linhas da matriz geradora G, fornecerão as palavras-código correspondentes as 2^k possíveis mensagens. Note que os elementos de G pertencem ao GF(q).

Seja $\underline{u} = (u_0 , u_1 , \ldots , u_{k-1})$ uma seqüência de informação ou mensagem a ser codificada, onde u_i , $0 \leq i \leq k-1$ pertence a GF(q).

O codificador transforma cada mensagem $\underline{u}$ em uma palavra-código através da aplicação de G em $\underline{u}$, isto é,

$$\underline{v} = \underline{u} . G = \underline{u}_0\, g_0 + \underline{u}_1\, g_1 + \ldots + \underline{u}_k\, g_k =$$

$$= (v_0 , v_1 , \ldots , v_{n-1}).$$

Em geral, a proposição de novos códigos de bloco lineares se resume na apresentação de uma matriz geradora resultante da utilização das mais variadas estruturas matemáticas, tais como geometria projetiva, combinatória, teoria dos números, etc...

Uma interpretação equivalente à matriz geradora de um código de bloco será utilizada na descrição de um código convolucional, porém, mantidas as especificações de cada classe de códigos.

Para tal, seja o par (i,j) correspondente a i-ésima entrada e j-ésima saída do codificador convolucional onde $0 \leq i \leq k-1$ e $0 \leq j \leq n-1$ (Veja Fig. 2.3.2.1). Para cada valor do par (i,j) teremos associada submatrizes geradoras $g_i^{(j)}$ da correspondente matriz geradora G.

Essas submatrizes são explicitamente dadas por

$$g_i^{(j)}=(g_{i,0}^{(j)},g_{i,1}^{(j)},g_{i,2}^{(j)},\ldots,g_{i,m}^{(j)}),$$

que podem ser vistas como a "resposta ao impulso" a cada par (i,j). Dessa forma, poderemos interpretar as equações de codificação como obtidas através da convolução entre a seqüência de entrada $\underline{u}$ e as seqüências geradoras $g_i^{(j)}$, isto é, $v^{(j)}= \underline{u} * g^{(j)}$.

Após a codificação, as seqüências de saída, $v^{(j)}$, $0 \leq j < n-1$, são multiplexadas em uma única seqüência codificada

$$v = (v_0^{(1)}v_0^{(2)},\ v_1^{(1)}v_1^{(2)},\ v_2^{(1)}v_2^{(2)},\ \ldots)$$

Entrelaçando e arranjando as seqüências geradoras em forma matricial, obtemos a matriz geradora G .

A matriz geradora para um código (n,k,m) é dada por

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & & & \\ & G_0 & G_1 & \cdots & G_{m-1} & G_m & & \\ & & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & \\ & & & & \cdot & & & \\ & & & & & \cdot & & \\ & & & & & & \cdot & \end{bmatrix}$$

onde cada G_l é uma submatriz $k \times n$, cujos elementos são :

$$G_l = \begin{bmatrix} g_{1,l}^{(1)} & g_{1,l}^{(2)} & \cdots & g_{1,l}^{(n)} \\ g_{2,l}^{(1)} & g_{2,l}^{(2)} & \cdots & g_{2,l}^{(n)} \\ \cdot & \cdot & \cdot & \cdot \\ g_{k,l}^{(1)} & g_{k,l}^{(2)} & \cdots & g_{k,l}^{(n)} \end{bmatrix}$$

e os espacos vazios de G são nulos.

Dessa forma, a seqüência semi-infinita de informação,

$\underline{u} = (u_0 , u_1 , \ldots) = (u_c^{(1)} u_c^{(2)} \ldots u_c^{(k)}$ $u_1^{(1)} u_1^{(2)} \ldots u_1^{(k)} , \ldots)$ é transformada em uma palavra-código $\underline{v} = (v_0 , v_1 , \ldots) = (v_c^{(1)} v_c^{(2)} \ldots v_c^{(n)} ,$ $v_1^{(1)} v_1^{(2)} \ldots v_1^{(n)} , \ldots)$ através da relação :

$$\underline{v} = \underline{u} \cdot G$$

EXEMPLO : Seja o codificador binário (2,1,3) apresentado na Fig. 2.3.2.1 , abaixo .

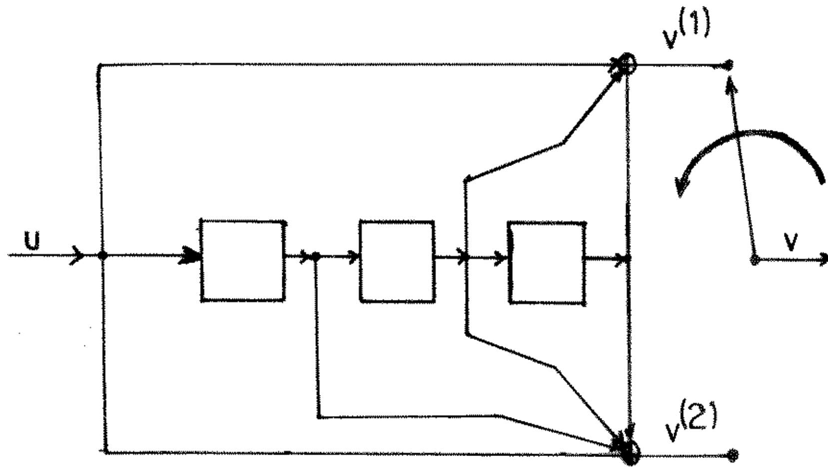


Fig.2.3.2.1 : Código convolucional binário (2,1,3)

As seqüências geradoras deste código são, respectivamente,

$$g^{(1)} = (1 , 0, 1 , 1)$$

$$g^{(2)} = (1 , 1, 1 , 1)$$

Assim , para a seqüência de informação $\underline{u}$ = (1 0 1 1 1) as componentes da palavra-código serão dadas por:

$$v^{(1)} = (1\ 0\ 1\ 1\ 1) * (1\ 0\ 1\ 1) = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 1)$$

$$v^{(2)} = (1\ 0\ 1\ 1\ 1) * (1\ 1\ 1\ 1) = (1\ 1\ 0\ 1\ 1\ 1\ 0\ 1)$$

e, conseqüentemente, a palavra-código resultante será :

$$\underline{v} = (\ 11\quad 01\quad 00\quad 01\quad 01\quad 01\quad 00\quad 11\ )$$

A matriz geradora G para este código será dada por :

$$G = \begin{bmatrix} g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & g_2^{(1)} g_2^{(2)} & \cdots & g_m^{(1)} g_m^{(2)} & & \\ & g_0^{(1)} g_0^{(2)} & g_1^{(1)} g_1^{(2)} & \cdots & g_{m-1}^{(1)} g_{m-1}^{(2)} & g_m^{(1)} g_m^{(2)} & \\ & & g_0^{(1)} g_0^{(2)} & \cdots & g_{m-2}^{(1)} g_{m-2}^{(2)} & g_{m-1}^{(1)} g_{m-1}^{(2)} & \\ & & & \cdot & & & \\ & & & & \cdot & & \\ & & & & & \cdot & \end{bmatrix}$$

$$= \begin{bmatrix} 11 & 01 & 11 & 11 & & & & \\ & 11 & 01 & 11 & 11 & & & \\ & & 11 & 01 & 11 & 11 & & \\ & & & 11 & 01 & 11 & 11 & \\ & & & & 11 & 01 & 11 & 11 \end{bmatrix}$$

Podemos verificar que para a mesma seqüência de entrada $\underline{u} = (1, 0, 1, 1, 1)$, obtemos pela aplicação de G a palavra-código encontrada anteriormente, isto é,

$$\underline{v} = \underline{u} \cdot G$$

$$= (1\ 0\ 1\ 1\ 1) \cdot \begin{bmatrix} 1\,1\,0\,1\,1\,1\,1\,1 & & & & \\ & 1\,1\,0\,1\,1\,1\,1\,1 & & & \\ & & 1\,1\,0\,1\,1\,1\,1\,1 & & \\ & & & 1\,1\,0\,1\,1\,1\,1\,1 & \\ & & & & 1\,1\,0\,1\,1\,1\,1\,1 \end{bmatrix}$$

$$= (11\ ,\ 01\ ,\ 00\ ,\ 01\ ,\ 01\ ,\ 01\ ,\ 00\ ,\ 11)$$

Convém notar que as linhas de G são réplicas das linhas precedentes, porém deslocadas de n-colunas a direita.

FUNÇÃO DE TRANSFERÊNCIA : Como a palavra-código $\underline{v}$ é uma combinação linear das linhas da matriz G, formando portanto um grupo, um código convolucional (n,k,m) é um código linear. Sabemos da descrição da matriz geradora que a operação utilizada para obtenção da palavra-código resultante , $\underline{v}$, do processo de codificação da seqüência de informação $\underline{u}$ é a convolução. Todavia, este procedimento analítico de determinação da palavra--código $\underline{v}$ não é simples. Um procedimento alternativo e bastante simples, vem através do uso de transformadas. É de conhecimento geral que em um determinado domínio a operação sendo realizada é a convolução, então através do uso de transformadas a correspondente operação no domínio transformado será a multiplicação. Com isto ,temos que cada elemento da seqüência de informação $\underline{u}$ pode ser visto como o coeficiente de um polinômio na variável independente D. Conseqüêntemente, a operação de convolução, efetuada anterior mente, tornar-se-á uma multiplicação polinomial sobre GF(2).

Este polinômio recebe o nome de polinômio gerador , quando aplicado a seqüência geradora.

Um polinômio gerador de um código pode

ser determinado diretamente do seu diagrama esquemático, uma vez que podemos associar a cada registro um atrasador (D) de uma unidade de tempo, como também podemos associar a sequência de conexões (1 = conectado , 0 = não-conectado) de um registro a cada saída à seqüência dos coeficientes do polinômio gerador.

EXEMPLO : Para o código (2,1,3) anterior, encontramos os seguintes polinômios geradores :

$$g^{(1)} = 1 + D^2 + D^3$$

$$g^{(2)} = 1 + D + D^2 + D^3$$

Para a seqüência de informação $\underline{u}$ =(1 0 1 1 1), teremos $u(D) = 1 + D^2 + D^3 + D^4$. Conseqüentemente , a palavra codificada será :

$$v(D) = v_1(D^2) + D\, v_2(D^2)$$

$$v(D) = 1 + D + D^3 + D^7 + D^9 + D^{11} + D^{14} + D^{15}$$

que é a mesma obtida no exemplo anterior.

Uma vez que o codificador pode ser visto como um sistema linear com $u^{(i)}(D)$, representando a i-ésima seqüência da entrada, $v^{(j)}(D)$ a j-ésima seqüência de saída , o polinômio gerador $g_i^{(j)}(D)$ pode ser interpretado como uma função de transferência da entrada i para a saída j.

Como em qualquer sistema linear com k-entradas e n-saídas, existe um total de k.n funções de transferência,então a matriz geradora G(D) será dada por :

$$G(D) = \begin{bmatrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \dots & g_1^{(n)}(D) \\ g_2^{(1)}(D) & g_2^{(2)}(D) & \dots & g_2^{(n)}(D) \\ \cdot & \cdot & \cdot & \cdot \\ g_k^{(1)}(D) & g_k^{(2)}(D) & \dots & g_k^{(n)}(D) \end{bmatrix}$$

Como conseqüência, temos $G(D)$ sendo a matriz função de transferência .

<u>REPRESENTAÇÃO MODULAR DE CÓDIGOS DE BLOCO LINEARES</u> : O conceito de representação modular de código de bloco, básicamente estabelece uma forma de determinação da matriz geradora através do conhecimento da distribuição dos pesos de Hamming das palavras código. [3]

Sendo assim, passaremos a nos ater com detalhes sobre este tópico, pois o mesmo é relevante para o processo de resolução do problema proposto nesta tese.

Assim, seja G uma matriz geradora de um código linear (n,k). Seja $rank(G) = k$, desse modo $(q^k - 1)$ possíveis colunas distintas podem ser obtidas tal que seus elementos de colunas constituam os $(q^k - 1)$ tipos ou combinações de q elementos tomados k a k. Se o rearranjo das colunas é irrelevante, um código pode ser descrito por uma lista de número de colunas $\underline{d} = (a_o, \dots, a_{q^k-1})$, onde os $\underline{a}$ s representam as colunas tipo i.

É interessante ressaltar que a matriz

$$K = M^t . G$$

onde t significa transposta, de dimensão $(q^k - 1) \times n$, tem todas as linhas não nulas e são apresentadas como uma combinação linear das linhas da matriz geradora G e, assim, tem todos os vetores

códigos não-nulos, apresentados como linhas. Um caso importante é aquele na qual o código é gerado pela matriz M . Esta então, fornecerá a matriz geradora uma vez que $C = M^t . M$.

Note que C é simétrica e contém uma coluna de cada possível tipo .

No caso de interesse, o corpo de Galois contém somente dois elementos. Assim, uma lista de pesos das (2^k-1) palavras-código não nulas podem ser encontradas como sendo as componentes do vetor $\underline{w}$, isto é,

$$\underline{w} = \underline{a} . C \qquad \text{ou} \qquad \underline{w}^t = C . \underline{a}^t$$

A matriz C considerada como uma matriz de números reais é não singular e, portanto, admite uma inversa. Esta inversa pode ser obtida através da transformação de "0" em "-1" e "1" em "1", e dividindo-se cada elemento por 2^{k-1} . O primeiro passo para se provar isto, é mostrar que duas colunas distintas de C tem elementos "-1" em comum nas 2^{k-2} linhas.

As linhas de C juntamente com o vetor $\underline{0}$ formam um grupo, desde que sejam o espaço das linhas de M .

Considere o conjunto que contém o elemento "0" na i-ésima linha e j-ésima coluna . É facilmente verificado que elas formam um sub-grupo, pois podemos estabelecer quatro sub-matrizes, cada uma com um número igual de elementos, denominados sub-grupos, sendo estas submatrizes formadas pelo conjunto contendo cada linha com "0" na coluna i e "1" na coluna j; o conjunto contendo cada linha com elementos "1" na coluna i e "0" na coluna j; e o conjunto de linhas com elementos "1" tanto na linha i, quanto na coluna j.

Podemos também mostrar que cada coluna contém 2^{k-1} elementos.

Uma vez que a matriz C é simétrica, se multiplicarmos C por ela mesma, os elementos da diagonal terão valor 2^{k-1}, enquanto que todos os elementos fora dela terão valor 2^{k-2}. Então,

$$C^2 = 2^{k-2} (I + J)$$

onde I representa a matriz identidade e J representa a matriz com todos os elementos iguais a 1.

Podemos verificar facilmente que

$$C \cdot J = 2^{k-1} \cdot J$$

$$\text{Como } I = \frac{1}{2^{k-1}} (2C^2 - 2^{k-1} J) =$$

$$= \frac{1}{2^{k-1}} (2C^2 - CJ)$$

então

$$C^{-1} = \frac{2C - J}{2^{k-1}}$$

Assim, se a lista de pesos, na ordem pela qual a palavra-código aparece na equação $\underline{w} = \underline{a}.C$ é dada, o vetor representação modular pode ser encontrado e, conseqüêntemente o código é determinado exceto pelo rearranjo das colunas.

Para um dado conjunto de pesos, é possível determinar a correspondente matriz geradora, se este conjunto puder ser escrito na forma de um vetor $\underline{w}$, de modo que $\underline{a} = \underline{w}.C^{-1}$ apresente componentes que são inteiros não-negativos.

Como uma aplicação, considere o caso em que o código tenha suas palavras com pesos iguais a w_o. Somente um arranjo dos elementos de $\underline{w}$ é possível. O produto $\underline{w}.C$ tem, então, cada componente igual a $w_o.2^{-(k-1)}$, uma vez que cada linha de C^{-1} possui um termo positivo a mais do que o negativo. Como $\underline{a} = \underline{w} . C^{-1}$ precisa ter componentes inteiras, $\underline{w}$ deve ser múltiplo de 2^{k-1}, isto é, $w_o = t \cdot 2^{k-1}$. Desse modo, cada componente de $\underline{a}$ vale t e, o código consiste de t colunas de cada tipo.

EXEMPLO : Seja $\underline{w}$ o vetor peso de um código com a distribuição de pesos dado por $\underline{w}$ = (2,2,4,3,3,3,3). Para tal, iremos adotar que k=3, pois o número de palavras-código iguala a 8.

Seja M, a matriz 3X7 dada por

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

onde as colunas de M são representações binárias das respectivas colunas. Então, a matriz C será dada por

$$C = M^t . M = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

A matriz C^{-1} é dada por :

$$C^{-1} = (1/4) \cdot \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ -1 & 1 & 1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 & 1 \end{bmatrix}$$

Assim,

$$\underline{a} = \underline{w} \cdot C^{-1} = 1/4 \cdot (4,4,0,4,4,4,0)$$

$$= (1,1,0,1,1,1,0)$$

Note que o vetor $\underline{a}$ é constituido por elementos dos tipos 1,2,4,5 e 6. Conseqüêntemente, G será dada por

$$G = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

2.4 - CÓDIGOS DE MEMÓRIA UNITÁRIA

Seja (n,k,m), um código convolucional com taxa $r = k/n$ e m memórias.

Sabemos que um código convolucional $(\tilde{n},k,m)$ pode ser considerado como um código $(\tilde{n} = M.n,\ k= M.k\ )$ com $M=1$, simplesmente por representar

$$G_0' = \begin{bmatrix} G_0 & G_1 & \cdots & G_{m-1} \\ & G_0 & \cdots & G_{m-2} \\ \cdot & \cdot & \cdot & \cdot \\ & & & G_0 \end{bmatrix}$$

e

$$G_1' = \begin{bmatrix} G_m & & & \\ G_{m-1} & G_m & & \\ \cdot & \cdot & \cdot & \cdot \\ G_1 & G_2 & \cdots & G_m \end{bmatrix}$$

Estes dois códigos são equivalentes, pois para uma mesma seqüência semi-infinita de dígitos de informação , produzem a mesma seqüência semi-infinita codificada. Assim, se definirmos a complexidade como sendo M.k , ambos apresentando este mesmo valor, e para uma dada taxa, o valor máximo da distância mínima é obtido para M =1, dentre o sub-conjunto de códigos convolucionais.

Dessa forma, o processo de codificação para tais códigos reduz simplesmente a :

$$v^{(j)} = u^j . G_0 + u^{j-1} . G_1 \text{ , para } j = 0,1,2 \qquad (2.4.1)$$

onde $v^{(j)}$ é a seqüência codificada, u^j é a seqüência de

informação e $v^{(j)} = 0$ para todo $j<0$.

A representação do codificador dos códigos de memória unitária é mostrada na Fig. 2.4.1.

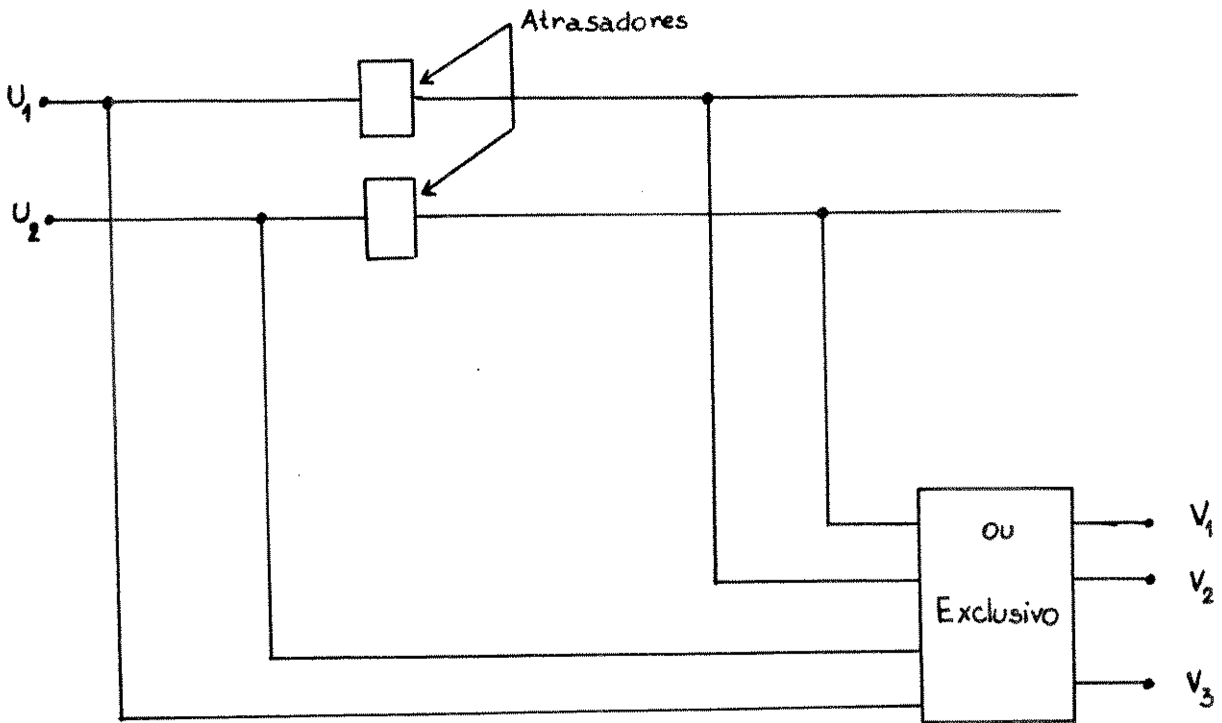


Fig. 2.4.1: Codificador de memória unitária.

Para facilitar o entendimento do problema que iremos tratar, é interessante apresentarmos um exemplo de códigos de memória unitária, haja vista a sua importância.

EXEMPLO : Seja o código de memória unitária (3,2,1) mostrado na Fig. 2.4.2 , abaixo.

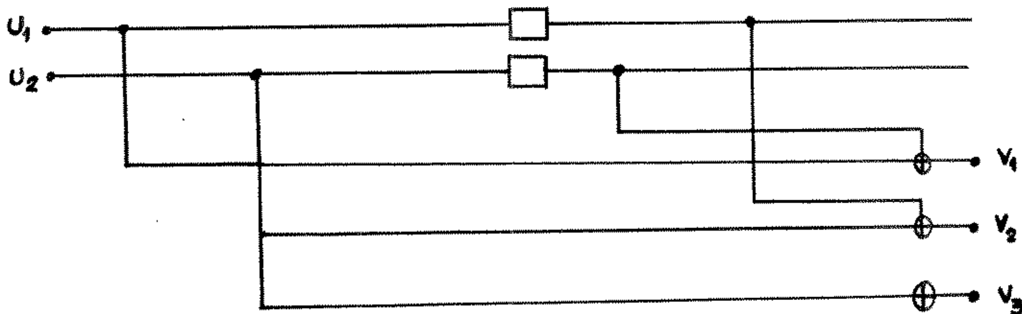


Fig. 2.4.2 : Codificador (3,2,1).

Queremos determinar a distância mínima deste código. Como m=1 e k=2, o número de estados é $2^{mK}=4$,sabemos que o número de transições de cada estado é dado por $2^k=4$. Logo, utilizando a representação de diagrama de estados particionados, teremos para o codificador (3,2,1) a representação mostrada na Fig. 2.4.3 .

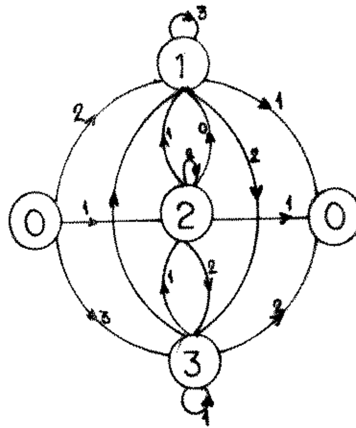


Fig. 2.4.3 : Diagrama de Estados Particionados.

Os valores que aparecem nos ramos de transição entre os estados na Fig. 2.4.3 são os pesos de Hamming da palavra-código ramo. A palavra-código ramo é obtida através das operações indicadas na equação (2.4.1).

Por exemplo, o peso de Hamming correspondente a transição do estado 1 para o estado 3 é, como segue :

$$v_t = [\ 1\ 1\ ] . \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} + [\ 0\ 1\ ] \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$= [\ 0\ 1\ 1\ ]$$

$$\underline{w}(\ v_t\ ) = 2$$

A distância mínima deste código é determinada pela soma dos valores dos ramos de transição saindo do estado zero e retornando ao estado zero.

Como pode ser facilmente constatada, $d_{min} = 2$.

2.5 - ALGORITMO DE DECODIFICAÇÃO DE VITERBI

2.5.1 - INTRODUÇÃO

Se um código convolucional possui restrição de memória pequena e baixas taxas podemos (enumerá-las) decodificar as palavras-código usando o método de gerações sucessivas.

Por outro lado, sabemos que para um código convolucional, com restrição de memória e com taxas altas, a quantidade de cálculos necessárias para enumerar (decodificar) todas as possíveis palavras-código torna-se inviável por enumeração exaustiva.

Assim, alguns códigos com restrição de memória relativamente grandes e taxas $r = 1/n$, com valores de memória em torno de 10, podem ser decodificados com eficiência e, enumerados, usando o algoritmo de máxima verossimilhança, ou seja, o algoritmo de Viterbi.

2.5.2 - DIAGRAMA DE TRELIÇA

Antes de introduzirmos o procedimento de aplicação do algoritmo de Viterbi, faz-se necessário estabelecer a representação do diagrama de estado do codificador evoluindo no tempo, isto é, representar cada janela de tempo com um diagrama de estado particionado. Esta estrutura resultante é denominada "**DIAGRAMA DE TRELIÇA**" .

Para tal, seja

$$G(D) = [\ 1 + D\ ,\ \ 1 + D^2\ ,\ \ 1 + D + D^2\ ]$$

a matriz função de transferência do código (3,1,2) e U(D) uma seqüência de informação de comprimento L=5 sobre GF(2).

Para este caso, temos que o diagrama de treliça contém (L+m+1) níveis ou janelas de tempo, enumerados de 0 a L+m, onde L é o comprimento da seqüência de informação e m , o número de memórias .

Assumindo que o codificador sempre inicializa a codificação estando no estado zero e para ele retorna, as primeiras m+1 janelas de tempo correspondem à saída e retorno do codificador ao estado zero. Como nem todos os estados podem ser pesquisados nas primeiras m ou nas últimas m janelas de tempo,somente a parte central da treliça é que possibilitará que todos os estados intermediários sejam possíveis de serem pesquisados.

A Fig. 2.5.2.1 apresenta o diagrama de treliça para o código (3,1,2). Note que existem dois ramos deixando e entrando em cada estado. O ramo superior deixando cada estado na janela de tempo i, representa a entrada $u_i = 1$, enquanto que o ramo inferior, representa a entrada $u_i = 0$. Cada ramo é rotulado com as n saídas V_i correspondentes, e cada uma das 2^L palavras-código de comprimento $N = n(L+m)$ é representada por um caminho único através da treliça.

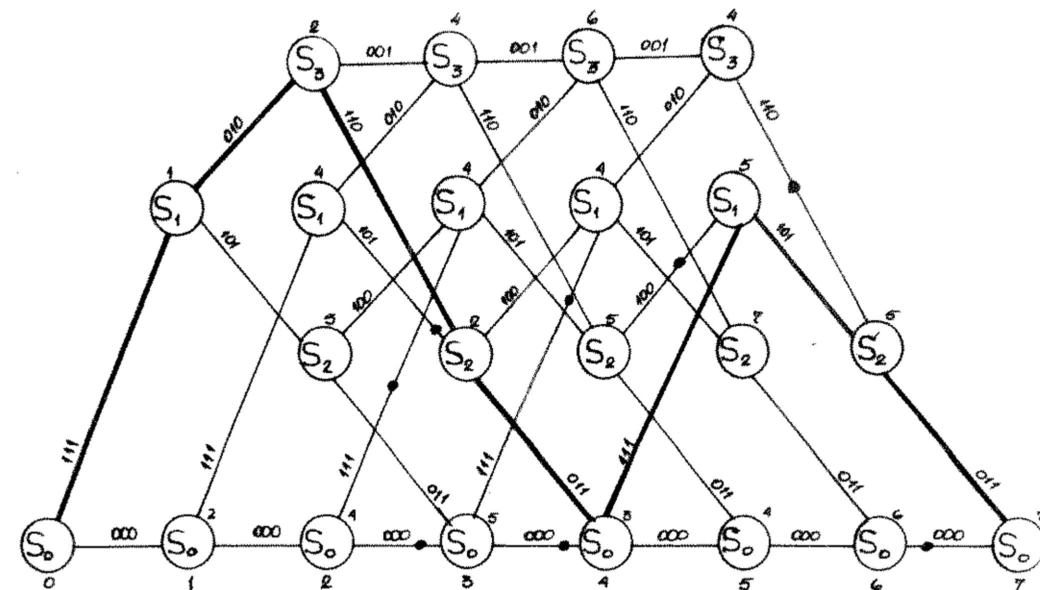


Fig. 2.5.2.1 : Diagrama de Treliça do Código.

Em geral, para um código (n,k,m), e uma seqüência $\underline{u}$ de informação de comprimento k.L, existem 2^k ramos deixando e entrando em cada estado, e 2^{KL} caminhos distintos através da treliça correspondendo a 2^{KL} palavras-código.

Como veremos, no item 2.5.3 , a treliça montada desta forma, possui uma aplicação deveras importante, qual seja, a de encontrar a palavra-código com a menor distância mínima.

2.5.3 - O ALGORITMO DE VIRTERBI

Um codificador de máxima verossimilhança para um determinado tipo de canal sem memória, escolhe como palavra-código $\underline{v}$, aquela que maximiza o log da função

probabilidade de transição do canal. Tal função recebe o nome de MÉTRICA associada ao caminho v.

O algoritmo de Viterbi, encontra o caminho através da treliça, com a maior métrica, isto é, o caminho com a máxima verossimilhança, por isso é conhecido também pelo nome de DECODIFICADOR DE MÁXIMA VEROSSIMILHANÇA .

Seja $R = (r_0 , r_1 , \ldots , r_{L+m-1})$ uma seqüência recebida. O algoritmo de Viterbi processa essa seqüência de modo iterativo. A cada passo ele compara a métrica de todos os caminhos entrando em cada estado e armazena a maior métrica ou métrica acumulada .

Ao caminho associado a esta maior métrica acumulada, atribui-se o nome de SOBREVIVENTE.

Para um código (n,k,m) , sobre $GF(2)$, existem 2^k sobreviventes a partir da m-ésima janela de tempo até a L-ésima janela de tempo: cada um dos 2^K estados. Depois de L existem poucos estados quando o codificador está retornando para o estado zero. Finalmente, na janela de tempo L+m, existe somente um estado, o estado zero, e assim, somente um sobrevivente. Dessa forma, o algoritmo termina, pois apresenta o caminho de máxima verossimilhança.

No caso especial de um canal binário simétrico (BSC) com probabilidade de transição $p < 1/2$, a seqüência recebida, R , é binária e a função log-probabilidade é dada por

$$\log P(R/V) = d(R,V) \cdot \log \frac{p}{1-p} + N \log(1-p)$$

onde $d(R,V)$ é a distâcia de Hamming entre R e V. Como $\log p/(1-p)$ é menor que zero e $N \cdot \log(1-p)$ é constante para qualquer V, um decodificador de máxima verossimilhança, para um canal BSC, escolherá V como palavra-código v , que é o estimador da palavra-código V produzido pelo codificador baseado na seqüência recebida R, que minimiza a distância de Hamming :

$$d(R,V) = \sum_{i=0}^{L+m-1} d(R_i , V_i) = \sum_{i=0}^{N-1} d(r_i , v_i)$$

Assim, na aplicação do algoritmo de Viterbi o canal BSC, temos que distância de Hamming $d(R_i , V_i)$ vem a ser a métrica do ramo, enquanto que $d(r_i, v_i)$ vem a ser a métrica do bit. Desse modo , o algoritmo precisa encontrar o caminho na treliça que fornece a menor métrica acumulada.

2.5.4 - EXEMPLO DE APLICAÇÃO PARA O CANAL BSC.

Assuma que a palavra-código no diagrama de treliça do código convolucional (3,1,2) , apresentado no ítem 2.3.3 (cuja sequência é $\underline{u}$ = (1 0 1 1 1)) , é transmitida através de um canal BSC e que a seqüência recebida seja

R = (110 , 110 , 110 , 111, 010 , 101 , 101)

O caminho de máxima verossimilhança apresentado na Fig. 2.5.2.1, através das linhas cheias, é dado por

$\underline{v}$ = (111 , 010 , 110 , 011 , 111 , 101 , 011)

A correspondente seqüência de informação decodificada é dada por

$\underline{u}$ = (1 1 0 0 1)

Se o caminho de máxima verossimilhança na treliça possui uma métrica acumulada igual a 7 , significa dizer que nenhum outro caminho através da treliça difere de R em menos que 7 posições.

Isto indica uma amarração no valor da métrica dos dois caminhos entrando naquele estado. Se o sobrevivente final percorre qualquer um destes estados, existe mais do que um caminho de máxima verossimilhança cujas correspondentes distâncias de Hamming são idênticas. Quando isto

ocorre, qualquer um dos caminhos é selecionado, arbitráriamente, como sobrevivente. Esta arbitrariedade não afeta a probabilidade de erro de decodificação.

CAPITULO 03

CAPÍTULO 3

PROGRAMAÇÃO DINAMICA E KNAPSACK COMBINATORIAL

3.1 - INTRODUÇÃO

A Pesquisa Operacional pode ser entendida como o conjunto de intersecção entre a arte e a ciência, pois engloba em seu contexto a arte de modelar matemáticamente determinada situação de forma bem definida e a ciência de deduzir os métodos computacionais mais convenientes para o modelamento proposto. Assim sendo, pode-se entende-la também, como uma "tradutora" de um problema formulado verbalmente, em um problema de programacao matemática que procurará apresentar a melhor solução possível, levando-se em conta todos os detalhes que delimitam o conjunto das possíveis soluções, também denominado conjunto das restrições do modelo.

Dessa forma, a Pesquisa Operacional apresenta-se através de técnicas que tentam otimizar uma determinada situação.

Dentre as possíveis técnicas de programação proposta pela Pesquisa Operacional, encontram-se duas que tomarão lugar de destaque dentro deste trabalho, a saber a programação dinâmica e a programação inteira (knapsack) , que passarão a mostrar sua importância na aplicação a códigos de memória unitária.

Dessa forma, apresentaremos neste Capítulo, noções gerais de Programação Dinâmica e Programação Inteira , bem como exemplos, para que induza o leitor a conveniência de suas aplicações no problema proposto nesta tese.

3.2 - PROGRAMAÇÃO DINAMICA

Como mencionado anteriormente, a Programação Dinâmica , é uma das possíveis técnicas de otimização dentro da pesquisa operacional .

A programação dinâmica poderia ser entendida como uma técnica matemática de aproximação a otimização, assim como indução matemática é uma aproximação a demonstração de teoremas.

A idéia principal da programação dinâmica, consiste em relacionar uma combinação de decisões a serem tomadas que estejam inter-relacionadas entre si e que, não só otimizem o problema na sua totalidade, mas como em qualquer parte do mesmo.

Sua maior dificuldade consiste no fato de não possuir uma estrutura padrão de formulação, isto é, cada problema recebe um equacionamento específico para sua situação individual. Ou seja, dificilmente encontra-se uma formulação matemática reaproveitável para outro problema, sem que a estrutura básica deste seja equivalente ao do problema já equacionado.

Talvez resida neste fato, a maior dificuldade na elaboração da formulação matemática de um problema por programação dinâmica; ou na conveniência, ou não, em aplicá-la ao problema que se deseja resolver. Dessa forma, torna-se necessária uma familiaridade maior com essa técnica, que só poderá ser alcançada através dos estudos de casos já solucionados pela mesma, desenvolvendo-se assim, o conhecimento necessário para se decidir quando sua aplicação será conveniente.

Existe, contudo, um problema protótipo da utilização da programação dinâmica, chamado "problema da diligência". Este problema, em especial, possui a mesma estrutura básica referida anteriormente. Ao comparar a estrutura de um novo problema, deve-se verificar a existência, ou não, de semelhanças. Caso as estruturas sejam análogas, pode-se fazer uso da programação dinâmica , mas como já salientado, com a formulação totalmente especifica para o novo caso em questão.

A programacao dinâmica, quando passível

de aplicação, fornece com grande eficiência a solução ótima desejada, com muito menos esforço computacional, comparada ao método da enumeração exaustiva, pelo fato de começar com uma pequena parte do problema, encontrar a otimalidade para esta parte ("ótimo parcial"), ampliar o problema e a partir do ótimo encontrado anteriormente, encontrar o novo ótimo parcial. Este processo é repetido até que o problema original seja resolvido por completo. Uma vez que esta técnica inicia através de ótimos parciais, a verificação da condição de piores soluções certamente não é considerada e, assim, a quantidade de buscas até se alcançar o ótimo total torna-se bem inferior a enumeração exaustiva , diminuindo, dessa forma, o tempo de computação.

A programação dinâmica pode ser dividida em programação dinâmica determinística e programação dinâmica probabilística.

A seguir procuraremos detalhar as afirmações efetuadas nesta introdução, através de exemplos ilustrativos com o objetivo de elucidar e encaminhar para a solução do problema
a ser apresentado.

3.2.1 - ESTRUTURA BASICA DO PROBLEMA DE PROGRAMAÇÃO DINAMICA

Todo problema de programação dinâmica, a fim de permitir ser caracterizado como tal, deve contar com a seguinte estrutura básica :

1. Estágios ;
2. Estados ;
3. Decisões ;
4. Política do problema;
5. Relação recursiva que identifica a política ótima.

Entende-se por estágio, as etapas seqüênciais necessárias para definir o processo como um todo, ou seja , cada parte do problema.

As opções para se completar os estágios são chamadas decisões, isto é, dentro de cada estágio conta-se com algumas opções de escolhas; estas possíveis escolhas são as possíveis decisões.

Entende-se por estado, a condição do processo num dado estágio, assim, cada estágio tem um número de estados associados a ele.

Uma política , deve ser entendida como uma seqüência de decisões - uma decisão para cada estágio do processo - cujo principal efeito é o de transformar o estado atual em um estado associado ao estágio seguinte. Dessa forma, é a política quem fornece a transição de estágio a estágio, através dos estados encontrados de modo conveniente. A política ótima, tomada no estágio atual, é encontrada através de uma relação recursiva que faz uso da política ótima encontrada no estágio precedente. Deve-se salientar o fato de que a busca da política ótima do problema, tem início a partir da política ótima encontrada para o último estágio e retrocede, desse modo, até alcançar a política ótima no primeiro estágio, caracterizando, portanto, a política ótima total desejada.

Vê-se , portanto , que o conhecimento do estado atual do problema, fornece toda a informação sobre o comportamento dos estágios já conhecidos do sistema que é necessária para a determinação da política ótima no estágio posterior. Ainda mais, dado o estado atual, verifica-se que a política ótima para os estágios posteriores independe da política adotada nos estágios anteriores, isto é, ela é fornecida exclusivamente com relação a situação encontrada no estágio atual.

Resumindo : O método da programação dinâca a partir da estrutura básica já caracterizada acima, parte do último estágio de um problem de N estágios e determina a política ótima parcial através da relação recursiva, que a identifica, para deixar o estado atual. Uma vez encontrada a política parcial para o estágio N, repete-se o processo a partir

do estado ótimo encontrado para esse estágio, mas com relação ao estágio seguinte, assim, em cada estágio, determina-se a melhor política para deixar cada estado e completar o processo, supondo-se que todos os estágios precedentes foram concluídos, utilizando-se os resultados obtidos para o estágio seguinte.

Em termos de notação caracterizaremos :

x_n - variável de decisão do estágio n (n=1, ..., N)

$f_n(s, x_n)$ - valor ótimo da função objetivo, dado que o sistema começa no estado s do estágio n e x_n é escolhido.

$f_n^*(s)$ - valor ótimo da $f_n(s, x_n)$ sobre todos os possíveis valores de x_n .

Para cada um dos N estágios , começando-se pelo último, monta-se o seguinte quadro apresentado na Fig.3.2.1.1.

x_n / s	$f_n(s, x_n)$ = relação recursiva conveniênte	$f^*_n(s)$	x^*_n

Fig 3.2.1.1 - Quadro genérico de estágio.

Note que a primeira coluna fornece o estado ,s, em que se está atualmente;a segunda coluna fornece as possíveis alterações efetuadas de acordo com a relação recursiva , quando se passa do estado s para o x_n ; a terceira coluna fornece o melhor valor de $f_n(s)$ levando-se em conta o objetivo do problema; e, finalmente, a quarta coluna, apresenta os estados posteriores ao estado s atual, que fornecem como valor ótimo $f^*_n(s)$.

Preenchido o último quadro, isto é, ao se alcançar o estágio 1, a política ótima do problema está definida e, portanto, o problema proposto já possui uma , ou mais, soluções.

A decisão inicial, x^*_n , é portanto, conhecida. Uma vez definida a decisão no estágio atual, deve-se retornar ao quadro do estágio anterior para encontrar a próxima decisão. Para tal, toma-se o valor obtido na decisão ótima do estágio anterior. Este passará a ser o estado s do quadro do atual estágio . Segue-se, através do quadro, a linha associada ao já referido s , e encontra-se , a nova decisão associada agora a esse novo estágio. Este processo é repetido até que a decisão do último estágio seja definida.

Evidentemente, pode-se encontrar duas ou mais decisões ótimas dentro de um mesmo estágio. Neste caso, deve--se considerar cada uma delas isoladamente a partir deste ponto , pois fornecerão soluções ótimas alternativas, isto é, soluções diferentes sem, contudo,alterar o valor ótimo total encontrado.

3.2.2 - EXEMPLO PROTÓTIPO DE PROGRAMAÇÃO DINÂMICA

Com a finalidade de dirimir possíveis dúvidas provenientes da notação que caracterizam o modelo sob o ponto de vista da técnica de programação dinâmica, apresentaremos nesta seção um exemplo protótipo conhecido na literatura de Pesquisa Operacional por " problema da diligência ". Desse modo tentaremos, também, deixar de forma clara e objetiva, os passos efetuados estágio-a-estágio para que a sistemática da programação dinâmica seja assimilada e melhor entendida pelo leitor menos familiarizado com o método.

Descrição do problema da diligência :

O problema da diligência parte da situação hipotética, onde retornando ao velho oeste, um carteiro tem de entregar correspondência através de territórios habitados por índios não muito amistosos, saindo de um determinado território e entregando a correspondência em um território fixado, isto é, tanto o seu território de saída quanto o de chegada são fixados.

Para tal serviço, ele utiliza como meio de locomoção de um território a outro a diligência. Esta diligência não só presta serviços ao correio, como também às pessoas interessadas em deslocar-se dos territórios por onde elas passam. Visto que a viagem apresenta um certo risco de vida, devido aos possíveis ataques indígenas, foi elaborada uma política de seguro de vida para os passageiros. Notando que evidentemente a rota com menor custo de política de seguro de vida era a que apresentaria menor perigo aos passageiros, o carteiro logo deduziu que este deveria ser o seu itinerário na entrega da correspondência, visto que ele não necessita necessáriamente passar por tódos os territórios.

A Fig.3.2.2.1 ilustra um "mapa" dos territórios a serem atravessados, com os respectivos custos de políticas de seguro de vida associados a cada rota.

O carteiro quer saber, portanto, qual a rota que minimiza o custo total da política de seguro de vida.

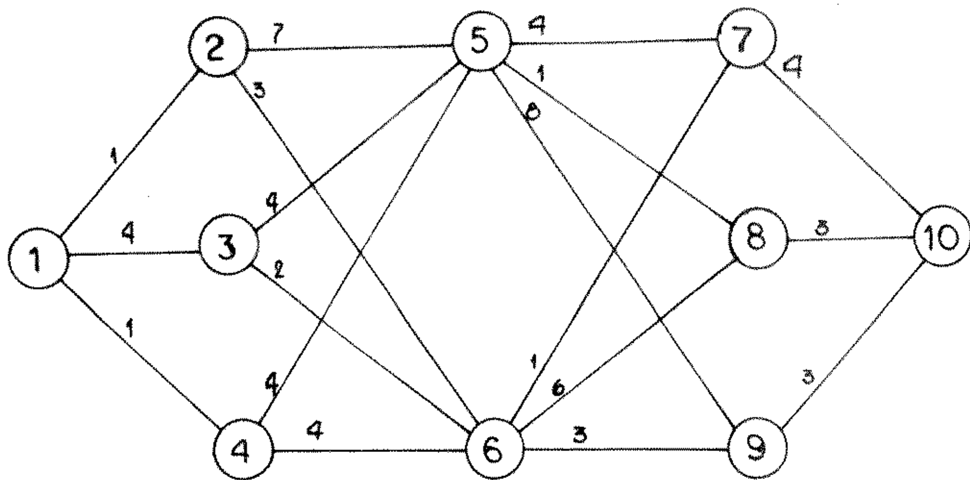


Fig. 3.2.2.1- Territórios do Problema da Diligência com respectivos custos de seguros de vida.

Temos, portanto, que determinar a melhor rota para sair do território 1 e chegar ao território 2.

Solução :

Estrutura do Problema :

1. Estados : Cada um dos territórios a ser escolhidos.(1 a 10)

2. Estágios : Cada uma das etapas da viagem até que o objetivo (território 10) seja alcançado. (1 a 4)

3. Decisões : A escolha de qual território escolher como "parada" em cada etapa da viagem.

4. Política : Escolher o caminho com menor custo associado.

5. Recursividade: O custo total no estágio atual deve ser a soma dos menores custos dos estágios antecedentes somado ao menor custo do estágio atual.

A seguir estabeleceremos a notação a ser utilizada neste exemplo.

s - estado atual ($s = 1, \dots, 10$)

x_n - variável que decide qual o destino imediato a ser tomado após o estágio n ($n = 1,2,3,4$);

$f_n(s,x_n)$ - custo da melhor política total encontrada para os estágios restantes, dado que o carteiro esta no estado s e seleciona x_n como detino imediato .

x_n^* - valor de x_n que minimiza $f_n(s, x_n)$.

$f_n^*(s)$ - valor mínimo correspondente a $f_n(s, x_n)$.

$c_{s\,x_n}$ - custo da política para ir de s a x_n .

Dessa forma, a relação recursiva pode ser dada por :

$$f_n(s, x_n) = c_{s\,x_n} + f_{n+1}^*(x_{n+1})$$

3.2.3 - TIPOS DE PROGRAMAÇÃO DINAMICA

Muito embora, a estrutura básica da programação dinâmica já tenha sido desenvolvida, torna-se imperativo descrever, mesmo que brevemente, os dois tipos em que a mesma pode ser apresentada : programação dinâmica determinística e programação dinâmica probabilística .

A programação dinâmica determinística caracteriza-se pelo fato do estado no próximo estágio ser completamente determinado pelo estado e pela política de decisão do estágio atual.

A idéia aqui, pode ser melhor apresentada através do diagrama mostrado na Fig. 3.2.3.1.

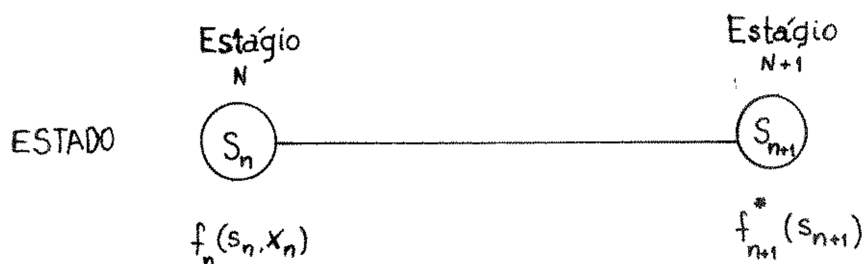


Fig. 3.2.3.1 - Diagrama do Problema de Programação Dinâmica Determinístico

Vemos, portanto, que se o resultado de cada decisão for conhecido exatamente, temos um Problema de Programação Dinâmica Determinística.

Apenas para completar a idéia, o problema de controle de estoques, quando focalizado através da Programação Dinâmica é Determinístico.

Não obstante, o problema de programação dinâmica probabilístico, difere do Problema de Programação Dinâmica Determinístico, pelo fato de que o estado no próximo estágionão é completamente determinado pelo estado e pela política de decisão do estágio atual.

No problema de Programação Dinâmica Probabilístico existe uma distribuição de probabilidade, para ser utilizada, a fim de que o próximo estado seja conhecido. Contudo, esta distribuição de probabilidade continua sendo completamente determinada pelo estado e pela política de decisão do estágio atual.

Apresentamos na Fig. 3.2.3.2, um esquema da idéia geral do Problema de Programação Dinâmica Probabilístico.

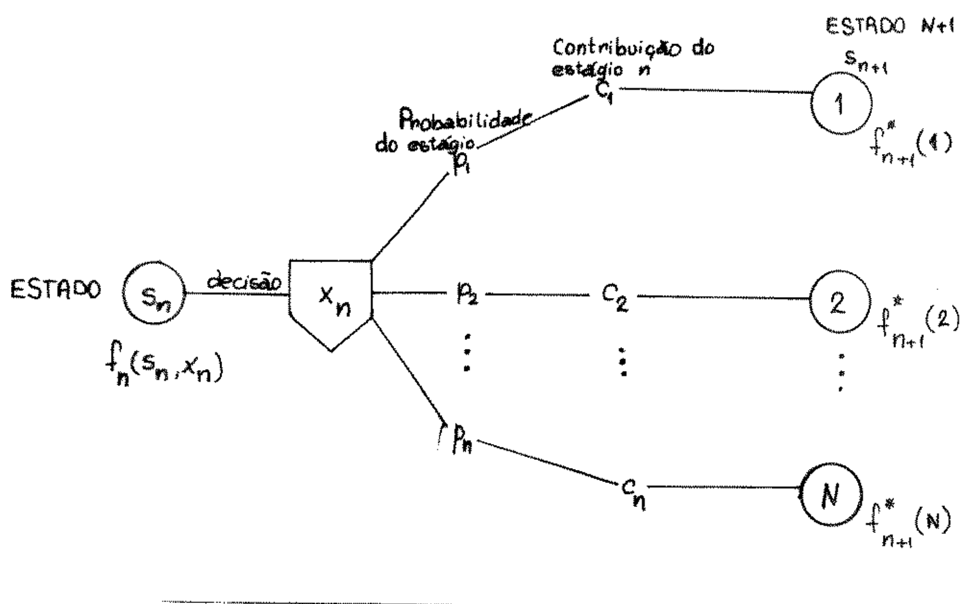


Fig. 3.2.3.2 - Diagrama do Problema de Programação Dinâmica Probabilístico.

Quando o diagrama acima é completado de modo a incluir todos os possíveis estados e decisões, ele é acrescido significativamente de tamanho e é chamado de árvore de decisões, por permitir a visualização de todos os caminhos a serem tomados. Assim, se a árvore de decisões for muito extensa, torna-se inviável a tentativa de visualizar as possíveis soluções através da mesma.

Devido a estrutura probabilística, a complexidade da relação entre as variáveis $f_n(s_n , x_n)$ e a $f^*_{n+1}(s_{n+1})$ é maior do que a complexidade entre as mesmas variáveis no problema de programação dinâmica determinístico .

Todavia, tanto no problem de Programação Dinâmica Determinística, quanto no problema de Programação Dinâmica Probabilística, é muito usual querer determinar uma seqüência inter-relacionada de decisões que requerem uma formulação recursiva apropriada, relacionada a cada problema individual, necessitando de um esforço computacional muito menor do que na enumeração exaustiva.

Apenas para efeito de comparação, se o problema tem 10 estágios e 10 possíveis decisões em cada estágio, a enumeração exaustiva considera as 10^{10} combinações, enquanto que a Programação Dinâmica necessita não mais de 10 cálculos.

3.3 - KNAPSACK COMBINATORIAL .

Como mencionado anteriormente, a Pesquisa Operacional conta com diversas técnicas que procuram descrever o problema considerado através do modelamento matemático mais conveniente.

Quando ao modelar-se o problema, percebe-se que ele é constituído de um conjunto de restrições lineares, cujas variáveis são necessáriamente não negativas, onde a possível solução deva maximizar ou minimizar uma equação também linear regida pelo objetivo do problema proposto, diz-se que formulou-se um problema de Programação linear .

Naturalmente, na prática, encontramos situações onde o resultado de uma otimização deva fornecer valores inteiros, como no caso em que devemos encontrar o número ótimo de funcionários para operarem determinadas máquinas da seção de uma fábrica, por exemplo. Torna-se inviável, portanto, uma solução como "um homem e meio" para operar na máquina do tipo "x".

Através da necessidade de contornar-se eventualidades como esta, os problemas formulados por Programação Linear receberam o acréscimo de restrições que forçam o aparecimento de soluções inteiras.

Um problema de programação linear com apenas algumas variáveis inteiras (o modelo pode apresentar algumas variáveis não-inteiras) recebe o nome de programação inteira

mixta . Entretanto, se todas as variáveis devem ser necessariamente inteiras, temos um problema de programação inteira .

Quando, dentro do contexto da Programação Inteira, temos um problema com uma restrição principal e as demais, necessariamente sendo com relação a positividade dos coeficientes, ele recebe o nome especial de problema knapsack ou problema da mochila .

Certos problemas de programação inteira, como comprovado por Dantzig [ ], podem ser resolvidos como problemas de programação linear, como é o caso do Problema de Designação, Transporte e Fluxo Máximo. Entretanto, quando a solução encontrada não apresentar as soluções inteiras necessárias, pode-se aproximar as soluções a partir das já encontradas. Os métodos de aproximações mais conhecidos são :

a) Técnica do Plano de Corte (Gomory);

b) Métodos Enumerativos;

c) Algoritmos de Particionamento (Benders);

etc ...

Os méritos de cada um dos métodos acima não serão levados em considerações detalhadas, entretanto,apresentamos abaixo, uma breve citação, de modo bem geral, a fim de darmos uma idéia do procedimento de cada um deles.

a) Técnica do Plano de Corte : Este método tem como objetivo principal,deduzir restrições suplementares a partir das restrições de integralidade e apresentar a solução do problema. A cada iteração, uma nova restrição convenientemente selecionada da a partir dos resultados obtidos na iteração anterior, é acrescida ao modelo, na tentativa de obter um resultado ótimo e inteiro.

O Método de Gomory, ou Algoritmode Plano

de Corte, é o mais utilizado atualmente. Também são conhecidos os algoritmos desenvolvidos por Glover e Young, chamados Algoritmo Primal de Plano de Corte.

É importante notar que uma vez que aumentamos uma nova restrição a cada iteração, na tentativa de encontrarmos o ótimo, o sistema aumenta consideravelmente de tamanho, o quepode vir a inviabilizar a aplicação do método.

b) Métodos enumerativos : Neste método, todas as possíveis soluções aparecem representadas implícitamente ou explicitamente. Como o próprio nome já diz, o método enumera todas as possíveis "candidatas" a solução. Novamente podemos perceber que dependendo do tamanho do sistema ele se tornará inviável.Este método é muitas vezes denominado de "Algoritmo de Ponto ou Nó" e sua representação esquemática é através de uma árvore de soluções.

c) Algoritmos de Particionamentos : Este algoritmo consiste em transformar um problema de programação inteira mixta em um problema de programação inteira. Devido ao grande número de restrições que precisam ser geradas e que dificulta, portanto, a resolução da formulação para o caso "inteiro", este algoritmo resolve o problema inteiro equivalente ao problema mixto através da resolução de uma série de problemas lineares e inteiros . Devemos ressaltar que os méritos desse método cabe a Benders.

Através da breve descrição de alguns métodos de Programação Inteira, pode-se deduzir que a carga computacional implícita em cada acréscimo de restrição, torna o custo operacional muitas vezes significativo, dependendo do porte do sistema em questão. Dessa forma, faz-se conveniente obter as possíveis soluções através de aproximações que otimizem o valor da função objetivo. Tal problema, torna-se conhecido pelo nome de Técnica de otimização combinatorial .

Tendo em vista os detalhes do problema, pode-se encontrar uma representação através de uma técnica de otimização combinatorial para um problema de programação inteira, mesmo que complexo.

Ainda que se consiga explicitar o conjunto

das possibilidades de resolução do problema, o esforço computacional dispendido não justificaria o emprego do método de otimização combinatorial em alguns casos. Novamente nos deparamos com a necessidade de conhecer uma técnica que enumere parcialmente um número controlável de possibilidades e que, implicitamente, todas as outras soluções, de uma maneira sistemática, sejam verificadas, evitando-se seguir caminhos que certamente não levarão a soluções convenientes. Tal técnica recebe o nome de "Backtrack".

Dentre as técnicas de backtrack encontra-se uma que deixaremos abordada de modo especial, por saber ser eficiente numa aplicação futura deste trabalho, denominada "BRANCH-AND-BOUND" [5] .

O Branch-and-Bound é uma técnica de enumeração implícita e explícita : explícita, porque enumera cada uma das possibilidades e resolve, sucessivamente, o problema linear resultante e, implícita, porque de acordo com as restrições de integralidade, positividade e outras relacionadas ao modelo, são verificadas todas as possibilidades de soluções de modo "inteligente", os caminhos que certamente conduzirão a soluções desapropriadas.

As seções que se seguem destinam-se a uma explicitação do método Backtrack relatado acima, contudo voltada sempre para o problema de programação inteira do tipo knapsack. Assim, como aplicamos o método de Backtrack, que é um método de Otimização Combinatorial, para resolver o problema do knapsack , dizemos que temos em mãos um knapsack combinatorial .

3.3.1 - O PROBLEMA KNAPSACK.

O nome Knapsack vem da situação hipotética de um viajante que leva para sua viagem somente uma mochila ("knapsack") carregando seus pertences.

É fácil deduzir que o peso de cada elemento a ser colocado na mochila, bem como o seu tamanho , devem ter significados importantes na escolha do número de vezes que este elemento aparecera dentro da mochila.

Certamente, o viajante desejará carregar

consigo a máxima quantidade de diferentes ítens com peso total menor que o suportado pela mochila (ou pelo próprio viajante), que deverá ser fornecido préviamente.

Assim, o objetivo é determinar quais itens ,e quantas unidades dos itens escolhidos,serão colocados na mochila.

Em termos de notação temos :

b - peso total limitante;

a_j - peso do j-ésimo ítem;

c_j - valor do j-ésimo ítem;

x_j - número de itens do tipo j a ser colocado na mochila.

Formalizando-se matemáticamente o problema de Programação Inteira do tipo Knapsack temos :

Problema :

$$\text{máx} \sum_{j=1}^{n} c_j \cdot x_j$$

sujeito a :

$$\sum_{j=1}^{n} a_j \cdot x_j \leq b$$

$$x_j \geq 0 \text{ e inteiro } (j=1, 2, \ldots, n)$$

Pode-se perceber que o problema Knapsack definido acima, pode ser visto também como um Problema 0/1 , ou

seja, leva o ítem j ($x_j = 1$), ou não leva ($x_j = 0$).

Dessa forma, embora seja um problema enunciado facilmente, não deve ser visto com descaso quanto a aplicações : situações industriais, problemas de investimento de capital, problemas de cargas, cortes de madeira (folhas de papel ou metálicas), etc ... [5] são alguns dos problemas encontrados em situações reais.

Existem muitos métodos para resolver um problema knapsack e muitos que também explicam a metodologia necessária para reduzir um problema inteiro em um problema Knapsack. [5] [3]

Entretanto, como já foi comentado, procura se resolver o Knapsack através de um método de otimização combinatorial.

3.3.3 - TÉCNICA DO BACKTRACK

Muito embora este trabalho faça uso apenas de backtrack faz-se conveniente apresentar algumas peculiaridades inerentes a mesma, a fim de que se compreenda sua conveniência nesta aplicação.

Trata-se o Backtrack de uma técnica de otimização combinatorial, que busca a forma sistemática, ou freqüêntemente enumerativa, de soluções para o problema apresentado.

Por tratar-se de uma técnica enumerativa uma melhor visualisação de seu procedimento de busca é através de uma árvore.

Para melhor compreensão , apresentamos a árvore a seguir.

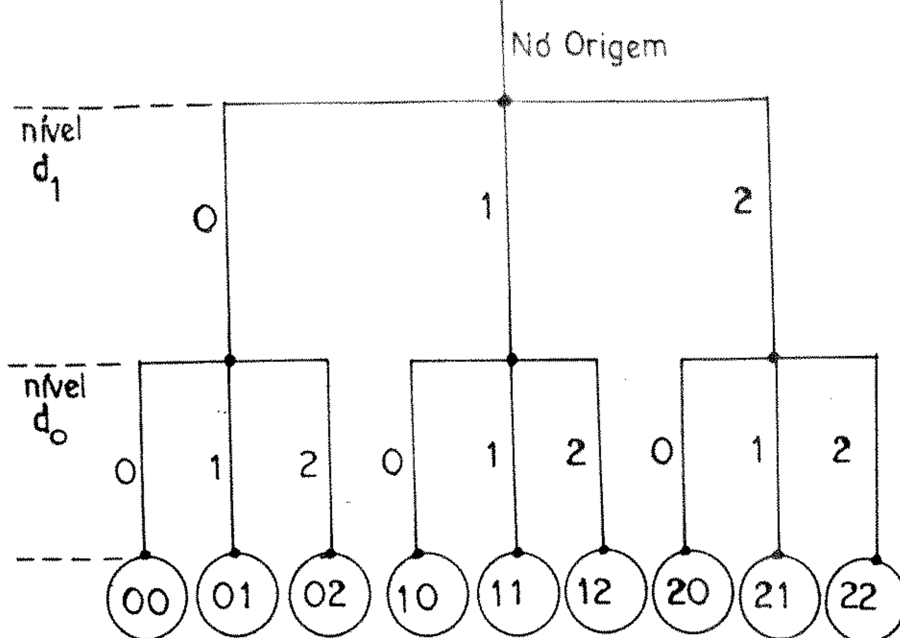


Fig. 3.3.3.1 : Árvore geradora de dígitos na base três.

Vemos que a árvore acima gera números de dois dígitos na base três. A escolha de cada dígito é feita através da opção escolhida em cada nível d_j (j= 0, 1). Uma vez escolhido o dígito do nível d_j , encontramos uma sub-árvore de escolhas para o dígito a ser escolhido no novo nível d_{j-1} .

No caso acima, possuímos uma árvore com uma estrutura uniforme, isto é, o número de ramos que saem de qualquer um dos nós, em um mesmo nível, são iguais.

Na Fig. 3.3.3.1, todas as possíveis soluções são apresentadas ao fim do nível d_o (no caso de árvore uniforme). Caso não tivéssemos uma árvore uniforme, como a apresentada na Fig. 3.3.3.2 abaixo ,

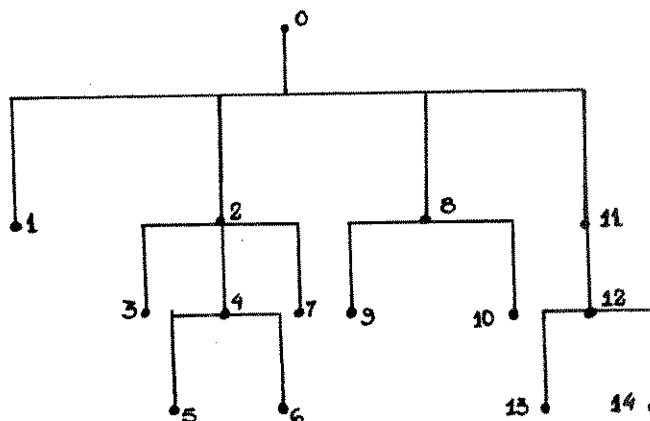


Fig. 3.3.3.2 : Árvore não uniforme .

escolhemos em cada nó, o próximo ramo a ser examinado. Caso não estejamos no último nível da árvore apresentada, mas o ramo escolhido no nível anterior tenha nos conduzido a um nó que não possua mais derivações de ramos , nó terminal , temos uma solução e, portanto, depois de armazenado este resultado retrocedemos de forma a retornar ao nó do nível anterior a partir do nó que nos conduziu ao nó terminal do nível atual. Uma vez retornado ao nível anterior, verificamos outro ramo que emana daquele nó e, assim, fazemos novas verificações de soluções em todos os ramos dos níveis posteriores.

Vê-se dessa forma, que o Backtrack ocorre sempre que um nó terminal é encontrado e deseja-se voltar ao nível anterior.

O Backtrack , enunciado da forma comentada acima, recebe o nome de " busca normal ".

Convém salientarmos, ainda, que o tempo necessário para fazer a procura na árvore , é proporcional ao número de nós na árvore, portanto, este é o principal problema a ser cuidadosamente estudado, quando utilizamos um algoritmo Backtrack. Se não pudermos contar com um computador de alta velocidade, esta busca exaustiva, torna-se impraticável, isto é, se possuirmos uma árvore com n-níveis e r-ramos saindo de cada nó, teremos r^n nós terminais.

Contudo, de acordo com as necessidades do problema que temos em mãos, podemos evitar a verificação de ramos que certamente conduzirão a nós terminais inconvenientes, isto é, de acordo com nossas conveniências efetuar o Backtrack , e portanto, a volta ao nível anterior sem atingir o nó terminal a que o ramo atual conduziria. Assim, o número de buscas diminuem consideravelmente e a utilização da técnica torna-se convenientemente aplicável. Chamamos este aspecto de busca de " deteção de impasse " . [7]

Atualmente, o algoritmo de Backtrack incorporando deteção de impasse é freqüentemente o mais direto e eficiente método de aproximação ao método enumerativo.

3.3.3.1 - BRANCH- AND- BOUND.

A técnica do Branch-and-Bound trata-se de uma variação especial da técnica de Backtrack, que tem por finalidade a resolução de um conjunto de problemas de programação linear.

Assim, um procedimento de resolução de um problema de programação inteira do tipo Knapsack, seria por intermédio da enumeração do método Branch-and-Bound.

Tratando-se de um método enumerativo, uma melhor visualização seria através do estabelecimento da árvore de possibilidades.

A idéia principal deste método, reside no fato de não ramificar para um nó solução cujo custo seria mais alto que o da solução com custo mínimo, determinada anteriormente.

Convém salientarmos o fato de que se uma solução mais otimizada for encontrada, o custo mínimo é atualizado , assim como os limitantes, para novas propostas de soluções. Também é importante comentar que o custo de um nó que deu origem a outras ramificações, também chamado de "nó pai", é sempre menor ou igual ao custo de seus "nós filhos", que seriam os nós ramificados nó pai. Desse modo impedimos buscas desnecessárias caso o custo mínimo atual, quando comparado ao novo valor encontrado para um novo nó, seja menor.

Como o Branch-and-Bound é uma técnica de enumeração implícita/explícita, sua eficiência depende principalmente da habilidade com que usamos esta vantagem no modelamento do problema, a fim de evitarmos gastos computacionais desnecessários. Mesmo assim, existem técnicas empregadas com o intuito de acelerar a convergência do Branch-and-Bound, tais como: Surrogate constraints, penalidades, heurísticos, etc ... [3] ,[5] , [4] .

Vemos, portanto, que o uso desta técnica seria o mais conveniente para o problema que nos propusemos a

resolver. Entretanto, por ser este trabalho uma primeira tentativa na sistematização de códigos ótimos convolucionais, não contamos com informações adicionais necessárias para a escolha de uma função objetivo conveniente e segura ao modelo knapsack. Sendo assim, por ser um trabalho novo em seu objetivo, devemos ter a prudencia de enumerar todas as soluções convenientes, através do método Backtrack usual, para que uma vez comprovada a possibilida de tal sistematização, encontremos os limites onde tais soluções ótimas se encontram seguramente e, assim , usarmos técnicas que convirjam mais rápidamente a solução, como a técnica de Branch--and-Bound ou até mesmo a de Programação Dinâmica, posteriormente. Sendo assim, embora os resultados tenham sido obtidos através do método backtrack, deixaremos as considerações necessárias quanto a técnica do Branch-and-Bound para esta aplicação, na sugestão de certeza de eficiência em um trabalho futuro.

Entretanto, a idéia principal deste método parte do princípio de "dividir e subjulgar", isto é, dividir regiões factíveis, ou seja, regiões de possíveis soluções, em mais sub-divisões convenientes à busca da integralidade e, então, subjulgar o novo problema a verificação da otimalidade.

Em termos mais formais: Um programa linear inteiro é um programa linear acrescido pela restrição de integralidade, assim, uma solução ótima para o problema linear será, na realidade, um limitante superior para o problema linear inteiro. Portanto, qualquer ponto inteiro factível, é sempre um limitante inferior no valor da função objetivo do problema linear ótimo. Dessa forma, subdividimos a região factível do problema linear e fazemos avaliação do problema Inteiro, baseado nessas subdivisões.

Logo, o método do Branch-and-Bound subdivide a região factível para obter os limitantes

$$\underline{z} \leq z^* \leq \bar{z} \quad \text{em} \quad z$$

onde $\underline{z}$ - limite inferior inteiro;

z^* - valor ótimo para o problema;

$\bar{z}$ - limite superior inteiro;

No caso de estarmos maximizando o problema $\underline{z}$ é o maior valor dentre todos os pontos inteiros factíveis encontrados. O limite superior é dado pelo valor ótimo do problema linear associado, ou pelo maior valor da função objetivo em qualquer verificação da solução.

Depois de considerada a sub-divisão atual, precisamos efetuar outra sub-divisão e analizá-las.

Algumas possíveis soluções são ditas " sondadas ", caso o problema linear sub-dividido seja infactível, ou tenhamos alcançado uma solução inteira, que é o objetivo do problema, ou se o valor ótimo encontrado não melhorou o valor ótimo encontrado anteriormente, isto é, caso piore a função objetivo.

A fim de uma melhor visualização da idéia do método, apresentamos na Fig. 3.3.3.1.1.1 o algoritmo da técnica do Branch-and-Bound para um problema de Programação inteira de maximização. [7]

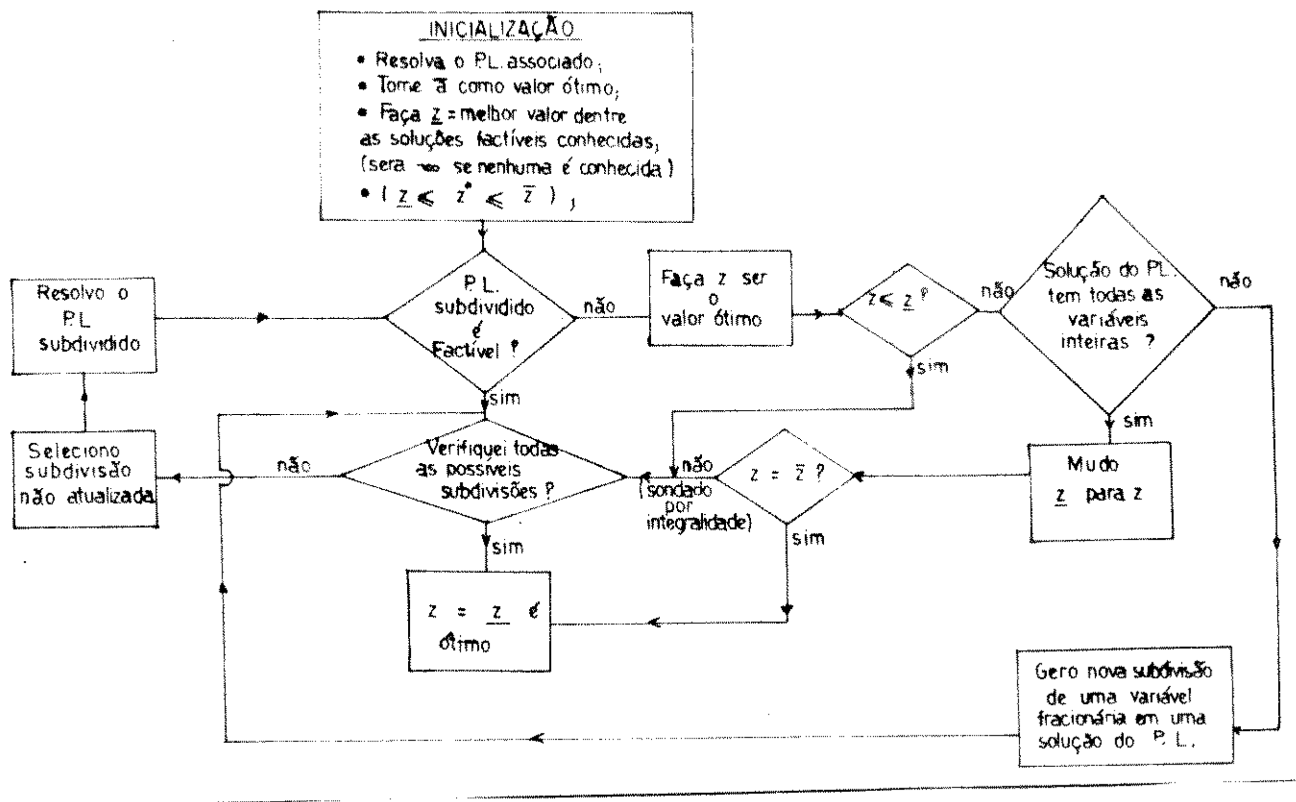


Fig. 3.3.3.1.1.1 : Algoritmo Branch-and-Bound para um problema de Programação Inteira de maximização.

3.3.3.1.2 - BRANCH-AND-BOUND APLICADO AO KNAPSACK.

Visto que o nosso interesse será aplicar o Branch-and-Bound a resolução de um problema Knapsack, certas considerações devem ser significativas, antes da aplicação do método.

A primeira delas é reordenar as variáveis de modo que , levando-se em conta a formalização matemática do Knapsack apresentado no ítem 3.3.1, a restrição apareça na forma a obedecer :

$$c_j / a_j \geq c_{j+1} / a_{j+1}$$

onde :

c_j - custo

a_j - coeficiente da restrição

Esta atitude é levada em consideração, de modo a obter que o primeiro ítem seja o de maior valor, em termos de valor por unidade de peso, em relação aos demais.

Desde que estamos otimizando a função objetivo, não ramificamos de um nó se o seu valor for inferior ao encontrado como melhor, até então.

Mais ainda: Uma vez estando no nível n, o nó possua valor e peso associado e nosso objetivo é a busca do nó ótimo, se o valor deste nó "pai" não for melhor que o atual ótimo, podemos retroceder para o nível (n-1) a procura de outra solução conveniente, pois o método garante que os nós provenientes do nó "pai", os chamados nós "filhos", não apresentarão soluções melhores que as apresentadas por ele. Dessa forma, dizemos que os ramos, a partir deste nó pai, estão sondados .

Assim, se $\sum_{j=1} c_j \cdot x_j$ define o valor do nó, no nível n, obtemos a seguinte expressão.

$$\sum_{j=1}^{n} c_j \cdot x_j + B - \sum_{j=1}^{n} a_j \cdot x_j \cdot c_{n+1} / a_{n+1} \qquad (3.3.3.1.2.1)$$

se

$$b - \sum_{j=1}^{n} a_j \cdot x_j \geq a_i$$

para algum $i>n$, ou

$$\sum_{j=1}^{n} c_j \cdot x_j \qquad \text{se} \quad b - \sum_{j=1}^{n} a_j \cdot x_j < a_i \qquad (3.3.3.1.2.2)$$

para todo $i > n$.

Vemos da equação (3.3.3.1.2.1) que $(\sum_{j=1}^{n} c_j \cdot x_j)$ representa o valor de todas as variáveis já assinaladas ; $b - \sum_{j=1}^{n} a_j \cdot x_j$ representa a quantidade restante de peso, uma vez que o nível é o n-ésimo [1].

Uma vez que as variáveis são ordenadas , o valor do novo peso limitante, multiplicado por (c_{n+1} / a_{n+1}), fornecido de x_{n+1}, poderia não ser inteiro. Sendo essa expressão o valor do nó, este valor seria na realidade um limitante superior para os nós "filhos" que aparecerão a seguir. Se o valor desse nó pai for menor ou igual (no caso do objetivo ser maximizar) ao valor da melhor solução factível atual,então podemos abandonar as buscas neste ramo, isto é, sondamos o ramo, pois não encontraremos melhores soluções.

Quando

$$b - \sum_{j=1}^{n} a_j \cdot x_j < a_i$$

para todo $i > n$, então nenhum outro ítem poderá ser colocado no Knapsack, portanto, armazenaremos $\sum_{j=1}^{n} c_j \cdot x_j$ como sendo o valor do nó e, o nó , é deixado.

Dessa forma, o Branch-and-Bound torna-se uma valiosa ferramenta para encontrar soluções de um problema Knapsack.

Apresentaremos a seguir, um exemplo detalhando tal aplicação.

Seja o problema Knapsack apresentado por :

$$\text{MÁX} \quad Z = x_1 + 3x_2 + 5x_3 + 9x_4$$

s.a.

$$2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10$$

$$x_j \geq 0 \qquad \text{e inteiros}$$

Assim, reordenando as variáveis de modo a obter

$$c_j \,/\, a_j \geq c_{j+1} \,/\, a_{j+1}$$

reescrevemos o problema da forma :

$$\text{máx} \quad z = 9x_1 + 5x_2 + 3x_3 + x_4$$

s.a

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \leq 10$$

$$x_j \geq 0 \qquad \text{e inteiro}$$

Podemos perceber que, para o problema acima as variáveis podem assumir os valores máximos apresentados abaixo.

Máx x_1 =1 , pois INT[10/7] =1

Máx x_2 = 2 , pois INT[10/4] = 2

Máx x_3 = 3 , pois INT[10/3] = 3

Máx x_4 = 5, pois INT[10/2] = 5

Obs: Os valores acima devem ser entendidos como a quantidade máxima de itens que caberiam dentro da "mochila", sem ultrapassar o peso limitante , caso quisessemos levar apenas um único tipo. Sendo assim, os valores que cada variável pode assumir é apresentado abaixo :

$$x_1 = \{0,1\} \; , \; x_2 = \{0,1,2\} \; , \; x_3 = \{ \; 0,1,2,3\} \quad e \; x_4 = \{0,1,2,3,4\}$$

Assim, com a metodologia do Branch-and-Bound, descrita préviamente, obtemos como solução a árvore apresentada na Fig. 3.3.4.1.4.1 .

Nos nós N_3 , N_4 , N_6 e N_7 os pesos excedem o peso total limitante. Os nós N_7 , N_{11} , N_{12} e N_{13} sondados, pois apresentam valores menores que o valor ótimo atual e como sabemos que nós filhos provenientes do mesmo nó pai, nunca apresentam soluções melhores que as dele, verificações posteriores tornam-se desnecessárias e o problema apresentado será concluído sem sequer ter entrado no nível 4.

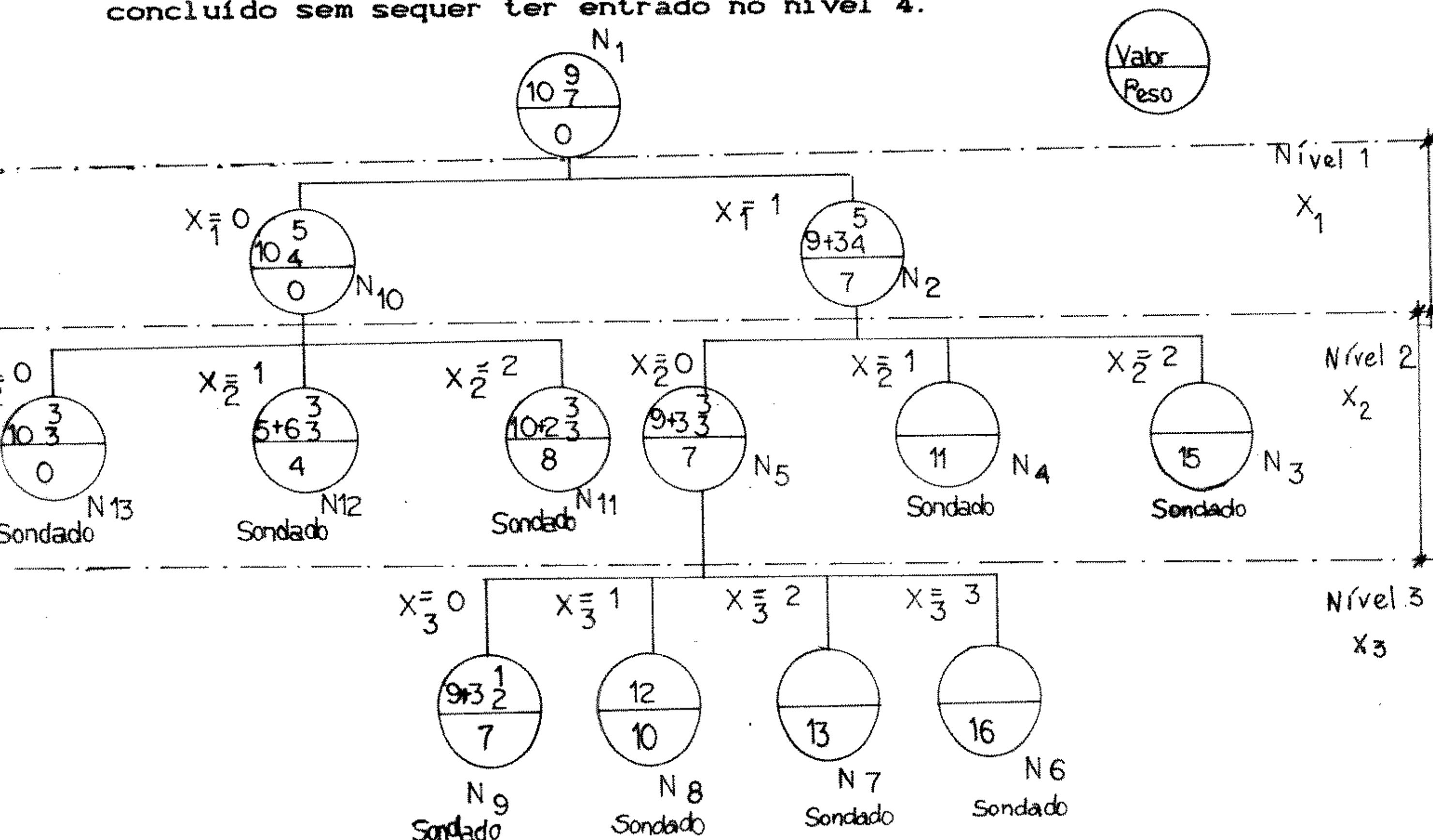


Fig. 3.3.3.1.4.1 : Árvore enumerativa do Problema.

CAPITULO 04

CAPITULO 4

PROBLEMA "KNAPSACK COMBINATORIAL APLICADO AOS CÓDIGOS DE MEMÓRIA UNITÁRIA

4.1- INTRODUÇÃO

Neste capítulo iremos estabelecer formalmente o problema "knapsack combinatorial" inerente aos códigos de memória unitária. Esta forma matemática tornou-se viável devido ao problema de determinação de códigos convolucionais ótimos possuirem uma equivalência com o problema de fluxo em redes, como mostrado em [6].

Para que não haja perda de continuidade na apresentação, iremos na seção 4.2 apresentar a equivalência dos problemas mencionados, bem como os teoremas que se fazem necessários. Na seção 4.3, o problema "knapsack combinatorial" será estabelecido. Na seção 4.4 o algoritmo de Viterbi será apresentado. Estas seções formam a estrutura básica do algoritmo implementado, para a solução do problema em mãos, de uma forma sistemática. Gostaríamos de salientar que esta é uma contribuição original em direção a solução de um problema NP-completo no seu pior caso.

4.2- EQUIVALÊNCIA DOS PROBLEMAS : DETERMINAÇÃO DE CÓDIGOS ÓTIMOS E FLUXO EM REDES

Um grafo direcionado $G=[N,A]$, consiste de um conjunto finito N, onde $N = \{ 1,2,3, \ldots, n\}$, e de um conjunto A de pares ordenados (i,j), não necessariamente distintos, isto é, $A = \{ (i,j) : i,j \in N \}$. Os elementos de N e A serão chamados de estados e ramos, respectivamente.

Suponha que cada ramo (i,j) de um grafo direcionado tenha associado a si um número não negativo c_{ij} ,a

capacidade de (i,j), representando a quantidade máxima de algum "produto", que pode chegar em j vindo de i ao longo de (i,j), por unidade de tempo, em uma condição de estado fixa.

A fonte, nó 1, é a entrada do fluxo na rede e, o nó mais profundo na árvore, o nó n, é o fim do fluxo na rede. Matemáticamente, os ramos x_{ij}, são definidos como um conjunto de números não-negativos, satisfazendo as seguintes restrições :

$$\sum_i x_{ij} - \sum_k x_{jk} = \begin{cases} -\phi & , \text{ se } j = 1 \\ 0 & , \text{ se } j=1,n \\ \phi & , \text{ se } j=n \end{cases} \qquad (4.2.1)$$

$$0 <= x_{ij} <= c_{ij} \qquad (4.2.2)$$

Note que o fluxo é conservado em cada nó exceto na fonte e no último nó, e que cada fluxo do ramo, x_{ij}, é limitado superiormente,pela sua capacidade c_{ij}, que neste caso é a máxima distância de Hamming do estado i ao estado j.

Seja a seguinte conjectura :

(4.2.3) Se existir um codificador convolucional não-sistemático e invariante no tempo com parâmetros K e r=b/n tal que $M_i(A') = M_k(A') = D^p$ para todo k=i, e $M_i(A') = M^j(A') = D^p$ para todo i=j, e vice-versa, com $p = n.\ 2^{k-1}$, então o código é ótimo. Onde :

A' = matriz de transição aumentada (i.é., com os estados finais e iniciais incluídos);

$M_i(A')$ = matriz produto dos elementos não nulos da i-ésima linha de A';

$M^j(A')$ = produto dos elementos não-nulos da j-ésima coluna de A';

$r = b/n$ = taxa

(para facilidade notacional, iremos usar b ao invés de k);

K = restrição de comprimento = m+1;

D^p = valor de cada ramo fornecido pela função de transferência, onde p é o fluxo total;

Assim, se um código satisfizer conjectura (4.2.3), então as condições (4.2.1) e (4.2.2) são encontradas.

Portanto, sob o ponto de vista desta conjectura, o problema de encontrar códigos convolucionais ótimos, não-sistemáticos, é caracterizado pelo Fluxo Máximo na Rede .

O seguinte teorema estabelece o fluxo máximo que o codificador convolucional ótimo invariante no tempo satisfaz.

TEOREMA 4.2.4 [6] : Para um código convolucional não-sistemático,invariante no tempo, com taxa r=b/n e restrição de comprimento K, o fluxo uniforme é dado por

$$\phi = n.\ 2^{b-1}$$

PROVA : Para a taxa r=b/n e restrição de comprimento K, o número de estados é $2^{b(K-1)}$. O número de transições de cada estado é 2^b . O número total de ramos é dado por $2^{b.K}$. Por outro lado, a saída do codificador tem comprimento n e assim, existem 2^n possíveis seqüências de saída.

Seja c a razão entre o número de ramos

e o número de seqüências de saída. Se c é maior que 1, então este número especifica quantas vezes a seqüência de saída será repetida. Se c é maior que 1, então ele fornecerá a proporção da seqüência de saída que estão sendo usadas.

Seja $w_T(c')$ o peso total das seqüências de saída , onde

$$w_T(c) = \sum i \binom{n}{i} = [d/dt] ((1+a)^n)_{a=1}$$

$$= n.2^{n-1}$$

então, $c = w_T(c')$ fornece o peso total do diagrama de estado. Desde que, existem $2^{b(k-1)}$ estados, o fluxo uniforme é

$$\phi = [c . w_T(c')] / 2^{b(k-1)}$$

$$= n.2^{b-1}$$

<u>C. Q. D.</u>

Dessa forma, a busca do código ótimo convolucional para qualquer K e taxa r=b/n , reduz-se ao seguinte problema :

Dado um diagrama de estado particionado, onde todos os pesos dos ramos são limitados inferiormente por zero e superiormente por c_{ij} , onde os (i,j) são as máximas distâncias de Hamming de cada ramo, encontre os fluxos dos ramos, as distâncias dos ramos, tal que a distância mínima para sair do estado zero e para ele retornar é maximizada sujeita ao fluxo máximo ϕ .

Uma rede com estas cracterísticas é chamada rede limitada . Assim, o problema de encontrar códigos

convolucionais bons pelo uso de técnicas heurísticas, foi transformado em um problema bem estruturado que se resume em encontrar o fluxo na rede.

Inerente a esta transformação, existe alguns problemas combinatoriais que, na maioria dos casos, podem ser facilmente resolvidos, tornando possível a determinação de

códigos convolucionais através de cálculos manuais.

Para acelerar o processo, a distância mínima do código é calculada na exclusão dos "candidatos potenciais" da classe dos códigos ótimos.

O seguinte teorema estabelece um limite superior para a distância de qualquer código convolucional (invariante no tempo, periódicamente variante no tempo, variante no tempo), tão bom quanto o dos códigos de treliça não-lineares recorrentemente gerados, com parâmetros K e r=b/n, mdc(b,n)=1, que é muito bom para restrições de comprimento pequenas. Ele generaliza o limite superior de Heller , para códigos convolucionais com restrição de comprimento K e taxa r=b/n.

Em particular, temos :

TEOREMA 4.2.5 [6]: Para qualquer código convolucional com restrição de comprimento K e taxa r=b/n, mdc(b,n)=1, a distância mínima e limitada por

$$d_{min} <= \min_{p\geq 1} \{ [2^{p-1}/2^{p}-1].(n/b).(p+b(K-1))\}$$

PROVA : É sabido que o código convolucional binário representado com M bits de informação é um código de grupo que apresenta uma matriz geradora de dimensão M X n.(M+(K-1)) para taxa r=1/n. Para taxa r=b/n, mdc(b,n)=1, desde que tenhamos b k-registros de deslocamento em paralelo, determinar um código implica na necessidade de inserirmos b(K-1) dígitos conhecidos. O comprimento total de bits de informação é b.M e, assim, a matriz geradora tem, agora, b.M linhas e (n/b)(b.M + b(K-1)) colunas.

Para códigos de grupo binário, a distância de Hamming entre palavras-código é equivalente ao peso

de palavras-código não nulas. Assim, se todas as 2^{bM} palavras-código são arranjadas como linhas de uma matriz, então qualquer coluna, excluindo a toda nula, tem metade uns e metade zeros [ ]. Assim, o peso total das 2^{bM} palavras-código é limitado superiormente por

$$w_T <= 2^{bM-1} (n/b).(b.M + b(k-1))$$

Desde que $2^{bM}-1$ palavras-código são não-nulas e seus pesos mínimos precisam ser menores ou iguais aos seus pesos médios, a distância satisfaz

$$d_{min} <= [2^{bM-1}/ 2^{bM} - 1].(n/b).(b.M + b(K-1))$$

Desde que este limite conserva-se para qualquer bM, ele também conserva-se para p<bM . O que nós realmente queremos é o menor limite superior. Este é encontrado se minimizarmos o lado direito da equação em relação a p , assim

$$d_{min} <= \min_{p\geq 1} [2^{p-1}/ 2^{p} - 1].(n/b).(p + b.(K-1)) = d_{limite}$$

onde [a] , significa o maior inteiro menor ou igual a a .

C. Q. D.

O limite superior melhorado desenvolvido por ODENWALDER , onde d_{limite} é impar e r=1/n, é também aplicável a taxas r=b/n, isto é, um código linear ou tem todas as palavras-código com peso par ou metade tem peso par e metade tem peso ímpar. Assim, se d_{limite} é ímpar para algum ótimo c , então o peso total de 2^c palavras-código é limitado por

$$2^{c-1} \cdot d_{limite} + (2^{c-1} - 1).(d_{limite} + 1) <= w_T$$

$$2^{c-1} \cdot d_{limite} + (2^{c-1}-1).(d_{limite} +1) <= 2^{c-1}(n/b).(c+b(K-1))$$

assim,

$$d_{limite} <= [2^{c-1} /2^{c}-1]. \{(n/b).(c+b(K-1)) + 2^{1-c} - 1\} \quad (4.2.6)$$

Se d_{limite} é impar e (4.2.6) não é satisfeita, então d_{limite} pode ser decrescido de uma unidade.

EXEMPLO : Tomemos K=2 e a taxa r=2/3. O objetivo é encontrar o codificador convolucional ótimo.

Pelo teorema 4.2.4 , $\phi = n.2^{b-1} = 3.2^{2-1} = 6$.

O diagrama de estado particionado é mostrado na Fig. 4.2.6.

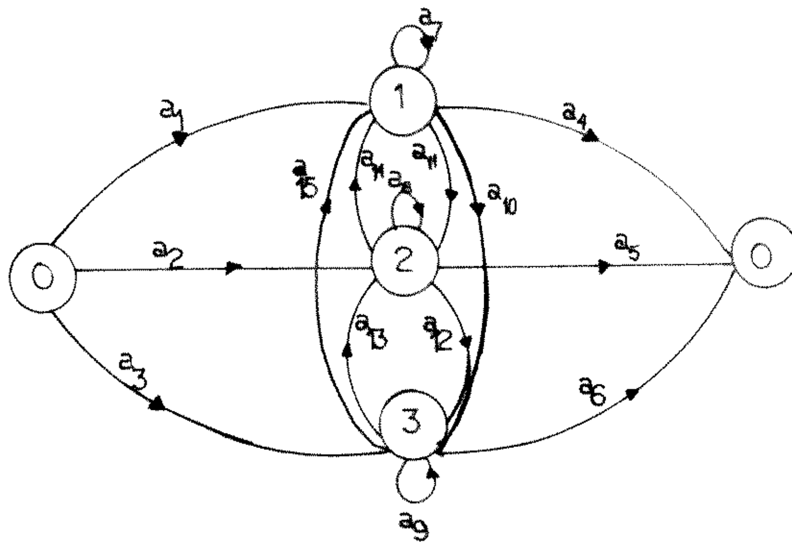


Fig. 4.2.6. : Diagrama de Estado Particionado para $K = 2$ e $r = 2/3$.

então, $a_1 + a_2 + a_3 = 6$ e $a_4 + a_5 + a_6 = 6$, mas os a_i são as distâncias de Hamming dos ramos que representam as palavras-código. Assim, temos que encontrar valores para a_i tal que : 1) o fluxo seja 6 no estado zero e a conservação do fluxo seja válida para todos os estados restantes; 2) a distância mínima seja máxima; 3) o peso do diagrama de estado seja 24.

O peso de um diagrama de estado gerado por um codificador com restrição de comprimento K e taxa $r=b/n$ é dado pela soma de todos os pesos dos ramos. Do teorema 4.2.1, temos que o fluxo saindo e entrando em cada estado é $\phi = n.2^{b-1}$, desde que existem $2^{b(k-1)}$ estados,

$$w_T = \sum_{i=1}^{2^{b(k-1)}} a_i = \phi \cdot 2^{b(k-1)}$$

Assim, provamos :

LEMA 4.2.8 [6] : O peso total de um diagrama de estado gerado por um codificador com parametro K e $r=b/n$, $mdc(b,n)=1$, é dado por

$$w_T = \phi \cdot 2^{b(k-1)} = (n/q) \cdot 2^{bk}$$

A mesma aproximação usada para encontrar a configuração do codificador com parâmetros K e $r=1/n$ é também aplicável a taxas $r = b/n$.

Seja $\underline{u} = (u_1 , u_2 , u_3 , \dots)$ a seqüência de dados de entrada que entra no codificador de registros de deslocamentos, com u_i tendo comprimento b, e G_i são submatrizes, de dimensão bXn, da matriz geradora G do código convolucional.

As palavras-código de saída são dadas por $x = u \cdot G$. Assim, o problema combinatorial é encontrar as linhas de G_i , $i=0,1,2, \dots , k-1$, tais que as condições 1), 2) e 3) sejam satisfeitas. Pelo teorema 4.2.5, $d_{min} \leq 4$, para $d_{min} = 4$, todas as possíveis G_i fornecem códigos catastróficos, então $d_{min} = 3$ é o valor ótimo para esta classe. Nas próximas seções, veremos este problema mais detalhadamente. Uma possível solução do problema combinatorial é :

$$G_0 = \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix} \quad e \quad G_1 = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \end{vmatrix}$$

4.2 - PROBLEMA "KNAPSACK" COMBINATORIAL

Quando a conjectura 4.2.3 é satisfeita, existe conservação do fluxo em cada estado no diagrama de estado particionado. Apesar do fluxo máximo e da conservação do fluxo

serem propriedades importantes, eles somente fornecem a busca em um conjunto cuja cardinalidade é menor que aquela do problema original . Para uma restrição de comprimento grande, este novo conjunto tem um número de elementos considerável.

O menor limite superior para a distância livre (d_{free}) é outro parametro que precisa ser usado no processo , que reduzirá a busca. Assim, o fluxo máximo e a distância livre são os únicos parâmetros que tem de ser examinados.

Para taxas r=b/n, o número de transições entre quaisquer estados, menos o zero, é 2^b . Para o estado zero, este número é $2^b -1$ no diagrama de estado particionado. Assim, o número de caminhos deixando e entrando no estado zero em K ramos é $2^b -1$. Seja d o conjunto ordenado que contém todas as $2^b -1$ distâncias, isto é ,

$$d = \{ d_1 , d_2 , d_3 , \ldots , d_{2^b -1} \} \quad (4.3.1)$$

É natural esperarmos que

$$d_{free} \geq \min \{ d \} \quad (4.3.2)$$

O critério de otimalidade a ser adotado quando comparamos conjuntos ordenados é do tipo lexicográfico, isto é, dado 2 conjuntos ordenados X e Y

$$X = (x_1, x_2, x_3, \ldots, x_n)$$

$$Y = (y_1, y_2, y_3, \ldots, y_n)$$

$$X > Y \quad \text{se} \quad x_i = y_i \quad \text{para} \quad 1 \le i \le p \qquad (4.3.3)$$

$$x_{p+1} > y_{p+1}$$

Do teorema 4.2.4 sabemos que o fluxo máximo é $\phi = n \cdot 2^{b-1}$, que é igual a soma de todas as distâncias de Hamming saindo ou entrando em um estado. Desde que tenhamos um codificador com K elementos armazenadores , então K é um limite superior na soma de pesos de todo d_i pertencente ao ramo de maior caminho K, isto é,

PROBLEMA I :

$$\sum_{i=1}^{B} a_i \cdot d_i < K \cdot \phi \qquad (4.3.4)$$

sujeito a

$$\sum_{i=1}^{B} a_i = 2^{b} - 1 \quad \text{e} \quad d_1 = d_{free} , \quad d_2 = d_{free} + 1 , \quad \text{e assim}$$

por diante .

B é uma constante com nenhum significado principal reservado a ela.

Uma vez conhecidos os a_i 's, o próximo passo é resolver um problema combinatorial , como descrito a seguir : Encontre as submatrizes, de dimensão bXn, G_i , $i = 0,1,2,\ldots, k-1$ cujos elementos pertencem a GF(2), tal que a combinação linear de suas linhas satisfaça a solução do Problema

I. Se isto não for possível, temos que rearranjar os valores para a_i no ponto onde o problema combinatorial iguala-se a nova solução do Problema I . Quando isto ocorre, encontramos o codificador que irá ser lexicograficamente ótimo. Este procedimento não elimina a possibilidade de encontrarmos código catastrófico. Para restrições de comprimento pequeno, é fácil verificar se os valores encontrados para G_i resultam em um código catastrófico pela simples regra de independência linear entre as colunas de G_i , mas para restrição de comprimento longo, uma busca exaustiva poderia ser utilizada.

Tomemos um exemplo para demonstrar o procedimento. Considere o caso onde K=2 e r=2/5. Sabemos do teorema 4.2.4 que $\phi = n.2^{b-1} = 5(2)=10$. Do teorema 4.2.5, temos que $d_{min} \leq 6$, então o Problema I é determinado como segue

(A)

$$\sum_{i=1}^{B} a_i \cdot d_i \leq K.\phi \qquad a_1 d_1 + a_2 d_2 + \ldots + a_b \ d_b \leq 2.\phi$$

(B)

$$\sum_{i=1}^{B} a_i = 2^b - 1 \qquad a_1 + a_2 + \ldots + a_B = 3$$

De (A) e (B) temos

1) $a_1 = 1 \quad a_2 = 2$; 2) $a_1 = 2 \quad a_3 = 1$

Claramente a solução é 1). Assim,

$$d = \langle 6,7,7 \rangle$$

O problema combinatorial é encontrar as matrizes G_0 e G_1 , de dimensão bXn, tais que d seja encontrado. Mas d_i , i= 1,2,3 é a soma dos ramos dos pesos de Hamming que deixam e entram no estado zero.

Seja w_i , $1 <= i <= 6$, tal que

$d_1 = w_1 + w_4$, $d_2 = w_2 + w_5$, $d_3 = w_3 + w_6$

Uma possível solução é

$w_2 = 4$ $\qquad$ $w_5 = 2$

$w_1 = 3$ $\qquad$ $w_4 = 4$

$w_3 = 3$ $\qquad$ $w_6 = 4$

e assim,

$$G_0 = \begin{vmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{vmatrix} \qquad G_1 = \begin{vmatrix} 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix}$$

Por outro lado, se depois de um número de tentativas não existir a igualdade entre a solução do problema combinatorial e a do Problema I, o próximo passo é diminuir de uma unidade o valor da distância livre original e começar novamente o processo de busca descrito acima até que a igualdade entre as soluções aconteça.

Dessa forma, uma vez conhecido o código em questão, e conseqüêntemente a sua taxa, e substituindo os dados nas expressões (4.2.4) e (4.2.5) , obtemos ϕ e $d_0 = d$

Note que o valor apresentado em (4.2.2) pode não ser inteiro necessariamente, sendo assim, devemos considerar os inteiros que aproximam o valor encontrado, superiormente e inferiormente, e considerar ambos os casos no andamento do problema.

Em posse do d_0 escolhido, devemos passar ao próximo passo que seria determinar o vetor $\underline{a}$. O vetor $\underline{a}$ apresenta, em seus elementos, a quantidade de vezes que cada d_i aparece no vetor espectro de Hamming onde temos que d_i é dado por

$d_i = d_{i-1} + 1$, para $i = 1, \ldots, q^b - 1$ e aparece de modo a obtermos :

$$\sum_{i=1}^{q^k - 1} a_i \, . \, d_i \leq 2 . \phi$$

Ainda mais, sabemos que os elementos do vetor a, apresentado como possível vetor peso, deve ainda obedecer a seguinte restrição :

$$\sum_{i=1}^{q^b - 1} a_i = (q-1)\,(q^b - 1)$$

onde $a_i \geq 0$ inteiro, e $d_i = d_{i-1} + 1$ com d_o escolhido a priori por razões que serão melhores esclarecidas a seguir.

Vemos, portanto, que possuimos apenas uma restrição realmente significativa, que seria de preenchimento do flúxo máximo multiplicado por dois.

De acordo com os conceitos apresentados no Capítulo 3, podemos induzir facilmente o nosso problema a ser um problema de Programação Inteira do tipo "Knapsack", onde a função objetivo poderia ser minimizar a_o , uma vez que a_o é o coeficiente relacionado ao d_o escolhido (que é mínimo).

Entretanto, no problema em questão, não é de fundamental importância a função objetivo do knapsack, mas sim as diferentes formas de "preenche-lo", no momento.

O método escolhido para resolver tal knapsack descrito acima , foi o Backtrack , apresentado e discutido no Capítulo 3, por ser , comojá comentado, o mais conveniente .

Sendo assim, o Knapsack Combinatorial resol vido através do método do Backtrack, será aplicado com o intuito de encontrarmos os vetores a_i 's mais convenientes.

EXEMPLO : Suponhamos que para um deteminado código de memória unitária, tenhamos encontrado $\phi = 6$, $d_o = 4$ e $b = 2$. Assim,sabemos que o número de elementos do vetor a é dado

por $q^b - 1 = 2^2 - 1 = 3$. Sendo assim, o knapsack a ser resolvido tem como restrição :

$$4\ a_1 + 5\ a_2 + 6\ a_3 = 12$$

Podemos perceber pela igualdade apresentada acima, que a quantidade máxima que cada a_i pode assumir é determinada por:

$$a_i = \text{INT}\ (\ 2\ \phi\ /\ d_i\ )$$

e assim para :

a_1 ———— INT(12/4)=3 ———— $a_1 \in \{0,1,2,3\}$

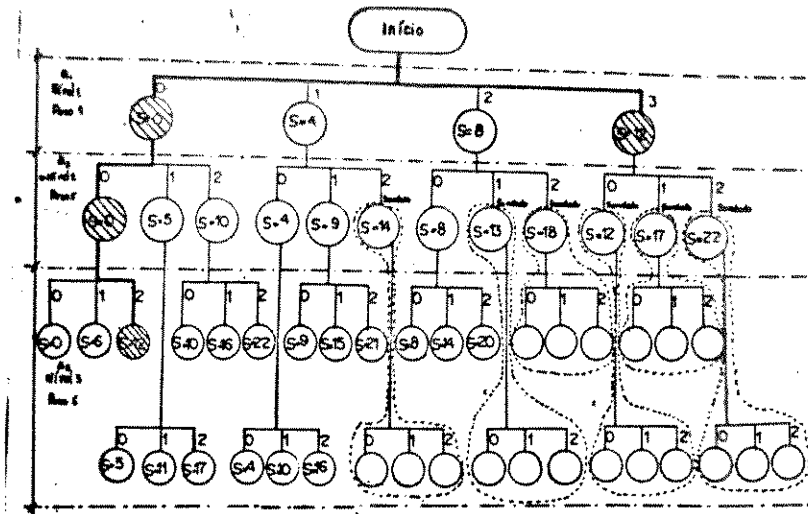
a_2 ———— INT(12/5)=2 ———— $a_2 \in \{0,1,2\}$

a_3 ———— INT(12/6)=2 ———— $a_3 \in \{0,1,2\}$

Os d_i nada mais são do que os "pesos" dos níveis da árvore que representa este problema. Assim

$$P_{a_1} = 4 \quad , \quad P_{a_2} = 5 \quad e \quad P_{a_3} = 6$$

A árvore deste problema é apresentada abaixo :



Legenda ---- - Solução sondada antes de entrar no nível 3

▨ - Possível solução

s - Soma acumulada

Fig. 4.1 - Solução em forma de árvore, representando o problema .

Deste modo as soluções que seriam apresentadas como resposta do knapsack são :

$$\underline{a}^{(1)} = (3,0,0) \qquad \text{e} \qquad \underline{a}^{(2)} = (0,0,2)$$

Como podemos verificar, a soma dos elementos do vetor $\underline{a}^{(1)}$ fornece $q^{b} - 1 = 3$, o que não ocorre com a soma dos elementos do vetor $\underline{a}^{(2)}$, que é igual a 2. Logo , somente o vetor $\underline{a}^{(1)}$ é solução conveniente para este problema, como resposta do knapsack combinatorial.

Sendo assim, as componentes do vetor $\underline{a}$, conveniente, quando não nulas, fornecem o número de ramos que entram e deixam o estado zero, na representação do diagrama de estados particionados, com distância de Hamming d_i .

Logo , o vetor $\underline{a}$ fornece o número de ramos com determinado peso d_i , ou seja, o vetor $\underline{a}$, na realidade, é o vetor coeficiente dos D na expressão da função

de transferência do código apresentado.

Tendo o vetor $\underline{a}$, podemos continuar em busca de mais uma informação preciosa: o **"VETOR ESPECTRO DE HAMMING"**, que é denotado por :

$$\underline{D} = (D_0 , D_1 , \ldots D_{q^b-1}) .$$

O elemento a_i do vetor $\underline{a}$ indica o numero de elementos do vetor $\underline{D}$ que terão valor d_i .

Para o exemplo fornecido anteriormente, temos $\underline{a} = (3,0,0)$. O vetor espectro de Hamming associado será $\underline{D} = (4,4,4)$, pois o elemento não negativo no vetor $\underline{a}$,está associado ao $d_0 = 4$. Se, por hipótese, tivéssemos encontrado como possível solução $\underline{a} = (2,1,0)$, o vetor $\underline{D}$ associado seria $\underline{D} = (4,4,5)$, pois o novo vetor $\underline{a}$ possui a informação de que em $\underline{D}$ existem 2 elementos iguais a d_0 e 1 elemento igual a d_1 .

Agora torna-se claro o porque da soma dos elementos do vetor $\underline{a}$ ter que ser igual a $(q - 1)(q^b -1)$, para apresentar uma solução conveniente.

Este vetor $\underline{D}$ é o **"ESPECTRO DAS DISTANCIAS DE HAMMING"** de um código de bloco equivalente, sobre o caminho de dois ramos do código de memória unitária em consideração.

Tendo os pesos das palavras-ramo do código, necessitamos determinar a matriz geradora do código de bloco equivalente, que apresenta palavras-código com peso de Hamming dados pela solução do Problema Knapsack Combinatorial.

Através do conceito de representação modular de código de bloco, apresentado no Capítulo 2, poderemos determinar, de forma sistemática, a matriz geradora do código apresentado.

A aplicação do conceito de representação

modular de códigos de bloco à solução do problema knapsack, resulta em um conjunto $T = \{ t_i \}$, onde t_i são os vetores de valores inteiros, de dimensão $(q-1)(q^b-1)$, especificando a matriz geradora.

Sem dúvida, somente os vetores t_i, cujas componentes $t_i > 0$, são considerados apropriados para o "tipo" em questão.

O próximo passo seria enumerar o vetor t_i, isto é, a enumeração das sub-matrizes formadas pelos valores dos vetores "tipo" correspondentes. Contudo, uma vez que concretizada a enumeração, não existe a garantia de que o código resultante não seja catastrófico, devemos, portanto, fazer tal verificação.

A técnica mais eficiente de verificação de catastroficidade é aquela que apresenta o conceito de matriz de transição avaliada.

Caso a matriz geradora determinada apresente um código catastrófico, podemos ainda rearranjar as colunas desta matriz geradora apresentada, de forma a encontrar uma matriz geradora de um código equivalente ao anterior e não catastrófico.

Para uma melhor compreensão do que foi descrito nos parágrafos anteriores, apresentamos um exemplo completo, para elucidar o que acabamos de expor.

EXEMPLO : Seja o código de memória unitária (3,2,1) . Assim, a taxa r é dada por $r = b/n = 2/3$; o número de elementos do corpo de Galois é dado por $q = 2$; o número de memórias por $m=1$.

A equação (1) fornece o fluxo , $\phi = 3(2) = 6$.

Substituindo os dados na equação (2), encontramos

$$d_o \leq \min_{p \geq 1} \left(\frac{2^{p-1}}{2^p - 1} \frac{3}{2} (p+2) \right)$$

Encontramos como valor de d_o, um número real e, assim, devemos considerar $d_o = 3$ e $d_o = 4$.

Assumindo $d_o = 4$, se encontrar uma matriz geradora conveniente, não será necessário fazer novo teste para $d_o = 3$, pois quanto maior for a distância mínima que é apresentada por d_o, melhor será a capacidade de correção do código dado.

Como b= 2, teremos $2^b - 1 = 2^2 - 1 = 3$ elementos componentes do Knapsack Combinatorial.

Assim,

$$a_o d_o + a_1 d_1 + a_2 d_2 = 2 . \phi$$

isto é ,

$$4 a_o + 5 a_1 + 6 a_2 = 12$$

O método de resolução do Knapsack Combinatorial é o Backtrack, apresentado no exemplo 1.

A heurística de que a soma dos elementos dos vetores apresentados como solução do Método Backtrack deva ser igual a $2^b - 1$, apresenta como solução conveniente, a=(3,0,0). O vetor espectro de Hamming é, portanto, D =(4,4,4).

Usando representação modular de códigos de bloco para a determinação da matriz geradora correspondente, temos:

$$c = D \; . \; H = (4,4,4) \cdot \begin{vmatrix} 1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & -1 \end{vmatrix} = (4,4,4)$$

onde a matriz H é obtida recursivamente através da disposição dos elementos na cruz marcada na própria matriz. Uma vez que $H = (M^t M)$, onde M é a matriz que apresenta como coluna j, o vetor do tipo j em binário. Como o código é (3,2,1) , M possui dimensão b X n e apresenta as colunas ordenadas como números binários.

Ao resultado encontrado por c , devemos multiplicar pelo fator de correção:

$$\frac{1}{2^{b-1}}$$

Assim,

$$c' = c \; . \; \frac{1}{2^{2-1}} = (4,4,4) . \; \frac{1}{2}$$

$$= (2,2,2)$$

O novo vetor c' = (2,2,2), fornece a informação do número de tipos existentes para a matriz geradora G,

isto é, cada coordenada i, do vetor $\underline{c}$, está associada ao tipo i.

Sendo assim, a quantidade de elementos em cada coordenada i, indica a quantidade de tipos i que aparecem na matriz geradora G.

Uma vez conhecido c', determinamos a matriz geradora G. Como $\underline{c}$ = (2,2,2), temos na matriz geadora, 2

UNICAMP
BIBLIOTECA CENTRAL

colunas do tipo 1, 2 do tipo 2 e 2 do tipo 3.

Cada tipo i é apresentado como colunas da matriz G de forma que o número i, é apresentado em binnário. A quantidade de elementos é o número de vezes que cada coluna de tipo i aparece na matriz geradora G.

Dessa forma,

$$G = \begin{vmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{vmatrix}$$

Apresentando G na forma "echelon", temos

$$G = \begin{vmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{vmatrix}$$

Assim, as sub-matrizes do código de memória equivalente são :

$$G_0 = \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{vmatrix} \qquad \text{e} \qquad G_1 = \begin{vmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

Já de posse de G_0 e G_1 , o diagrama de estados particionado é facilmente obtido.

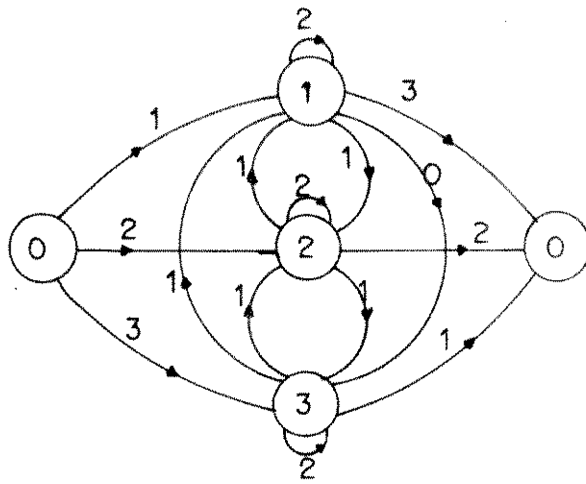


Fig. 4.2 - Diagrama de Estados Particionados.

Vemos, pelo diagrama apresentado na Fig. 4.2 acima, que a distância mínima para sairmos do estado 0 e, para ele retornar, é dada por 2, para o código em questão.

Note que colocamos G_0 e G_1 na forma "echelon", pois caso apresentassemos G do modo como foi obtida, teríamos nitidamente verificado a catastroficidade do código, pois tendo

$$G_0 = \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix} \quad \text{e} \quad G_1 = \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

verificamos facilmente que as linhas de G_0 e de G_1 são idênticas.

Por outro lado, se usarmos a forma "echelon" para G_0 e para G_1, ou vice-versa, teremos

$$G_0 = \begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{vmatrix} \quad \text{e} \quad G_1 = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix}$$

que resultaria, novamente, num código catastrófico.

Deveríamos continuar efetuando troca de colunas em G até que a catastroficidade aparente fosse destruída.

O exemplo acima , apresenta claramente os passos que serão apresentados no algoritmo descrito no próximo capítulo, onde todos os esclarecimentos necessários para a obtenção de um melhor efeito de sistematização, são fornecidos.

4.3 - ENUMERAÇÃO DAS PALAVRAS CÓDIGO

As seções dos Capítulos 2 e 3 enunciaram os principais conceitos a serem aplicados e discutidos neste item.

Sendo assim, uma vez que o Algoritmo de Viterbi determina a distância de Hamming mínima e que o Problema de Programação Dinâmica conhecido como "Problema da Diligência", já discutido, busca a rota mínima, é evidente a similaridade dos problemas apresentados.

Assim, seja

d_{s,x_n} = distância de Hamming do estado s para o estado x_n;

$f(s_n, x_n)$ = distância mínima total para os estágios restantes, dado que estou no estado s e seleciono como destino imediato a variável x_n ;

x_n = variável de decisão do estágio n ;

x^*_n = estado que minimiza $f(s_n, x_n)$. (Distância de Hamming);

f^*_n = valor mínimo correspondente a $f_n(s_n, x_n)$;

e a função recursiva dada por :

$$f_n(s, x_n) = d_{s,x_n} + f^*_{n+1}(x_{n+1})$$

Sob a formulação apresentada acima, tem-se o algoritmo de Viterbi estruturado sob o ponto de vista da Programação Dinâmica para a enumeração das palavras-código do código de memória unitária.

Convém salientarmos que d_{s,x_n} determina a ligação de estado a estado, na treliça. Assim, para efeito de redução de memória, caso não haja ligação de determinado estado a outro em alguma das "janelas de tempo" da treliça, não haverá o d_{s,x_n} correspondente.

Note que esta formulação mostra também ser conveniente, quando aplicada a treliças variantes no tempo.

A variável s deve ser entendida como um contador de ligações de estado a estado, isto é, se existe ligação do estado i para o estado j, no diagrama de estados, o "estado" s

recebe um número que irá referenciar a esta ligação sempre que solicitado. Isto evita armazenarmos "zeros" indicando a não-existência de ligação de estados, como já salientado.

A $f^*(1,1)$ apresentará a distância mínima e se somarmos todas as soluções que fornecem esta distância mínima, poderemos verificar a eficiência da matriz geradora apresentada como resultado da seção 4.2 anterior.

Suponhamos que um código apresenta a seguinte treliça para uma janela de tempo :

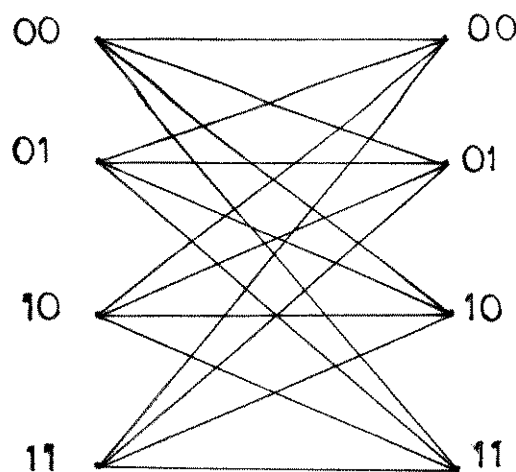


Fig. 4.3 - Diagrama de Treliça para uma janela de tempo.

A medida que aumentamos o número de janelas de tempo, devemos verificar quantos caminhos possíveis, com a menor distância minima total saem do estado 00 na primeira janela de tempo e retornam para este estado 00 após m janelas de tempo.

Isto é, para cada valor fixado do número de janelas de tempo (m) o algoritmo de Viterbi fornecerá o d_{min} e a quantidade de caminhos correspondentes à este d_{min}.

Armazena-se estes valores e repete-se este propcedimento para todas as janelas de tempo intermediárias até que

se atinja o comprimento de restrição do código. Dessa forma, obtemos uma tabela da forma apresentada na Fig. 4.4 abaixo.

Quantidade de Janelas	D_{min}	Quantidade de Caminhos
2	6	2
3	7	2
4	9	3
5	10	4
6	11	2
7	12	2
8	13	2
9	14	2
10	15	2
11	16	2
12	17	2
13	18	2
14	19	2
15	20	2

Fig. 4.4 - Quadro associado ao código (5,2,1).

O menor d_{min} encontrado nesta tabela, deve ser igual ao d_o escolhido, caso estejamos testando a veracidade dos resultados obtidos na seção 4.2, e ainda mais, o menor d_{min} encontrado na coluna dos d_{min} da tabela apresentada pela Fig. 4.4 , fornece um limitante para o código dado chamado d_{free} , ou distância livre do código.

A importância em conhecermos a distância livre do código reside no fato da probabilidade de erro depender fortemente deste valor.

EXEMPLO : Para o código (5,2,1), apenas como ilustração do comentário realizado acima, encontramos a Fig. 4.4 .

Teremos, portanto, d_{free} = 6.

A título de ilustração do processo de

enumeração utilizado no algoritmo, considere o seguinte exemplo:

EXEMPLO : Considere o código de memória unitária (n,k,1). A treliça parcial correspondente á este código é mostrada na Fig. 4.3. Assumiremos por conveniência que estamos interessados na enumeração das palavras códigos contidas no intervalo de até 10 janelas de tempo, para qualquer instante de tempo, j, fixado na treliça. Veja Fig. 4.5.

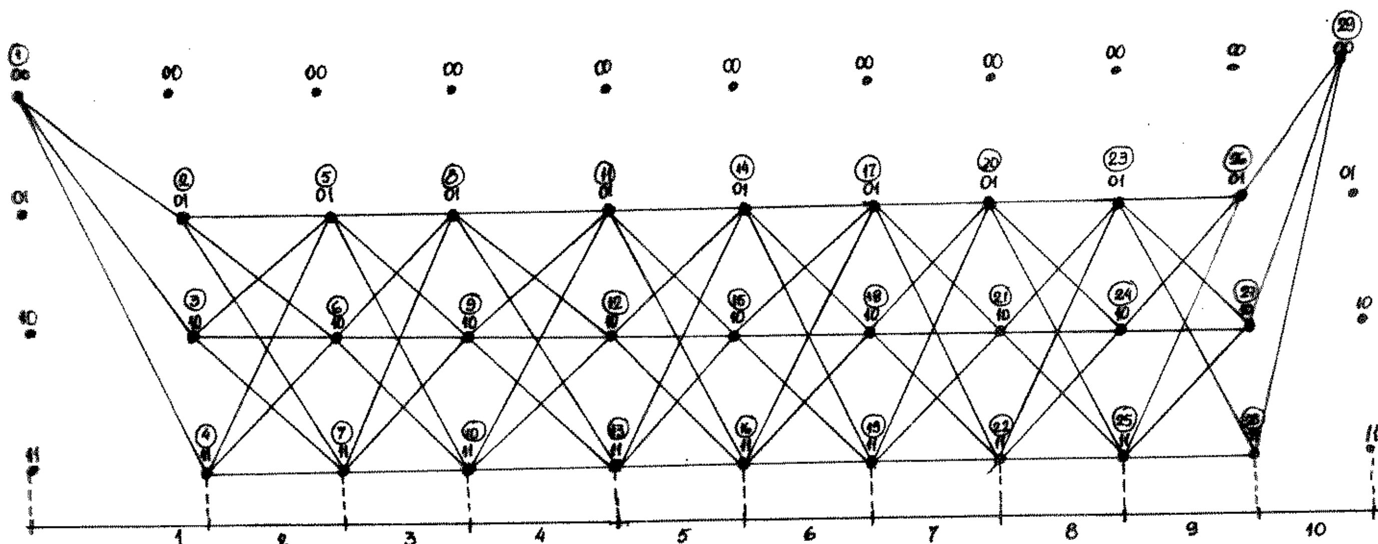


Fig. 4.5 - Treliça com 10 janelas de Tempo.

Como saímos do estado 00 e retornamos ao estado 00, então a janela 1 só possui as ligações saindo de 00 e a janela 10 só possui as ligações que retornam ao estado 00.

Uma vez que não pode existir um retorno ao estado final 00 antes de alcançarmos a janela 10, da janela 2 a janela 9 não existem ligações de estados conduzindo a um retorno

ao estado 00.

O número que se encontra dentro do círculo ao alto de cada estado ligado é o "estado" associado ao problema de Programação Dinâmica.

Uma vez que a técnica de Programação Dinâmica atua do último estágio ao primeiro estágio e estamos associando a idéia de estágio a janela de tempo, o processo iniciará na janela de tempo 10 e prosseguirá até alcançar a janela de tempo 1, determinando ,assim, a distância mínima do percurso.

Vemos que na janela de tempo 10, existem 3 ligações de estados ao 00, da janela 9 a janela 2 existem 3 estados ligados a 3 estados e, na janela 1 apenas uma ligação é feita, a do estado 00.

Da janela 2 a janela 9 vemos que as ligações irão se repetir, portanto, não existe a necessidade de armazenar todas as distâncias, mas apenas a de uma janela de tempo, pois podemos perceber, claramente, que a distância do estado 23 ao 26 será a mesma do estado 20 = 23 -(número de ligações) ao estado 23 = 26 - (número de ligações), onde o número de ligações é igual a 3 . Isto se repete nas janelas posteriores.

Aplicando a técnica de Programação Dinâmica apresentada no Capítulo 3 a treliça apresentada pela Fig. 4.6, encontramos em $f^*(1,1)$, o d_{min} do percurso associado a esta quantidade de janelas de tempo.

Repetimos o processo para vários valores de Janelas de Tempo e associamos ao menor d_{min} encontrado entre todas as verificações feitas, o limitante d_{free} .

CAPITULO 05

CAPÍTULO 5

DESCRIÇÃO DO ALGORITMO PROPOSTO E RESULTADOS

5.1 - INTRODUÇÃO

Neste Capítulo apresentaremos a implementação sistemática do algoritmo proposto no Capítulo 4.

Na seção 5.2, apresentaremos uma descrição geral do sistema. Este é o algoritmo propriamente dito, entretanto sem pormenores quanto a sua estrutura.

Na seção 5.3 apresentaremos as especificações das sub-rotinas dos programas relativos a descrição do sistema da seção 5.2 .

A seção 5.4 contém, em forma de fluxogramas, a descrição das variáveis, bem como o funcionamento do algoritmo já descrito no Capítulo 4.

Apresentaremos na seção 5.5 os resultados obtidos, através da utilização do algoritmo proposto, na sua forma original.

5.2 - DESCRIÇÃO GERAL DA IMPLEMENTAÇÃO COMPUTACIONAL DO ALGORITMO

O sistema que será apresentado abaixo, tem por finalidade principal apresentar a matriz geradora de um código de memória unitária, quando são fornecidos como dados de entrada as variáveis : q = ordem do Corpo de Galois, n = número de entradas do código e m = número de memórias do código.

Os resultados fornecidos no Capítulo 4, através de uma forma sistemática, nada mais são do que um exemplo do andamento do algoritmo aqui descrito.

Embora aparentemente simples e de aplicação imediata, devemos comentar o fato de que os blocos que constituem cada parte do algoritmo devem interagir de forma eficiente para que o andamento do mesmo, em sua totalidade seja conveniente. Assim sendo, apresentaremos a seguir os programas fundamentais para a formalização da idéia que tentamos unir dos capítulos precedentes.

PROGRAMA DADOS.PAS - Entrada de dados do sistema e estimativa do D_ZERO.

PROGRAMA KNAP_5.PAS - Resolve o knapsack através da técnica do Backtrack e fornece o vetor tipo.

PROGRAMA MAT_VIT.PAS - Constroi a matriz geradora, verifica a catastroficidade, encontra pesos das transições entre estados e processa o algoritmo de decodificação de Viterbi por Programação Dinâmica, apresenta a distância livre do código ou D_FREE.

Em forma de diagrama de blocos teríamos :

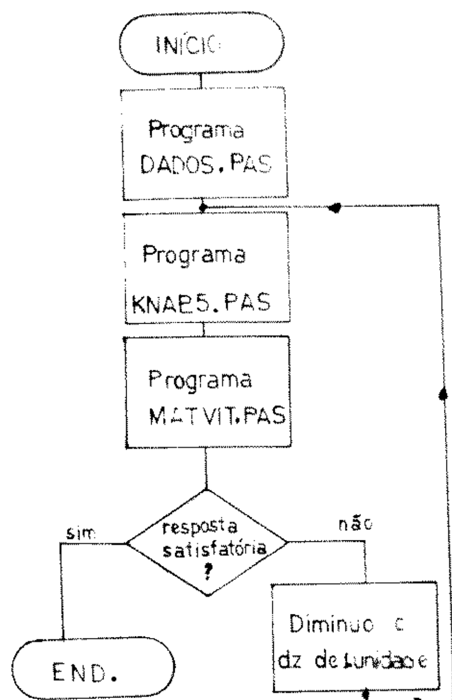


Fig. 5.2.1 - Diagrama de Blocos do Sistema.

Apresentaremos na seção 5.3 a seguir, as especificações computacionais relativas ao Programa Knap5.pas ficando os Programas Dados.pas e Matvit.pas nos apêndices A e B respectivamente.

5.3 - ESPECIFICAÇÕES COMPUTACIONAIS DO ALGORITMO

O algoritmo fornecido nesta tese, teve sua implementação computacional da seguinte forma :

Linguagem de Programação : Turbo-Pascal Versão 4.0;

Equipamento Utilizado : Sid 501 com co-processador numérico.

Apresentaremos a seguir as especificações computacionais do algoritmo passo-a-passo com respeito as variáveis de cada programa pertencente ao sistema.

5.3.1 - PROGRAM KNAP_5.PAS

A versão inicial que deu origem a este algoritmo, pode ser encontrada em [1] e conta com sua versão 1.4 alterada de acordo com as necessidades desta tese.

(A) NOME DO PROGRAMA : Knapsack_5

(B) COMANDO DE CHAMADA : Knap_5.pas

(C) PROPÓSITO DO PROGRAMA : O objetivo deste programa é enumerar todas as soluções que satisfaçam as restrições :

$$S = \begin{cases} \sum_{i=0}^{n-1} a_i d_i = \phi & (5.3.1.1) \\ \sum_{i=0}^{n-1} a_i = n & (5.3.1.2) \\ a \in \mathbb{N} = \{0,1,2,\ldots\}, \quad i = 0,1,\ldots,n-1 & (5.3.1.3) \end{cases}$$

Ou seja, encontrar todos os possíveis vetores a, já comentado no Capítulo 4, convenientes a formação do vetor espectro de Hamming, d , e o vetor tipo, tip.

(D) **COMENTÁRIOS SOBRE O PROGRAMA** : Uma das maneiras de computacionalmente se gerar todas as soluções factíveis de S, é gerar todas as soluções inteiras positivas que satisfazem uma

das restrições e, numa segunda etapa, verificar quais satisfazem também a outra restrição. Para que este processo seja computacionalmente eficiente será necessário escolher a restrição que tenha um menor número de soluções ou, então, uma combinação linear das restrições que satisfaça esta condição.

Uma particularidade dos coeficientes da restrição (5.3.1.1) é que $d_i = d_0 + i$, de forma que :

$$\sum_{i=0}^{n-1} d_i\, a_i = d_0 \sum_{i=0}^{n-1} a_i + \sum_{i=0}^{n-1} i\, a_i = \phi$$

Substituindo (5.3.1.2) nesta equação obtemos :

$$\sum_{i=1}^{n-1} i\, a_i = \phi - n\, d_0 \qquad (5.3.1.4)$$

Pode-se mostrar que o número de soluções inteiras da (5.3.1.4) é muito menor que o número de soluções inteiras da (5.3.1.1).

Como existem duas restrições de igualdade, o vetor de soluções de $\underline{a}$ tem n-2 graus de liberdade. Fixando-se n-2 elementos de $\underline{a}$, os dois elementos restantes são determinados univocamente a partir das duas equações. Observando-se (5.3.1.4) pode-se notar que o coeficiente de a_0 é igual a 0 (uma variável a menos na enumeração das soluções) e o coeficiente de a_1 é igual a 1, logo se a_2 , a_3 , ... , a_{n-1}

são inteiros, então (5.3.1.4) é satisfeita por um valor inteiro de $\underline{a}$ dado por :

$$a_1 = \phi - n\, d_o - \sum_{i=2}^{n-1} i\, a_i \qquad (5.3.1.5)$$

A equação (5.3.1.1) tem propriedades análogas para a_o, de modo que podemos obter :

$$a_o = n\,(d_o + 1) - \phi + \sum_{i=2}^{n-1} (i-1)\, a_i \qquad (5.3.1.6)$$

Para se enumerar todas as soluções inteiras e positivas que satisfazem (5.3.1.4) , devemos garantir que $a_1 >= 0$, ou seja , queremos todos os valores inteiros e positivos de a_i , i>1, tal que

$$\sum_{i=2}^{n-1} i\, a_i \leq \phi - n\, d_o \qquad (5.3.1.7)$$

Iniciando o processo de enumeração por a_{n-1} , pode-se determinar o valor máximo que uma variável a_r pode assumir durante o processo de enumeração por :

$$r\, a_r \leq \left\{ \phi - n\, d_o - \sum_{i=r+1}^{n-1} i\, a_i \right\} - \sum_{i=2}^{r-1} i\, a_i$$

O termo a direita, fora da chave, é obviamente não negativo, logo podemos definir um limitante superior a_r , que é função de a_{r+1} , a_{r+2} , ... , a_{n-1} e que será denotado por u_r , na forma ($\lfloor\ \rfloor$ representa o

inteiro menor ou igual) :

$$a_r \leq u_r = \left\lfloor \left\{ \phi - n\, d_o - \sum_{i=r+1}^{n-1} i \; a_i \right\} / \; r \right\rfloor \qquad (5.3.1.8)$$

A partir desta definição de u_r, o processo de enumeração passa a ser simplesmente a geração de todos os valores de a que satisfazem $0 <= a_i <= u_i$, começando a enumeração a partir de a_{n-1}.

Para cada solução que satisfaça (5.3.1.8) é necessário verificar se o valor de a_o correspondente (5.3.1.6) não é negativo. Se não for, então a solução pertence a S.

A restrição $a_o \geq 0$ pode ser incluída diretamente ao processo de enumeração, pela sua substituição na restrição (5.3.1.6) :

$$\sum_{i=2}^{n-1} (i-1) \; a_i \geq \phi - n \, (d_o + 1)$$

Reagrupando os termos, temos :

$$(r - 1)\, a \geq \left\{ \phi - n\, (d_o + 1) - \sum_{i=r+1}^{n-1} (i - 1)\, a_i \right\} - \sum_{i=2}^{r-1} (i - 1)\, a_i$$

Um limitante superior para o último termo pode ser calculado resolvendo-se o problema :

$$\text{MAX} \quad Z = \sum_{i=2}^{r-1} (i-1)\, a_i$$

s.a.

$$\sum_{i=2}^{r-1} i \; a_i \leq \phi - n\, d_o - \sum_{i=r}^{n-1} i \; a_i$$

A solução deste problema é :

$$z^* = \frac{(r-2)}{(r-1)} \left[\phi - n\, d_o - \sum_{i=r}^{n-1} i \; a_i \right]$$

Substituindo este resultado, chegamos a :

$$a_r \geq l_r = \phi - n\,(d_o - r - 1) - \sum_{i=r+1}^{n-1} (i + 1 - r)\; a_i$$

(5.3.1.9)

Este limitante inferior de $\underline{a}$ é definido de forma semelhante a u_r, o que permite usar o mesmo processo de enumeração, mas usando valores inteiros de a_i tal que $l_i \leq a_i \leq u_i$, para i= 2,3, ..., n-1. Os valores de a_o e a são calculados por (5.3.1.5) e (5.3.1.6) e, pelas condições impostas acima, devem ser inteiros e não-negativos.

(E) DESCRIÇÃO DOS ARQUIVOS ACRESCENTADOS PARA A CONVENIÊNCIA DO PROGRAMA AO OBJETIVO DA TESE

NOME	TIPO	DESCRIÇÃO
VET_A.DAT	text	armazena coordenada i do vetor a, que possui a quantidade solut[i], isto é, armazena o resultado do knapsack na forma de índice-quantidade
TIPO.DAT	text	armazena o vetor tipo que é dado por code[1,i] div code_mod.
IN_TESE.DAT	text	arquivo que armazena code_in (= entradas), code_out (= saídas) e o d_o que proporcionou solução conveniente.

5.4 - RESULTADOS

Apresentaremos, a seguir, os resultados obtidos através do algoritmo proposto , um na sua forma de saída original do programa e os demais nas tabelas I e II.

Como resultado do algoritmo proposto, a seguir apresentamos nas Tabelas I e II uma amostra dos códigos obtidos.

A matriz geradora G é apresentada na forma de suas duas sub-matrizes G_0 e G_1, em representação octal.

Por exemplo, o código com taxa 2/8 tem sua matriz geradora $G = [G_0 , G_1]$ representada por

$$G_0 = \begin{vmatrix} 274 \\ 616 \end{vmatrix} \qquad G_1 = \begin{vmatrix} 076 \\ 706 \end{vmatrix}$$

tal que (274) ≅ 010 111 100 ≅ 010 111 10

TABELA I

TAXA	D_{FREE}	VETOR ESPECTRO	MATRIZ GERADORA
[3] 2/3	3	(4 , 5, 3)	G_0 - 56 G_1 - 73
[2] 2/4	5	(5, 6, 5)	G_0 - 60 54 G_1 - 74 14
[4] 2/5	6	(6 ,7 ,7)	G_0 - 34 46 G_1 - 36 46
[2] 2/6	8	(8, 8, 8)	G_0 - 35 47 G_1 - 17 63
[4] 2/7	9	(9, 9, 10)	G_0 - 170 634 G_1 - 174 614
[2] 2/8	10	(10, 10, 10)	G_0 -274 616 G_1 -076 706

[5] 2/9	12	(12, 12, 12)	G_0 - 176 617 G_1 - 077 707
[5] 2/10	13	(13, 13, 14)	G_0 - 0770 7074 G_1 - 0774 7034
[6] 2/11	14	(14, 15, 15)	G_0 - 1770 6076 G_1 - 0776 7036
2/12	16	(16, 16, 16)	G_0 - 0776 7037 G_1 - 0377 7417
2/13	17	(18, 17, 17)	G_0 - 07760 70374 G_1 - 03774 74174
2/14	18	(20, 18, 18)	G_0 - 03774 74176 G_1 - 01776 76076
2/15	20	(20, 20, 20)	G_0 - 03776 74077 G_1 - 01777 76037
2/16	21	(22, 21, 21)	G_0 - 037760 740774 G_1 - 017774 760374
2/17	22	(24, 22, 22)	G_0 - 017774 760376 G_1 - 007776 770176
2/18	24	(24, 24, 24)	G_0 - 017776 760177 G_1 - 007777 770077
2/19	25	(26, 25, 25)	G_0 - 0177760 7601774 G_1 - 0077774 7700774

2/20	26	(28, 26, 26)	G_0 - 0077774 7700776 G_1 - 0037776 7740376

TABELA II

	TAXA	D_{FRE}	VETOR ESPECTRO	MATRIZ GERADORA
[3]	3/4	4	(5, 4, 5, 4 ,5, 4, 5)	G_0 - 14 24 70 G_1 - 44 50 60
[4]	3/5	5	(5, 5, 6, 5,6, 8, 5)	G_0 - 30 42 54 G_1 - 16 62 24
[2]	3/6	6	(6, 6, 6, 6, 6, 8,10)	G_0 - 30 51 16 G_1 - 37 43 45
[3]	3/7	8	(8, 8, 8, 8, 8, 8, 8)	G_0 - 740 170 314 G_1 - 524 344 710
[3]	3/8	8	(8 ,8, 8,9, 9, 11,11)	G_0 - 170 610 236 G_1 - 076 706 114

apresentando resultados obtidos para os seguintes dados :

```
TAXA                      : r = 2 / 13
ORDEM DO CORPO            : 2
MEMORIA                   : 1
QUANTIDADE DE ELEMENTOS : 3
FLUXO MAXIMO              : 52
MAIOR DISTANCIA MINIMA  : 17

SOLUCAO :
VETOR ESPECTRO DE HAMMING :
( 18 17 17 )

VETOR TIPO                :
( 9 , 8 , 9 )

VETOR TIPO MATRIZ GERADORA :
( 3 , 4 , 4 , 6 , 4 , 5 )


************  MATRIZ GERADORA ***********

0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1


******* VITERBI POR PROG. DINAMICA ******

JANELAS DE TEMPO  DISTANCIA MINIMA  QTD. DE CAMINHOS
------------------------------------------------------
        2                   17                  2
        3                   19                  1
        4                   22                  1
        5                   24                  1
D_free = 17
```

OBS: saída original do programa.

CAPITULO 06

CAPÍTULO 6

CONCLUSÃO

Visto a existencia da dificuldade em se sistematizar um procediemeto de busca de códigos convolucionais ótimos, devido a falta de uma estrutura matemática bem definida, convém ressaltar a importância dos resultados aqui alcançados, numa tentativa inédita e com sucesso, da apresentação de um algoritmo que busca sistematizar, através de técnicas de otimização, o problema da obtenção de códigos convolucionais ótimos através de códigos de memória unitária invariantes no tempo.

Embora a eficiência do programa elaborado para a tese, descrito nos capítulos 4 e 5 bem como apresentado nos apêndices A, B e C, esteja condicionado a taxas relativamente altas, convém ressaltar o fato de que este problema poderá ser contornado através de técnicas de Processamento Paralelo, evitando , dessa forma, a demora na verificação de soluções ótimas e podendo , *portanto, elevar a quantidade de taxas ótimas conhecidas.* Fica assim, além dessa sugestão para trabalhos futuros, a utilização deste trabalho para aplicações em códigos variantes no tempo; em determinação de proteção desigual de erros; na utilização em sistemas concatenados; em aplicações a problemas de codificação genética, etc...

Convém ressaltar a importância do algoritmo na apresentação de um código ótimo para qualquer taxa desejada, na aplicação à otimização de transmissão em faixas limitadas. Visto que a faixa de transmissão é definida préviamente e, a razão entre a faixa de transmissão e a faixa de recepção deva ser o mais próximo de 1, encontramos a faixa de recepção desejada e, uma vez que a faixa de recepção é determinada pelo inverso da taxa do código, podemos encontrar a taxa do código que tornaria a transmissão o mais eficiente possível e, assim, através do algoritmo proposto, encontrarmos a matriz geradora do código ótimo para esta faixa.

No Capítulo 3 foi apresentadaa toda a

formulação matemática necessária a utilização da Técnica do Backtrack aplicada ao problema do Knapsack , bem como apresentamos os conceitos necessários à aplicação do Branch-and-Bound, por termos a certeza de que será o método mais apropriado para um trabalho futuro. Infelizmente não pudemos fazer uso de tal método, por se tratar de um trabalho pioneiro, sendo portanto necessária a prudência na verificação de todas as possíveis soluções a uma má escolha da função objetivo. Sendo assim, preferimos optar pelo Backtrack usual, neste trabalho.

Foi apresentado também a enumeração das palavras-código através da utilização de Programação Dinâmica no Algoritmo de Viterbi. Dessa forma apresentamos a quantidade de caminhos que apresentam a maior distância mínima, encontrados para uma quantidade de janelas de tempo pré-determinadas.

Convém ressaltar que a implementação computacional deste algoritmo foi feita em um micro-computador, o que torna o algoritmo de fácil utilização uma vez que, para pequenas taxas, não é necessário um computador mais sofisticado.

O programa permite também algumas alterações de forma a efetuar tentativas de obtenção do mesmo algoritmo mas para códigos variantes no tempo, o que fica como sugestão para um trabalho futuro.

APÊNDICE A : PROGRAMA DADOS.PAS

(A) NOME DO PROGRAMA.........: Dados.Pas;

(B) COMANDO DE CHAMADA.......: Dados.pas;

(C) PROPÓSITO DO PROGRAMA....: Entrada de dados do sistema e cálculo do D_ZERO estimado,d0 ;

(D) SUB-ROTINAS DO PROGRAMA..: ENT_DADOS e CALC_D_ZERO ;

(E) DESCRIÇÃO DAS VARIÁVEIS..:

NOME	TIPO	DESCRIÇÃO
Q	integer	número de elementos do Corpo de Galoi
B	integer	número de entradas do codificador
N1	integer	número de saídas do codificador
M	integer	número de memórias (no caso m = 1)
N	integer	quantidade de estados com excessao do todo nulo
FI	integer	fluxo na rede
d_o	integer	valor de D_ZERO estimado
AU_o	integer	variável auxiliar em cálculo comum
I	integer	variável auxiliar em cálculo comum
P_1	integer	variável auxiliar em cálculo comum
P	integer	variável auxiliar em cálculo comum
MIN	real	valor mínimo para o d0
D[ ]	vetor real	auxiliar no cálculo do d0

(F) DESCRIÇÃO DOS ARQUIVOS DO PROGRAMA

NOME	TIPO	DESCRIÇÃO
DAD.DAT	TEXT	Armazena dados de entrada ($q, m, b. n_1$, 2*fi, n)
ENT_KNAP.DAT	TEXT	Armazena entrada de dados para o programa Knap_5.pas (b, n_1, d_0)

(A) **NOME DO PROGRAMA...........:** MAT.VIT.PAS

(B) **COMANDO DE CHAMADA.........:** Mat.Vit.pas

(C) **PROPÓSITO DO PROGRAMA......:** A partir dos dados recebidos do Programa Knap_5.pas, encontrar a matriz geradora do código ótimo e fazer a verificação do resultado obtido, efetuando o processo de decodificação de Viterbi por Programação Dinâmica. Caso o código seja não catastrófico imprimir resultados.

(D) **SUB-ROTINAS DO PROGRAMA......:** TRANSF, CONST_GERA, MULT_G_U, SOMA_VET, IMPRI_PESOS, PESOS, TROCA_COL, INVERTE, MÍNIMO, D_FREE , TESVI , AP_RESULT

1 - SUB-ROTINA TRANSF

(A) **PROPÓSITO DA SUB-ROTINA.....:** Transformar um número em binário.

(B) **DESCRIÇÃO DAS VARIÁVEIS....:**

NOME	TIPO	DESCRIÇÃO
t	integer	auxiliar em cálculo
j1	integer	número a ser trasformado em binário
Y[·]	vetor real	armazena o número em binário
ax4	integer	variável auxiliar no procedimento de transformar em binário

2 - SUB-ROTINA CONST_GERA

(A) PROPÓSITO DA SUB-ROTINA.....: Construir a matriz geradora do código.

(B) DESCRIÇÃO DAS VARIÁVEIS.....:

NOME	TIPO	DESCRIÇÃO
l	integer	variável contadora
j1	integer	variável local auxiliar em cálculo
j3	integer	variável local auxiliar em cálculo
j2	integer	variável local auxiliar em cálculo
dd[•]	vetor real	contém os tipos para formar a geradora
k[•, •]	matriz integer	matriz geradora do código

3 - SUB-ROTINA MULT_G_U

(A) PROPÓSITO DA SUB-ROTINA.....: Fazer a multiplicação da seqüência de informação, u , pela matriz matriz geradora k

(B) DESCRIÇÃO DAS VARIÁVEIS.....:

NOME	TIPO	DESCRIÇÃO
i	integer	variável local auxiliar em cálculo
j	integer	variável local auxiliar em cálculo
vs	integer	resultado da multiplicação do vetor u pela coluna da matriz k
p	integer	indica as colunas da matriz geradora ,k[•,•], que serão utilizadas
k[•,•]	matriz integer	matriz geradora
u[•]	vetor integer	seqüência de informacão passada por parâmetro
v1[•]	vetor integer	armazena o resultado da multiplicação u .K módulo 2
r1	real	variável auxiliar que armazena a divisão v1/2

OBS. : A matriz geradora K[K1 ,K2] formada por duas sub-matrizes a serem denominadas K1[• ,•] e K2[• ,•] .

4 - SUB-ROTINA SOMA_VET

(A) PROPÓSITO DA SUB-ROTINA.....: Efetuar a soma de v_1 com v_2, módulo 2, onde v_1 e v_2 são vetores dados por $v_1 = u_0 \; K_1$ e $v_2 = u_1 \; K_0$.

(B) DESCRIÇÃO DAS VARIÁVEIS.....:

NOME	TIPO	DESCRIÇÃO
i	integer	variável local auxiliar na indexação do vetor v .
v[•]	vetor integer	vetor binário resultante da soma de v_1 com v_2 .
r1	integer	variável que armazena a divisão do vetor v por 2

5 - SUB-ROTINA IMPRI_PESOS

(A) PROPÓSITO DA SUB-ROTINA......: O propósito desta sub-rotina é o de apresentar o peso de Hamming, de dada transição, no vídeo (ou na impressora, com pequena alteração, se de necessidade do usuário).

(B) DESCRIÇÃO DAS VARIÁVEIS......:

NOME	TIPO	DESCRIÇÃO
i	integer	auxiliar em cálculos
j	integer	auxiliar em cálculos
u_o[•]	vetor integer	vetor binário que indica o estado anterior, ou o estado de saída na transição.
u_1[•]	vetor integer	vetor binário que indica o estado atingido na transição ou estado atual
peso	integer	variável que contém, o peso de Hamming associado a transição

6 - SUB-ROTINA PESOS

(A) PROPÓSITO DA SUB-ROTINA : Calcular os pesos de Hamming associados a cada transição, verificando a existência de "loops", isto é, transições com estados de saída e chegada iguais, possuem peso de Hamming nulo e, conseqüentemente, catastroficidade do código. Esta sub-rotina efetua, também, o armazenamento dos pesos, nos arquivos convenientes, para uso posterior no algoritmo de Viterbi por Programação Dinâmica, apresentado pela sub-rotina TESVI.

(B) DESCRIÇÃO DAS VARIÁVEIS :

NOME	TIPO	DESCRIÇÃO
pes_dif.dat	text	armazena elementos de transições diferentes, isto é, que nem saiam do estado todo nulo ou que não retornem a ele.
pes_sai.dat	text	armazena pesos das transições que tem saída no estado todo nulo.
pes_ig.dat	text	armazena pesos das transições que possuem chegada no estado todo nulo.
ct_pes_dif	integer	contador de elementos de pes_dif.dat.
ct_pes_sai	integer	contador de elementos de pes_sai.dat
ct_pes_ig	integer	contador de elementos de pes_ig.dat
j1	integer	variável que assume valores de 0 até n, indicando o estado de saída.
ax4	integer	variável auxiliar para a transformar j1 em binário , com "b" algarismos

NOME	TIPO	DESCRIÇÃO
t4	integer	variável que atua como índice do vetor estado de saída , $u_o[\cdot]$.
t5	integer	variável que assume valores de 0 a n, indicando o estado de chegada na transição.
t6	integer	variável que atua como índice do vetor estado de chegada, $u_1[\cdot]$.
peso	integer	acumulador do peso da transição o corrida entre os estados j1 e t5.
j	integer	variável auxiliar no armazenamen to do peso. Varia de 1 a n1.
cat	integer	indicador de "catastroficidade" do código, quando cat=1.
cst[•]	vetor integer	auxiliar no armazenamento conve niente, para leitura posterior, do arquivo pes_dif.

7 - SUB-ROTINA TROCA_COL

(A) PROPÓSITO DA SUB-ROTINA : Realiza a troca de colunas na matriz geradora tentando, dessa forma, evitar a combinação linear entre as linhas da matriz geradora que provocariam , certamente, a catastroficidade do código.

(B) DESCRIÇÃO DAS VARIÁVEIS :

NOME	TIPO	DESCRIÇÃO
dl	integer	indica a coluna a ser trocada no vetor tipo da matriz geradora, ou seja, em dd[•].
ad	real	variável que indica a quantidade existente em dd[•], sem alteração

8 - SUB-ROTINA **MINIMO**

(A) PROPÓSITO DA SUB-ROTINA : Fazer o cálculo do caminho mínimo acumulado até o estado s , bem como a quantidade de caminhos que possuem tal valor mínimo e os estados anteriores, que chegando em s , fornecem tal caminho mínimo.

(B) DESCRIÇÃO DAS VARIÁVEIS :

NOME	TIPO	DESCRIÇÃO
j2	integer	variável local, contadora do índice do vetor sa[•]
fo[•]	vetor integer	vetor que armazena o ótimo, isto é, o valor associado ao caminho mínimo para o estado **s**
x[•]	vetor integer	vetor que possui a finalidade de armazenar o estado anterior ao estado atual (ou estado de saída), quando ct[•] = 1, que proporcionou o valor fo[•] para atingir o estado atual **s**; quando ct[•] >1, o vetor x[ ] armazena o índice de sa[•] a partir do qual estarão armazenados os estados anteriores que chegaram estado **s** com valor do caminho mínimo fornecido por fo[•].

NOME	TIPO	DESCRIÇÃO
j	integer	variável local auxiliar em cálculos
k	integer	variável local auxiliar em cálculos
ct[•]	vetor integer	contador da quantidade de caminhos que chegam ao estado **s** com peso mínimo
ax.dat	text	arquivo auxiliar que armazena os pesos acumulados, para um dado estado **s** de chegada
c[•]	vetor integer	guarda os pesos acumulados lidos de ax.dat, para maior facilidade de manipulação computacional na busca do mínimo
min	integer	menor valor acumulado até o estado **s**
sa[•]	vetor integer	vetor que armazena os caminhos, quando existem mais de um, que alcançaram o estado **s** atual com peso mínimo
s	integer	estado atual
i9	integer	variável contadora

9 - SUB-ROTINA AP_RESULT

(A) PROPÓSITO DA SUB-ROTINA : Enviar os dados de entrada e os resultados obtidos para a impressora.

(B) DESCRIÇÃO DAS VARIÁVEIS :

NOME	TIPO	DESCRIÇÃO
i	integer	variável local auxiliar em cálculos
j	integer	variável local auxiliar em cálculos
j1	integer	variável local auxiliar em cálculos
j2	integer	variável local auxiliar em cálculos
camin	integer	quantidade de caminhos lidos de viter.dat
x[•]	vetor integer	vetor local que armazena os elementos lidos de vet_a.dat
dm	integer	distância mínima lida de viter.dat

(A) **PROPÓSITO DA SUB-ROTINA** : Apresentar a distância livre do código, D_FREE , e compará-la com o d_o estimado, a fim de comprovar a catastroficidade, ou não, do código apresentado pela matriz geradora K[• , •].

(B) **DESCRIÇÃO DAS VARIÁVEIS** :

NOME	TIPO	DESCRIÇÃO
df[•]	vetor integer	vetor que armazena o valor para o menor caminho encontrado nas diferentes quantidades de janelas de tempo
min	integer	variável local que armazena o menor valor do vetor df[•]
fr	integer	contador de elementos do vetor df[.]
cn	integer	variável que verifica o comportamento das distâncias mínimas encontradas com relação as diferentes janelas de tempo, para evitar a catastroficidade do código

11 - SUB-ROTINA TESVI

(A) PROPÓSITO DA SUB-ROTINA : Testar a não catastroficidade da matriz geradora encontrada, K[. , .] , através do algoritmo de Viterbi por Programação Dinâmica.

(B) DESCRIÇÃO DAS VARIÁVEIS :

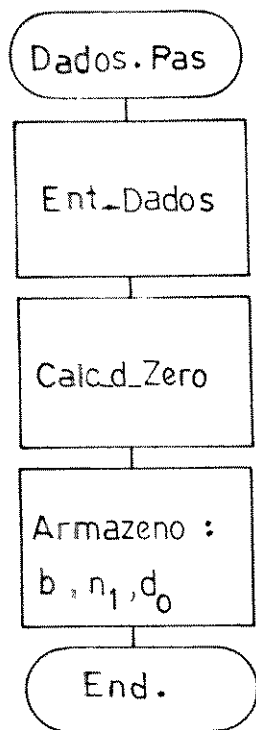
NOME	TIPO	DESCRIÇÃO
qj	integer	quantidade de janelas de tempo
viter.dat	text	arquivo que armazena distância mínima
jt	integer	número máximo de janelas de tempo
nv	integer	variável que armazena a quantidade inicial de janelas de tempo
j2	integer	variável índice do vetor sa[.]
ne	integer	número total de estados
etg	integer	estágio, ou janela de tempo, em que estou verificando o caminho mínimo
co	integer	contador de estados já verificados
qc	integer	quantidade calculada de elementos a serem lidos dos arquivos de pesos

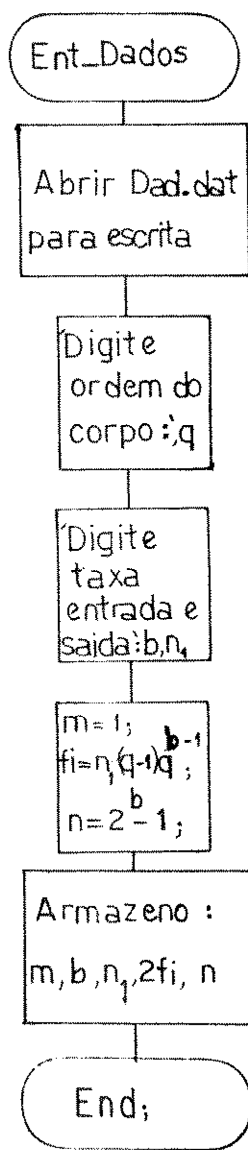
NOME	TIPO	DESCRIÇÃO
el	integer	quantidade de estados "ligados", ou seja, quantidade de possíveis transições na janela atual
ela	integer	quantidade de estados ligados na janela de tempo, ou estágio anterior
s	integer	estado atual ou lido de arquivo
j1	integer	estado anterior
j	integer	variável local auxiliar em cálculos
f1	integer	variável local auxiliar em cálculos
avet	integer	acumulador ótimo para o estado s
st	integer	valor lido de arquivo, que fornece o peso da transição
fo[·]	vetor integer	valor ótimo acumulado até o estágio j1
so	integer	somador de caminhos com a mesma distância mínima
gr1.dat	text	arquivo auxiliar na contagem de caminhos
gr2.dat	text	arquivo auxiliar na contagem de caminhos

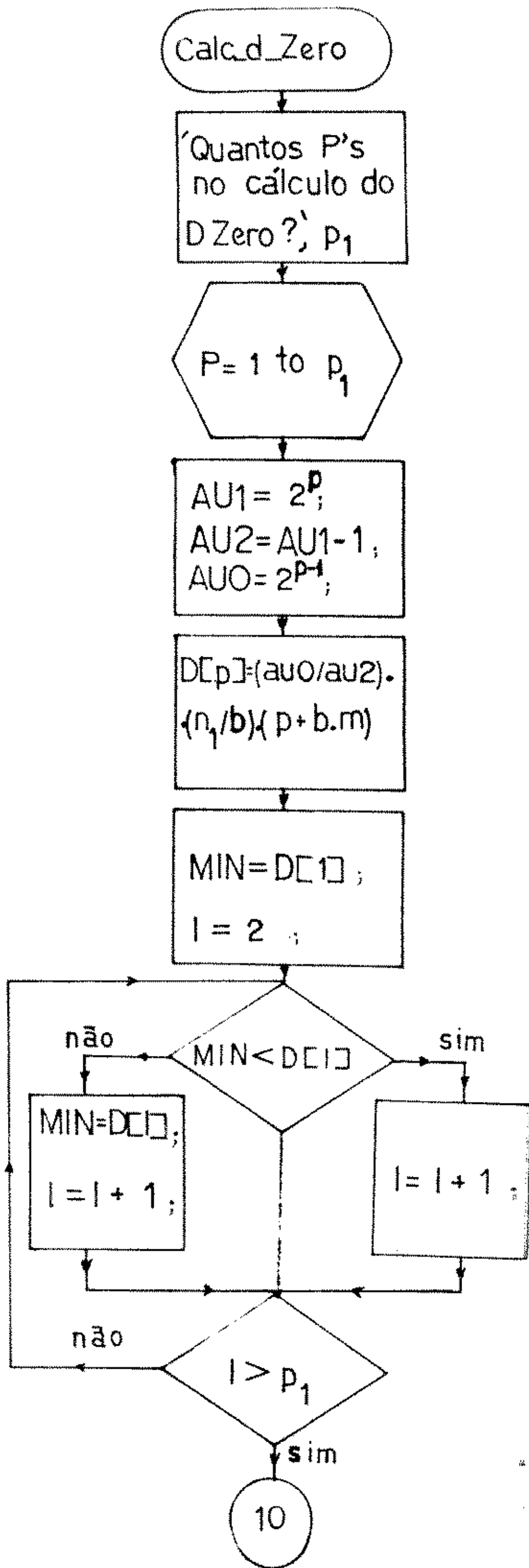
12 - ALGUMAS CONSIDERAÇÕES ÀS VARIÁVEIS DO PROGRAMA PRINCIPAL

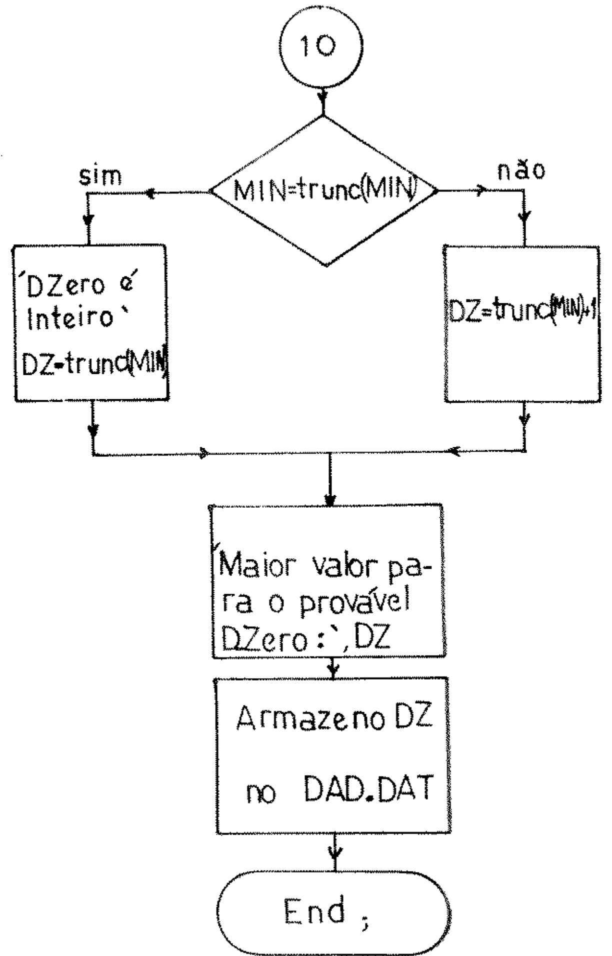
NOME	TIPO	DESCRIÇÃO
pa	integer	indicador de construção conveniente, ou não, da matriz geradora
c	integer	variável local auxiliar na leitura de arquivos
tip[•]	vetor integer	vetor tipo
dd[•]	vetor integer	vetor tipo de construção da matriz geradora

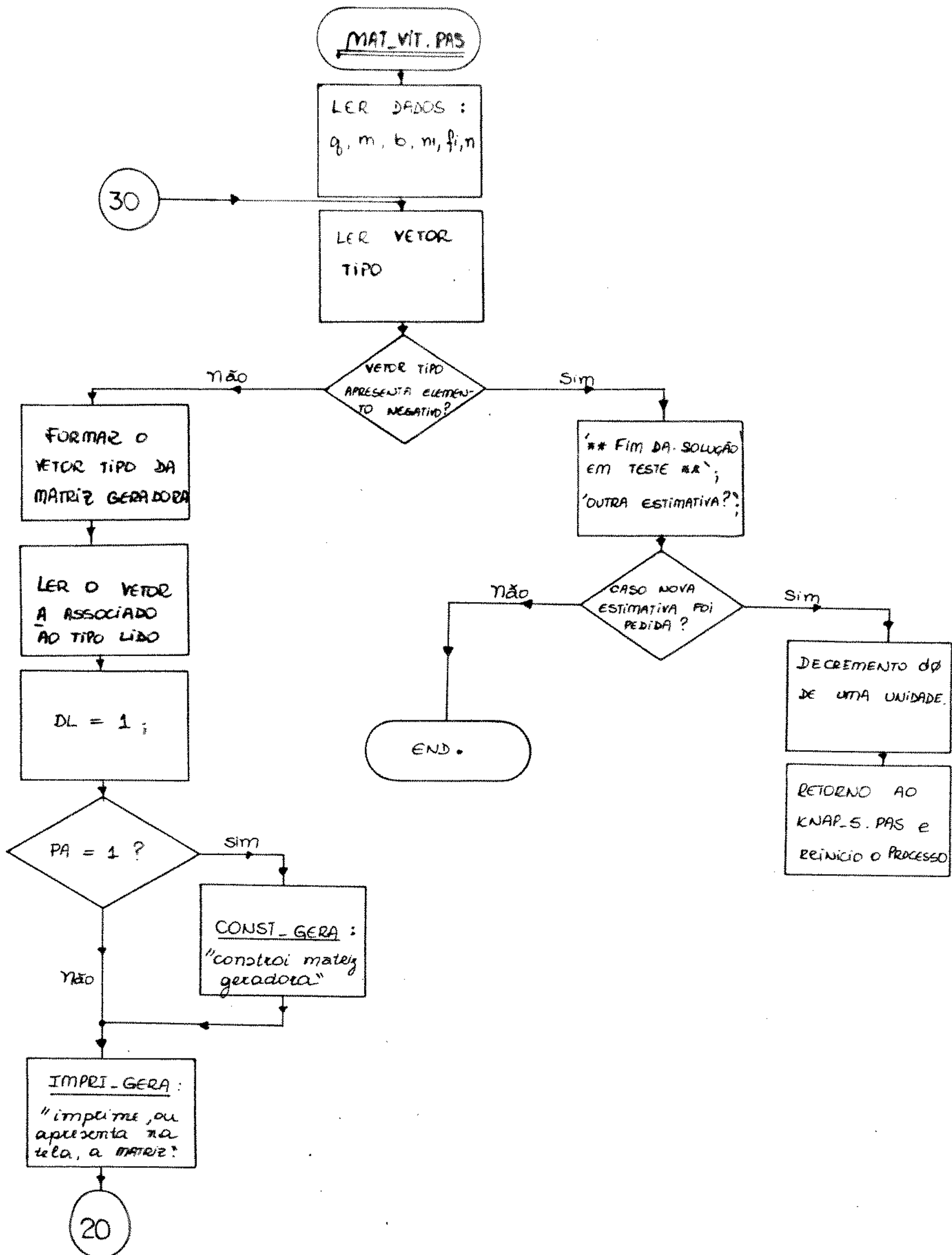
APENDICE C : FLUXOGRAMAS

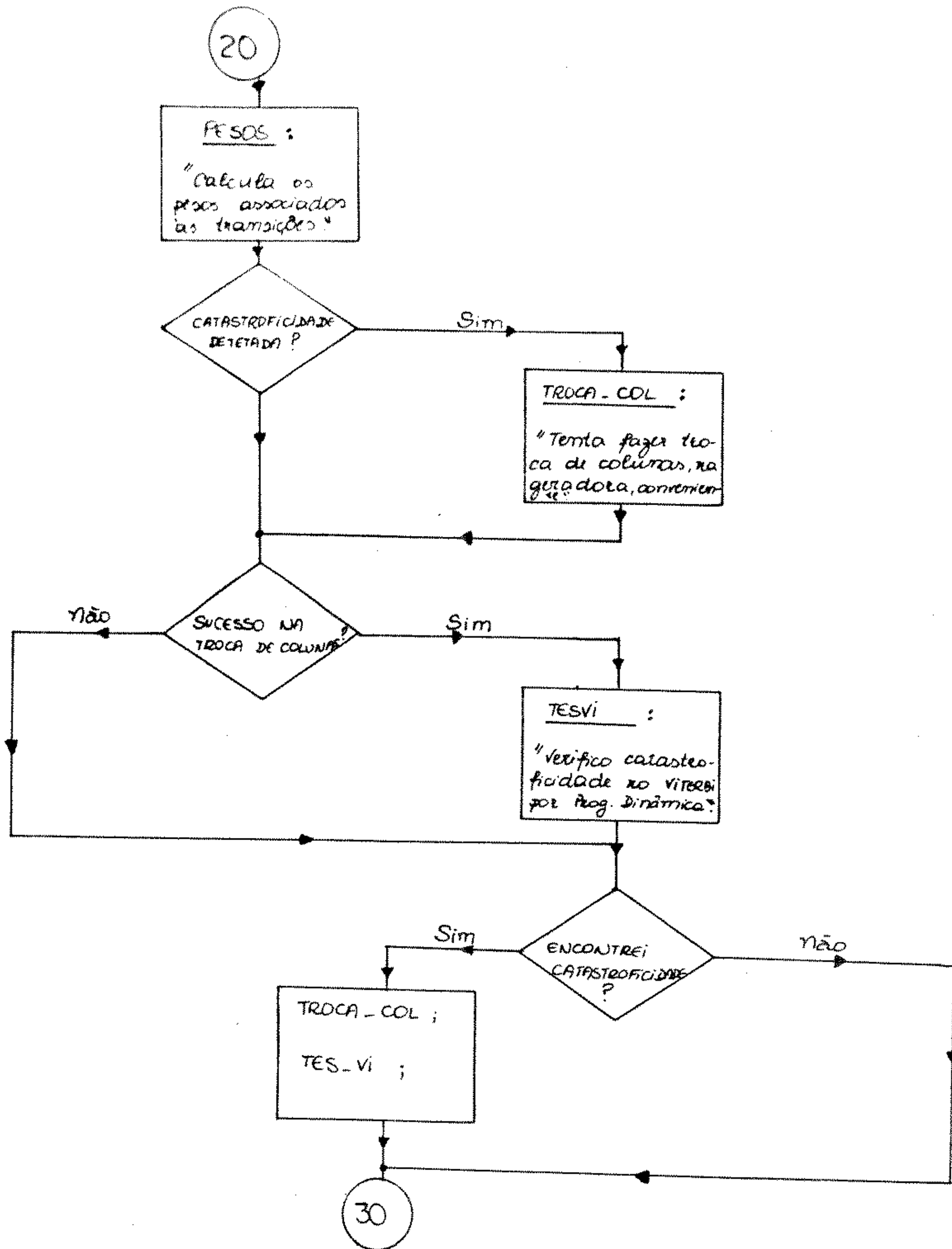

Dados.Pas
Ent_Dados
Calc_d_Zero
Armazeno :
b , n1 , d0
End.


Ent_Dados
Abrir Dad.dat para escrita
'Digite ordem do corpo:',q
'Digite taxa entrada e saida':b,n1
m=1;
fi=n1(q-1)q b-1;
n=2 b - 1;
Armazeno :
m,b,n1,2fi, n
End;

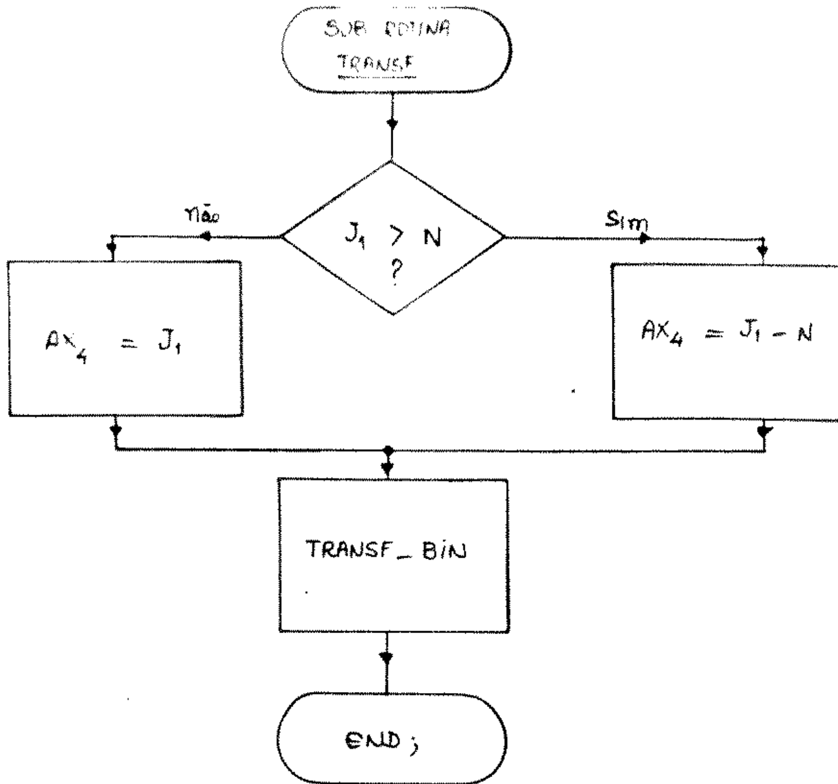

Calc_d_Zero
´Quantos P's no cálculo do D Zero?`, p1
P= 1 to p1
AU1= 2^p;
AU2=AU1-1;
AUO= 2^(p-1);
D[p]=(auO/au2).(n1/b).(p+b.m)
MIN=D[1];
I = 2;
MIN<D[I]
não
sim
MIN=D[I];
I=I+1;
I=I+1;
não
I>p1
sim
10

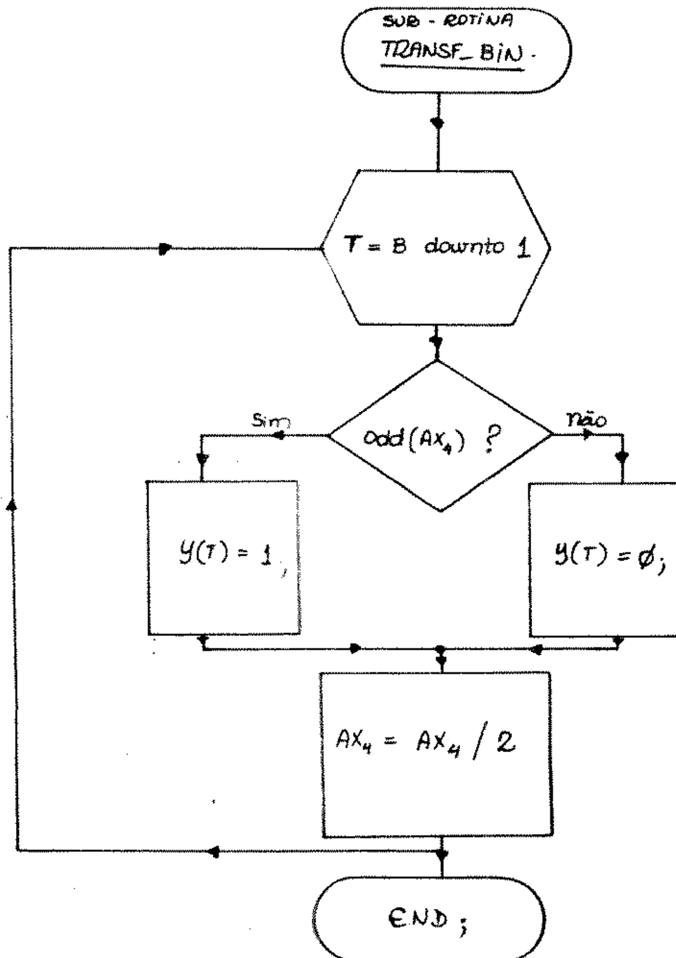

10
MIN=trunc(MIN)
sim
não
´DZero é Inteiro`
DZ=trunc(MIN)
DZ=trunc(MIN)+1
´Maior valor para o provável DZero:`,DZ
Armazeno DZ no DAD.DAT
End;

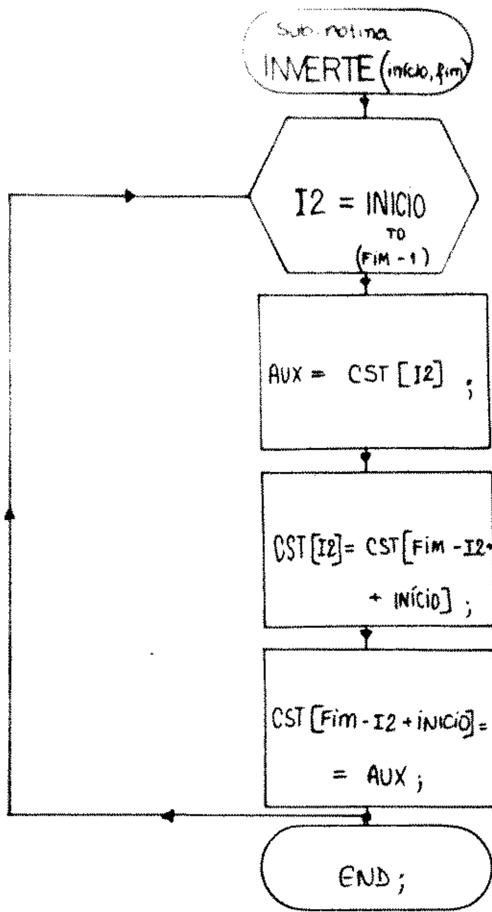

MAT_VIT.PAS
LER DADOS :
q, m, b, ni, fi, n
30
LER VETOR
TIPO
VETOR TIPO
APRESENTA ELEMENTO NEGATIVO?
não
sim
FORMAR O
VETOR TIPO DA
MATRIZ GERADORA
LER O VETOR
A ASSOCIADO
AO TIPO LIDO
DL = 1 ;
PA = 1 ?
sim
não
CONST_GERA :
"constroi matriz
geradora"
IMPRI_GERA :
"imprime, ou
apresenta na
tela, a matriz"
20
'** FIM DA SOLUÇÃO
EM TESTE **';
'OUTRA ESTIMATIVA?';
CASO NOVA
ESTIMATIVA FOI
PEDIDA ?
não
sim
END.
DECREMENTO do
DE UMA UNIDADE.
RETORNO AO
KNAP_5.PAS e
REINICIO O PROCESSO

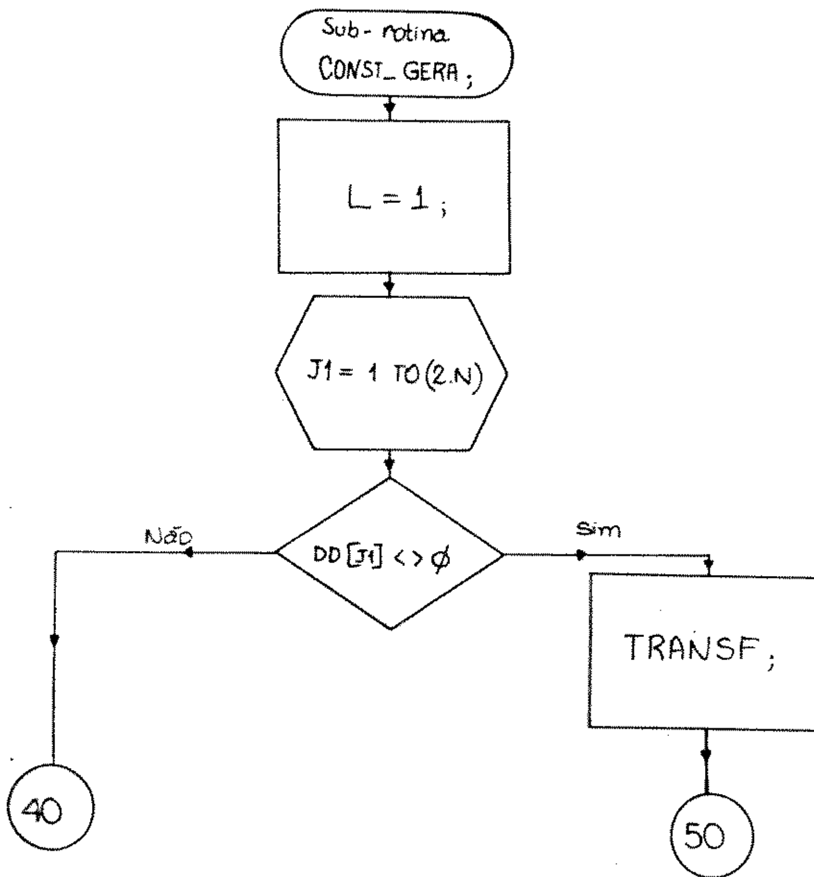


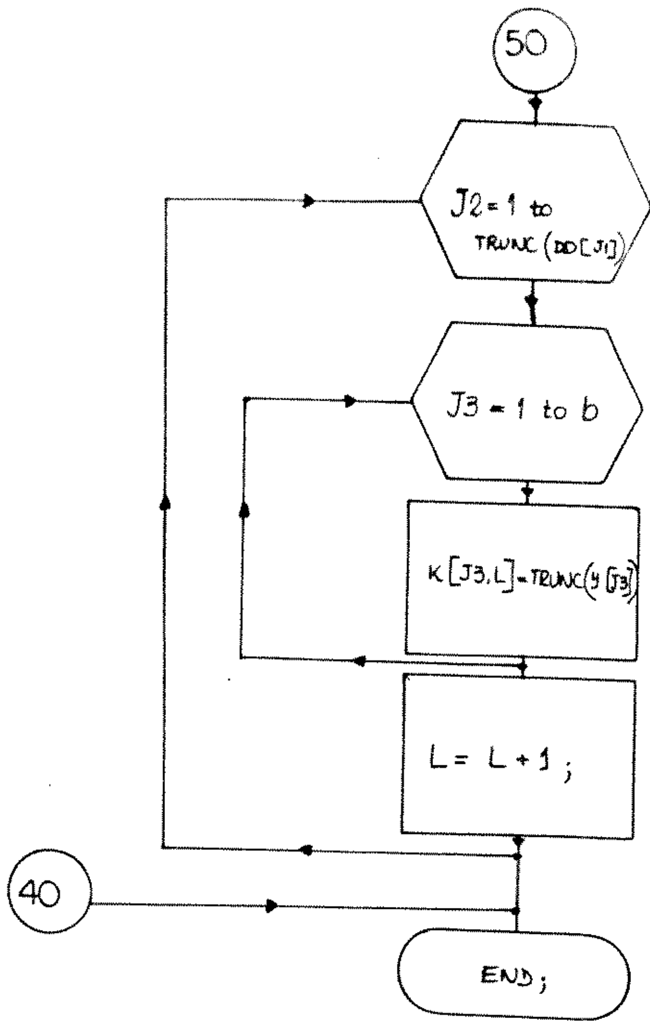
20
PESOS :
"Calcula os pesos associados às transições"
CATASTROFICIDADE DETETADA ?
Sim
TROCA_COL :
"Tenta fazer troca de colunas, na geradora, conveniente"
Não
SUCESSO NA TROCA DE COLUNAS?
Sim
TESVI :
"Verifico catastroficidade no VITERBI por Prog. Dinâmica"
Sim
ENCONTREI CATASTROFICIDADE ?
Não
TROCA_COL ;
TES_VI ;
30

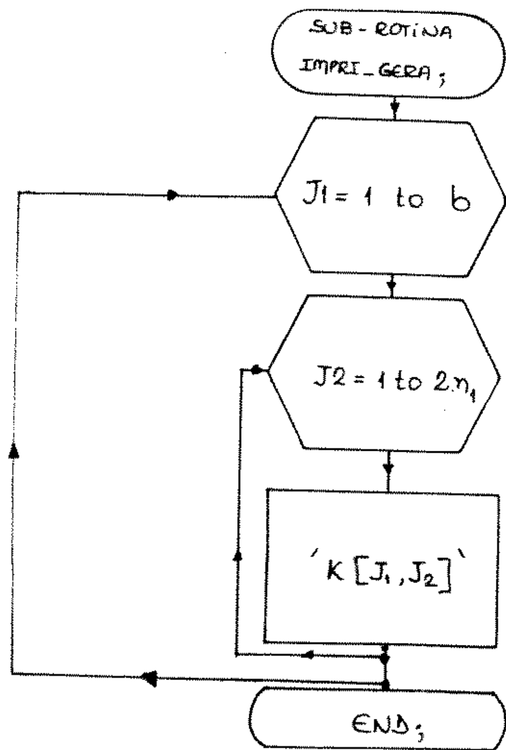

SUB ROTINA
TRANSF
$J_1 > N$
?
não
sim
$AX_4 = J_1$
$AX_4 = J_1 - N$
TRANSF_BIN
END;

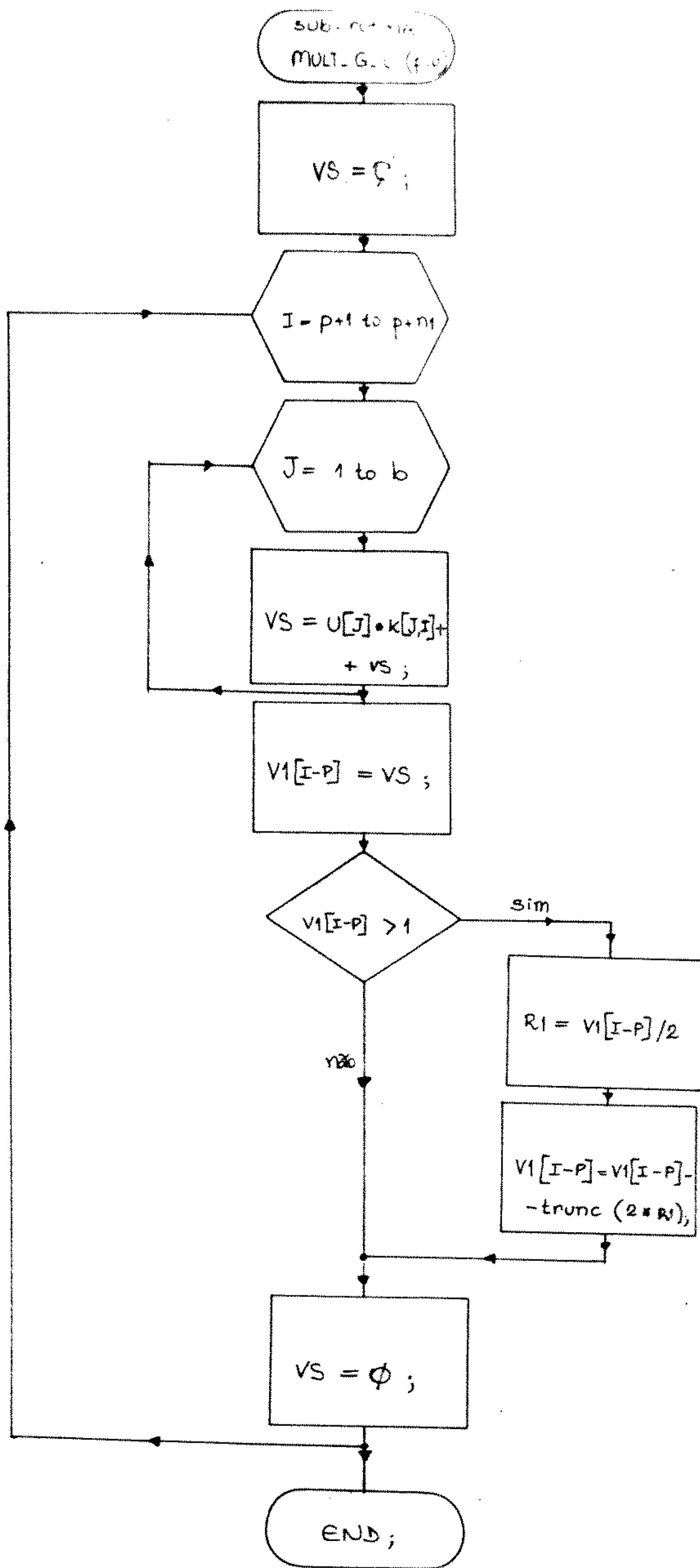

SUB - ROTINA
TRANSF_BIN.
T = B downto 1
odd(AX_4) ?
sim
não
$y(T) = 1$;
$y(T) = \emptyset$;
$AX_4 = AX_4 / 2$
END;


Sub-rotina
INVERTE (início, fim)
I2 = INICIO
TO
(FIM - 1)
AUX = CST [I2] ;
CST [I2] = CST [FIM - I2 +
+ INÍCIO] ;
CST [FIM - I2 + INICIO] =
= AUX ;
END ;

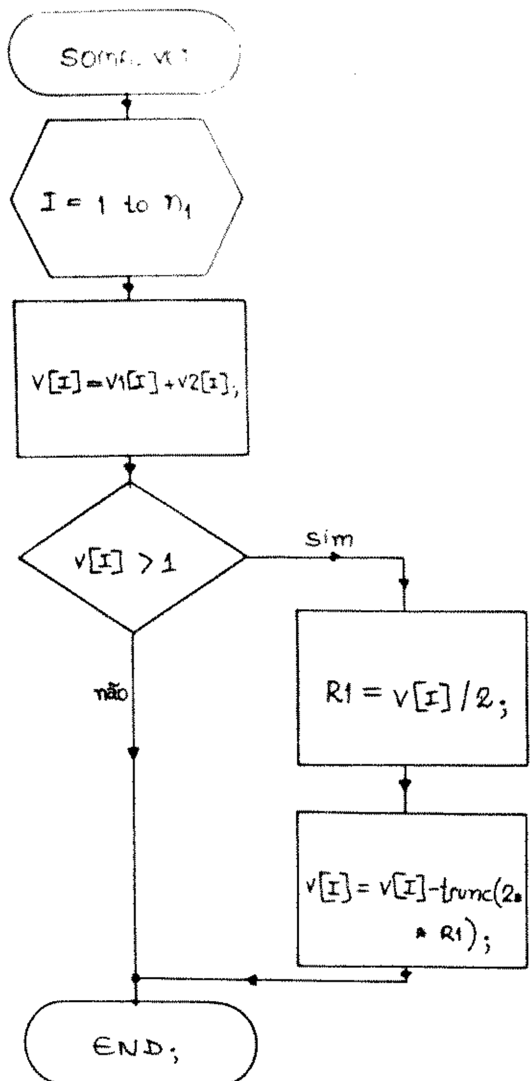

Sub-rotina
CONST_GERA ;
L = 1 ;
J1 = 1 TO (2.N)
DD [J1] <> Ø
Não
Sim
TRANSF ;
40
50

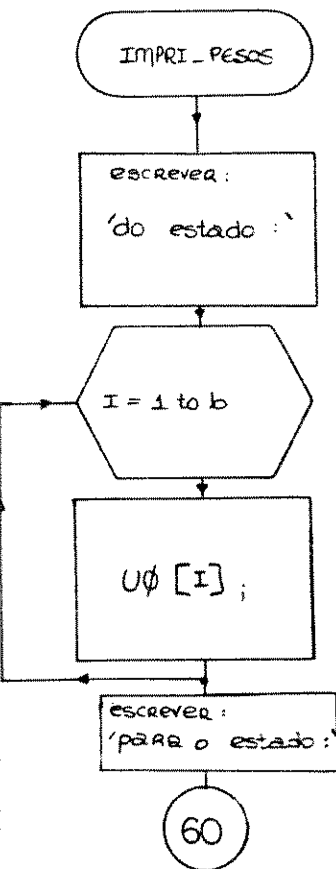

50
J2 = 1 to
TRUNC (DD[J1])
J3 = 1 to b
K [J3, L] = TRUNC (y [J3])
L = L + 1 ;
40
END;

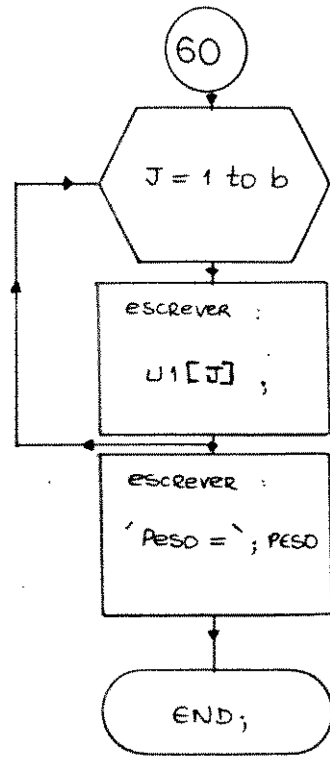

SUB - ROTINA
IMPRI_GERA ;
J1 = 1 to b
J2 = 1 to 2.n1
'K [J1, J2]'
END;

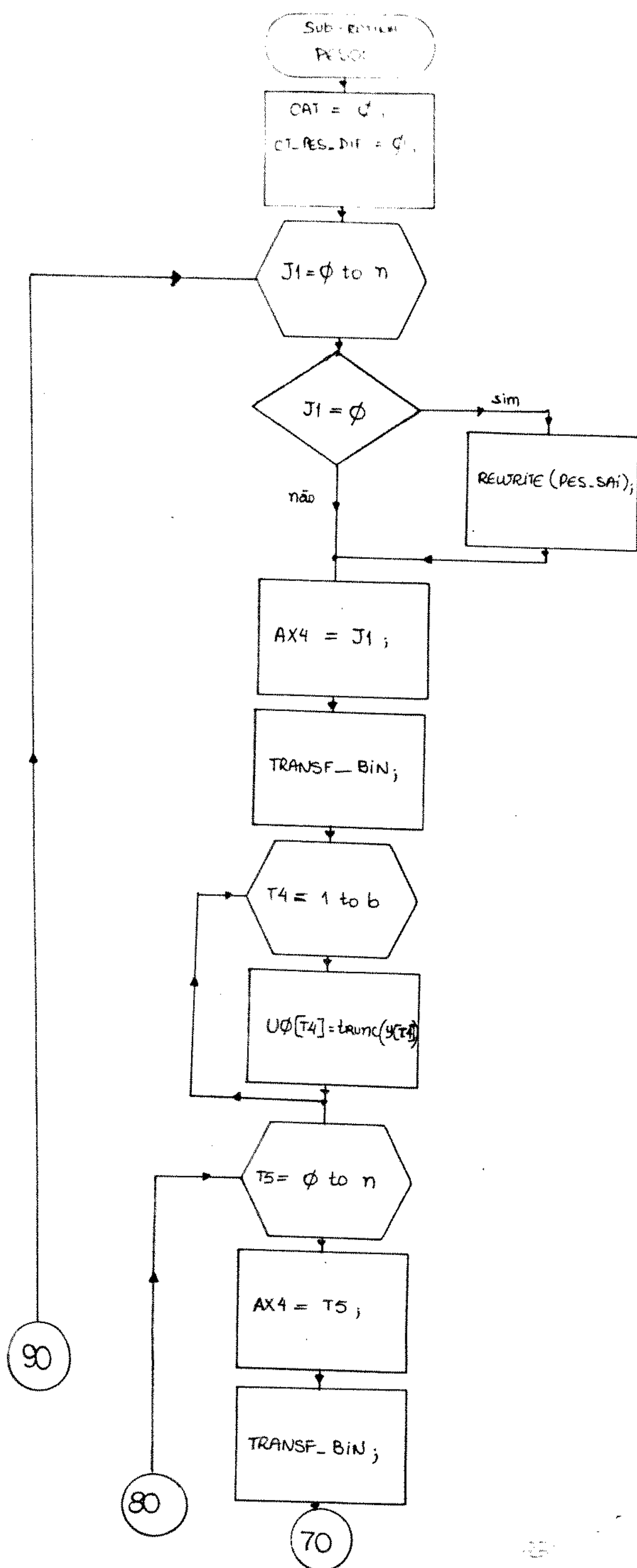


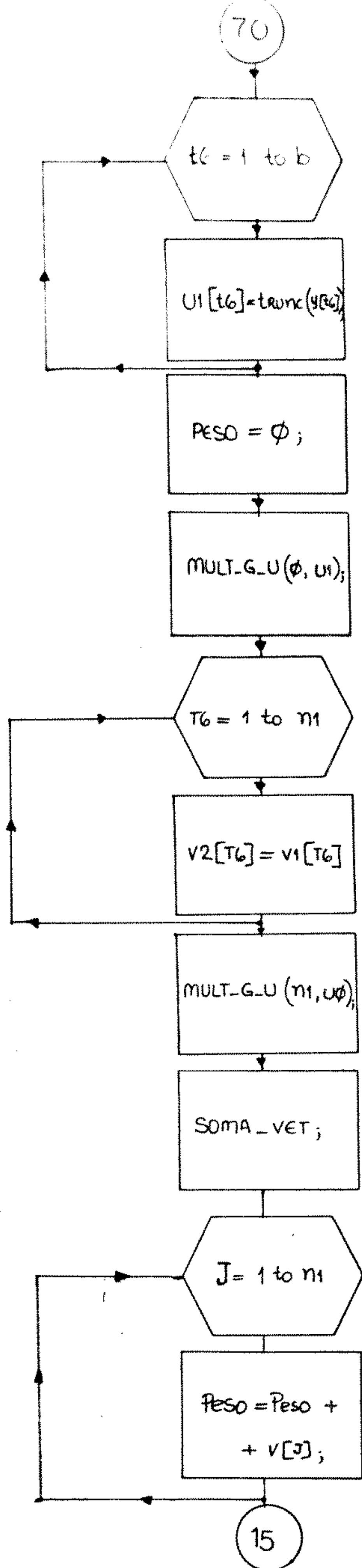
VS = Ç ;
I = p+1 to p+n1
J = 1 to b
VS = U[J] * K[J,I] + + VS ;
V1[I-P] = VS ;
V1[I-P] > 1
sim
não
R1 = V1[I-P]/2
V1[I-P] = V1[I-P] - - trunc (2 * R1),
VS = Ø ;
END;

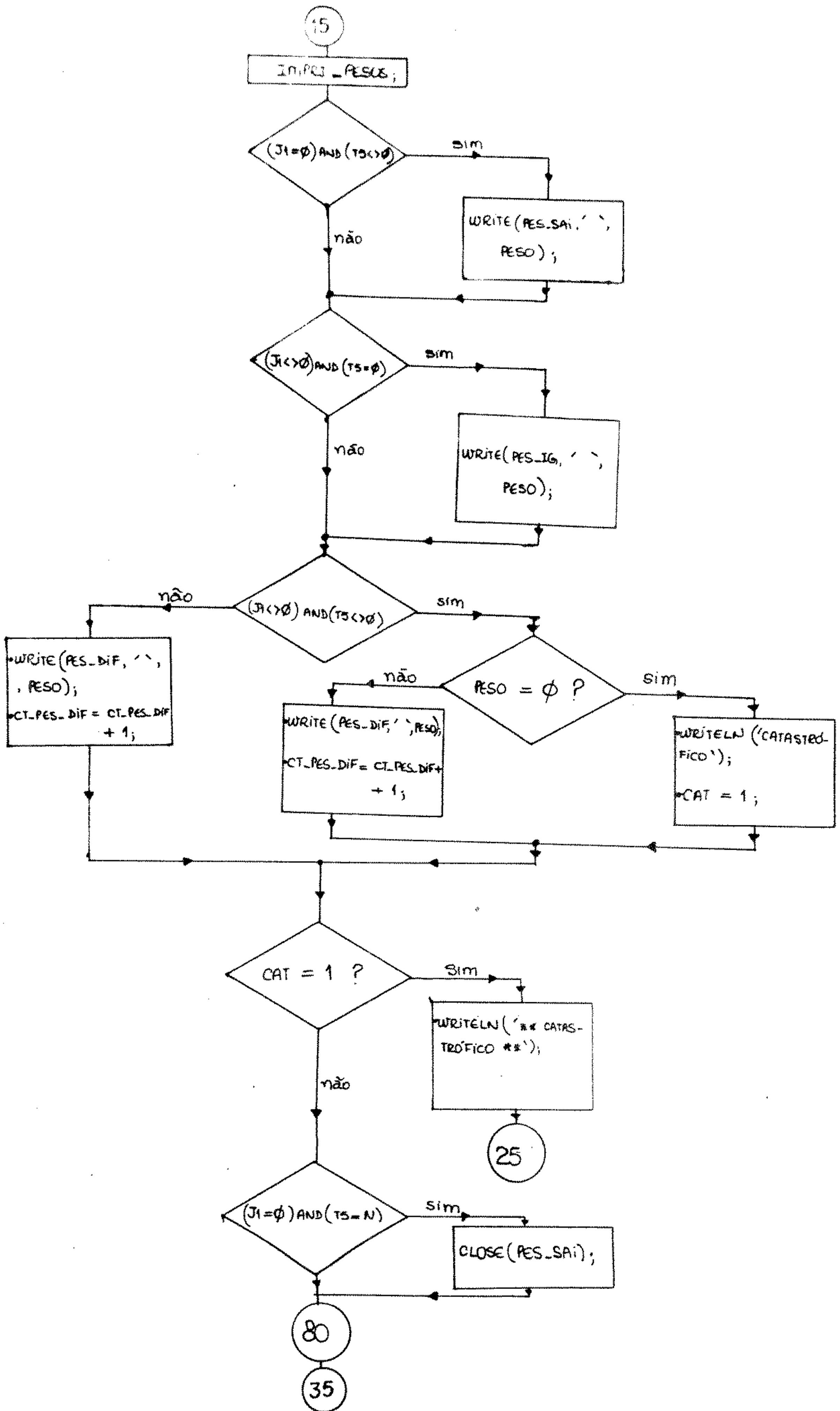

SOMA_VET
I = 1 to n1
V[I] = V1[I] + V2[I];
V[I] > 1
sim
não
R1 = V[I] / 2;
V[I] = V[I] - trunc(2 * R1);
END;

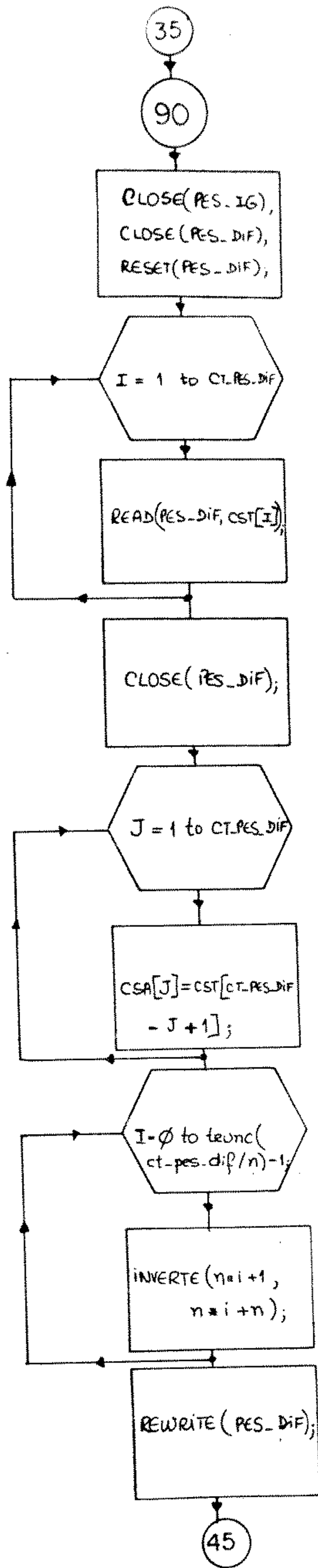

IMPRI_PESOS
escrever:
'do estado :'
I = 1 to b
U∅ [I];
escrever:
'para o estado:'
60

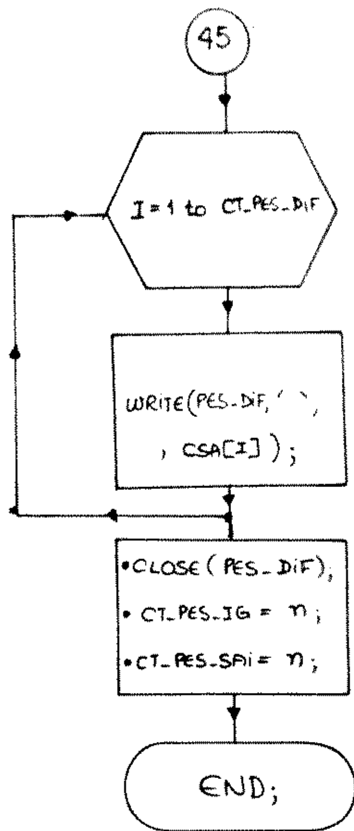

60
J = 1 to b
escrever :
U1[J] ;
escrever :
'Peso = '; PESO
END;

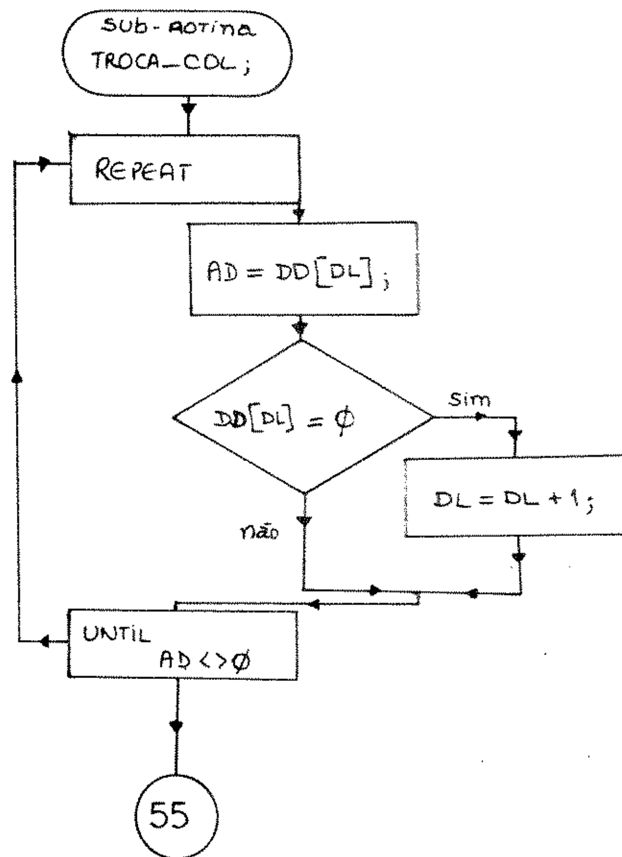

CAT = Ø ;
CT_PES_DIF = Ø ;
J1 = Ø to n
J1 = Ø
sim
REWRITE (PES_SAI);
não
AX4 = J1 ;
TRANSF_BIN;
T4 = 1 to b
UØ[T4] = trunc(Y[T4])
T5 = Ø to n
AX4 = T5 ;
TRANSF_BIN ;
90
80
70

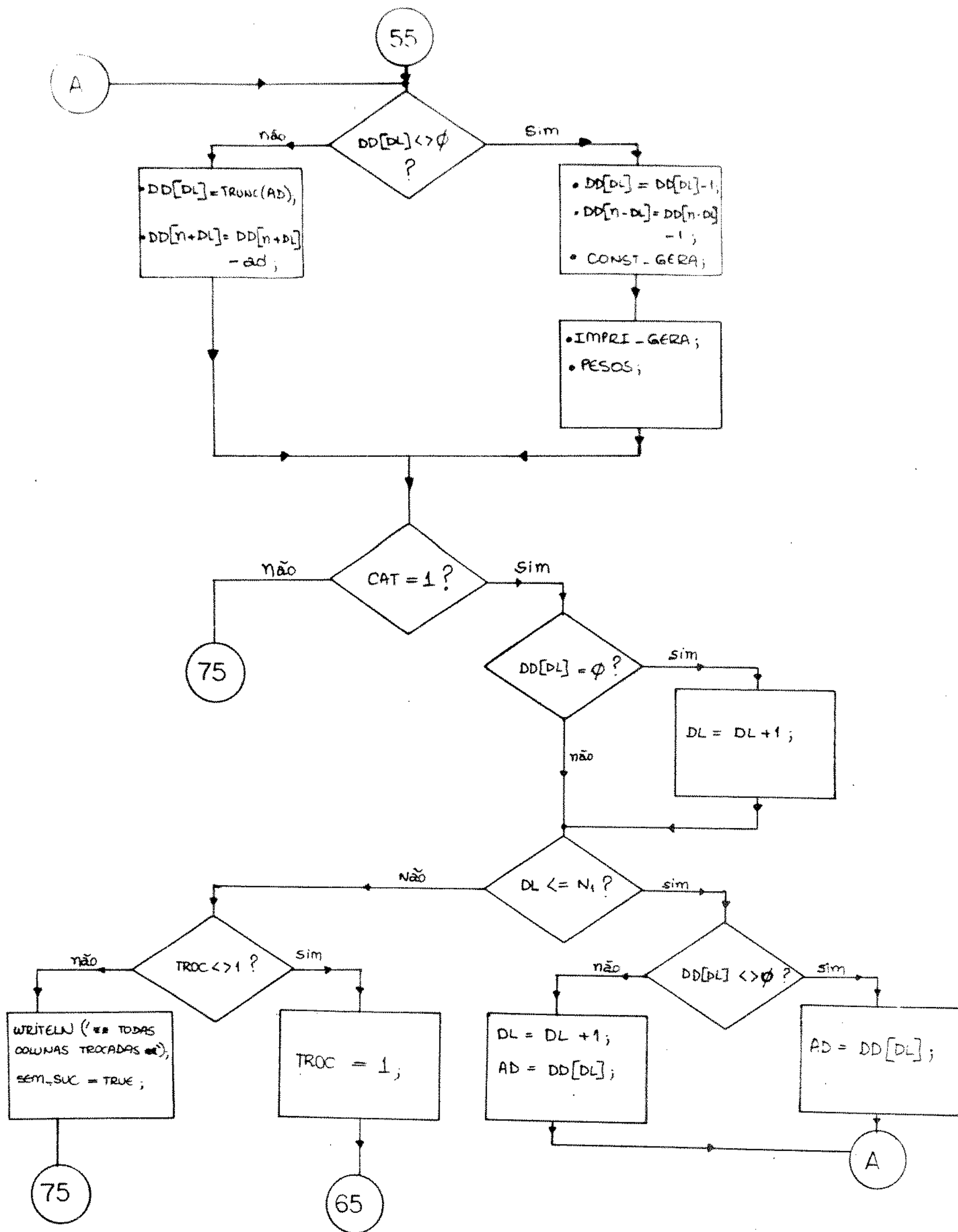

70
t6 = 1 to b
U1[t6]=trunc(y[t6])
PESO = ∅;
MULT_G_U(∅, U1);
T6 = 1 to n1
V2[T6] = V1[T6]
MULT_G_U (n1, U∅);
SOMA_VET;
J= 1 to n1
Peso = Peso +
+ V[J];
15

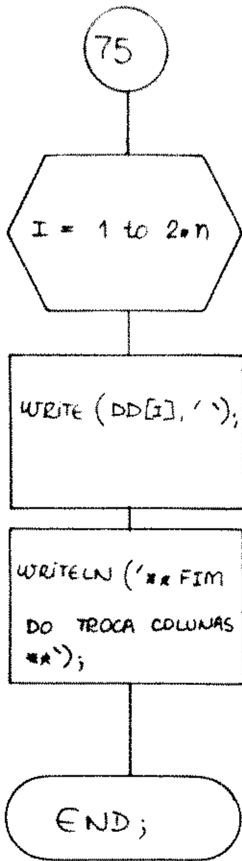

15
IMPRI_PESOS;
(J1=Ø) AND (T5<>Ø)
sim
não
WRITE(PES_SAI,' ',
PESO);
(J1<>Ø) AND (T5=Ø)
sim
não
WRITE(PES_IG,' ',
PESO);
não
(J1<>Ø) AND (T5<>Ø)
sim
WRITE(PES_DIF,' ',
, PESO);
CT_PES_DIF = CT_PES_DIF
+ 1;
não
PESO = Ø ?
sim
WRITE(PES_DIF,' ',PESO);
CT_PES_DIF = CT_PES_DIF+
+ 1;
WRITELN('CATASTRÓ-
FICO');
CAT = 1;
CAT = 1 ?
sim
não
WRITELN('** CATAS-
TRÓFICO **');
25
(J1=Ø) AND (T5=N)
sim
CLOSE(PES_SAI);
80
35

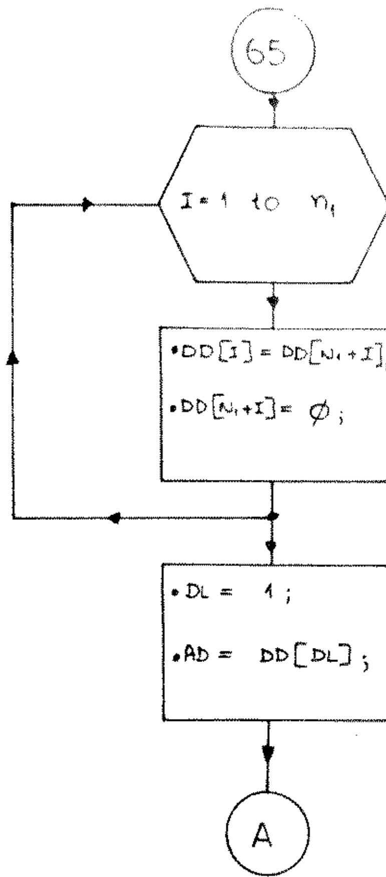

35
90
CLOSE(PES_IG),
CLOSE(PES_DIF),
RESET(PES_DIF);
I = 1 to CT_PES_DIF
READ(PES_DIF, CST[I]);
CLOSE(PES_DIF);
J = 1 to CT_PES_DIF
CSA[J] = CST[CT_PES_DIF
- J + 1];
I = Ø to trunc(
ct_pes_dif/n)-1;
INVERTE(n*i+1,
n*i+n);
REWRITE(PES_DIF);
45

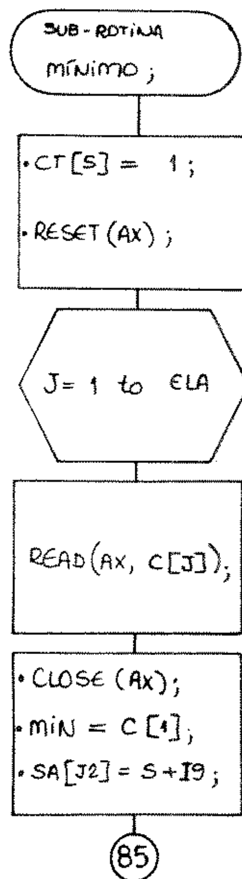

45
I = 1 to CT_PES_DIF
WRITE(PES-DIF, ' ',
, CSA[I]);
•CLOSE(PES_DIF);
• CT_PES_IG = n;
•CT_PES_SAI = n;
END;


SUB-ROTINA
TROCA_COL;
REPEAT
AD = DD[DL];
DD[DL] = Ø
Sim
não
DL = DL + 1;
UNTIL
AD <> Ø
55


55
A
DD[DL] <> ∅ ?
não
sim
• DD[DL] = TRUNC(AD);
• DD[n+DL] = DD[n+DL] - ad;
• DD[DL] = DD[DL]-1;
• DD[n-DL] = DD[n-DL] -1;
• CONST_GERA;
• IMPRI_GERA;
• PESOS;
CAT = 1 ?
não
sim
75
DD[DL] = ∅ ?
sim
DL = DL +1 ;
não
DL <= N1 ?
não
sim
TROC <> 1 ?
não
sim
WRITELN ('** TODAS COLUNAS TROCADAS **');
SEM_SUC = TRUE ;
TROC = 1;
75
65
DD[DL] <> ∅ ?
não
sim
DL = DL +1;
AD = DD[DL];
AD = DD[DL];
A


75
I = 1 to 2*n
WRITE (DD[I], ' ');
WRITELN ('** FIM DO TROCA COLUNAS **');
END;


65
I = 1 to n1
•DD[I] = DD[N1+I];
•DD[N1+I] = Ø;
•DL = 1;
•AD = DD[DL];
A


SUB-ROTINA MÍNIMO;
•CT[S] = 1;
•RESET (AX);
J = 1 to ELA
READ (AX, C[J]);
•CLOSE (AX);
•MIN = C[1];
•SA[J2] = S+I9;
85

85

EST_INIC = S+I9;

K = 2 to Ela

min ≥ C[K]?

sim

min <> C[K]?

não

J2 = J2 + 1;

CT[S] ≥ 2?

sim

J2 = J2 - 1;

não

• SA[J2] = EST_INIC + +K - 1;

• CT[S] = CT[S] + 1;

J2 = J2 + 1;

sim

min = C[K];

CT[S] > 1

não

SA[J2] = EST_INIC + K-1;

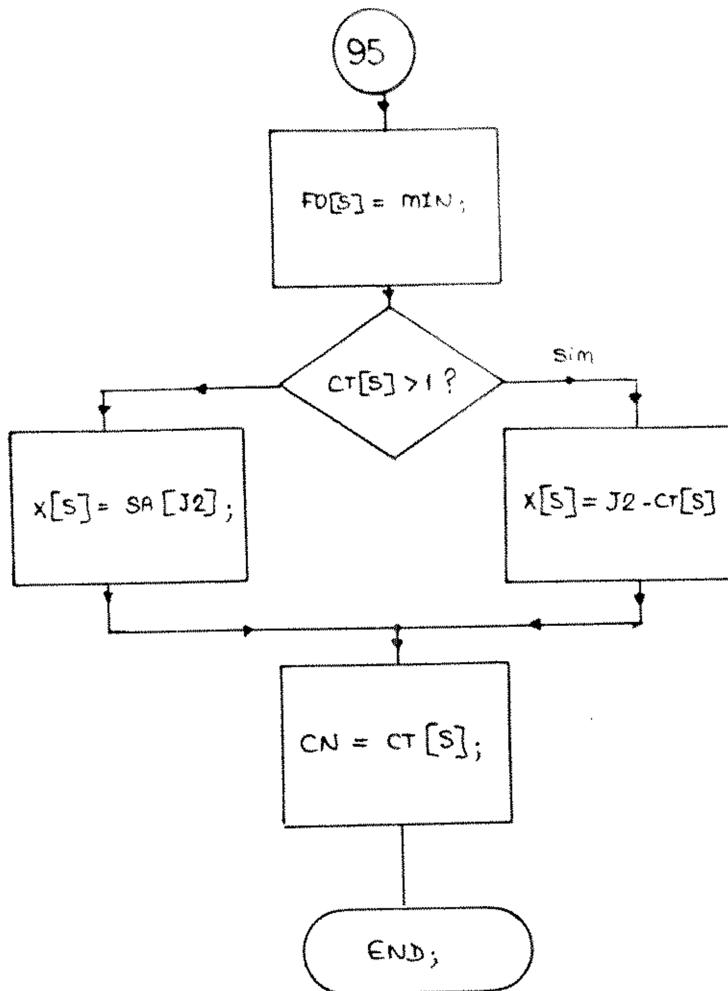
sim

• J2 = J2 - CT[S];

• SA[J2] = EST_INIC + K-1;

• CT[S] = 1;

95



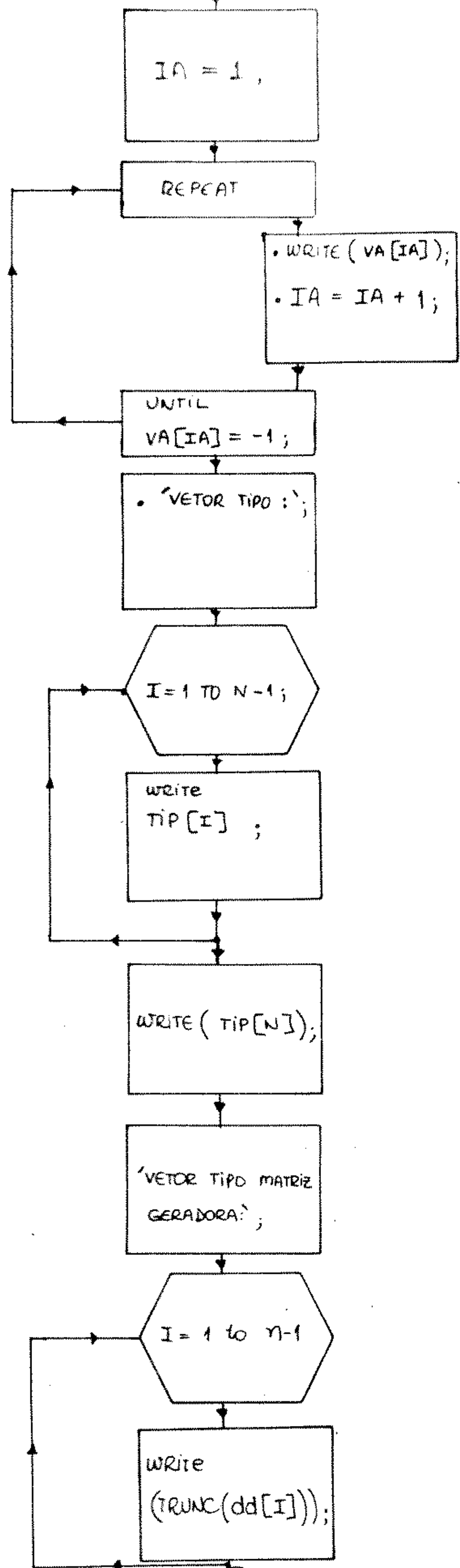
SUB-ROTINA
AP_RESULT;

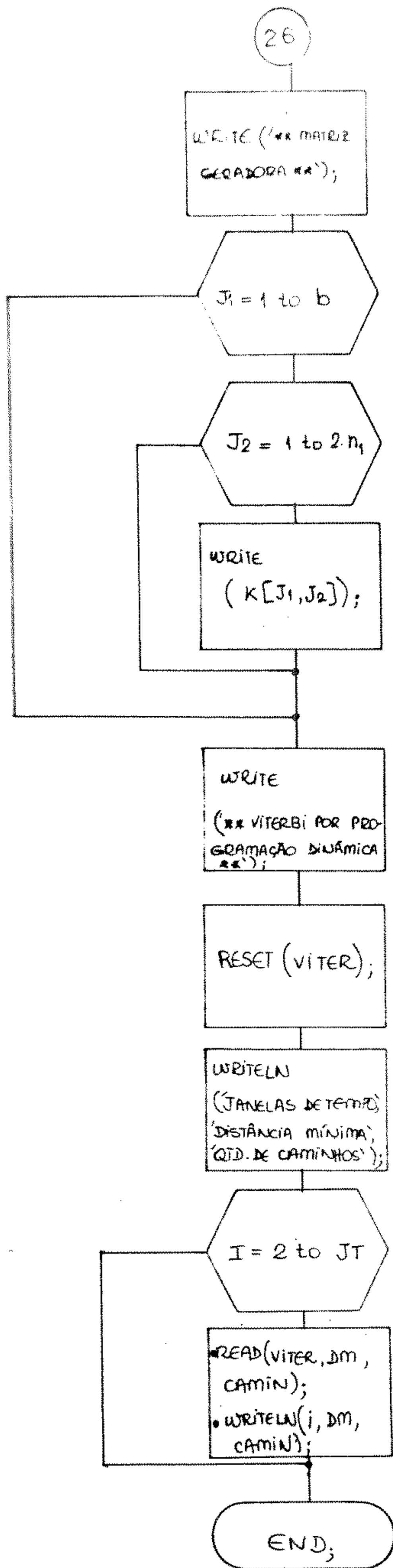
• 'TAXA :', b, '/', n_k;
• 'ORDEM DO CORPO :', q;
• 'MEMÓRIA :', 1;

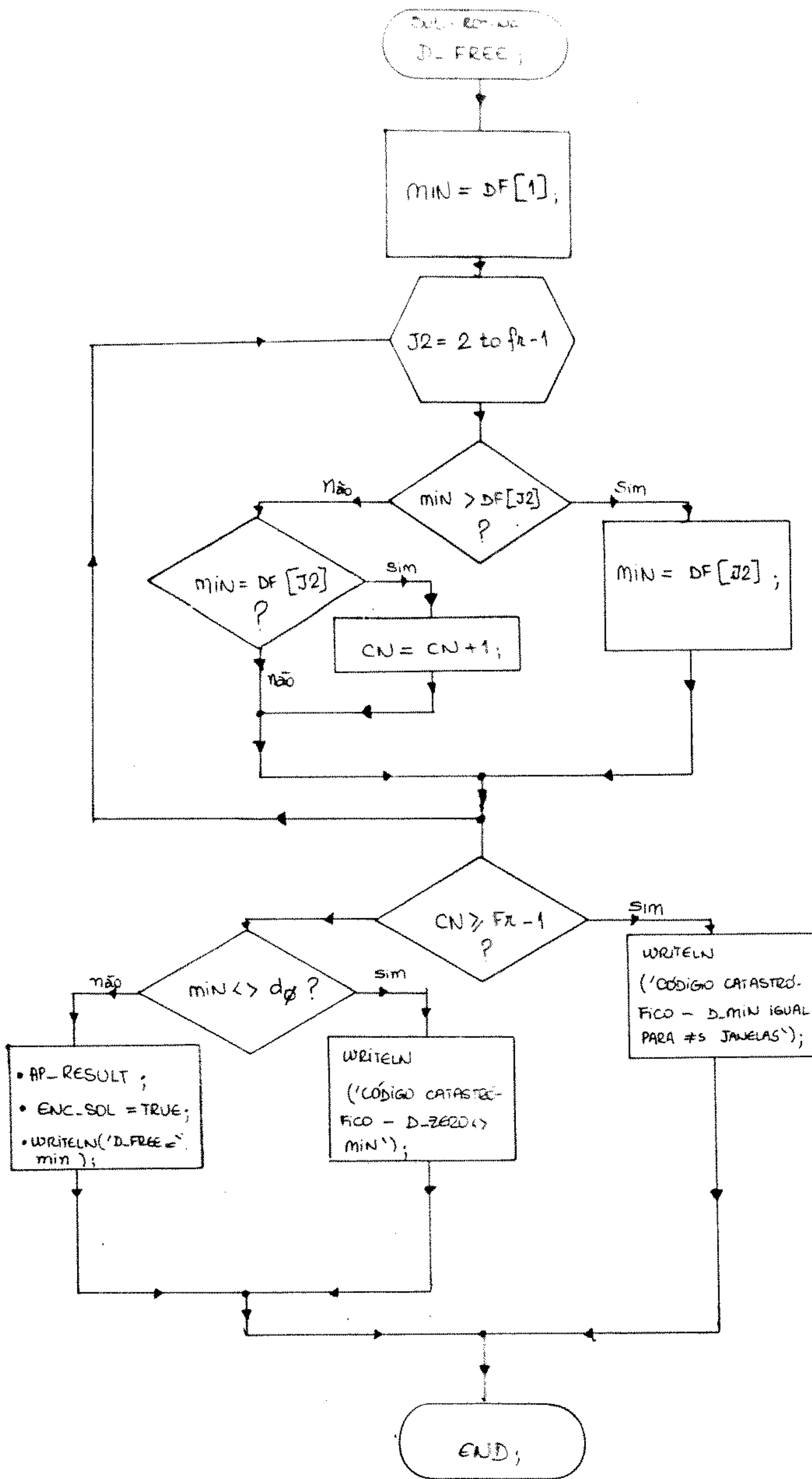
• 'Qtdade de Elementos :', n;
• 'Fluxo Máximo :', fi;
• 'maior d_{min} :', dø;

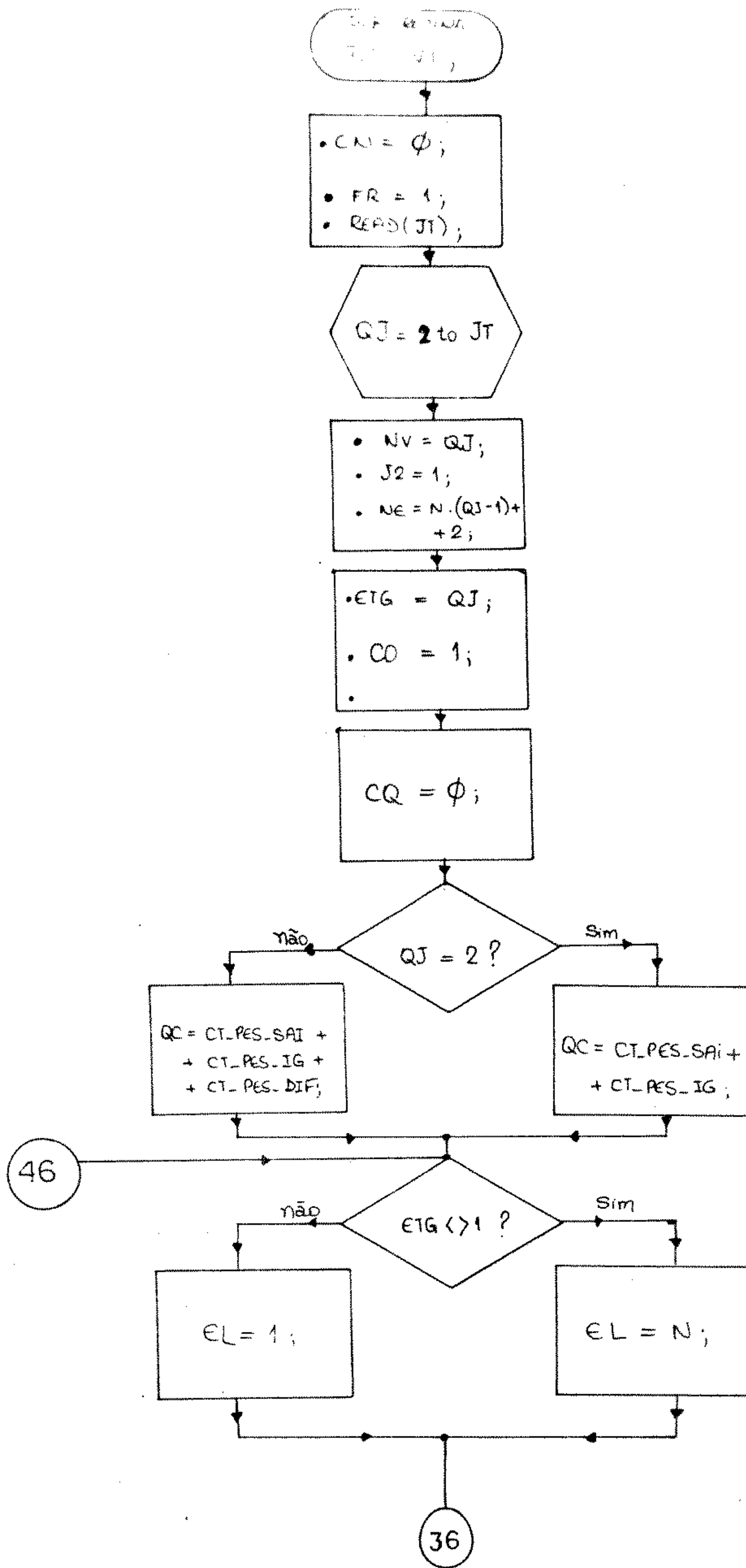
• 'SOLUÇÃO :';
• 'VETOR ESPECTRO DE HAMMING :';

16


16
IA = 1 ,
REPEAT
• WRITE (VA [IA]);
• IA = IA + 1 ;
UNTIL
VA [IA] = -1 ;
• 'VETOR TIPO :' ;
I = 1 TO N - 1 ;
WRITE
TIP [I] ;
WRITE (TIP [N]);
'VETOR TIPO MATRIZ
GERADORA:' ;
I = 1 to n-1
WRITE
(TRUNC(dd[I]));
26


26
WRITE ('** MATRIZ GERADORA **');
J1 = 1 to b
J2 = 1 to 2·n1
WRITE (K[J1,J2]);
WRITE ('** VITERBI POR PROGRAMAÇÃO DINÂMICA **');
RESET (VITER);
WRITELN ('JANELAS DE TEMPO', 'DISTÂNCIA MÍNIMA', 'QTD. DE CAMINHOS');
I = 2 to JT
READ(VITER, DM, CAMIN);
WRITELN(i, DM, CAMIN);
END;


D_FREE ;
MIN = DF[1];
J2 = 2 to fr-1
min > DF[J2] ?
não
sim
MIN = DF[J2] ;
MIN = DF [J2] ?
sim
CN = CN + 1;
não
CN ≥ Fr - 1 ?
sim
WRITELN ('CÓDIGO CATASTRÓ-FICO – D_MIN IGUAL PARA ≠S JANELAS');
min <> dø ?
não
sim
• AP_RESULT ;
• ENC_SOL = TRUE;
• WRITELN('D_FREE =' min);
WRITELN ('CÓDIGO CATASTRÓ-FICO – D_ZERO <> MIN');
END;



• CN = Ø ;
• FR = 1 ;
• READ(JT) ;
QJ = 2 to JT
• NV = QJ ;
• J2 = 1 ;
• NE = N .(QJ-1)+ + 2 ;
• ETG = QJ ;
• CO = 1 ;
CQ = Ø ;
não
QJ = 2 ?
Sim
QC = CT_PES_SAI + + CT_PES_IG + + CT_PES_DIF;
QC = CT_PES_SAi + + CT_PES_IG ;
46
não
ETG <>1 ?
Sim
EL = 1 ;
EL = N ;
36

36

ETG = NV ?

não

I9 = 1 to CL

- S = NE - CO;
- J1 = S + I9;

J = 1 to ELA

ETG = 1 ?

não

READ(PES_DIF, ST);

sim

READ(PES_SAI, ST);

- AVET = ST + FO[J1];
- WRITE(AX, AVET);
- J1 = J1 + 1;

- WRITE(AX, -1);
- mínimo;
- CO = CO + 1;

- ETG = ETG - 1;
- ELA = EL;

56

sim

S = NE - N;

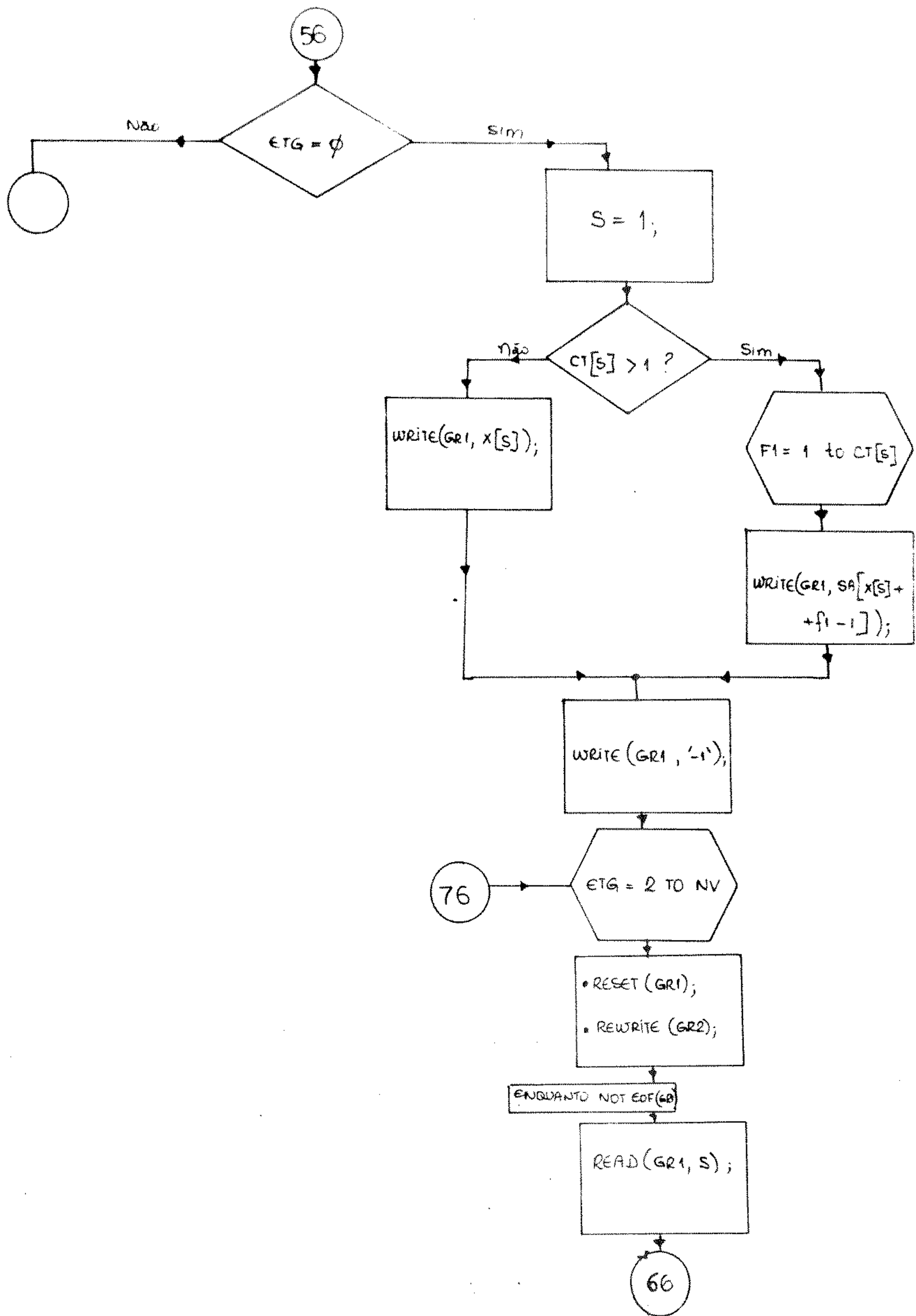
I = 1 to N

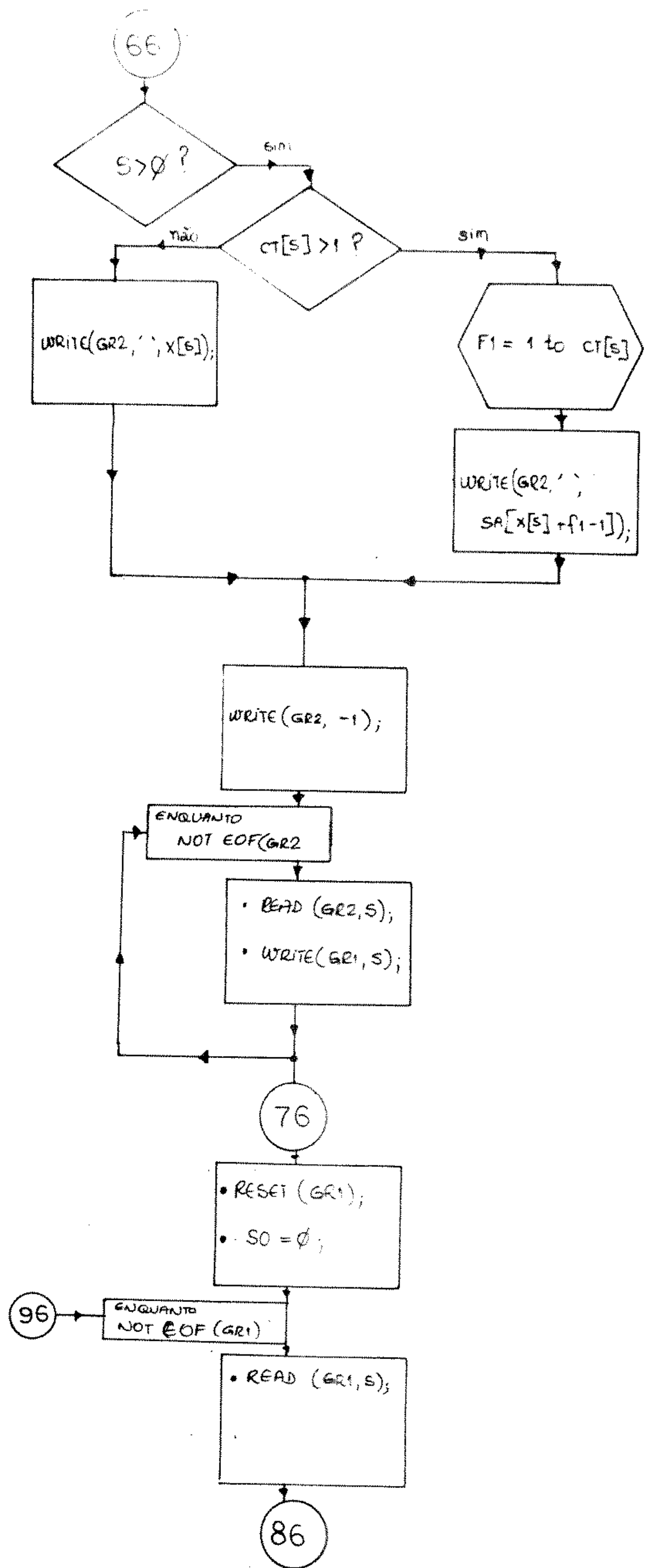
- X[S] = NE;
- READ(PES_IG, FO[S]);

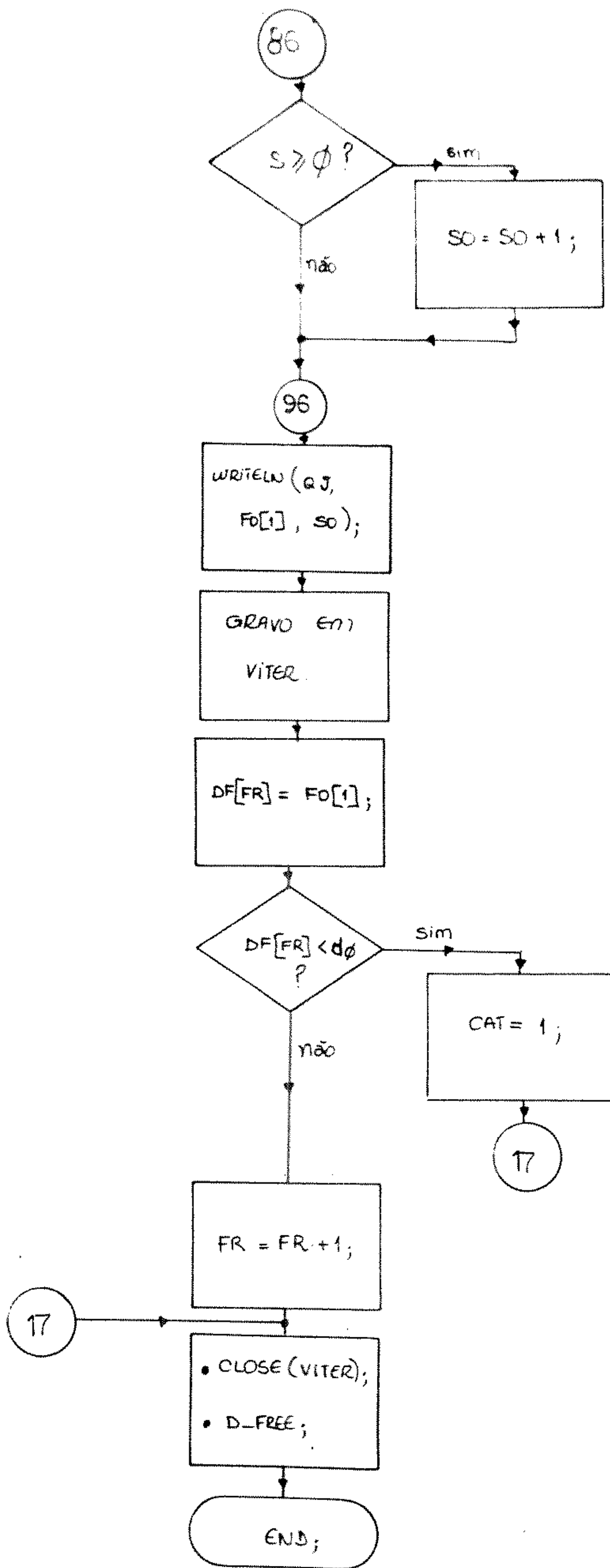
- CT[S] = 1;
- CO = CO + 1;
- S = S + 1;

- CLOSE(PES_IG);
- ETG = ETG - 1;
- ELA = EL;

46


56
ETG = ∅
Não
Sim
S = 1;
CT[S] > 1 ?
Não
Sim
WRITE(GR1, X[S]);
F1 = 1 to CT[S]
WRITE(GR1, SA[X[S] + +f1 - 1]);
WRITE (GR1, '-1');
76
ETG = 2 TO NV
• RESET (GR1);
• REWRITE (GR2);
ENQUANTO NOT EOF(GR)
READ (GR1, S);
66


66
S>Ø ?
sim
CT[S] > 1 ?
não
sim
WRITE(GR2,' ',X[S]);
F1 = 1 to CT[S]
WRITE(GR2,' ',
SA[X[S]+F1-1]);
WRITE(GR2, -1);
ENQUANTO
NOT EOF(GR2
• READ (GR2,S);
• WRITE(GR1,S);
76
• RESET (GR1);
• SO = Ø;
96
ENQUANTO
NOT EOF (GR1)
• READ (GR1,S);
86



86
S ≥ Ø ?
sim
SO = SO + 1 ;
não
96
WRITELN (QJ,
FO[1] , SO);
GRAVO EM
VITER
DF[FR] = FO[1];
DF[FR] < dØ
?
sim
CAT = 1 ;
17
não
FR = FR + 1;
17
• CLOSE (VITER);
• D_FREE;
END;

BIBLIOGRAFIA

BIBLIOGRAFIA

CAPITULO 1

[1] Elias,P. ; "Codding for noise channels.", IRE.CONV.REC., Part.4 , pp 37-47 , 1955.

[2] Lin, N. Lee , "Short unity-memory byte-oriented Binary Convolutional Codes having maximal Free Distance.", IEEE Trans. Inform. Theory, Vol. IT-22, pp. 349-352;

[3] Palazzo Jr., Reginaldo , "Analisys of Periodic Linear and Nonlinear Trellis Codes ", Ph.D. Dissertation, University of California, Los Angeles - USA, 1984.

[4] Pallazo Jr., Reginaldo , "Unity-Memory Codes : A Knapsack Problem? ", 2nd Swedish USSR Int. Workshop on Inform Theory , Granna , Sweden, April 1985.

[5] Palazzo Jr., Reginaldo , "New short Constraint lenght Convolutional Codes Arrived from a Network Flow Approach", IEEE Int. Symposium of Inform. Theory, Bringhton, England , 1985.

[6] Viterbi, A.J. ; " Error bounds for Convolutional Codes and an Assymptotically Optimum Decoding Algorithm.", IEEE Trans. Inform. Theory, Vol. IT-15, pp 260-269, April 1967.

[7] Forney, G. D.; "Convolutional Codes I : Algebraic Structure", IEEE. Trans. Inform. Theory, Vol. IT-16, pp. 720 - 738 , November 1970.

[8] Massey, J. L. ; " Error Bounds for Tree Codes, Trellis Codes , and Convolutional Codes with Encoding and Decoding Procedures", CISM Courses and Lectures n. 216, Springer-Verlang,

CAPÍTULO 2

[1] Lin, S. , " An introduction to Error-Correcting Codes ", Pretence Hall, Englewood Cliffs, N.J. 1970

[2] Viterbi, J. Andrew e Omura, Jim K. , "Principles of Digital Communication and Coding ", Mc Graw - Hill Kogahusha Ltda, International Student Edition, 1979.

[3] Peterson, W.W. e Weldon, E.J. , "Error Correcting Codes", MIT Press, Cambridge , Massachussets , 1972.

[4] Rocha Jr., V.C. e Palazzo Jr.,R. ,"Códigos Corretores de Erros" , ISICTs 87 -International Symposium on Information and Coding Theory , Campinas/SP -Brazil 1987.

CAPÍTULO 3

[1] Hu, T.C. , "Combinatorial Algorithms" , Addison Wesley Publishing CO , 1982.

[2] Bellman, R. , "Notes on the Theory of Dynamic Programming IV - Maximization Over Discrete Sets ", Naval Research Logistics Quartely 3, 67-70 , 1956.

[3] Kendell, K. and S.,Zionts; "Solving Integer Programming Problems by Aggregatin Constraints ", School of Management Working Paper n.155 , S.U.N.Y. at Buffallo , November , 1972.

[4] Balas, E. , "Discrete Programming by the Filter Method" , Operations Research 15(5) , 915-957 , 1967.

[5] Salking,H. M. , "Integer Programming " - Addison-Wesley Publishing Company , 1975.

[6] Hu, T. C., "Combinatorial Algorithms" , Addison-Wesley Publishing CO , 1982.

[7] Colomb,S. W and Baumert, L.D. ,"Backtrack Programming", J.Assoc.Comput.Match 12, 516 - 1965.

[8] Wells, M. B. , "Elements of Combinatorial Computing", Cap. 4. Pergamon Press, 1971.

[9] Kolesar, P. ,"A Branch-and-Bound Algorithm for the Knap sack Problem ", Management Science, 13(9), 723-735 , 1967.

[10] Greemberg, H. and Hegerich, R. , " A Branch search Algorithm for the Knapsack Problem", Management Science , 16(5), 327-332. , 1970

[11] Bradley/Hax/Magnanti, "Applied Mathematical Programming", Addison-Wesley Publishing Company , 1977.

[12] Wagner, H. M. , "Principles of Operations Research", 2nd Edition , Pretence Hall Inc., Englewood Cliffs, New Jersey , 1975.

[13] Bronson, Richard , "Pesquisa Operacional" , Shaum- Mc Graw Hill , 1985.

CAPÍTULO 4

[1] Palazzo Jr., R. , "Unit-Memory Codes : A Knapsack Problem? ", 2nd Swedisch URSS Int. Workshop on Inform. Theory ,Granna , Sweden , April , 1985.

[2] Palazzo Jr, R. , "New Short Constraint Lenght Convolutional Codes Derived from a Network Flow Approach ", IEEE.Int. Symposium of Inform. Theory, Bringhton , England , 1985.

[3] Rosemberg, W.J. , "Structural Properties of Convolutional Codes ", Ph.D. Dissertation, University of California, Los Angeles, 1971.

[4] Palazzo Jr., R. , Young, M.C.P. e Cruz, R.C.F. ,"Uma contribuição ao Problema Combinatorial da Mochila para Códigos de Memória Unitária", Contrato Telebrás 208/87 , RT 45 , ATA, NOV.1987.

[5] Young, M.C.P. , Palazzo Jr., R. e Cruz, R.C.F. ," O algoritmo de Viterbi revisado por Programação Dinâmica", Contrato Telebrás 208/87,RT 51 , ATA , ABRIL 1988.

[6] Palazzo Jr., R , "Analisys of Periodic Linear and Nonlinear Trellis Codes ", PhD Dissertation, University of California, Los Angeles, USA - 1984.

[7] Palazzo Jr., R , Young, M.C.P. and Cruz,R.C.F. ; " On the combinatorial Knapsack Problem in Unity Memory Codes ", International Symposium on Information and Coding Theory , ISICT 87, Campinas/SP. Brazil.

CAPÍTULO 5

[1] SAID; AMIR - "Um método de síntese estática de redes de transmissão para auxílio ao planejamento a longo prazo". DENSIS - FEE/UNICAMP, TESE DE MESTRADO - 1988.

[2] Lin Nan Lee, "Short Unit-Memory byte-oriented Binary Convolutional Codes Having Maximal Free Distance. ", IEEE. Trans. Inform. Theory, Vol. IT 22, pp.349-352 , May 1975.

[3] Paaske, A. "Short Binary Convolutional Codes with maximal free distance for rates 2/3 and 3/4 ", IEEE Trans. Inform. Theory , Vol. IT-24 , pp. 264-268 , March 1978.

[4] Daut, D. G. , Modestino, J. W. and Wismer, L. D. , "New Short Constraint Lenght Convolutional Codes Construction for Selected Rational Rates .", IEEE Trans. Inform. Theory, Vol. IT-28, pp. 794-800, September 1982.

[5] Lauer, G. S. , "Some Optimal Partial Unit-Memory Codes.", IEEE Trans. Inform. Theory , Vol. IT - 25, pp. 240-243, March 1979.